Application Note Book

As the distance between the device and the master becomes longer, the effects of cable length and lumped capacitance must be considered. Using the lowest capacitance cable available will provide the greatest operating distances for a given pullup value. This application note discusses techniques for

DALLAS SEMICONDUCTOR

TABLE OF CONTENTS

MICROCONTROLLERS

SCSI TERMINATION

DIGITAL POTENTIOMETERS

DALLAS SEMICONDUCTOR SALES OFFICES

TIMEKEEPING

DALLAS
SEMICONDUCTOR

Application Note 4
DS1213, DS1216, DS1613
SmartSocket/SmartWatch Options

SMARTSOCKET/SMARTWATCH OPTIONS
The DS1213 SmartSocket, DS1613 Partitionable SmartSocket, and DS1216 SmartWatch product families are designed to accept several user modifications. Please review the DS1213, DS1613, and DS1216 data sheets for normal operation before modification.

DS1213 AND DS1613 SMARTSOCKET OPTIONS
(Reference Figure 1)

Option 1: Power Supply Tolerance
The standard DS1213 and DS1613 socket products are manufactured such that power–fail detection occurs between 4.75 volts and 4.50 volts, giving a 5% supply operating range. This range can be changed to a 10% supply with power–fail detection occurring between 4.50 volts and 4.25 volts. Follow the procedure below:

- cut metal trace labeled "TOL"
- short together metal tabs labeled "T"

Option 2: Density Upgrade
This option applies to the DS1213B and DS1213D SmartSockets only. The DS1213B can be upgraded from 8K x 8 to 32K x 8 memory and the DS1213D can be upgraded from 128K x 8 to 512K x 8 memory by performing the following:

- cut metal traces identified by a hash mark labeled "U"
- short together square metal pads labeled "G"

DS1216 SMARTWATCH OPTIONS
(Reference Figure 2)

Option 1: RESET Disconnect
All DS1216 SmartWatch sockets are manufactured such that the RST signal to the real–time clock is located at pin 1 of the socket. If for a given application the RESET signal is not required, or not desired, this signal can be permanently disconnected as follows:

- cut metal trace labeled "RES"

Option 2: Density Upgrade
This option applies to the DS1216B and DS1216D SmartWatch sockets only. As with the DS1213B and DS1213D, the DS1216B and DS1216D can be upgraded from 8K x 8 to 32K x 8 memory and 128K x 8 to 512K x 8 respectively as follows:

- cut metal traces identified by a hash mark labeled "U"
- short together square metal pads labeled "G"

DS1213 AND DS1613 SMARTSOCKET FAMILY Figure 1
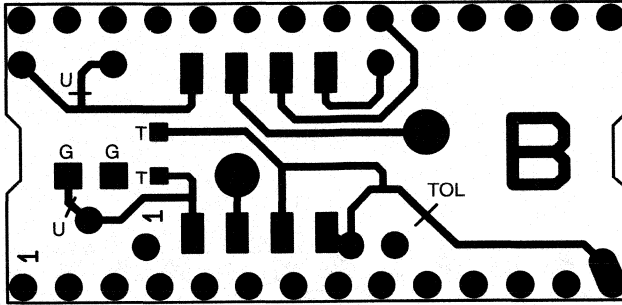
DS1213B
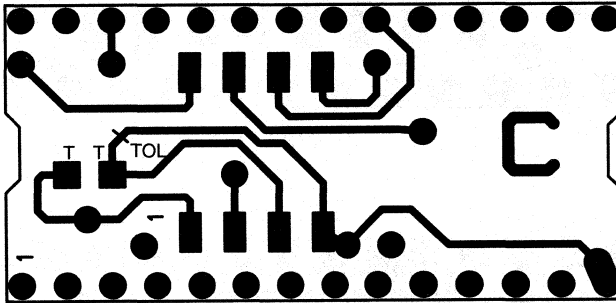
DS1213C
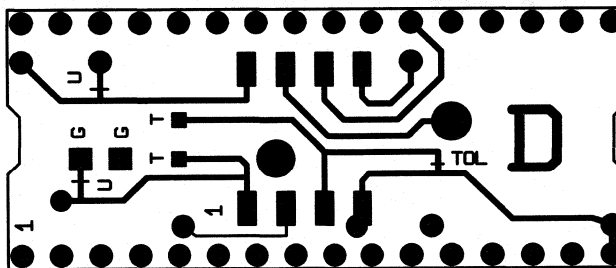
DS1213D

DS1213 AND DS1613 SMARTSOCKET FAMILY Figure 1 (cont'd)
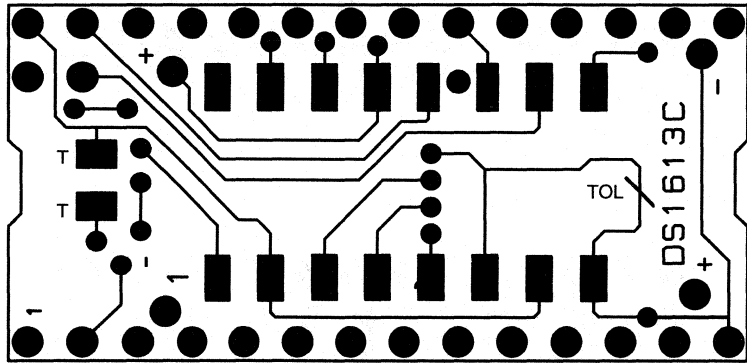
DS1613C
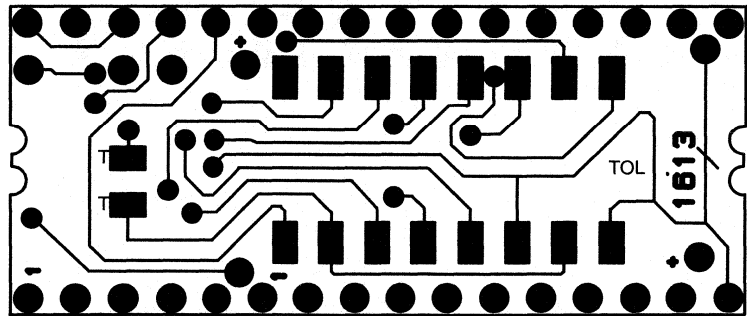
DS1613D

DS1216 SMARTWATCH FAMILY Figure 2
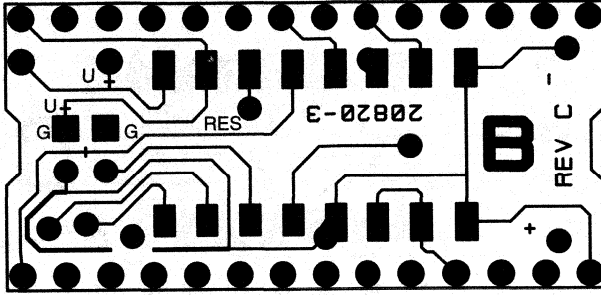
DS1216B
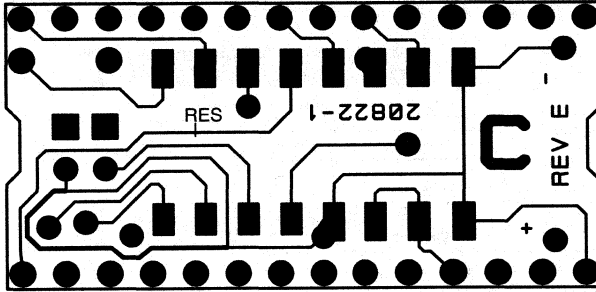
DS1216C
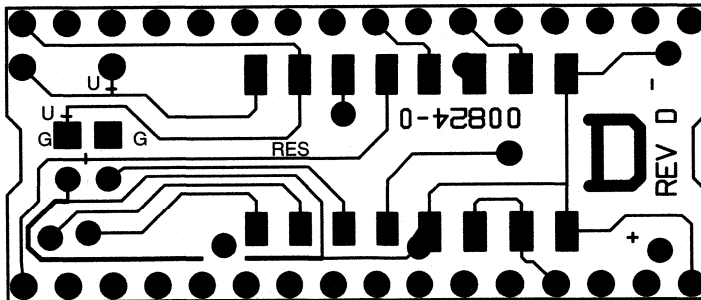
DS1216D

DS1216 SMARTWATCH FAMILY Figure 2 (cont'd)

DS1216D
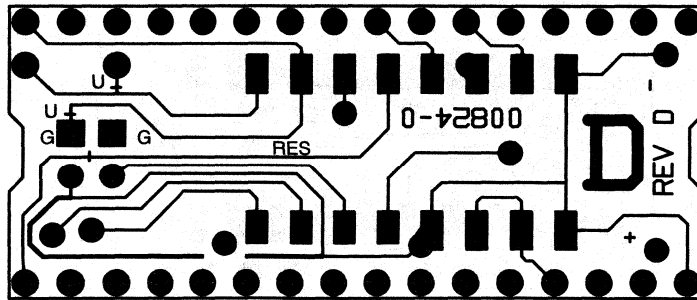
DS1216F
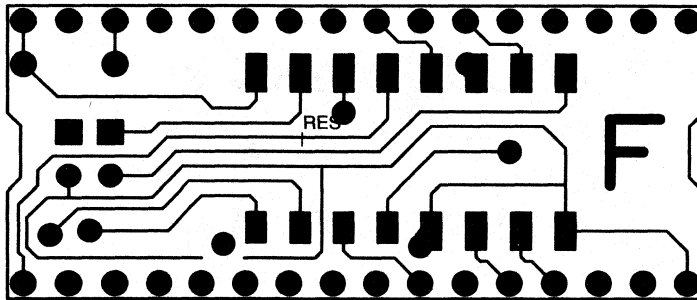
DALLAS
SEMICONDUCTOR

Application Note 30
Recording Power Cycling
Information Using the DS1602/DS1603

PIN ASSIGNMENT

```
V_CC   ●  □ 1
RST       □ 2
DQ        □ 3
NC        □ 4
CLK       □ 5
OSC       □ 6
GND       □ 7
```

DS1603 MODULE

```
RST □  1   8 □  V_CC          RST □  1   8 □  V_CC
DQ  □  2   7 □  X1            DQ  □  2   7 □  X1
CLK □  3   6 □  X2            CLK □  3   6 □  X2
GND □  4   5 □  V_BAT         GND □  4   5 □  V_BAT
```

DS1602   DS1602S
8 PIN DIP (300 mil)   8 PIN SOIC (200 mil)
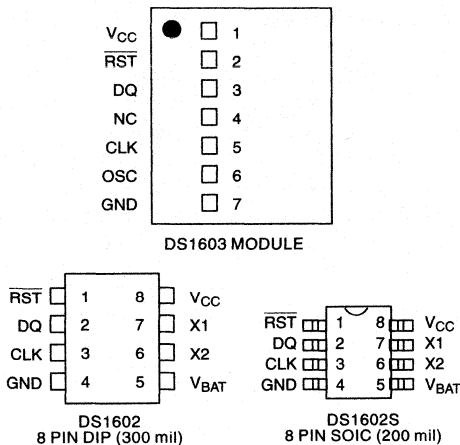
DESCRIPTION

The DS1602 and DS1603 from Dallas Semiconductor offer a simplified hardware solution for keeping time as well as tracking powered up time of a system. The DS1602 and DS1603 can be read and written directly by a microprocessor or microcontroller using simple software; however, a more creative software algorithm can be used to track years, months, days, day of week, time of day, etc. In addition, power–up time and number of power–up cycles can also be tracked using the DS1602/DS1603 with appropriate software.

The continuous counter and power–on counter in the DS1602/DS1603 are 32–bit counters which count in seconds and can be read and written through the DS1602/DS1603 3–wire serial interface. For the most basic implementation

1. the continuous counter will be set once and left to increment until it reaches its maximum value;

2. the powered up counter will be initially cleared once, and left to increment until it reaches its maximum value.

With these two assumptions, each counter has the ability to count up to a maximum value of $(2^{32}-1)$ seconds, or 4.29×10^9 seconds (about 136 years).

For a system that needs 100+ years of continuous timekeeping ability, the entire 32–bit counters may be required; but for users where the maximum continuous counter time required could be about 5 years, the unused counter bits space can be put to better use as memory bits for storing power–up cycling information.

As seen in Figure 1, DS1602/DS1603 can be partitioned to provide a continuous time counter and a power–up time counter with the capability to count up to 4.75 years, leaving the remaining higher bits of the counter available as a read/write nonvolatile memory.

The software implementation requires the use of three registers so a third register must be mapped into the available two as in Figure 1.

An example of how the counters may be used to accomplish this task follows.

MAPPING THREE REGISTERS ONTO TWO Figure 1

NO. OF POWER–UP CYCLES X 128	CONTINUOUS TIME		NO. OF POWER–UP CYCLES	POWERED UP TIME

NO. OF POWER–UP CYCLES	CONTINUOUS TIME	POWERED UP TIME

090794 1/3
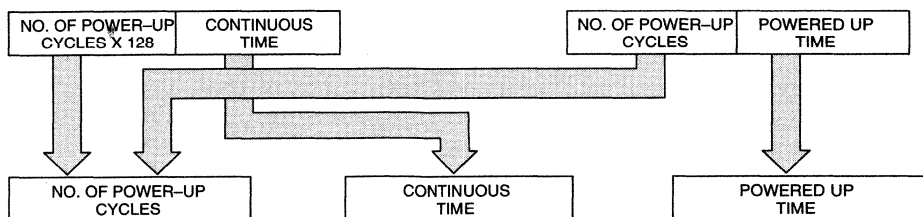
CONTINUOUS COUNTER MAP

Bits 1–24: Remain as continuous timebase measurement, up to 16.7×10^6 seconds or 0.53 years.

Bit 25: Buffer or overflow bit; for when the continuous time counter reaches its maximum value and has not been read and reset by the processor. This bit also serves to separate the counter part of the register from the part which will be used as memory bits.

Bits 26–28: Number of years continuous time has been running, x 0.53.

Bits 29–32: Number of power cycles x 128. These four bits serve as a register which is incremented once for each full count reached in bits 26–32 of the power–up counter.

POWERED UP COUNTER MAP

Bits 1–24: Remain as nonvolatile seconds measurement of powered up seconds, storing up to 16.7×10^6 seconds, or 0.53 years.

Bit 25: Buffer or overflow bit; for when the power–up counter reaches its maximum value and has not been read and reset by the processor. This bit also serves to separate the counter part of the register from the part which will be used as memory bits.

Bits 26–32: The high seven bits of the power–up counter are the 1-127 count storage area for the number of power–up cycles the DS1602 or DS1603 has seen.

With this discipline and the proper software algorithms in place, the DS1602 or DS1603, power–on time and continuous time are maintained by the DS1602/DS1603's self-contained counters while the number of power–up cycles and years of elapsed time x 0.53 is maintained in the higher order bits of the counters which are used as memory.

This implementation requires that a microcontroller must be prepared to read/write the DS1602 or DS1603 at least once every year.

For Continuous Time Tracking

When the lower 24 bits of the continuous counter have exceeded 0.53 years and set bit 25 to 1, the controller must read the continuous counter, determine the status of bit 25, and if 1, clear the bit and increment the value in bits 26–28 by one half year. If bit 25 is not set, the lower 24 bits of the register have yet to reach 0.53 years and can continue counting.

Once the value in bits 26–28 has reached 111, or 7 x 0.53 years, the continuous time counter can continue to count up to 1.06 years in the lower 24 bits plus the overflow of 0.53 in bit 25 for a maximum value of 9 x 0.53 years.

FOR CONTINUOUS TIME TRACKING Figure 2
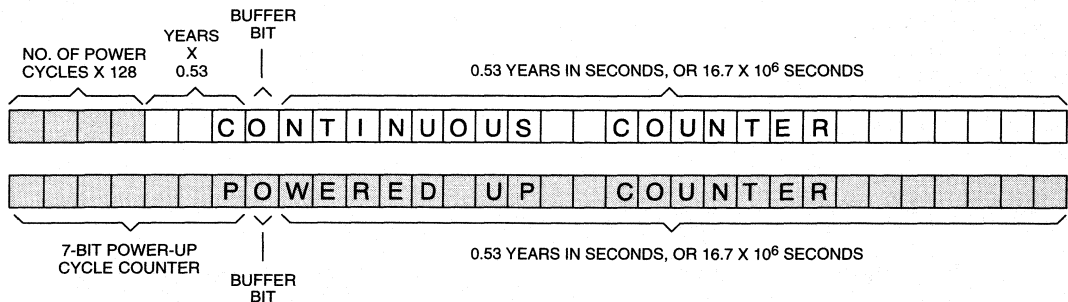
For Power-up Time Tracking

When the lower 24 bits of the power–up counter have exceeded 0.53 years and set bit 25 of the counter to 1, the controller must read the power–up counter, determine the status of bit 25, and if 1, clear the bit and store the value in external memory so that the power–up counter can continue to count. The maximum power–up time that can be stored in this way is 2 x 0.53 years within the DS1602/DS1603.

FOR POWER-UP TIME TRACKING Figure 3
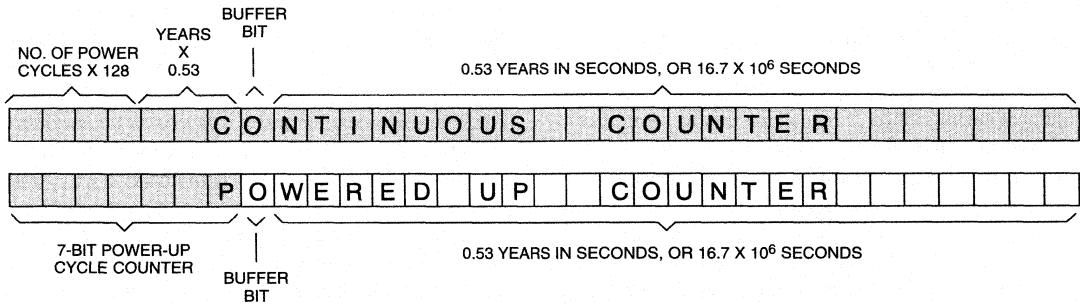
For Number of Power-up Cycles Tracking

Performing this function with the DS1603 or DS1602 is primarily a software task. When originally written with a start value or cleared, bits 25–32 of the power–up counter must be set to 0. Upon each power–up thereafter, the controller or processor connected to the DS1603 must read the power–up counter and examine the value stored in the high seven bits. If the value is less than 1111111, then the controller must increment the value and write it back to the seven higher order bits. If the value in the higher order bits is 1111111, the controller must set the value of 0000000, read the value in the high four bits of the continuous time counter, increment it by one, and write the new value back to the high four bits. Using this software algorithm, the DS1603 or DS1602 can be used to record and store up to 2,047 power cycles.

FOR NUMBER OF POWER-UP CYCLES TRACKING Figure 4

DALLAS
SEMICONDUCTOR

Application Note 52
Using the Dallas Phantom Real Time Clocks
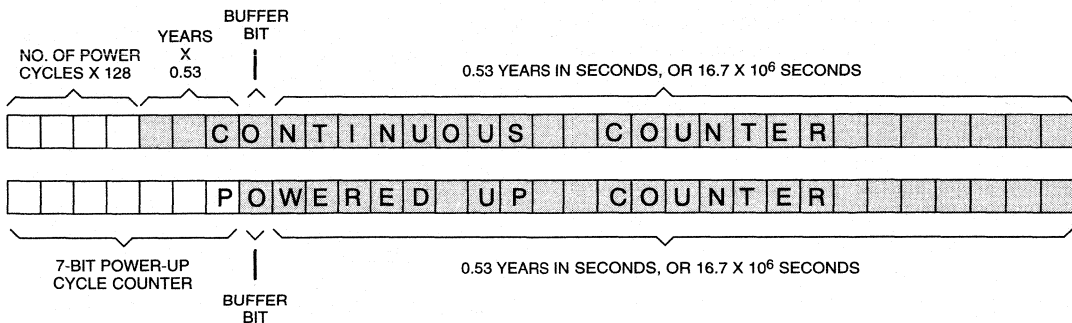
DESCRIPTION

The Dallas Phantom Real Time Clocks are a family of devices that offer the combination of a transparent CMOS timekeeper and a nonvolatile static RAM meeting the standard JEDEC bytewide pinouts. Some varieties of the Dallas Phantom Real Time Clocks also provide a transparent CMOS timekeeper for use with ROM. The timekeeper is transparent to the RAM/ROM memory map because it does not occupy any of the existing RAM/ROM locations. These devices are termed "Phantom" because the timekeeper is accessed only when a predetermined 64–bit pattern has been received by the device. When the timekeeper is not being accessed, the RAM/ROM can be accessed normally. The timekeeper keeps track of hundredths of seconds, seconds, minutes, hours, day, date, month, and year information. In the absence of power, a lithium energy source maintains the timekeeping operation and retains data in the CMOS static RAM.

FAMILY OVERVIEW

DS1215

The heart of the Dallas Phantom Real Time Clock family is the DS1215 Phantom Time Chip. This integrated circuit is a combination of a CMOS timekeeper and a nonvolatile memory controller. In the absence of power, an external battery maintains the timekeeping operation and retains data in the CMOS static RAM. The watch keeps track of hundredths of seconds, seconds, minutes, hours, day, date, month, and year information. The last day of the month is automatically adjusted for months with less than 31 days, including correction for leap year every four years. The real time clock operates in one of two formats: 12–hour mode with an AM/PM indicator or a 24–hour mode . The nonvolatile controller supplies all the necessary support circuitry to convert a CMOS RAM to a nonvolatile memory. The DS1215 can also be used to provide timekeeping functions with ROM.

DS1216

Stemming from the DS1215 are the DS1216 Smart-Watch Intelligent Sockets. The SmartWatch is a 600 MIL wide DIP socket with a built–in DS1215 (providing timekeeping functions and a nonvolatile RAM controller), an embedded lithium energy source, and a 32.768 kHz crystal. When the socket is mated with a bytewide CMOS static RAM, it provides a complete solution to problems associated with memory volatility and uses a common energy source to maintain time and date. The DS1216 can also be mated with a ROM to provide time-keeping capability only. Figures 1 and 2 show the basic interface of a SmartWatch with RAM inserted and a SmartWatch with ROM inserted, respectively.

DS1243Y, DS1244Y, DS1248Y

The DS124x Nonvolatile SRAM with Phantom Time Clock modules are the final members of the Dallas Phantom Real Time Clock family. These devices are fully nonvolatile static RAM with a built–in Phantom clock, embedded lithium energy source, and 32.768 kHz crystal. These devices operate identical to a DS1216 with a RAM inserted. The DS124x Nonvolatile SRAM with Phantom Time Clock modules will maintain over 10 years of data retention in the absence of power.

Perhaps the best way to sum up the Dallas Phantom Real Time Clock family is as follows. The DS1215 Phantom Time Chip is the basic building block that provides timekeeping and a nonvolatile memory controller. The DS1216 then adds a crystal and lithium energy source to the DS1215 and encapsulates them all in a socket that will accept either a RAM or a ROM. Finally, the DS124x modules contain both nonvolatile RAM and timekeeping features in a ready–to–use package.

The entire Dallas Phantom Real Time Clock family is shown in Table 1.

Table 1

DS1215	Phantom Time Chip
DS1216B	SmartWatch/RAM 16K/64K
DS1216C	SmartWatch/RAM 64K/256K
DS1216D	SmartWatch/RAM 256K/1M
DS1216E	SmartWatch/ROM 64K/256K
DS1216F	SmartWatch/ROM 64K/256K/1M
DS1243Y	64K NV SRAM with Phantom Clock
DS1244Y	256K NV SRAM with Phantom Clock
DS1248Y	1024K NV SRAM with Phantom Clock

APPLICATION

The Dallas Phantom Real Time Clock family offers two features that will greatly enhance a system. The first feature is nonvolatile RAM capability. The second feature is that the Phantom Clock is transparent to the RAM and therefore timekeeping capacity can be added to a system without changing the existing hardware. All that is required is an existing bytewide memory socket. The retrofit capability is maximized through the transparent interfaces supported by the Phantom Time Chip. Also advantageous to the designer is that an upgrade path is provided to higher RAM densities with the DS124x modules or to higher RAM/ROM densities with the DS1216.

It should be mentioned that there is some software overhead that is associated with having timekeeping functions that are transparent to RAM as will be discussed in detail below. If it is determined that a transparent clock is not necessary, then the DS164x Nonvolatile Timekeeping RAM family could offer an excellent solution to your timekeeping and nonvolatile SRAM needs. These offer nonvolatile SRAM with the Real Time Clock registers located in the RAM address space. Another possible solution are the DS1386 or DS1486 RAMified Watchdog Timekeepers which offer nonvolatile RAM and Real Time Clock as well as a few extra features including alarm function and Watchdog timer.

OPERATION

Nonvolatile RAM Operation

One important feature of the Dallas Phantom Real Time Clocks is that the nonvolatile RAM can be used to store system configuration data and since the clock is transparent to the RAM, no memory is lost to timekeeping needs. When the Phantom Clock is not accessed, the $\overline{CE}$ signal is passed on to the chip enable of the memory. Reading and writing to the RAM is identical to that of a standard RAM chip. Figure 1 illustrates a typical RAM/Time Chip interface. Note that this is the basic interface used for the DS1216 SmartWatch/RAM and the DS124x.

The Phantom Real Time Clock family performs circuit functions required to make a CMOS RAM nonvolatile. First, a switch is provided to direct power from the battery or V_{CC} supply depending on which is greater. The second function provided is power–fail detection. Power–fail detection occurs when V_{CCI} falls below V_{TP}, which is equal to $1.26 \times V_{BAT}$. When V_{CCI} goes out of tolerance, a comparator outputs a power–fail signal to the chip enable logic. The third function accomplishes write protection by holding the chip enable signal to the memory ($\overline{CEO}$) within 0.2 volts of V_{CC} or battery as long as V_{CC} is out of tolerance. During nominal power supply conditions the memory chip enable signal ($\overline{CEO}$) will track the chip enable signal ($\overline{CEI}$) sent to the socket with a maximum propagation delay of 20 ns.

Finally, an important consideration when using the DS1216 is to select a RAM that draws no more than a maximum of 1 µA. If the RAM data retention current is larger than 1 µA, the device will not meet the data retention expectations of more than 10 years at 25°C. In the 10 year data retention calculation for the DS1216, it is assumed that system power will be on 20% of the time. Perhaps the best way to insure that a data retention of 10 years at 25°C is met is to use one of the DS124x modules with self–contained nonvolatile RAM. The DS124x modules will insure data retention for 10 years regardless of how often the system power is on.

ROM Operation

The DS1215 and DS1216(E/F) can also be used in conjunction with a ROM. A typical ROM/Time Chip interface is illustrated in Figure 2. In this configuration, the ROM/$\overline{RAM}$ pin is connected to V_{CCO} to select the ROM mode of operation. Since ROM is a read–only device that retains data in the absence of power, battery

back–up and write protection is not required. As a result, the chip enable logic will force $\overline{CEO}$ low when power fails. The real time clock does retain the same internal nonvolatility and write protection as described in the RAM mode.

Real Time Clock Operation

The block diagram of Figure 3 illustrates the main elements of the Phantom Clock. Communication with the Phantom Clock is established by pattern recognition of a serial bit stream of 64 bits which must be matched by executing 64 consecutive write cycles containing the proper write data as shown in Figure 4. All accesses which occur prior to recognition of the 64–bit pattern are directed to memory via the chip enable output pin ($\overline{CEO}$). After recognition is established, the next 64 read or write cycles either extract or update data in the Phantom Clock and $\overline{CEO}$ remains high during this time, disabling the connected memory.

Data transfer to and from the Phantom Clock is accomplished with a serial bit stream under the control of chip enable input ($\overline{CEI}$), output enable ($\overline{OE}$), and write enable ($\overline{WE}$). Initially, a read cycle using the $\overline{CEI}$ and $\overline{OE}$ control of the Phantom Clock starts the pattern recognition sequence by moving a pointer to the first bit of the 64–bit comparison register. Next, 64 consecutive write cycles are executed using the $\overline{CEI}$ and $\overline{WE}$ control of the Phantom Clock. These 64 write cycles are used only to gain access to the Phantom Clock. However, the write cycles generated to gain access to the Phantom Clock are also writing data to a location in the mated RAM. The preferred way to manage this requirement is to set aside just one address location in RAM as a scratch pad for the Phantom Clock.

When the first write cycle is executed, it is compared to bit 0 of the 64–bit comparison register. If a match is found, the pointer increments to the next location of the comparison register and awaits the next write cycle. If a match is not found, the pointer does not advance and all subsequent write cycles are ignored until a read cycle is encountered which resets the comparison register pointer to the beginning of the 64–bit comparison register. If a read cycle occurs at any time during the pattern recognition process, the present sequence is aborted and the comparison register pointer is reset. Pattern recognition continues for a total of 64 write cycles as described above until all the bits in the comparison register have been matched (this bit pattern is

shown in Figure 4). With a correct match of the 64 bits, the Phantom Clock is enabled and data transfer to or from the timekeeping registers can proceed.

The next 64 cycles will cause the Phantom Clock to either receive or transmit data, depending on the level of the $\overline{OE}$ pin or the $\overline{WE}$ pin. Data will either be written to or read from the eight Phantom Clock registers shown in Figure 5. Cycles to other locations outside the memory block can be interleaved with $\overline{CE}$ cycles without interrupting the pattern recognition sequence or data transfer sequence to the Phantom Clock.

Figure 6 offers an example of pseudo code for both accessing the Phantom Clock embedded in a RAM through pattern recognition and interfacing with the clock registers. Another source code example is given in Figure 7. This code is used for interfacing with the 8051 microcontroller. Also, refer to the data book for timing diagrams for both read and write cycles.

Interfacing the Phantom Time Clock with a ROM is somewhat different from that of a RAM. This is due to the fact that no writes are made to a ROM. Since there are no $\overline{WE}$ or data in signals associated with the ROM, the Phantom Time Clock instead uses two address lines to access the real time clock as can be seen in Figure 2.

In summary, the operation of the Phantom Clocks is best defined as operating in two different modes. The first being the pattern match mode. In this mode, the Phantom Clock is transparent to the system yet monitors communication to the RAM waiting for a match of it's 64–bit access pattern. When the 64–bit access pattern has been written, the Phantom Clocks enter the clock access mode. In this mode, the eight phantom clock registers are available to be written or read and will stay in this mode until all eight registers have been accessed, until a reset has been executed, or until a power fail.

TROUBLESHOOTING

The Dallas Phantom Real Time Clocks have proven to be highly reliable and meet the published specifications. However, in the course of development, several common difficulties could be experienced. To reduce these difficulties, Dallas Semiconductor has gathered the common difficulties and pitfalls into a troubleshooting guide to assist users.

COMMON DIFFICULTIES

Cannot Access Clock Registers

Several items can cause this phenomena.

1. Comparison register pointer has has not been set to the first bit. The Phantom Real Time Clock hides behind the SRAM and waits for a match to it's 64–bit access pattern. In this mode (the pattern match mode), every write operation to the RAM will be interpreted as an attempt to match the access pattern by matching the value written to DQ0 (D for the DS1215) to the pattern bit pointed to by the pattern match pointer. It is possible that a partial match of the pattern can occur during normal operation of a system. It is best to assume that there is a partial match of the access pattern and that the comparison register pointer is not pointing to the first bit of the match pattern. Therefore, the comparison register pointer must be reset to the first pattern bit before writing the match pattern. This is accomplished by performing one read operation of the RAM before writing the match pattern.

2. Device is in clock access mode after system reset or interrupt. It is possible that during the course of previous operation, the Phantom Clock had been accessed, but had not gone back into pattern match mode before a system reset or interrupt occurred. In other words, data bits would be written to or read from the Phantom Clock registers rather than the RAM. A solution to this problem is to execute 65 consecutive read cycles immediately after an interrupt or system reset. This will insure that the device is taken out of the clock access mode (by reading a maximum of 64 bits) and will reset the comparison register pointer.

3. Access pattern has been input in reverse order. Insure that the pattern is input in the following order. Start with bit 0 of byte 0 continuing to bit 7 of byte 7.

4. Device is being reset. Insure that the device is not accidentally being reset. This can especially be a problem with the DS1216C, DS1216D, and DS1244Y where the reset pin is shared with an address pin. In this situation, that particular address line must never be taken low unless the reset bit (byte 4, bit 4) of the Phantom Clock is disabled, otherwise the device will be reset and the data transfer will be aborted.

5. Device is in constant write protect mode. If only one battery is being used for the DS1215, ensure that the BAT2 pin is grounded. If this pin remains floating it is possible that the device will think that a power–fail condition has occurred. This is due to the method in which power–fail is detected. A power–fail is determined to have occurred when V_{CC} is less than V_{TP}, which is equal to 1.26 x V_{BAT}. If V_{BAT2} is floating, it is possible that the node could float to a value such that V_{TP} is equal to V_{CC} and thus the device will always be in a write protect mode.

Device Will Not Oscillate

1. Oscillator enable bit is disabled. Insure that the oscillator enable bit (bit 5 of byte 4) is at logic 0.

2. Wrong crystal used (DS1215). Insure that the correct crystal is being used. It is very important that a crystal with a load capacitance of 6 pF is used. Dallas Semiconductor recommends Seiko part number DS–VT–200, Daiwa part number DT–26S, or equivalent.

3. Poor crystal connection (DS1215). To insure the greatest performance, insure that the crystal is placed as close as possible to the crystal input pins. It should also be mentioned that the DS1215 does not require load capacitors or feedback resistors.

Note: It should also be mentioned that it is difficult to determine if the device is oscillating by trying to measure the frequency with an oscilloscope probe. This is because of the loading caused by the scope probe which can kill the oscillator.

Timekeeping Inaccurate

1. Input pins driven higher than V_{CC}. It is very important to insure that input pins never go above V_{CC}. If any input is allowed to go above V_{CC}, it is possible that the oscillator may be briefly stopped which will cause the device to lose time.

2. Wrong crystal used (DS1215). For best accuracy, insure that the correct crystal is used.

RAM is Losing Data when Powered Down

This problem can occur especially in NMOS processors which become unstable at a higher voltage than CMOS processors. This problem manifests itself in the method in which power–fail is determined. Write protection is asserted when V_{CC} drops below V_{TP}, which is equal to $1.26 \times V_{BAT}$. Typically, the battery has a voltage of ~3.0V which leads to a V_{TP} of 3.78V. Therefore, in a power–down situation, if the processor becomes unstable at a V_{CC} of greater than ~3.78V (which is often the case for an NMOS processor), a spurious write cycle could corrupt the data in the Phantom Clock. The solution to this problem is to ensure that the processor is reset before it becomes unstable and thus prevent any unwanted writes from being executed. This can be accomplished by monitoring V_{CC} by one of Dallas Semiconductor's power monitors (the DS123x family) which generate a reset signal when V_{CC} is out of tolerance.

Cannot Read Consecutive Hundredths of Seconds

It is not possible to read consecutive hundredths of seconds because the access time to read the clock registers is too long.

COMMON PITFALLS

1. Device needs separate read and write signals. The Dallas Phantom Real Time Clocks were designed with Intel timing in mind. Therefore it is necessary to have separate read and write signals. It should be further stressed that simply complementing one signal to arrive at the other is not sufficient because this will cause the pattern match pointer to be reset during each write cycle since the $\overline{OE}$ signal will toggle whenever the $\overline{WE}$ signal toggles.

2. Battery attachment (DS1215). Any battery attached to the BAT1 or BAT2 pins must be connected directly to the pin. It should be noted that a diode should not be connected between the battery input pin and the battery. This is not necessary because internal reverse charging current protection circuitry is provided and is UL recognized (#E99151).

3. ROM/$\overline{RAM}$ pin (DS1215). Insure that the ROM/$\overline{RAM}$ pin is set to the correct value.

4. Reading and writing to clock registers. It is important that all 64 bits be read or written to when the clock registers have been accessed. If this is not done, the device will remain in the clock access mode.

5. Do not water wash DS1216 Intelligent Sockets. Water washing for flux removal must not be performed on the DS1216 Intelligent Sockets because contaminants in the water can cause discharging of the internal lithium energy source.

6. Crystal selection (DS1215). A 32.768 kHz quartz crystal, Seiko part number DS–VT–200, Daiwa part number DT–26S, or equivalent should be used. The crystal selected for use must have a specified load capacitance of 6 pF. The use of an incorrect crystal can kill the oscillator or cause accuracy problems. Also, the use of an external trim capacitor to adjust the oscillator is discouraged.

It is recommended that one of the Dallas SmartWatch or Nonvolatile SRAM with Phantom Time Clock devices be selected for the highest accuracy (±1 minute/month).

RAM/TIME CHIP INTERFACE Figure 1

CMOS STATIC RAM

A0 – AN

ADD
DATA I/O
DQ0–DQ7

WE

WE

OE

OE

CE
V$_{CC}$

DS1215

10 CEO V$_{CCO}$ 15
12 OE D 6
3 WE Q 7
11 CEI V$_{CCI}$ 16 +5 VDC
13 RST ROM/ 9
4 BAT$_1$ RAM 14
X$_1$ BAT$_2$
X$_2$

CE

RST

OR TIE TO GND FOR
ONE-BATTERY
OPERATION

BAT$_1$

BAT$_2$

1 2

32.768 KHz

ROM/TIME CHIP INTERFACE Figure 2

ROM

A0 – AN

ADD V$_{CC}$
DATA OUT
D0 – D7

A2

OE

OE

A0

CE

DS1215

6 D CEO 10
12 OE Q 7
3 WE V$_{CCI}$ 16 +5 VDC
11 CEI V$_{CCO}$ 15
13 RST ROM/ 9
4 BAT$_1$ RAM 14
X$_1$ BAT$_2$
X$_2$

CE

RST

OR TIE TO GND FOR
ONE-BATTERY
OPERATION

BAT$_1$

BAT$_2$

1 2

32.768 KHz
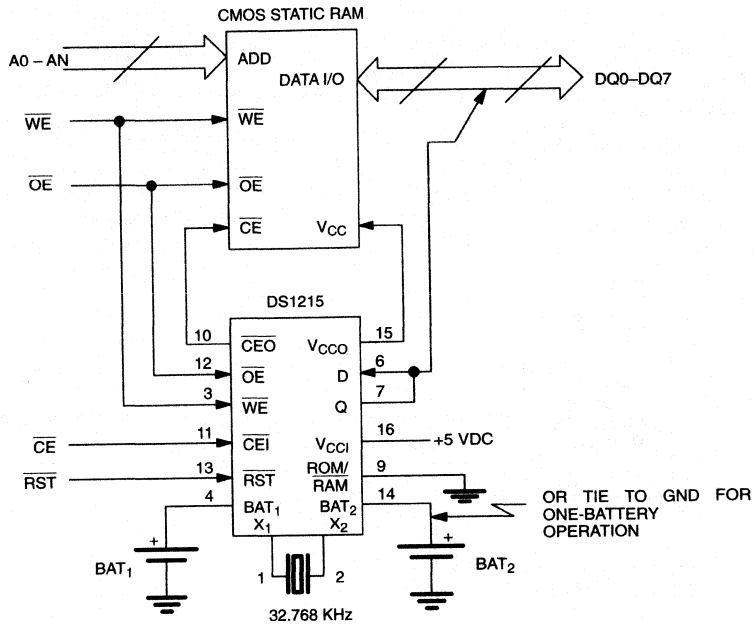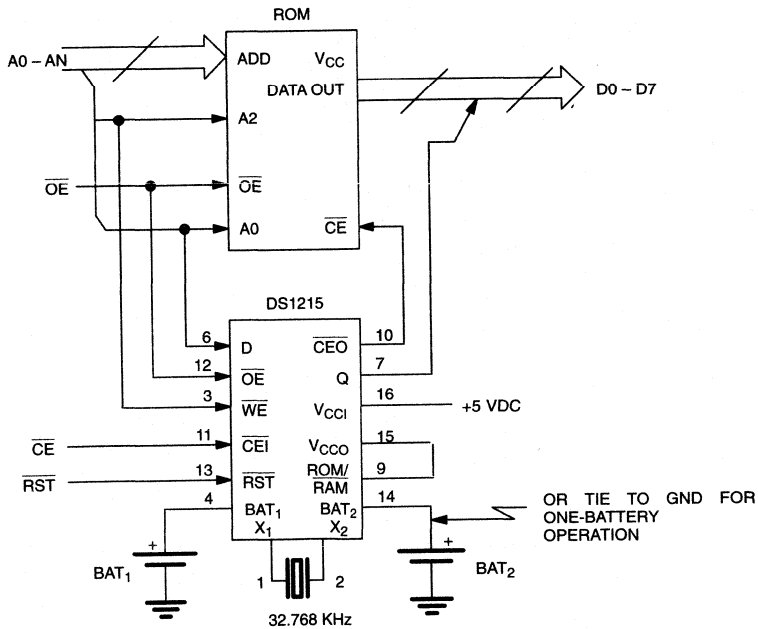
TIMING BLOCK DIAGRAM Figure 3
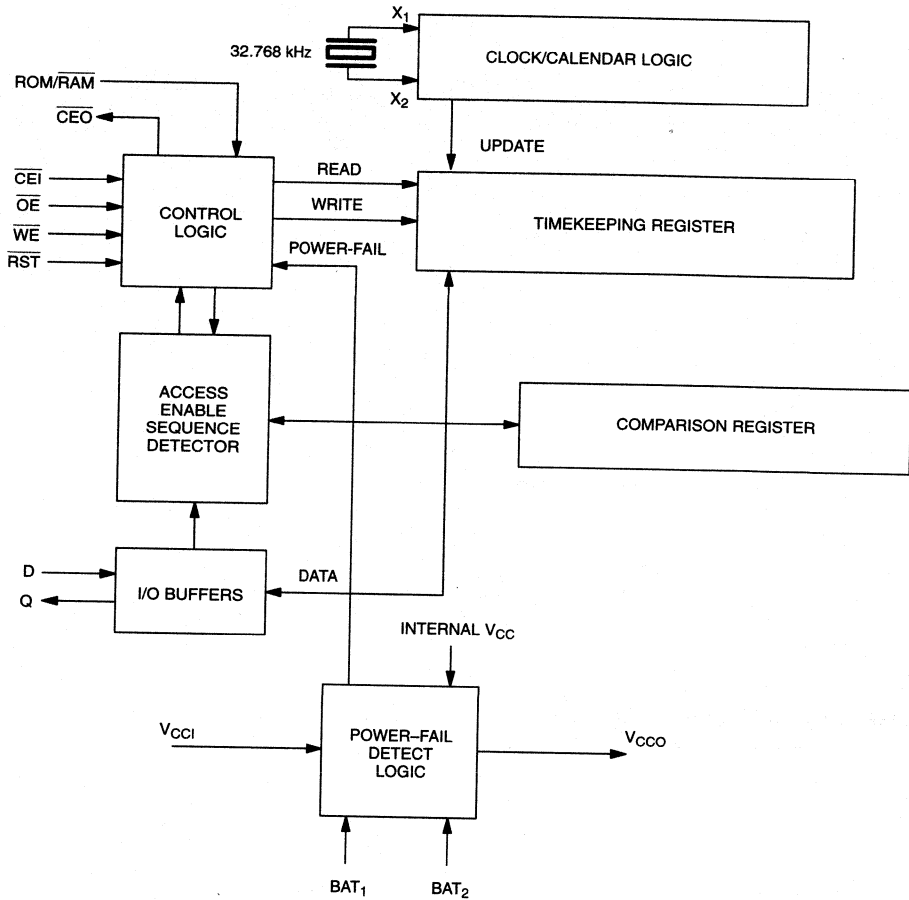
TIME CHIP COMPARISON REGISTER DEFINITION Figure 4

	7	6	5	4	3	2	1	0	HEX VALUE
BYTE 0	1	1	0	0	0	1	0	1	C5
BYTE 1	0	0	1	1	1	0	1	0	3A
BYTE 2	1	0	1	0	0	0	1	1	A3
BYTE 3	0	1	0	1	1	1	0	0	5C
BYTE 4	1	1	0	0	0	1	0	1	C5
BYTE 5	0	0	1	1	1	0	1	0	3A
BYTE 6	1	0	1	0	0	0	1	1	A3
BYTE 7	0	1	0	1	1	1	0	0	5C
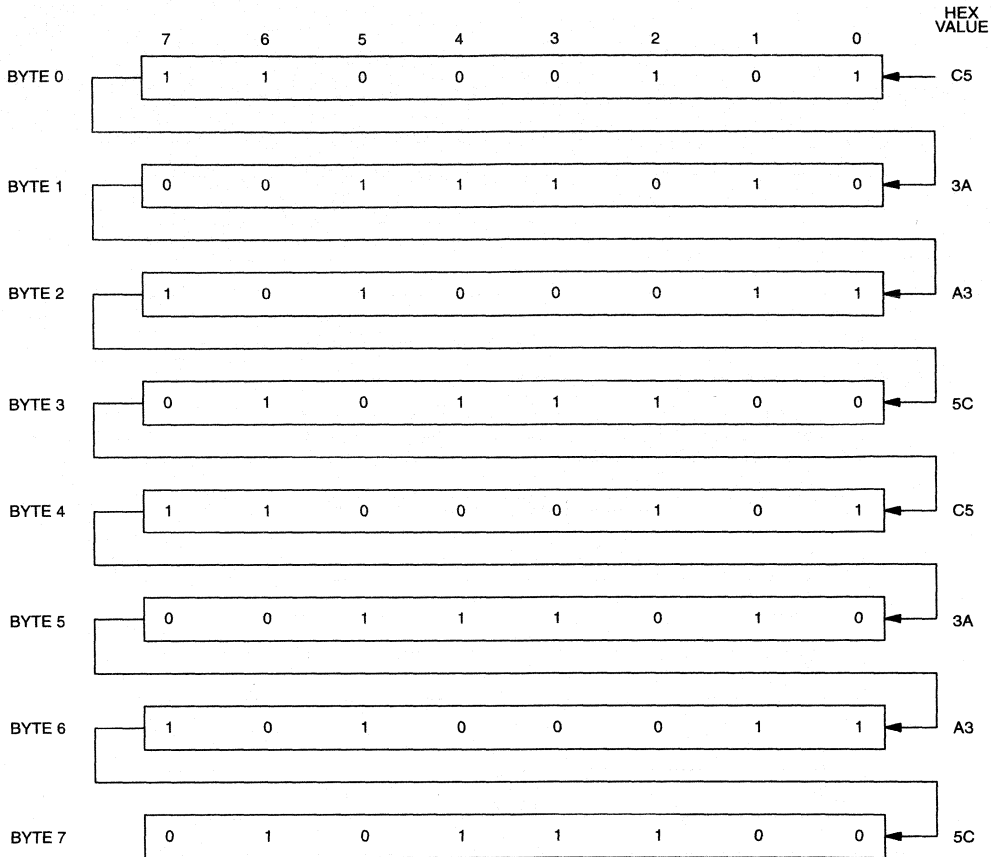
NOTE:

The pattern recognition in Hex is C5, 3A, A3, 5C, C5, 3A, A3, 5C. The odds of this pattern being accidentally duplicated and causing inadvertent entry to the Time Chip are less than 1 in 10^{19}. This pattern is sent to the Phantom Clock LSB to MSB.

TIME CHIP REGISTER DEFINITION Figure 5
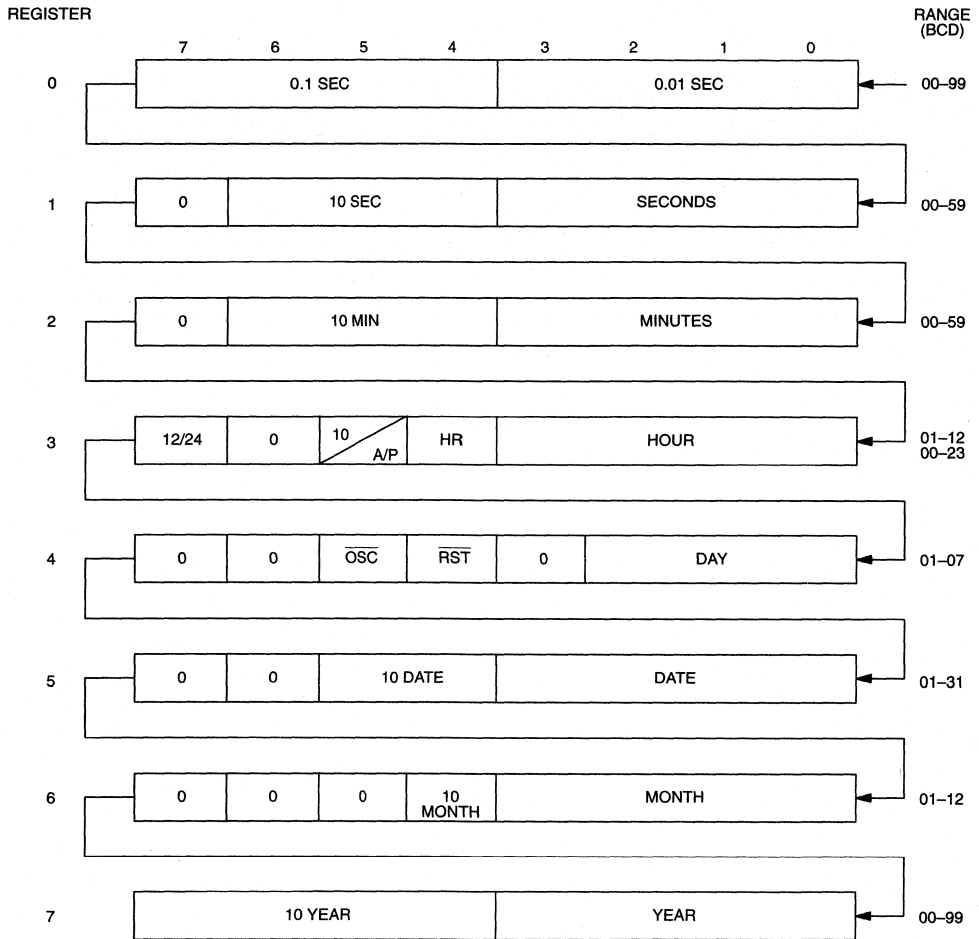
PSEUDO CODE Figure 6

```
* This code will access the Phantom Time Clock by sending the 64-bit   *
* access pattern.  Then the time data will be written to the clock and *
* finally the Phantom Time Clock will be accessed again and it will be *
* read.  The time information to be written is 12:00 PM Wednesday,      *
* January 1, 1992.  Also note that the oscillator has been enabled and  *
* reset has been disabled.                                             *

A :  Array[0..7] = (C5, 3A, A3, 5C, C5, 3A, A3, 5C)  (access pattern)
T :  Array[0..7] = (00, 00, 00, B2, 14, 01, 01, 92)  (time data)
X :  Byte at 1000                          (memory location 1000H)
D :  Array [0..7]
S :  Byte

* Send access pattern to Phantom Time Clock *
  FOR I = 0 TO 64  S = x          (perform 65 consecutive reads from x)
  FOR I = 0 TO 7                  (loop for 8 bytes)
     FOR J = 0 TO 7               (loop for 8 bits)
        X = A[I] SHR J            (write to X, shift bits right J)
     NEXT J
  NEXT I


* Write time data to Phantom Time Clock registers *
  FOR I = 0 TO 7                  (loop for 8 bytes)
     FOR J = 0 TO 7               (loop for 8 bits)
        X = T[I] SHR J            (write to X, shift bits right J)
     NEXT J


* Send access pattern to Phantom Time Clock *
  FOR I = 0 TO 64   S = X         (perform 65 consecutive reads from X)
  FOR I = 0 TO 7                  (loop for 8 bytes)
     FOR J = 0 TO 7               (loop for 8 bits)
        X = A[I] SHR J
     NEXT J
  NEXT I


* Read Phantom Time Clock registers *
  FOR I = 0 TO 7                  (loop for 8 bytes)
     D[I] = 0                     (initiate the byte)
     FOR J = 0 TO 7               (loop for 8 bits)
        D[I] = D[I] or (X and 1) SHL J   (position bits in byte)
     NEXT J
  NEXT I
```

EXAMPLE SOURCE CODE FOR 8051 MICROCONTROLLER Figure 7

```
;  8051CODE.DOC
;  RTC procedure to access the DS1215 Serial Timekeeper, or DS1216
;  SmartWatch using 8031, 8051 or 80C196
;
BIT_SEG SEGMENT BIT
        RSEG    BITSEG
WF:     DBIT    1
BYTE_SEG        SEGMENT DATA
        RSEG    BYTE_SEG
BUFF:   DS      8           ;Centi-sec:  00-99
;                           ;Seconds:  00-59
;                           ;Minutes:  00-59
;                           ;Hours:  01-12 / 00-23
;                           ;Day:1nHEX % RST off, n=DAY# 01-07
;                           ;Date:  01-31
;                           ;Month:  01-12
;                           ;Year:  00-99
CODESEG SEGMENT CODE
        RSEG    CODE_SEG
;************************
;*** MAIN PROGRAM GOES HERE
;************************
;
; Main program SETS WF for Read Mode and on return from RTC the BUFF will
; contain the 8 bytes of data read from the clock.  If WF is CLEARED then
; RTC will return after writing the 8 byte BUFF to the clock.
;
;    NOTE  !!! : Refer to the DS1215 (RAM MODE) or DS1216 data sheet.
;
RTC:    PUSH    PSW                 ;Save user registers.
        PUSH    ACC
        PUSH    B
        MOV     B, R0
        PUSH    B
        MOV     R0, #BUFF           ;Load pointer to start of table.
        LCALL   OPEN                ;Set up to open the DS1216.
        MOV     B, #8H              ;Load loop counter for 8 bytes.
        JNB     WF, WRITETIME       ;Read/Write mode check.
;
READTIME:       LCALL   RBYTE       ;Read one byte.
        MOV     @R0, A              ;Save in RTn temporary register.
        INC     R0                  ;Temporary data register pointer.
        DJNZ    B, READTIME         ;Loop to read 8 bytes.
        SJMP    ENDTIME             ;Done reading goto finish.
;
WRITETIME:      MOV     A, @R0      ;Load byte of data to be written.
        LCALL   WBYTE               ;Write one byte.
        INC     R0                  ;Temporary data register pointer.
        DJNZ    B, WRITETIME        ;Loop to write 8 bytes.
```

```
;
ENDTIME:        POP     B               ;Restore registers.
        MOV     R0, B
        POP     B
        POP     ACC

        POP     PSW
        RET                             ;Return to main calling program.
;
;
;
;
;**************************************
; SUBROUTINE TO OPEN THE CLOCK/CALENDAR
;**************************************
;
; This subroutine executes the sequence of reads and writes which
; is required in order to open communication with the timekeeper.
;
OPEN:   LCALL   CLOSE           ;Make sure it is closed.
        MOV     B, #4           ;Set pattern period count.
        MOV     A, #0C5H        ;Load first byte of pattern.
OPENA:  LCALL   WBYTE           ;Send out the byte.
        XRL     A, #0FFH        ;Generate next pattern byte.
        LCALL   WBYTE           ;Send out the byte.
        SWAP    A               ;Generate next pattern byte.
        DJNZ    B, OPENA        ;Repeat until 8 bytes sent.
        RET                     ;Return.
;
;****************************
;*** SUBROUTINE TO CLOSE CLOCK
;****************************
;
; This subroutine insures that the registers of the timekeeper
; are closed by executing 72 successive reads of the date and time
; registers.
;
CLOSE:  MOV     B, #9           ;Set up to read 9 bytes.
CLOSEA: LCALL   RBYTE           ;Read a byte.
        DJNZ    B, CLOSEA       ;Loop for 9 byte reads.
        RET                     ;Return
;
;*********************************
:*** SUBROUTINE TO READ A DATA BYTE
;*********************************
;
RBYTE:  PUSH    DPL             ;Save the data
        PUSH    DPH             ; pointer on stack.
        PUSH    B               ;Save the B register.
        MOV     DPTR, #RTCADDR  ;Enable the clock.
        MOV     B, #8           ;Set the bit count.
```

```
LI:        PUSH      ACC                ;Save the accumulator.
           MOVX      A, @DPTR           ;Input the data bit.
           RRC       A                  ;Move it to carry.
           POP       ACC                ;Get the accumulator.
           RRC       A                  ;Save the data bit.
           DJNZ      B, LI              ;Loop for a whole byte.
           POP       B                  ;Restore the B register.
           POP       DPH                ;Restore the data
           POP       DPL                ; pointer from stack.
           RET                          ;Return.
;
;
;
;**********************************
;*** SUBROUTINE TO WRITE A DATA BYTE
;**********************************
;
WBYTE:     PUSH      DPL                ;Save the data
           PUSH      DPH                ; pointer on stack.
           PUSH      B                  ;Save the B register.
           MOV       DPTR, #RTCADDR     ;Enable the clock.
           MOV       B, #8              ;Set the bit count.
LO:        PUSH      ACC                ;Save the accumulator.
           ANL       A, #1              ;Set up bit for output.
           MOVX      @DPTR, A           ;Output the data bit.
           POP       ACC                ;Restore the accumulator.
           RR        A                  ;Position next bit.
           DJNZ      B, LO              ;Loop for a whole byte.
           POP       B                  ;Restore the B register.
           POP       DPH                ;Restore the data
           POP       DPL                ; pointer from stack.
           RET                          ;Return.
;
;***************
;END OF PROGRAM
;***************
;
           END                          ;End of program.
```

DALLAS
SEMICONDUCTOR

Application Note 58
Crystal Considerations with Dallas Real Time Clocks

Dallas Semiconductor offers a variety of real time clocks (RTCs). The majority of these are available either as integrated circuits or modules. Modules include a real–time clock integrated circuit, crystal, and lithium energy source encapsulated in one package.

This application note is intended to help those customers who choose to use Dallas Semiconductor real time clock chips rather than modules and therefore need to attach their own crystal. The information contained in this article will be beneficial in maximizing accuracy and insuring proper operation of Dallas real time clocks by helping the customer to select the correct crystal to use and by providing a few basic guidelines that should be followed when placing the crystal on a PCB layout. This application note will also include an elementary discussion of the effect of temperature on the accuracy of real time clocks.

CRYSTAL SELECTION
In any crystal based oscillator circuit, the oscillator frequency is based almost entirely on the characteristics of the crystal that is used. It is important to select a crystal that meets the design requirements. In particular, the specified load capacitance (C_L) is a critical crystal parameter that is often overlooked. This parameter specifies the capacitive load that must be placed across the crystal pins in order for the crystal to oscillate at its specified frequency. The crystal manufacturer actually "trims" the crystal to oscillate at its nominal frequency for the given specified load capacitance. Note that the C_L is the capacitance that the crystal needs to "see" from the oscillator circuit, it is **not** the capacitance of the crystal itself.

As previously stated, the load capacitance that the crystal "sees" is due to the capacitance of the oscillator cir-

cuit itself. Any change in the load capacitance of the oscillator circuit will therefore have an affect on the frequency of that oscillator. Likewise, using a crystal that has a C_L that is different than the actual load capacitance of the circuit will also affect the frequency of the oscillator.

In general, using a crystal with a C_L that is larger than the load capacitance of the oscillator circuit will cause the oscillator to run faster than the specified nominal frequency of the crystal. Conversely, using a crystal with a C_L that is smaller than the load capacitance of the oscillator circuit will cause the oscillator to run slower than the specified nominal frequency of the crystal.

The majority of Dallas Semiconductor RTCs have an internal capacitance of 6 pF across the crystal input pins. Recent RTC offerings from Dallas Semiconductor have 12.5 pF capacitance or are software configurable for 6 pF or 12.5 pF. For proper operation and accuracy, a crystal that meets the parts requirements should be used. As mentioned above, using a crystal with the wrong C_L will cause the oscillator to run fast or slow. Limited characterization at Dallas Semiconductor has confirmed this. For example, limited characterization on the DS1485, which is designed for a 6 pF crystal, has revealed that the device will run approximately 4 minutes/month fast at room temperature (25°C) when a 12 pF crystal is used. The device had an accuracy within ±30 seconds/month when a 6 pF crystal was used.

Several vendors offer crystals that can be used with Dallas Semiconductor real time clocks. These vendors include Epson and KDS/Daiwa. Equivalent crystals from other crystal manufacturers can also be used. As a reference, see Table 1 for recommended crystal characteristics.

050696 1/7

CRYSTAL SPECIFICATIONS Table 1

PARAMETER	SYMBOL	MIN	TYP	MAX	UNITS	NOTES
Nominal Frequency	F_0		32.768		kHz	
Load Capacitance	C_L		6		pF	
Temperature Turnover Point	T_0	20	25	30	°C	
Parabolic Curvature Constant	k			0.042	ppm/°C	
Quality Factor	Q	40,000	70,000			
Series Resistance	R1			45	KΩ	
Shunt Capacitance	C_0		1.1	1.8	pF	
Capacitance Ratio	C_0/C_1		430	600		
Drive Level	D_L			1	μW	

RTC CRYSTAL SUPPLIERS

MANUFACTURER	MODEL	C_L	PACKAGE
KDS/Daiwa Crystal Corp.	DT–38 DT–381 DT–26S DT–261S DT–14 DT–26S 32.768 Hz DMX–2632.768 KHz DS–VT–200	6pF	Cylinder Cylinder Cylinder Cylinder Cylinder Cylinder Cylinder Cylinder
Epson Crystal Corp.	C–001R C–002RX C–004R C–005R MC–306 MC–405 MC–406	6 pF, 12.5 pF 6 pF, 12.5 pF 6 pF, 12.5 pF 6 pF, 12.5 pF 6 pF, 12.5 pF 6 pF, 12.5 pF 6 pF, 12.5 pF	Cylinder Cylinder Cylinder Cylinder SMT SMT SMT
Hooray Electronics	MMTF–32	6pF	Cylinder

CRYSTAL LOAD COMPENSATION

Dallas Semiconductor does not recommend using crystals that do not have a C_L that matches the RTCs specification because this will decrease the accuracy of the clock. We find however that customers sometimes choose to use 12.5 pF crystals with our 6 pF RTCs. For this situation it is possible to improve the decreased accuracy caused by the C_L mismatch. This can be accomplished by increasing the load capacitance that the crystal "sees" by connecting a capacitor in parallel with the crystal as shown in Figure 1. As a rule of thumb, the approximate capacitor value is equal to the specified load capacitance (C_L) of the crystal minus 6 pF (the approximate load capacitance of the real time clock oscillator circuit). For example, if a 12 pF crystal is being used, a 6 pF capacitor should be placed in parallel with it to improve the accuracy of the oscillator. A 12 pF crystal is adjusted by the crystal manufacturer to oscillate at its specified nominal frequency when a 12 pF load is present. A 6 pF capacitor is therefore added to the 6 pF load of the oscillator circuit to compensate for the additional load that the crystal needs in order to oscillate at its specified nominal frequency.

CRYSTAL CONFIGURATION WHEN 6 PF CRYSTAL IS NOT USED Figure 1
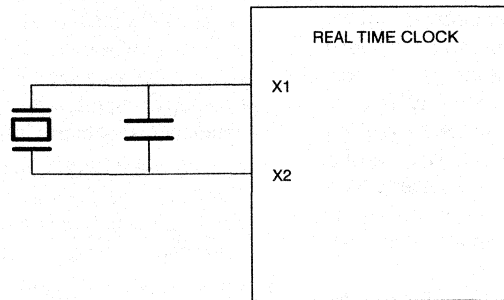
As mentioned, a crystal with a C_L of greater than 6 pF can be compensated with an external capacitor to improve the accuracy. However, it should be noted that the oscillator start–up time (the time it takes during initial power up for the oscillator to stabilize) will increase due to the increased capacitance in the feedback path of the oscillator. This capacitance decreases the loop gain of the oscillator which in turn increases the start–up time. For example, limited characterization has shown that the start–up time for a 6 pF crystal is typically less than a couple hundred milliseconds, but can increase to one or two seconds when a 12 pF crystal with a 6 pF capacitor in parallel are used.

NOISE AND CRYSTAL LAYOUT GUIDELINES

Since the crystal inputs of the Dallas Semiconductor real time clocks have a very high impedance (about 10^9 ohms), the leads to the crystal act like a very good antennae, coupling high frequency signals from the rest of the system. If a signal is coupled onto the crystal pins, it can either cancel out or add pulses. Since most of the signals on a board are much higher frequency than the 32.768 KHz crystal, it is more likely to add pulses where none are wanted. These noise pulses get counted as extra clock "ticks" and make the clock appear to run fast.

It is very simple to determine if noise is the cause of the inaccuracy of a real time clock. The following steps illustrate how this can be done.

1. Power the system up and synchronize the real time clock to a known accurate clock.

2. Turn system power off.

3. Wait for a period of time (two hours, 24 hours, etc.). The longer the time period, the easier it will be to measure the accuracy of the clock.

4. Turn system on again, read clock, and compare to the known accurate clock.

5. Re–synchronize the real time clock to the known accurate clock.

6. Keep system powered up and wait for a period of time equal to the period in step 3.

7. Read clock after waiting for the above period of time and compare to the known accurate clock.

By using the above steps, the accuracy of the clock can be determined both when the system is powered up and when the system is powered down. If the clock proves to be inaccurate when the system is powered up, but is accurate when the system is powered down, the problem is most likely due to noise from other signals in the system. However, if the clock is inaccurate both when the system is powered up and when it is powered down, then the problem is not due to noise from the system.

Since it is possible for noise to be coupled onto the crystal pins, care must be taken when placing the external crystal on a PCB layout. It is very important to follow a few basic layout guidelines concerning the placement of the crystal on the PCB layout to insure that extra clock "ticks" do not couple onto the crystal pins.

1. It is important to place the crystal as close as possible to the X1 and X2 pins. Keeping the trace lengths between the crystal and the real time clock as small as possible reduces the probability of noise coupling by reducing the length of the "antennae". Keeping the trace lengths small also decreases the amount of stray capacitance.

2. Keep the crystal bond pads and trace width to the X1 and X2 pins as small as possible. The larger these bond pads and traces are, the more likely it is that noise can couple from adjacent signals.

3. If possible, place a guard ring (tied to ground) around the crystal. This helps to isolate the crystal from noise coupled from adjacent signals. See Figure 2 for an illustration of using a guard ring around a crystal.

4. Try to insure that no signals on other PCB layers run directly below the crystal or below the traces to the X1 and X2 pins. The more the crystal is isolated from other signals on the board, the less likely it is that noise will be coupled into the crystal.

5. It may also be helpful to place a local ground plane on the PCB layer immediately below the crystal guard ring. This helps to isolate the crystal from noise coupling from signals on other PCB layers. Note that the ground plane needs to be in the vicinity of the crystal only and not on the entire board. See Figure 2 for an illustration of a local ground plane. Note that the perimeter of the ground plane does not need to be larger than the outer perimeter of the guard ring.

Note that care must be taken concerning the use of a local ground plane because of the stray capacitance that it introduces. This capacitance will be added to the crystal pins and if large enough could slow the clock down. Therefore, some factors must be taken into account when considering adding a local ground plane. For example, the capacitance due to the ground plane may be approximated by

$$C = \varepsilon A/t$$

where

ε = dielectric constant of the PCB
A = area of the ground plane
t = thickness of the PCB layer.

Therefore, to determine if a ground plane is appropriate for a given design, the above parameters must be taken into account to insure that the capacitance from the local ground plane is not sufficiently large enough to slow down the clock.

EXAMPLE OF CRYSTAL PLACEMENT ON PCB Figure 2
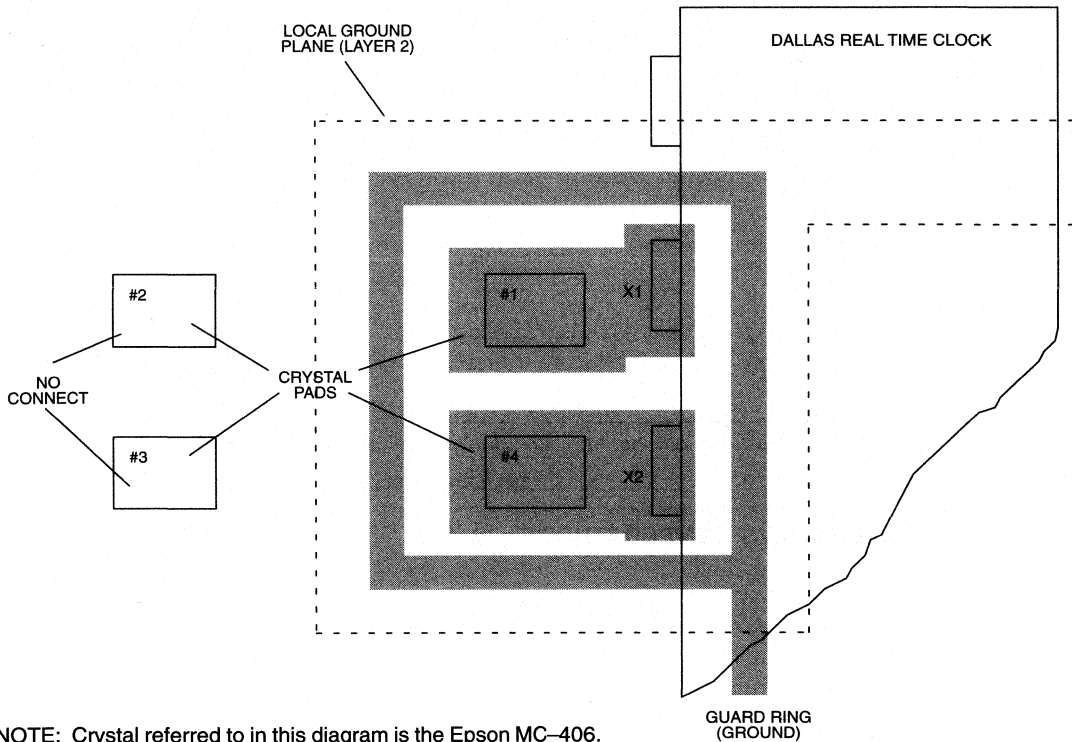
NOTE: Crystal referred to in this diagram is the Epson MC–406.

CLOCK ACCURACY OVER TEMPERATURE

The accuracy of a real time clock is directly dependent upon the frequency of the crystal. Therefore, since the resonant frequency of a crystal is dependent upon temperature, a real time clock will also be dependent upon temperature. The resonant frequency of a crystal is expressed in the following basic formula:

$$f = f0 + k * (T - T0)^2$$

where

f_0 = nominal frequency
k = parabolic curvature constant
T_0 = turnover temperature
T = temperature

The values of the above parameters can be found in the data sheet of the crystal being used. The temperature characteristic of a nominal Daiwa crystal is illustrated in Figure 3 where $f_0 = 32.768$ KHz, $k = -0.042$ ppm/°C, and $T_0 = 23$°C. As can be seen in this figure, frequency has a parabolic relationship to temperature – as temperature deviates from the ideal 23°C, the crystal frequency becomes increasingly slower.

Figure 4 shows the same basic curve, however, the Y axis has been changed to show the frequency deviation (in ppm) from the crystal's nominal frequency at 23°C. This curve illustrates more clearly how the frequency of the crystal will affect the accuracy of the clock. A frequency deviation of 23 ppm translates into an accuracy of approximately ±1 minute per month. With this in mind, a quick glance at Figure 4 will give an approximate expected accuracy at a given temperature.

The above information should help to provide a basic understanding of how temperature will affect Dallas Semiconductor real time clocks.

CRYSTAL FREQUENCY VS. TEMPERATURE Figure 3
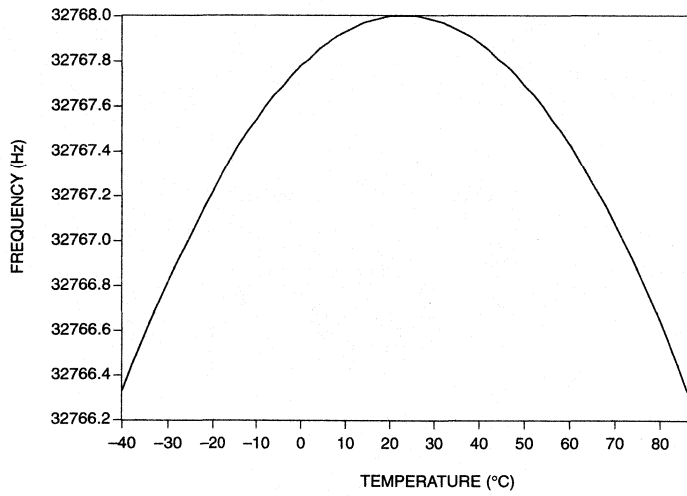
FREQUENCY DEVIATION VS. TEMPERATURE Figure 4
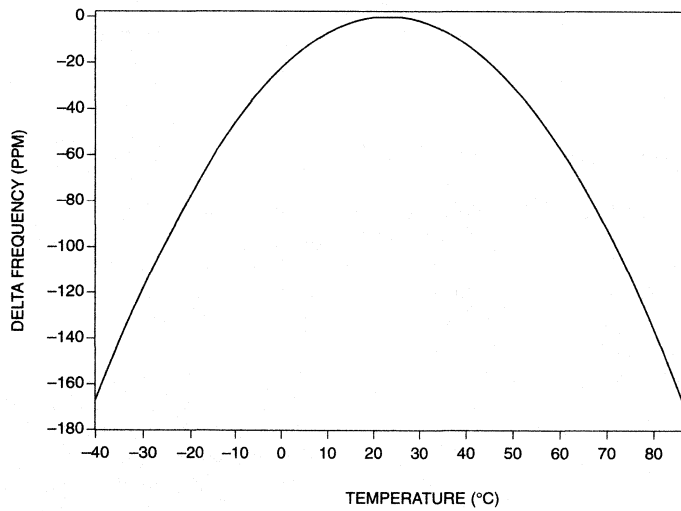
TROUBLESHOOTING

This section is provided as a summary of the most frequent causes of real time clock inaccuracies. Most of these problems have been mentioned earlier, but are repeated here as a quick reference. This section has been divided into two parts. The first part will consider the factors that cause a real time clock to run too fast and the second part will consider the factors that cause a real time clock to run too slow.

FAST CLOCKS

The following are the most common scenarios that cause a crystal–based real time clock to run fast.

1. Noise coupling into the crystal from adjacent signals: This problem has been extensively covered above. Noise coupling will usually cause a real time clock to be grossly inaccurate.

2. Wrong crystal: A real time clock will typically run fast if a crystal with a specified load capacitance (C_L) greater than 6 pF is used. The severity of the inaccuracy is dependent on the value of the C_L. For example using a crystal with a C_L of 12 pF will cause the real time clock to be about 3–4 minutes per month fast.

SLOW CLOCKS

The following are the most common scenarios that cause a crystal–based real time clock to run slow.

1. Overshoot on real time clock input pins: It is possible to cause a real time clock to run slow by periodically stopping the oscillator. This can be inadvertently accomplished by noisy input signals to the real time clock. If an input signal rises to a voltage that is greater than a diode drop ($\sim$0.3V) above V_{DD}, the ESD protection diode for the input pin will forward bias, allowing the substrate to be flooded with current. This, in turn, will stop the oscillator until the input signal voltage decreases to below a diode drop above V_{DD}.

This mechanism can cause the oscillator to stop frequently if input signals are noisy. Therefore, care should be taken to insure that there is no overshoot on input signals.

Another situation that is common to overshoot problem is having an input to the real time clock at 5V when the real time clock is in battery back–up mode. This can be a problem in systems that systematically shut down certain circuits, but keep others powered up. It is very important to insure that there are no input signals to the real time clock that are greater than the battery voltage when the device is in battery back–up mode.

2. Wrong crystal: A real time clock will typically run slow if a crystal with a specified load capacitance (C_L) less than 6 pF is used. The severity of the inaccuracy is dependent on the value of the C_L.

3. Stray capacitance: Stray capacitance between the crystal pins can slow a real time clock down. Therefore care must be taken when designing the PCB layout to insure that the stray capacitance is kept to a minimum.

4. Temperature: The further the operating temperature is from the crystal turnover temperature, the slower the crystal will oscillate. See Figures 3 and 4.

REFERENCES

1. Eaton, S. S. *Timekeeping Advances Through COS/MOS Technology*, RCA Application Note ICAN–6086.

2. Eaton, S. S. *Micropower Crystal–Controlled Oscillator Design Using RCA COS/MOS Inverters*, RCA Application Note ICAN–6539.

3. Meyer, R. G. "MOS Crystal Oscillator Design," *IEEE Journal of Solid–State Circuits*, Vol. SC–15, No. 2, pp. 222–227, April 1980.

4. Williamson, T. *Oscillators for Microcontrollers*, Intel Application Note AP–155.

DALLAS
SEMICONDUCTOR

GENERAL OVERVIEW

Many applications require a real time clock to keep track of absolute time. Often times these same applications could benefit by being told to perform certain functions at specific times. The Watchdog Timekeeper family from Dallas Semiconductor is a solution to systems that need both an accurate real time clock and interrupt capabilities at specific times. In addition the Dallas Watchdog Timekeeper family also provides an upgradeable nonvolatile RAM path.

Dallas Watchdog Timekeepers provide basic real time clock functions. The devices keep track of hundredths of seconds, seconds, minutes, hours, day, date, month, and year information. The time is accurately maintained by a very low power oscillator. The Watchdog Timekeeper modules which contain the Watchdog integrated circuit as well as a crystal and lithium battery provide an accuracy of ±1 minute per month. Because of the low power oscillator, the internal lithium battery provides 10 years of continuous operation in the absence of system power.

The Dallas Watchdog Timekeeper family is made up of the DS1283 and DS1284 integrated circuits as well as the DS1286, DS1386, and DS1486 modules. Table 1 lists the different devices that belong to this family. As can be seen in the table, this family provides an upgradeable nonvolatile RAM growth path.

THE DALLAS WATCHDOG TIMEKEEPER FAMILY Table 1

DEVICE	NVRAM	
DS1283	50 bytes	IC
DS1284	50 bytes	IC
DS1286	50 bytes	Module
DS1386–08	8K bytes	Module
DS1386–32	32K bytes	Module
DS1486	128K bytes	Module

One benefit of using Dallas Watchdog Timekeepers is that the devices are simple to use since the real time clock registers are mapped directly into the device's onboard RAM. This makes accessing the timekeeping registers analogous to accessing a byte in RAM. See Figure 1 for the memory map of the DS1486, for example. Note that the real time clock registers occupy only the top 14 bytes of the memory.

An additional benefit that simplifies the design effort when using the Watchdog family is that the DS1386 and DS1486 pinouts are very close to the standard JEDEC bytewide pinout for SRAM's. The two interrupt outputs and the square wave output are the only pins that differ from the JEDEC pinout. See Figure 2 for a comparison of standard JEDEC bytewide pinouts compared to the pinouts of the DS1386 and the DS1486.

INTERRUPTS

A key feature of the Dallas Watchdog Timekeepers is that two different interrupt outputs are provided – a time of day alarm and a watchdog interrupt. These two interrupts are controlled by the Command Register, time of day alarm registers, and watchdog alarm registers. The Command Register allows flexibility in the operation of the interrupts. See Appendix for a complete description of the Command Register bits.

Time of Day Alarm

The first type of interrupt is the Time of Day Alarm. The Time of Day Alarm allows the user to program the device to generate an interrupt at a specific time of day for a specific day of the week. Special mask bits in the alarm registers also make it possible for this alarm to generate an interrupt once per minute, once per hour, or once per day.

DS1486 RAMIFIED TIMEKEEPER REGISTERS Figure 1

Group	ADDRESS	BIT 7						BIT 0	RANGE
	0	0.1 SECONDS			0.01 SECONDS				00–99
	1	0	10 SECONDS		SECONDS				00–59
	2	0	10 MINUTES		MINUTES				00–59
	3	M	10 MIN ALARM		MIN ALARM				00–59
CLOCK, CALENDAR, TIME OF DAY ALARM REGISTERS	4	0	12/24	10 A/P	10 HR	HOURS			01–12+A/P / 00–23
	5	M	12/24	10 A/P	10 HA	HR ALARM			01–12+A/P / 00–23
	6	0	0	0	0	0	DAYS		01–07
	7	M	0	0	0	0	DAY ALARM		01–07
	8	0	0	10 DATE		DATE			01–31
	9	EOSC	ESQW	0	10MO	MONTHS			01–12
	A	10 YEARS			YEARS				00–99
COMMAND REGISTERS	B	TE	IPSW	IBH LO	PU LVL	WAM	TDM	WAF	TDF	
WATCHDOG ALARM REGISTERS	C	0.1 SECONDS			0.01 SECONDS				00–99
	D	10 SECONDS			SECONDS				00–99
USER REGISTERS	E								
	1FFFF								

COMPARISON OF WATCHDOG TIMEKEEPER AND JEDEC BYTEWIDE PINOUTS Figure 2

WATCHDOG TIMEKEEPERS

Left	Pin		Pin	Right
$\overline{\text{INTA}}$	1		32	V_{CC}
$\overline{\text{INTB}}$	2		31	SQW
NC	3		30	V_{CC}
A12	4		29	$\overline{\text{WE}}$
A7	5		28	NC
A6	6		27	A8
A5	7		26	A9
A4	8		25	A11
A3	9		24	$\overline{\text{OE}}$
A2	10		23	A10
A1	11		22	$\overline{\text{CE}}$
A0	12		21	DQ7
DQ0	13		20	DQ6
DQ1	14		19	DQ5
DQ2	15		18	DQ4
GND	16		17	DQ3

DS1386 8K X 8

JEDEC BYTEWIDE

Left	Pin		Pin	Right
NC	1		28	VCC
A12	2		27	$\overline{\text{WE}}$
A7	3		26	NC
A6	4		25	A8
A5	5		24	A9
A4	6		23	A11
A3	7		22	$\overline{\text{OE}}$
A2	8		21	A10
A1	9		20	$\overline{\text{CE}}$
A0	10		19	DQ7
DQ0	11		18	DQ6
DQ1	12		17	DQ5
DQ2	13		16	DQ4
GND	14		15	DQ3

8K X 8 SRAM

Left	Pin		Pin	Right
$\overline{\text{INTA}}$	1		32	V_{CC}
$\overline{\text{INTB}}$	2		31	SQW
A14	3		30	V_{CC}
A12	4		29	$\overline{\text{WE}}$
A7	5		28	A13
A6	6		27	A8
A5	7		26	A9
A4	8		25	A11
A3	9		24	$\overline{\text{OE}}$
A2	10		23	A10
A1	11		22	$\overline{\text{CE}}$
A0	12		21	DQ7
DQ0	13		20	DQ6
DQ1	14		19	DQ5
DQ2	15		18	DQ4
GND	16		17	DQ3

DS1386 32K X 8

Left	Pin		Pin	Right
A14	1		28	VCC
A12	2		27	$\overline{\text{WE}}$
A7	3		26	A13
A6	4		25	A8
A5	5		24	A9
A4	6		23	A11
A3	7		22	$\overline{\text{OE}}$
A2	8		21	A10
A1	9		20	$\overline{\text{CE}}$
A0	10		19	DQ7
DQ0	11		18	DQ6
DQ1	12		17	DQ5
DQ2	13		16	DQ4
GND	14		15	DQ3

32K X 8 SRAM

Left	Pin		Pin	Right
$\overline{\text{INTB}}$	1		32	V_{CC}
A16	2		31	A15
A14	3		30	$\overline{\text{INTA}}$/SQW
A12	4		29	$\overline{\text{WE}}$
A7	5		28	A13
A6	6		27	A8
A5	7		26	A9
A4	8		25	A11
A3	9		24	$\overline{\text{OE}}$
A2	10		23	A10
A1	11		22	$\overline{\text{CE}}$
A0	12		21	DQ7
DQ0	13		20	DQ6
DQ1	14		19	DQ5
DQ2	15		18	DQ4
GND	16		17	DQ3

DS1486 128K X 8

Left	Pin		Pin	Right
NC	1		32	VCC
A16	2		31	A15
A14	3		30	NC
A12	4		29	$\overline{\text{WE}}$
A7	5		28	A13
A6	6		27	A8
A5	7		26	A9
A4	8		25	A11
A3	9		24	$\overline{\text{OE}}$
A2	10		23	A10
A1	11		22	$\overline{\text{CE}}$
A0	12		21	DQ7
DQ0	13		20	DQ6
DQ1	14		19	DQ5
DQ2	15		18	DQ4
GND	16		17	DQ3

128K X 8 SRAM

Watchdog Alarm

The second type of interrupt is the Watchdog Alarm. The Watchdog Alarm allows the user to program the device to generate a periodic interrupt at a user defined interval. The user can program the device to generate an interrupt every 0.01 seconds to 100 seconds in 0.01 second increments. The watchdog interrupt is typically used in one of two different ways – as a microprocessor monitor or as a periodic interrupt.

Watchdog Alarm as Microprocessor Monitor

The Watchdog Alarm is often used as a microprocessor monitor in critical applications. In this function, the Watchdog Alarm is used to ensure that the microprocessor does not go out of control. For this type of application, the system is designed such that the microprocessor "checks in" with the Watchdog Timekeeper periodically by reading or writing to any of the watchdog alarm registers. Each time the microprocessor "checks in" with the Watchdog Timekeeper the watchdog timer is reset. If the microprocessor does not "check in" within the user specified watchdog interval, the Watchdog Alarm will generate an interrupt. This interrupt is used to reset the microprocessor. Figures 3 and 4 show two different ways that the Watchdog Timekeeper is interfaced with a microcontroller to monitor the system for an out of control condition.

In Figure 3, the interface between the DS1386–08 and the 68HC11 microcontroller is illustrated. In this diagram, $\overline{\text{INTA}}$ from the DS1386–08 is connected to the interrupt request (IRQ) pin of the 68HC11. In this example, the 68HC11 can be programmed such that it will reset the system if $\overline{\text{INTA}}$ is allowed to go active. Figure 4 illustrates the interface between the DS1386–08 and the 8051 microcontroller. In this example, the $\overline{\text{INTA}}$ pin is connected to one input of a 74LS122 one shot circuit. The output of the one shot is connected to the reset (RST) pin of the 8051. In this circuit, if the watchdog timer is allowed to time out, the $\overline{\text{INTA}}$ pin will go active, causing the one shot to provide a pulse to the 8051 reset and thus reset the 8051.

It should be noted that in both examples the interrupt output was not connected directly to the microcontroller reset. The reason for this is related to one of the options that is provided in the Command Register. The Command Register allows the interrupts to be programmed to generate either a pulse or a constant level signal in the event that the interrupts are activated. When the pulse option is provided, there is no problem with connecting the $\overline{\text{INTA}}$ output directly to the reset pin. However, if the constant level signal mode is selected, the microcontroller would be permanently locked in reset since there would be no way to clear the interrupt (since the interrupt output would permanently hold the microcontroller in reset). With this in mind, connecting the interrupts directly to the reset input is never recommended. Even if the designer programs the interrupts to run in pulse mode, it is still possible that the level mode could be accidentally activated by a software error which would cause the microcontroller to be permanently locked in reset.

Watchdog as Periodic Interrupt

A second popular way that the Watchdog Alarm is used is as a periodic interrupt. In this case the Watchdog Timekeeper is programmed to generate an interrupt and, in contrast to the above example, the intention is to allow the interrupt to occur. In this example the device will generate an interrupt periodically at a user defined interval. Using the interrupt in this manner is useful, for example, in data acquisition equipment. The interrupt is used to tell the system to collect data each time it is activated. The circuit illustrated in Figure 3 could be used for this type of application. In this example the microcontroller could be programmed to gather data whenever an interrupt is activated.

Interrupt as "Wake Up" Signal

One important feature of the interrupt outputs of the Watchdog Timekeeper is that they can be enabled even when system power is off and the device is in battery backed operation. This is particularly useful in applications where the system is shut off to conserve power when not in use. Either of the two interrupts can be used to "wake up" a system or specific circuitry within a system. The specified task can be completed and then the system can be shut off again. The system will remain powered down until the Watchdog Timekeeper tells the system to power up again. It should be mentioned that the interrupt outputs must be selected for active low operation to function when in battery backed mode.

WATCHDOG TIMEKEEPER INTERFACED WITH 68HC11 MICROCONTROLLER Figure 3
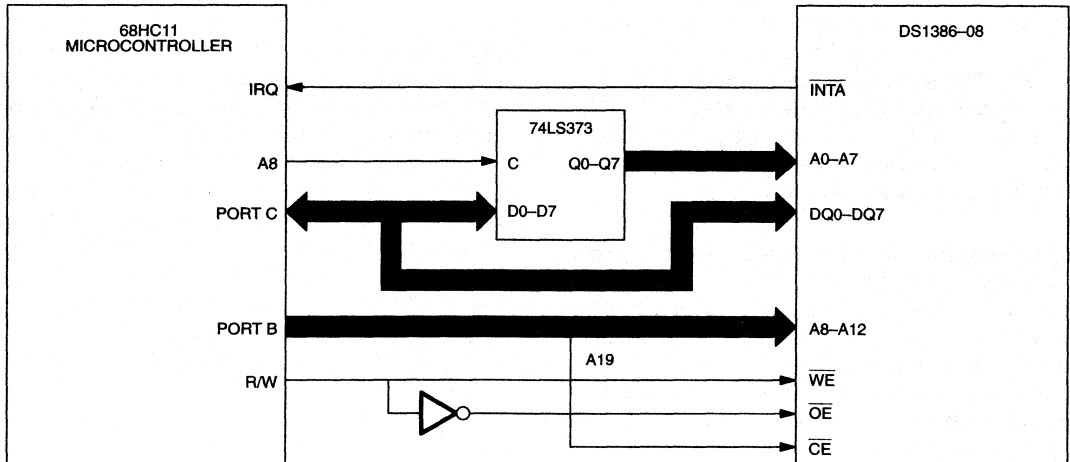
WATCHDOG TIMEKEEPER INTERFACED WITH 8051 MICROCONTROLLER Figure 4
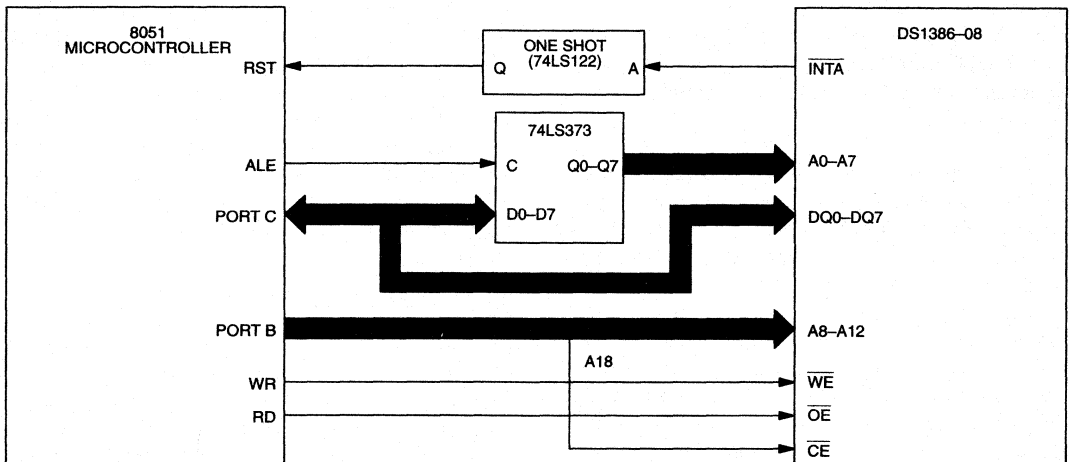
One important consideration when using an interrupt to "wake up" a system is to insure that no pins on the Watchdog Timekeeper are interfacing with any logic that is at a greater potential than the battery voltage when system power is turned off. If a pin is at a potential that is greater than the battery voltage when the system power is turned off, this will cause the positive ESD (electro–static discharge) protection diode to forward bias. This allows current to flood the substrate of the Watchdog Timekeeper which in turn can cause the oscillator to stop.

IDEAL APPLICATIONS

Dallas Watchdog Timekeepers are ideally suited for many different types of applications. In general, this product family is a perfect fit for any system that requires a real time clock, nonvolatile RAM, and interrupt capabilities. Furthermore, the upgrade path in RAM densities make this product even more attractive for designers who see the potential need for greater RAM densities. The near–JEDEC bytewide footprint and the easy software interface with the Watchdog Timekeeper further its appeal as a simple product to use.

The interrupt outputs provided by the Watchdog Timekeepers offer much flexibility to a designer. These interrupts can be used to signal a system to perform an event at a specific time and/or can be used in critical applications to monitor the microprocessor to insure that it does not run "out of control." Additionally, the interrupts can be operated even when the device is in battery backed operation. This allows the designer to use the Watchdog Timekeeper to "wake up" a system.

TROUBLESHOOTING

The Dallas Watchdog Timekeepers have proven to be highly reliable and meet the published specifications. However, during the course of development, some common difficulties could be experienced. This section is provided as a summary of the most common problems that could be encountered and gives the solution to those problems.

1. Clock is Inaccurate (i.e., greater than ±1 minute/month at 25°C)

It is possible to cause the Watchdog Timekeeper to run slow by periodically stopping the oscillator. This can be caused inadvertently by noisy input signals to the Watchdog Timekeeper. If an input signal rises to a voltage that is greater than a diode drop (~0.3 volts) above V_{CC}, the ESD protection diode for the input pin will forward bias allowing the substrate of the device to be flooded with current. This, in turn, can stop the oscillator until the input signal voltage decreases to below a diode drop above V_{DD}.

There is a second scenario that can also cause the clock to appear to run slow. This scenario is similar to that above. If the Watchdog Timekeeper is in battery back–up mode, it is important to insure that none of the input signals are at a potential that is greater than the battery voltage. This needs to be considered especially for applications that use the interrupt signals when the Watchdog Timekeeper is in battery back–up operation. Since the interrupt outputs are open drain, they need to have external pull up resistors. When the Watchdog Timekeeper is in battery back–up operation, it is necessary to insure that the interrupts are not pulled up to a potential that is greater than the battery voltage, otherwise the ESD protection diode will be forward biased which can cause the oscillator to stop running.

Also, it should be mentioned that the Watchdog Timekeepers will be the most accurate when run at room temperature (25°C). Figure 5 illustrates the accuracy of a typical real time clock over temperature. As can be seen from the graph, timekeeping accuracy is temperature dependent and becomes less accurate the further the ambient temperature deviates from +25°C.

REAL TIME CLOCK ACCURACY OVER TEMPERATURE Figure 5
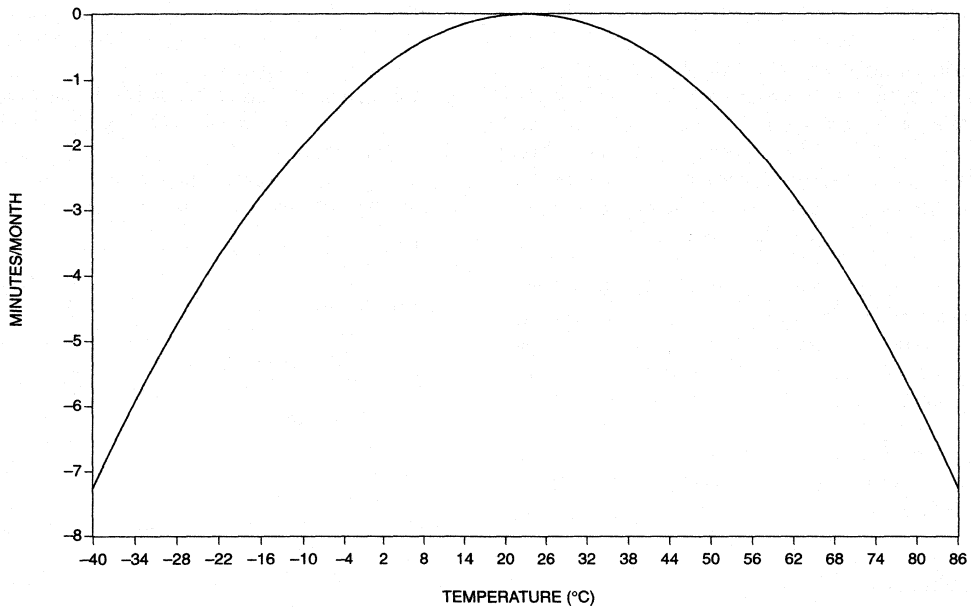
2. Interrupts Do Not Work

   Several situations can cause this problem.

   a. The interrupt outputs are open drain. Therefore, $\overline{INTA}$ needs an external pull–up resistor and $\overline{INTB}$(INTB) needs an external pull–up resistor when set for active low operation and needs an external pull–down resistor when set for active high operation. If the pull–up or pull–down resistors are not used, the interrupts will not function properly.

   b. When in battery back–up operation, only active low mode can be used for $\overline{INTB}$(INTB). Active high mode for the $\overline{INTB}$(INTB) pin will not function when the Watchdog Timekeeper is in battery back–up mode.

3. Cannot Write to Real Time Clock

   Insure that the TE (Transfer Enable) bit has been set to a logic 1. This allows data written in the outer buffers to be transferred into the real time clock registers.

4. Clock will not run

   Insure that the oscillator enable bit ($\overline{EOSC}$, bit 7 of register 9) is set to a logic 0.

5. Alarm Flags do not work

   Alarm flags are set only as long as the corresponding interrupt output is active. Therefore when the interrupts are set to operate in pulse mode, the alarm flag will be set only during the active pulse of the interrupt. If level mode operation is selected, the interrupt will remain until the alarm condition is cleared.

APPENDIX: COMMAND REGISTER

Address location 0Bh is the Command Register where mask bits, control bits and flag bits reside. The operation of each bit is as follows:

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
TE	IPSW	IBH/LO	PU/LVL	WAM	TDM	WAF	TDF

TE (Bit 7 Transfer enable) - This bit when set to a logic 0 will disable the transfer of data between internal and external clock registers. The contents in the external clock registers are now frozen and reads or writes will not be affected with updates. This bit must be set to a logic 1 to allow updates.

IPSW (Bit 6 Interrupt switch) - When set to a logic 1, $\overline{\text{INTA}}$ is the Time of Day Alarm and INTB/($\overline{\text{INTB}}$) is the Watchdog Alarm. When set to logic 0, this bit reverses the output pins. $\overline{\text{INTA}}$ is now the Watchdog Alarm output and INTB/($\overline{\text{INTB}}$) is the Time of Day Alarm output.

IBH/LO (Bit 5 Interrupt B Sink or Source Current) - When this bit is set to a logic 1 and V_{CC} is applied, INTB/($\overline{\text{INTB}}$) will source current (see DC characteristics I_{OH}). When this bit is set to a logic 0, INTB will sink current (see DC characteristics I_{OL}).

PU/LVL (Bit 4 Interrupt pulse mode or level mode) - This bit determines whether both interrupts will output a pulse or level signal. When set to a logic 0, $\overline{\text{INTA}}$ and INTB/($\overline{\text{INTB}}$) will be in the level mode. When this bit is set to a logic 1, the pulse mode is selected and $\overline{\text{INTA}}$ will sink current for a minimum of 3 ms and then release. INTB/($\overline{\text{INTB}}$) will either sink or source current, depending on the condition of bit 5, for a minimum of 3 ms and then release.

WAM (Bit 3 Watchdog Alarm Mask) - When this bit is set to a logic 0, the Watchdog Interrupt output will be activated. The activated state is determined by bits 1, 4, 5, and 6 of the COMMAND REGISTER. When this bit is set to a logic 1, the Watchdog interrupt output is deactivated.

TDM (Bit 2 Time of Day Alarm Mask) - When this bit is set to a logic 0, the Time of Day Alarm Interrupt output will be activated. The activated state is determined by bits 0, 4, 5, and 6 of the COMMAND REGISTER. When this bit is set to a logic 1, the Time of Day Alarm interrupt output is deactivated.

WAF (Bit 1 Watchdog Alarm Flag) - This bit is set to a logic 1 when a watchdog alarm interrupt occurs. This bit is read only.

The bit is reset when any of the Watchdog Alarm registers are accessed.

When the interrupt is in the pulse mode (see bit 4 definition), this flag will be in the logic 1 state only during the time the interrupt is active.

TDF (Bit 0 Time of Day Flag) - This is a read only bit. This bit is set to a logic 1 when a Time of Day alarm has occurred. The time the alarm occurred can be determined by reading the Time of Day Alarm registers. This bit is reset to a logic 0 state when any of the Time of Day Alarm registers are accessed.

When the interrupt is in the pulse mode (see bit 4 definition), this flag will be in the logic 1 state only during the time the interrupt is active.

DALLAS
SEMICONDUCTOR

Application Note 77
DS1585/87, DS1685/87 and DS17x85/87
Accessing Extended User RAM via Software

GENERAL OVERVIEW
The DS1585/87, DS1685/87 and DS17x85/87 include an additional block of extended user RAM. The memory capacity of each device varies as follows; the DS1585/87 provides 65,536 bits organized in an 8K x 8 block, the DS1685/87 provides 1,024 bits organized in a 128 x 8 block, and the DS17x85/87 provides 16,384, 32,768 or 65,536, bits organized in 2K x 8, 4K x 8, or 8K x 8, blocks respectively.

REGISTER PARTITIONING
Figure 1 illustrates how the register blocks have been partitioned into two separate banks, bank 0 and bank 1. A bank select bit, DV0 located in control register 0Ah (bit 4), is used to select which register bank to make accessible. When DV0 is written to a logic 0, bank 0 is selected and an additional 64 bytes of user RAM can be accessed. However, when DV0 is written to a logic 1, bank 1 is selected and the additional features, including the extended user RAM, can be accessed. The real time clock (RTC), control registers, and 50 bytes of user RAM are accessible from either bank, independent of the DV0 bit.

SOFTWARE COMMUNICATION PORTS
The extended user RAM communication ports reside in the bank 1 register block. The extended user RAM address ports are located in registers 50h and 51h, while the extended user RAM data port is located in register 53h. Register 50h contains the LSB address and register 51h contains the MSB address. The DS1685/87 requires only seven bits to address the extended RAM and therefore does not require the MSB address register, 51h. These three bank 1 registers provide the software interface necessary to access the

extended user RAM. The steps involved to read from and write to the extended RAM are listed below:

- Write the DV0 bit to a logic 1

- Write the LSB address to register 50h

- Write the MSB address (if required) to register 51h

- Read from or write to the data register, 53h

An automatic address increment feature, available with the DS17x85/87, simplifies the software required to access the extended user RAM. This feature can be enabled or disabled with a single bit, located in extended control register 4Ah, bit 5. This feature simplifies the software required to access consecutive RAM address locations.

PROTOCOL FOR PC APPLICATIONS
The processor I/O ports used to access CMOS RAM are 70h and 71h. Port 70h is the CMOS RAM address register and port 71h is the CMOS RAM data register. The flow chart shown in Figure 2 illustrates the software protocol for PC applications.

SUMMARY
The DS1585/87 also provides a hardware interface to access the extended user RAM, however, this requires additional hardware to implement. The extended user RAM software access method provides the user with the greatest flexibility when determining which RAM density is needed, without any hardware modifications, for the DS1685/87 and DS17x85/87 (2K, 4K, and 8K) devices.

REGISTER BLOCK PARTITIONING Figure 1
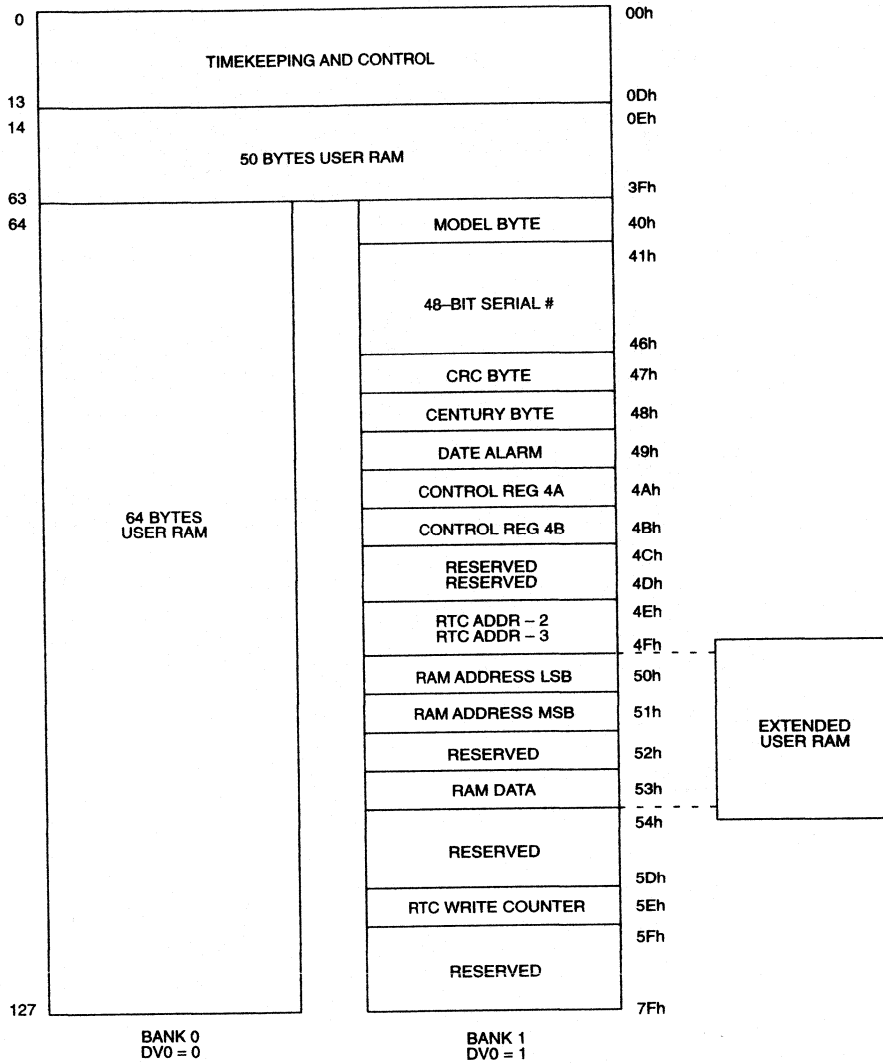
PC SOFTWARE PROTOCOL FLOW CHART Figure 2

START

DV0 = 1 ? — N → WRITE 0AH TO I/O PORT 70H → WRITE BIT 4 = 1 TO I/O PORT 71H

Y

RAM DENSITY > 256 X 8 ? — Y → DOES RAM ADDRESS LSB NEED UPDATING ? — Y → WRITE 50H TO I/O PORT 70H → WRITE RAM ADDRESS LSB TO I/O PORT 71H

N

N

DOES RAM ADDRESS NEED UPDATING ? — N

Y

DOES RAM ADDRESS MSB NEED UPDATING ? — N

Y

WRITE 50H TO I/O PORT 70H → WRITE RAM ADDRESS TO I/O PORT 71H

WRITE RAM ADDRESS MSB TO I/O PORT 71H ← WRITE 51H TO I/O PORT 70H

AUTO ADDRESS INC ON ? — Y

N

DS17X85/87 ONLY

WRITE 53H TO I/O PORT 70H

READ OR WRITE ?

READ → READ RAM DATA FROM I/O PORT 71H

WRITE → WRITE RAM DATA TO I/O PORT 71H

FINISHED ? — N

Y

END

DALLAS
SEMICONDUCTOR

Application Note 82
Using the Dallas Trickle Charge Timekeeper

DESCRIPTION

The Dallas Semiconductor DS1302 Trickle Charge Timekeeping Chip is a programmable 3–wire serial interface clock with a trickle charge circuit for using both rechargeable and non–rechargeable backup supplies. The real time clock/calendar provides seconds, minutes, hours, day, date, month, year information. The end of the month date is automatically adjusted for months with less than 31 days, including corrections for leap year. The clock operates in either the 24–hour or 12–hour format with an AM/PM indicator. The DS1302 also provides 31 bytes of nonvolatile SRAM for data storage. Interfacing the DS1302 with a microprocessor is simplified by using a synchronous serial communica-

tion. Only three wires are required to communicate with the clock/RAM: (1) $\overline{RST}$ (Reset), (2) I/O (Data Line), and (3) SCLK (Serial Clock). Data can be transferred to and from the clock/RAM one byte at a time or in a burst of up to 31 bytes. The DS1302 is designed to operate on very low power and retain data and clock information on less than 1 microwatt. The DS1302 is designed to be completely compatible with designs that are currently using the DS1202. This compatibility allows the DS1302 to be dropped directly into a DS1202 socket. Then the optional trickle charge circuit on the DS1302 can be used to backup the system time and data with a super cap or a rechargeable battery.

DS1302 PROGRAMMABLE TRICKLE CHARGER Figure 1

090794 1/6
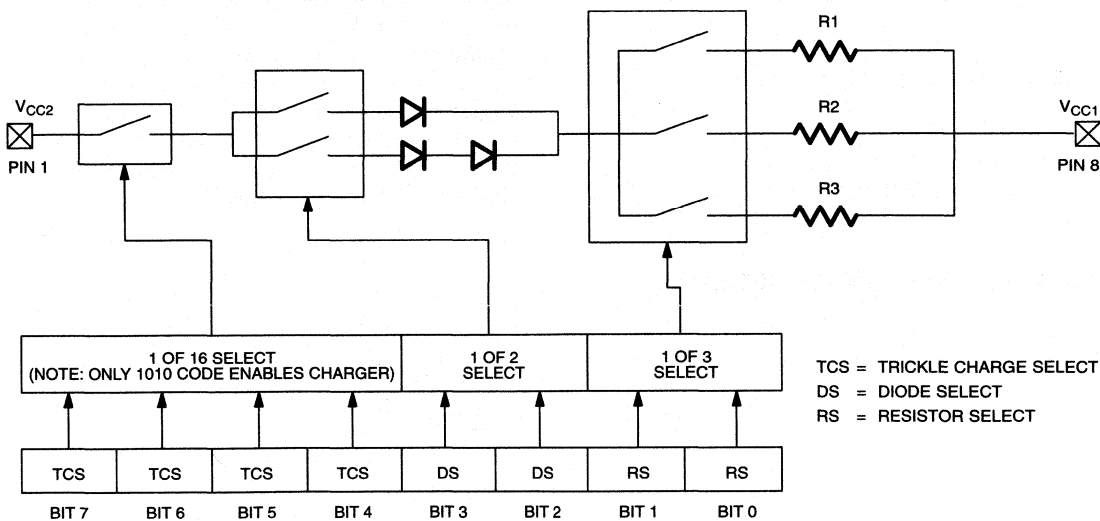
TRICKLE CHARGER

The trickle charge circuit is shown in Figure 1 along with the trickle charge register. To enable the trickle charger the desired path through the circuit must be selected and the appropriate pattern written to the trickle charge register. The trickle charge select (TCS) bits (bits 4 – 7) control the selection of the trickle charger. In order to prevent accidental enabling, only a pattern of 1010 will enable the trickle charger. All other patterns will disable the trickle charger. The DS1302 powers up with the trickle charger disabled. The diode select (DS) bits (bits 2 – 3) select whether one diode or two diodes are connected between V_{CC2} and V_{CC1}. If DS is 01, one diode is selected or if DS is 10, two diodes are selected. If DS is 00 or 11 the trickle charger is disabled independent of TCS. The RS bits (bits 0 – 1) select the resistor that is connected between V_{CC2} and V_{CC1}. The resistor selected by the resistor select (RS) bits is as follows:

RS BITS	RESISTOR	TYPICAL VALUE
00	None	None
01	R1	$2K\Omega$
10	R2	$4K\Omega$
11	R3	$8K\Omega$

If RS is 00 the trickle charger is disabled independent of TCS.

Diode and resistor selection is determined by the user according to the maximum current desired for battery or super cap charging. The maximum charging current can be calculated as illustrated in the following example. Assume that a system power supply of 5V is applied to V_{CC2} and a super cap is connected to V_{CC1}. Also, assume that the trickle charger has been enabled with 1 diode and resistor R1 between V_{CC2} and V_{CC1}. The maximum current I_{MAX} would therefore be calculated as follows:

$$I_{MAX} = (5.0V - \text{diode drop})/R1$$
$$\sim (5.0V - 0.7V)/2K\Omega$$
$$\sim 2.2 \text{ mA}$$

Obviously, as the super cap charges, the voltage drop between V_{CC2} and V_{CC1} will decrease and therefore the charge current will decrease (please see curves in Trickle Charge Characteristics section).

POWER CONTROL

The DS1302 can be powered in several different ways. The first method, shown in Figure 2, illustrates the DS1302 being supplied by only one power supply. In Figure 2a the power supply is connected to V_{CC2} (pin 1) and in Figure 2b the power supply is connected to V_{CC1} (pin 8). In each case the unused power pin, V_{CC1} or V_{CC2}, is grounded. The second method, Figure 3, illustrates the DS1302 being backed up using a non–rechargeable battery connected to V_{CC1}. In these two cases the trickle charge circuit has been disabled. In the final case, Figure 4, the DS1302 is being backed up by connecting a super cap, Figure 4a, or a rechargeable battery, Figure 4b, to V_{CC1}. In this case the trickle charge circuit has been enabled.

SINGLE POWER SUPPLY OPTION Figure 2
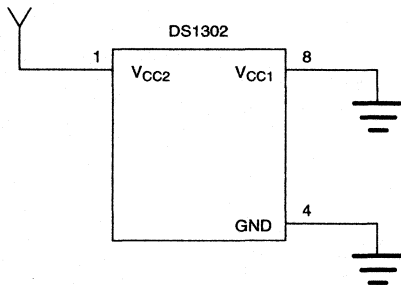
2a
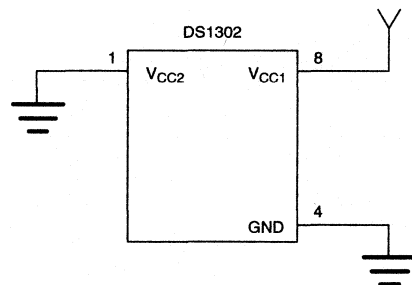
2b

NON–RECHARGEABLE BATTERY BACKUP Figure 3
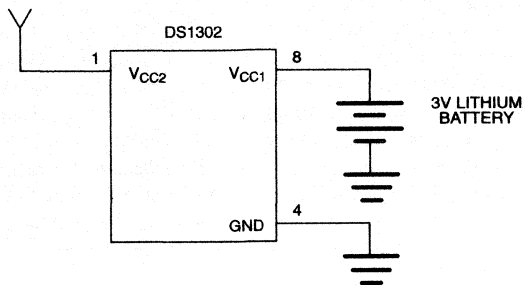
SUPER CAP OR RECHARGEABLE BATTERY BACKUP Figure 4
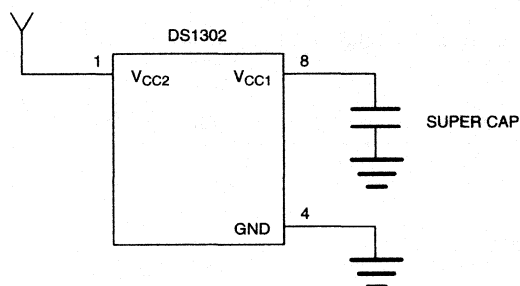
4a
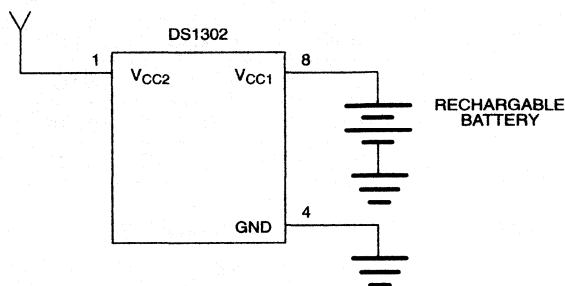
4b

TRICKLE CHARGE CHARACTERISTICS

Charging the Super Cap – As was discussed earlier the maximum current, I_{MAX}, required by the trickle charge circuit can be calculated by inserting the correct values selected in the trickle charge register into the following equation:

$$I_{MAX} = (V_{CC2} - \text{diode drop})/R$$

Table 1 contains the values of I_{MAX} for V_{CC2} values of 4.5V, 5.0V and 5.5V; 1 diode drop and 2 diode drops; resistor values of 2000Ω, 4000Ω and 8000Ω.

Also, the charging current can be modeled as a function of charge time. Both the super cap voltage and charging current as a function of time are represented in Figure 5. The equation to model the super cap voltage as a function of time is

$$V(t) = V_{MAX} [1 - e^{(-t/RC)}]$$

where

V(t)	=	Super Cap Voltage
V_{MAX}	=	(V_{CC2} – n Diode Drops), n=1,2
R	=	Internal Trickle Charge Resistor
C	=	Super Cap Capacitance

The time needed to charge the super cap to 95% of V_{MAX} is given in Table 2. Note that the time required to charge the super cap to 95% of the value of V_{MAX} is independent of the value of V_{MAX}. The equation which models the charging current as a function of time is given as

$$I(t) = V_{MAX}/R * e^{(-t/RC)}$$

where

I(t)	=	Charging Current
V_{MAX}	=	(V_{CC2} – n Diode Drops), n=1,2
R	=	Internal Trickle Charge Resistor
C	=	Super Cap Capacitance

Discharging the Super Cap – When modeling the DS1302 for the time to discharge the super cap the DS1302 characterization data was used to observe that the I_{CC1T}, Timekeeping Current through V_{CC1}, was linear. This implies that it is proper to represent the DS1302 as a resistive load, R_L, through which the super cap will be discharged. Using the data sheet spec of I_{CC1T} max of 0.3 μA at 2.5 V_{CC1} gives a value for R_L of 8.3MΩ. Then the equation modeling the discharging of the super cap is given by

$$V(t) = V_{MAX} * e^{(-t/R_L C)}$$

where

V(t)	=	Super Cap Voltage
V_{MAX}	=	(V_{CC2} – n Diode Drops), n=1,2
R_L	=	DS1302 Load Resistance
C	=	Super Cap Capacitance

The calculated values for the time required to discharge the super cap to 2V are given in Table 3 and a sample of the super cap voltage as a function of discharge time is given in Figure 6.

CALCULATED VALUES OF I_{MAX} Table 1

	2000Ω		4000Ω		8000Ω		
V_{CC2}	1 diode	2 diodes	1 diode	2 diodes	1 diode	2 diodes	UNITS
4.5V	1.90	1.55	0.95	0.78	0.48	0.39	mA
5.0V	2.15	1.80	1.08	0.90	0.54	0.45	mA
5.5V	2.40	2.05	1.20	1.03	0.60	0.51	mA

CHARGING TIME FOR SUPER CAP TO 95% OF V_{MAX} Table 2

	2000Ω	4000Ω	8000Ω	UNITS
Super Cap=0.047 F	4.7	9.4	18.8	minutes
Super Cap=0.47 F	46.9	93.9	187.7	minutes
Super Cap=1.5 F	149.8	299.6	599.2	minutes

SUPER CAP DISCHARGE TIME TO 2V Table 3

	0.047F		0.47F		1.5F		
V_{CC2}	1 diode	2 diodes	1 diode	2 diodes	1 diode	2 diodes	UNITS
4.5V	69.8	47.7	698.3	476.8	2228.7	1521.7	hours
5.0V	83.3	63.9	832.8	639.5	2657.9	2040.9	hours
5.5V	95.2	78.1	952.5	780.9	3039.8	2492.5	hours

SUPER CAP CHARGING CHARACTERISTICS Figure 5

SUPER CAP CHARGE TIME – 0.47F
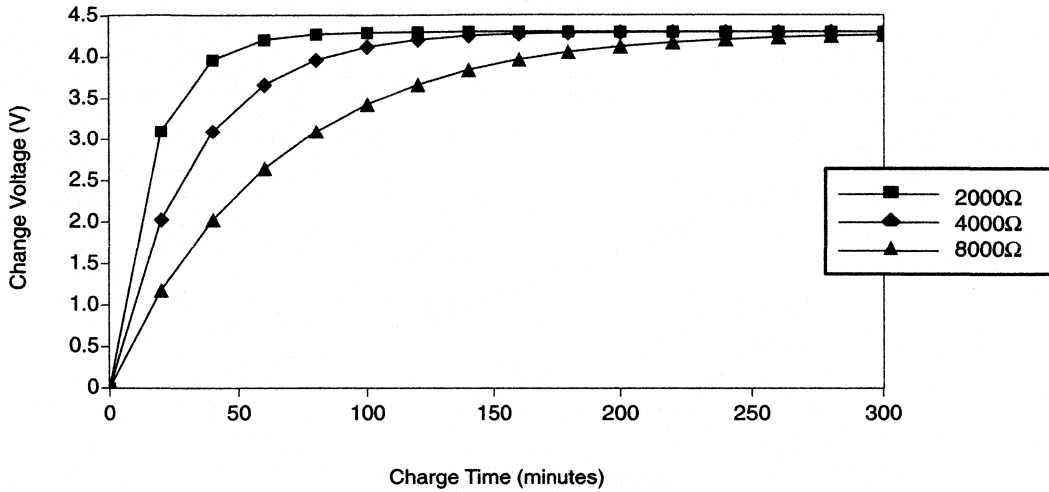
SUPER CAP CHARGE CURRENT – 0.47F
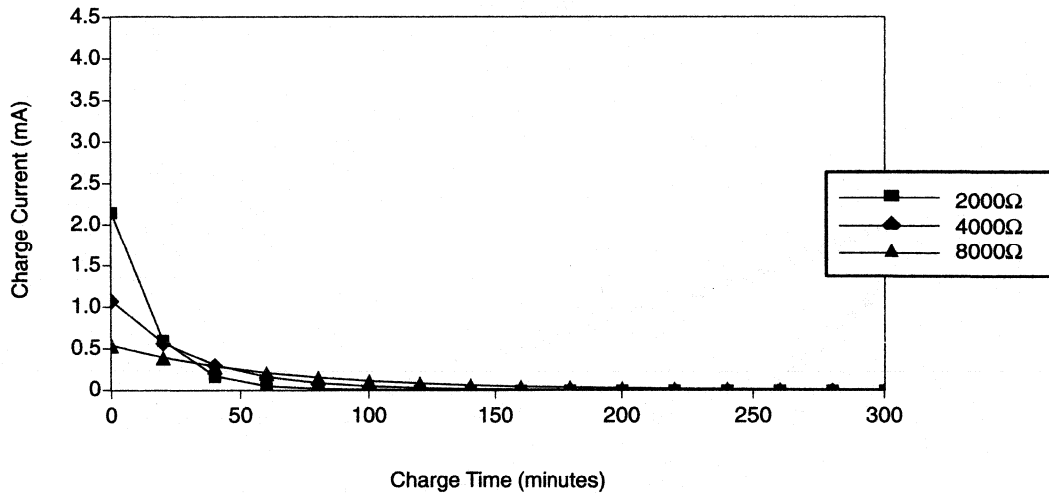
SUPER CAP DISCHARGING CHARACTERISTICS Figure 6

SUPER CAP DISCHARGE TIME
$V_{MAX}=4.3V$

SUPER CAP DISCHARGE TIME
$V_{MAX}=3.6V$

DALLAS
SEMICONDUCTOR

Application Note 90
Using the PC Clock Extended Features
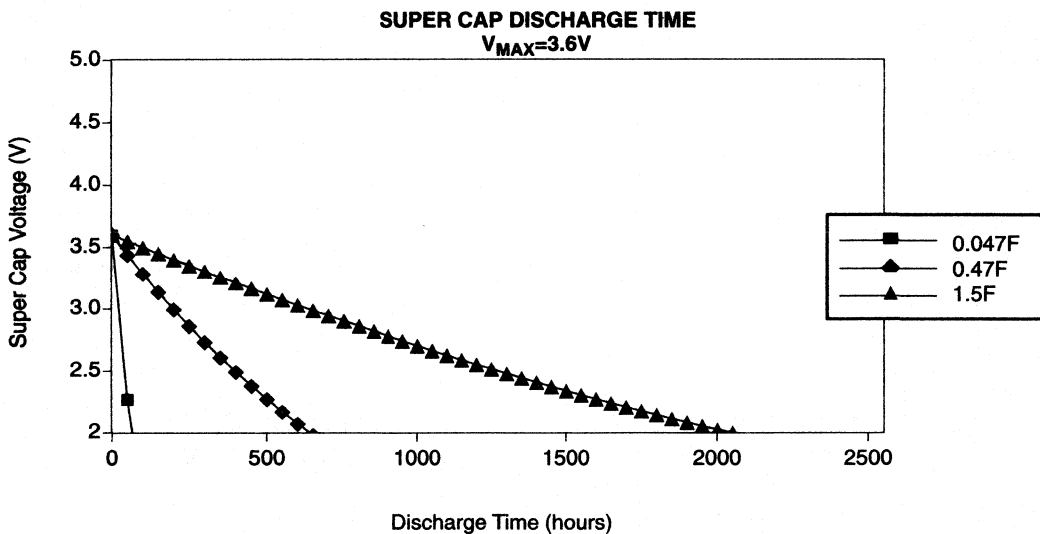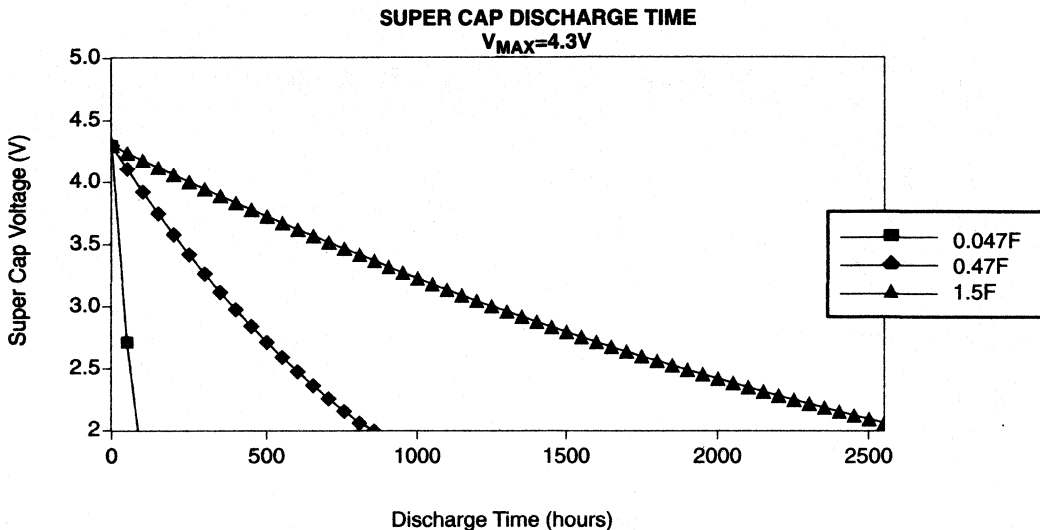
DESCRIPTION

This application note discusses the extended features for the following PC Real Time Clocks (RTCs): DS1585/87, DS1685/87, DS1688/91, DS17285/87, DS17485/87, DS17A84/86, and DS17885/87. Also included is an initialization flow chart for the PC RTCs. All of these devices incorporate these extended features unless otherwise noted.

EXTENDED FEATURES

Auxiliary Battery Input (V$_{BAUX}$)

The auxiliary battery input, V$_{BAUX}$, provides the power required to use the kickstart, wake up, power–on output, and the 32 KHz square wave in the absence of system power. The timekeeping and RAM data can also be maintained by this power source. If this input is going to be used, bit 7 in control register 4Bh (ABE) must be set to a logic 1. If this input is not going to be used then ABE must be set to a logic 0 and the V$_{BAUX}$ input tied to ground.

Kickstart Input ($\overline{KS}$)

The kickstart pin is intended to be used in the battery backed mode in conjunction with the $\overline{PWR}$ signal providing systems with power management control. A signal providing a ground closure will force the $\overline{PWR}$ signal to transition low. The $\overline{KS}$ pin can also be used as an interrupt input while V$_{CC}$ is applied.

Power–On Output ($\overline{PWR}$)

This output is open drain and requires an external pull–up resistor for proper operation. This signal is intended to be used in conjunction with $\overline{KS}$ and the wakeup alarm for power management features.

32 KHz Square Wave Output (SQW)

The SQW output pin can provide a 32 KHz square wave for power management purposes. The DS1685/87, DS1688/91, DS17285/87, DS17485/87, and DS17885/87 will provide the 32 KHz each time the system power, V$_{CC}$, is applied. The DS1585/87,

DS1689/93, and DS17A84/86 must be programmed, to output the 32 KHz, after the system power is applied.

Silicon Serial Number

A unique 64–bit silicon serial number is located in bank 1 registers 40h – 47h. The serial number is divided into three separate parts. The first byte, location 40h, contains a model number to identify the device type and revision. The following is a list of the model numbers currently assigned for the devices mentioned above:

Model #	Part #
70h	DS1585/87
71h	DS1685/87
72h	DS17285/87
73h	DS1688/91
73h	DS1689/93
74h	DS17485/87
78h	DS17885/87

The second part of the serial number is a unique 48–bit binary number located in registers 41h – 46h. The third part of the serial number is located in register 47h and contains a CRC number used to validate the data in registers 40h – 46h. This eight byte serial number is read only.

Century Byte

The century byte, located in bank 1 register 48h, will update every 100 years keeping track of the century.

Date Alarm Byte

The date alarm byte, located in bank 1 register 49h, can be used in conjunction with the time of day alarm providing a system with a wakeup alarm programmable up to 31 days.

Control Registers 4Ah and 4Bh

See the Register Description section.

082495 1/11

Extended RAM Software Port

The extended user RAM can be accessed through software via three internal registers located in bank 1 locations 50h, 51h, and 53h. Locations 50h and 51h are used for the RAM address and location 53h is used for the data transfer. This unique feature eliminates the need for external hardware and at the same time provides the ability to switch RAM densities without any hardware modification required. This feature is not available on the DS1688/91 or DS1689/93.

V_{CC} Elapsed Time Counter

A 32–bit V_{CC} powered elapsed time counter, located in bank 1 registers 54h (LSB) through 57h (MSB), will keep track of how long system power has been applied. This counter will update once per second as long as V_{CC} is within nominal limits and the oscillator and countdown chain are enabled. When V_{CC} falls outside of the nominal limits the counter is halted and the elapsed time is retained. This counter can be read or written at the users discretion. This feature is available only on the DS1688/91 and DS1689/93.

V_{BAT} Elapsed Time Counter

A 32–bit V_{BAT} powered elapsed time counter, located in bank 1 registers 58h (LSB) through 5Bh (MSB), will keep track of how long the system has been in service. This counter will run continuously and update once per second as long as V_{BAT} or V_{BAUX} is within nominal limits and the oscillator and countdown chain are enabled. This counter can be read or written at the user's discretion. This feature is available only on the DS1688/91 and DS1689/93.

Power Cycle Counter

A 16–bit power cycle counter, located in bank 1 registers 5Ch (LSB) and 5Dh (MSB), will keep track of the number of times a system is powered on and off. Each time system power, V_{CC}, is applied within nominal limits, the counter will be incremented by one. This counter can be read and written at the users discretion. This feature is available only on the DS1688/91 and DS1689/93.

Additional Serial Number

An additional 64–bit customer specific serial number or ROM is located in bank 1 registers 60h through 67h. This feature is available only on the DS1688/91 and DS1689/93.

Burst Mode

The DS17285/87, DS17485/87, and DS17885/87 are the only devices which provide the burst mode option. The model byte uniquely identifies these devices as having burst mode.

REGISTER DESCRIPTION

LOCATION	DESCRIPTION
0Ah	Control Register A
	Bit 7 Update In Progress (UIP) – (read only) 0=time/date can be read 1=update in progress

	Bits 6 – 4 Oscillator Control (DV2–DV0)
	 DV2=countdown chain
	     0=countdown chain enabled
	     1=resets countdown chain only if DV1=1
	 DV1=oscillator enable
	     0=oscillator off
	     1=oscillator on
	 DV0=bank select
	     0=original bank
	     1=extended registers

	Bits 3 – 0 Rate Selection (RS3–RS0) These bits define the square wave output frequency and the periodic interrupt rate.

LOCATION	DESCRIPTION
0Bh	Control Register B Bit 7 Halt Clock Updates (SET) 0=updates time/date once per second 1=time/date updates are inhibitied Bit 6 Periodic Interrupt Enable (PIE) 0=disabled 1=enabled Bit 5 Alarm Interrupt Enable (AIE) 0=disabled 1=enabled Bit 4 Update–Ended Interrupt Enable (UIE) 0=disabled 1=enabled Bit 3 Square Wave Enable (SQWE) 0=disabled 1=enabled Bit 2 Time and Date Data Mode (DM) 0=binary coded decimal (BDC) format 1=binary format Bit 1 Hour Format (24/12) 0=12 hour mode 1=24 hour mode Bit 0 Daylight Savings Time Enable (DSE) 0=disabled 1=enabled

LOCATION	DESCRIPTION
0Ch	Status Register C (read only) Bit 7 — Interrupt Request Flag (IRQF) – (read only) Bit 6 — Periodic Interrupt (PF) – (read only) Bit 5 — Alarm Interrupt Flag (AF) – (read only) Bit 4 — Update–Ended Interrupt Flag (UF) – (read only) Bits 3 – 0 — Reserved (read only logic 0)
0Dh	Status Register D (read only) Bit 7 — Valid RAM and Time (VRT) – (read only) 0=battery low; CMOS RAM invalid 1=battery good; CMOS RAM valid Bits 6 – 0 — Reserved (read only logic 0)
04Ah	Extended Control Register 4A Bit 7 — Valid RAM and Time 2 (VRT2) – (read only) 0=auxiliary battery is low 1=auxiliary battery is good Bit 6 — Increment in Progress (INCR) – (read only) 0=time/date has been updated 1=time/date is incrementing and checking alarms Bit 5 — Burst Mode Enable (BME) – See Note 1 0=single byte extended RAM reads and writes 1=auto increments address for extended RAM reads and writes Bit 4 — Reserved Bit 3 — Power Active Bar (PAB) 0=$\overline{PWR}$ pin is in the active low state 1=$\overline{PWR}$ pin is in the inactive high state Bit 2 — RAM Clear Flag (RF) 0=cleared state (must be written) 1=high to low transition detected on $\overline{RCLR}$ if RCE=1 Bit 1 — Wakeup Alarm Flag (WF) 0=cleared state (must be written) 1=wakeup alarm condition has occurred Bit 0 — Kickstart Flag (KF) 0=cleared state (must be written) 1=high to low transition detected on $\overline{KS}$ input

LOCATION	DESCRIPTION
04Bh	Extended Control Register 4B
	Bit 7 Auxiliary Battery Enable (ABE) 0=auxiliary battery input not being used 1=auxiliary battery input used for extended features
	Bit 6 Enable 32 KHz (E32K) 0=32 KHz disabled 1=32 KHz enabled to be output on the SQW pin
	Bit 5 Crystal Selection (CS) – See Note 2 0=oscillator configured for a crystal with a load capacitance of 6 pF 1=oscillator configured for a crystal with a load capacitance of 12.5 pF
	Bit 4 RAM Clear Enable (RCE) 0=RAM clear function is disabled 1=allows $\overline{RCLR}$ pin to clear user RAM
	Bit 3 Power Active Bar Reset Select (PRS) – See Note 3 0=$\overline{PWR}$ pin will go High–Z when entering power fail 1=$\overline{PWR}$ pin remains active when entering power fail
	Bit 2 RAM Clear Interrupt Enable (RIE) 0=disabled 1=enables RAM clear function to generate interrupts
	Bit 1 Wakeup Alarm Interrupt Enable (WIE) 0=disabled 1=enables wakeup alarm to generate interrupts
	Bit 0 Kickstart Interrupt Enable (KSE) 0=disabled 1=enables $\overline{KS}$ to generate interrupts

NOTES:

1. Burst Mode Enable (BME) is available only on the DS17285/87, DS17485/87, and DS17885/87 devices.

2. Cystal Selection (CS) is not available on the DS1585/87 or DS17A84/86 devices.

3. Power Active Bar Reset Select (PRS) is not available on the DS1585/87 or DS17A84/86 devices.

PC RTC INITIALIZATION PROCEDURE

```
        ┌──────────┐
        │  START   │
        └────┬─────┘
             │
             ▼
┌────────────────────────┐
│  CONTROL REGISTER B     │────▶▷  TO 1
│  INITIALIZATION         │
└────────────────────────┘
```

```
FROM 2 ▷──▶┌──────────────┐──▶┌────────────────────────┐
           │ READ STATUS  │   │  CONTROL REGISTER A     │
           │ REGISTER C   │   │  INITIALIZATION         │
           └──────────────┘   └───────────┬────────────┘
                                          │
                                          ▼
                                        TO 3
```

```
┌────────────────────────┐
│  CONTROL REGISTER 4B    │◀────◁  FROM 4
│  INITIALIZATION         │
└───┬────────────────────┘
    │
    ▼
  TO 5
```

```
FROM 6
  │
  ▼
┌────────────────────────┐
│  CONTROL REGISTER 4A    │────▶▷  TO 7
│  INITIALIZATION         │
└────────────────────────┘
```

```
FROM 8 ▷──▶┌────────────────────┐
           │  SET TIME, DATE,    │
           │  AND ALARMS         │
           └─────────┬──────────┘
                     │
                     ▼
                   TO 9
```

```
┌──────────────────────────────────┐
│  SWITCH TO ORIGINAL BANK BY        │◀────◁  FROM 10
│  LOCIC ANDing EFh WITH             │
│  CONTROL REGISTER A AND            │
│  WRITING CONTROL REGISTER A        │
└───────────────┬────────────────────┘
                │
                ▼
        ┌──────────────┐
        │   FINISH     │
        └──────────────┘
```

(5)

CONTROL REGISTER 4B
INITIALIZATION

KICKSTART
INTERRUPT
USED — N → SET BIT 0=0

↓ Y

SET BIT 0=1

WAKEUP ALARM
INTERRUPT
USED — N → SET BIT 1=0

↓ Y

SET BIT 1=1

RAM CLEAR
INTERRUPT
USED — N → SET BIT 2=0

↓ Y

SET BIT 2=1

POWER ACTIVE
BAR RESET
SELECT — High–Z → SET BIT 3=0

↓ ACTIVE

SET BIT 3=1

RAM CLEAR
FUNCTION
USED — N → SET BIT 4=0

↓ Y

SET BIT 4=1

CRYSTAL
SELECTION — 6 pF → SET BIT 5=0

↓ 12 pF

SET BIT 5=1

32 KHz
OUTPUT
USED — N → SET BIT 6=0

↓ Y

SET BIT 6=1

AUXILIARY
BATTERY
USED — N → SET BIT 7=0

↓ Y

SET BIT 7=1

WRITE CONTROL
REGISTER 4B → (6)

082495 9/11

55

```
                                    ( 7 )
                                      |
                                      v
          +---------------------------+
          |  CONTROL REGISTER 4A      |
          |  INITIALIZATION           |
          +---------------------------+
                      |
                      v
          +---------------------------+
          |  RESET                    |
          |  KICKSTART FLAG           |
          |  SET BIT 0=0              |
          +---------------------------+
                      |
                      v
          +---------------------------+
          |  RESET WAKEUP             |
          |  ALARM FLAG               |
          |  SET BIT 1=0              |
          +---------------------------+
                      |
                      v
          +---------------------------+
          |  RESET RAM                |
          |  CLEAR FLAG               |
          |  SET BIT 2=0              |
          +---------------------------+
                      |
                      v
```

POWER ACTIVE BAR STATE

ACTIVE LOW → SET BIT 3=0

INACTIVE HIGH → SET BIT 3=1

BIT 4 IS RESERVED

BURST MODE ON/OFF

OFF → SET BIT 5=0

YON → SET BIT 5=1

BITS 6 AND 7 ARE READ ONLY

WRITE CONTROL REGISTER 4A

(8)

(9)

HALT CLOCK UPDATES BY
LOGIC ORing 80h WITH
CONTROL REGISTER B AND
WRITING CONTROL REGISTER B

SET THE
TIME

WRITE THE SECONDS TO ADDRESS 00h
WRITE THE MINUTES TO ADDRESS 02h
WRITE THE HOUR TO ADDRESS 04h

WRITE THE DAY OF THE WEEK TO ADDRESS 06h
WRITE THE DAY OF THE MONTH TO ADDRESS 07h
WRITE THE MONTH TO ADDRESS 08h
WRITE THE YEAR TO ADDRESS 09h
WRITE THE CENTURY TO ADDRESS 48h (BANK 1)

SET THE
DATE

SET THE
ALARMS

WRITE THE SECONDS ALARM TO ADDRESS 01h
WRITE THE MINUTES ALARM TO ADDRESS 03h
WRITE THE HOURS ALARM TO ADDRESS 05h
WRITE THE DATE ALARM TO ADDRESS 49h (BANK 1)

START CLOCK UPDATES BY LOGIC
ANDing 7FhWITH CONTROL
REGISTER B AND WRITING
CONTROL REGISTER B

(10)

DALLAS
SEMICONDUCTOR

Application Note 100
DS1670 Portable System Controller

FEATURES

There is an increasing trend in the industry today towards portable battery–operated products. These products typically require extremely small form factors and low power. The DS1670 Portable System Controller is designed to help the portable system designer in meeting these two requirements by integrating key functions typically needed in these systems while consuming very little power.

Key functions frequently required in portable products include a real time clock; nonvolatile RAM control to insure that data is not lost in the event of a power down or battery fail; microprocessor monitoring to insure that the microprocessor does not "run out of control"; and analog–to–digital converters for interfacing with external environmental sensors, battery monitoring, etc. The DS1670 Portable System Controller replaces as many as four separate integrated circuits by incorporating all of these functions in a single device. The DS1670 provides a very accurate real time clock, nonvolatile SRAM controller, microprocessor monitor, and 3–channel 8–bit analog–to–digital converter. Communication with the device is established through a simple 3–wire interface.

USING THE DS1670

The DS1670 can be directly interfaced with any microprocessor or microcontroller through a 3–wire serial interface. The serial interface gives the microprocessor access to the DS1670 registers which consists of the Real Time Clock Registers, Control Register, Status Register, Watchdog Register, and ADC (Analog–to–Digital Converter) Register. Figures 1 and 2 illustrate a block diagram and address map of the DS1670, respectively. The following paragraphs briefly describe the operation of the various components of the DS1670 as well as the functions of the various control and status registers. For more detailed information please consult the DS1670 datasheet.

DS1670 BLOCK DIAGRAM Figure 1

TYPICAL APPLICATION Figure 2

```
                                                    SRAM
                                                    VCC
                                                    CE*

                              X1  X2                SRAM
                              DS1670               VCC
      386EX                   VCCO                  CE*
       or
      80186      P1.0    CS        CEOH*
                 P1.1    SCLK      CEOL*
                 P1.2    I/O       VBAT
                 INTO    INT                           SUPER
                 P1.3    ST*       VCC              CAPACITOR
               RESET*    RST*                    (OR RECHARGEABLE
                                                     BATTERY)
                 CS*     CEI*      AIN2    OTHER
                 BHE*    BHE*      AIN1    ANALOG      BATTERY
                 BLE*    BLE*      AIN0    INPUTS   (PRIMARY POWER)
                                          BATTERY
                                        "GAS GAUGE"
```
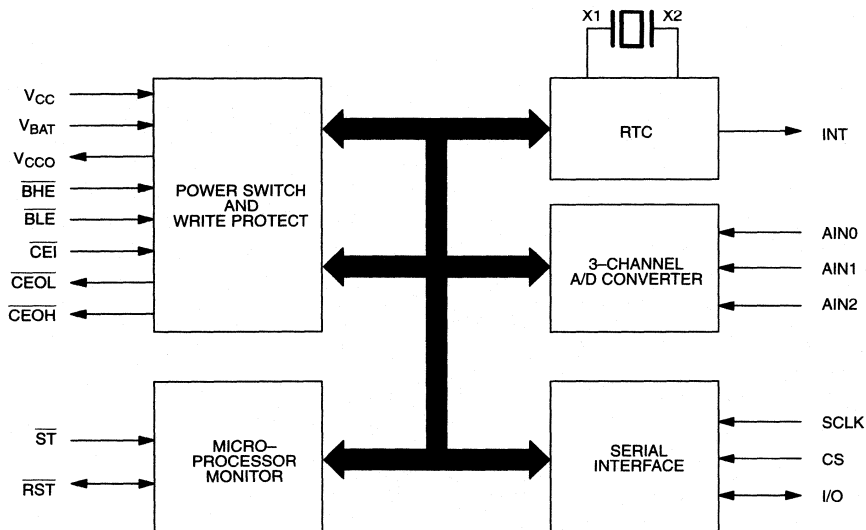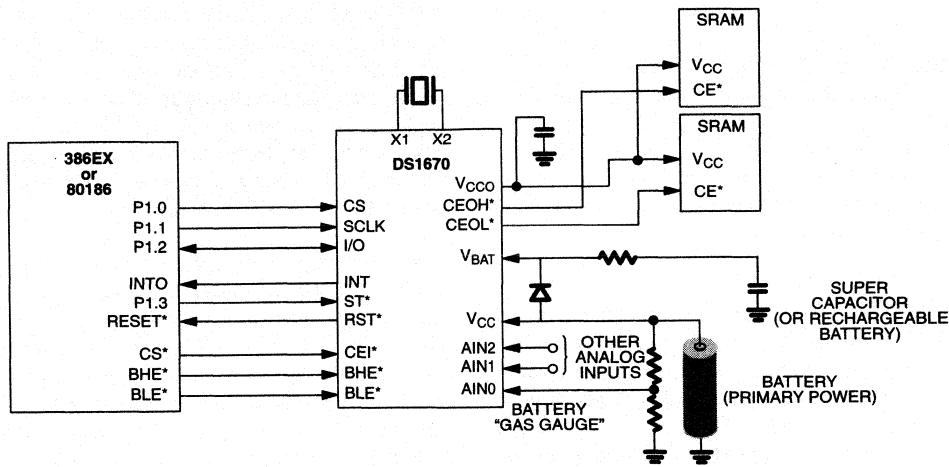
REAL TIME CLOCK

The Real Time Clock (RTC) provides seconds, minutes, hours, day, date, month, and year information with leap year compensation. The RTC has a back–up power supply input (V_{BAT}) that enables it to continue to keep time even when the primary supply connected to V_{CC} is removed. The RTC also provides an alarm interrupt which is functional both when the DS1670 is powered by the system power supply or when in battery back–up operation. Functionality while in battery back–up operation allows the alarm to "wake up" a system that is powered down.

NONVOLATILE SRAM CONTROL

The nonvolatile SRAM controller of the DS1670 provides two important functions. First, it insures a constant uninterrupted power supply to the system SRAMs. This is provided by the V_{CC} Output pin (V_{CCO}). V_{CCO} is internally connected to V_{CC} when V_{CC} is within nominal limits. However, when V_{CC} is below nominal limits, V_{CCO} is connected to the power source connected to V_{BAT} (typically a lithium battery, rechargeable battery, or super capacitor). The second important feature of the nonvolatile controller is that it gates the chip enable signals to the SRAMs. This insures that the microprocessor can access the SRAMs only when the primary power supply is within nominal limits, thus eliminating the corruption of data during power transients. It should also be noted that the DS1670 provides nonvolatile control for two separate SRAMs. The two inputs from the microprocessor are Byte High Enable ($\overline{BHE}$) and Byte Low Enable ($\overline{BLE}$). These inputs correspond to the two output Chip Enable Output High ($\overline{CEOH}$) and Chip Enable Output Low ($\overline{CEOL}$).

MICROPROCESSOR MONITOR

The microprocessor monitor circuitry of the DS1670 provides three basic functions.

1. A precision temperature–compensated reference and comparator circuit monitors the status of V_{CC}. When an out–of–tolerance condition occurs, an internal power fail signal is generated which forces the reset to the active state. When V_{CC} returns to an in–tolerance condition, the reset signals are kept in the active state for 250 ms to allow the power supply and processor to stabilize.

2. Pushbutton reset control: The DS1670 debounces a pushbutton input and guarantees an active reset pulse width of 250 ms.

3. Watchdog timer: The DS1670 has an internal timer that forces the reset signals to the active state if the strobe input is not driven low prior to watchdog time-out.

ANALOG–TO–DIGITAL CONVERTER

The DS1670 also provides a 3–channel 8–bit successive approximation analog–to–digital converter. The converter has an internal 2.55 volt (typical) reference voltage generated by an on–board band–gap circuit. The A/D converter is monotonic (no missing codes) and has an internal analog filter to reduce high frequency noise.

An A/D conversion is the process of assigning a digital code to an analog input voltage. This code represents the input value as a fraction of the full scale voltage (FSV) range. Thus the FSV range is then divided by the A/D converter into 256 codes (8 bits). The FSV range is bounded by an upper limit equal to the reference voltage and the lower limit which is ground. The DS1670 has a FSV of 2.55V (typical) which provides a resolution of 10mV. An input voltage equal to the reference voltage converts to FFh while an input voltage equal to ground converts to 00h. The relative linearity of the A/D converter is ±0.5 LSB.

The analog input (AIN0, AIN1, or AIN2) for an analog–to–digital conversion is selected by the condition of the Analog Input Select (AIS) bits in the Control Register. Conversions occur every 10 ms and the result is placed in the ADC Register.

TYPICAL APPLICATION

Figure 3 illustrates a typical application using the DS1670. In this application, the DS1670 is shown interfacing with a 386EX or 80186 microprocessor (microprocessors designed with embedded applications in mind). Communication between the microprocessor and the DS1670 is accomplished easily through three port pins (P1.0–P1.2) which are connected to the 3–wire serial interface pins of the DS1670 Serial Clock (SCLK), Chip Select ($\overline{CS}$), and serial input/output (I/O). This serial interface provides access to the real time clock, ADC, control, and status registers of the DS1670.

NONVOLATILE SRAM CONTROL

The DS1670 also provides nonvolatile control of external SRAMs in this circuit example. The Chip Select ($\overline{CS}$), Byte High Enable ($\overline{BHE}$), and Byte Low Enable ($\overline{BLE}$) outputs of the microprocessor are *gated* through the DS1670 to its Chip Enable Output Low ($\overline{CEOL}$) and chip Enable Output High ($\overline{CEOH}$) output pins. The $\overline{CEOL}$ and $\overline{CEOH}$ pins can be driven to the active state only when V_{CC} is within nominal limits (>2.88V) thus preventing the system from receiving or writing corrupted data. Note that the structure of the SRAM access controller allows the microprocessor to access data in wordwide format (16 bits wide by enabling $\overline{CEOL}$ and $\overline{CEOH}$ simultaneously) or byte–wide format (8 bit wide by enabling either $\overline{CEOL}$ or $\overline{CEOH}$).

Power to the SRAMs is provided from the V_{CCO} pin of the DS1670. When V_{CC} is within nominal limits (greater than 2.7 volts typical), V_{CCO} is internally connected to V_{CC}. However, when V_{CC} goes below nominal limits, V_{CCO} is internally connected to $V_{BAT,}$ thus insuring that the SRAM data is maintained in the event of a primary power supply failure. $V_{BAT,}$ is typically connected to a lithium battery, rechargeable battery, or super capacitor. Figure 3 illustrates V_{BAT} connected to a super capacitor. The super capacitor is charged whenever the power source connected to V_{CC} is a diode drop above V_{BAT} plus the voltage drop across resistor R1. R1 is used to decrease the charge current to the super capacitor. Note that in this application, the primary power source connected to V_{CC} is a battery. This could be a rechargeable battery pack or perhaps a series of AA cells. Regardless of the type of battery used, the DS1670 allows the battery to be changed without the loss of important data. Depending on the size and type of super capacitor or battery connected to $V_{BAT,}$ the data will be saved from several hours to several years without replacing the main system battery. It should be noted also that when using a super capacitor, the limiting factor is often the internal leakage of the capacitor itself. In other words, the battery back–up current consumed by the DS1670 and SRAMs is negligible compared to the internal leakage of the capacitor.

MICROPROCESSOR MONITORING

The illustrated typical application also takes advantage of the microprocessor monitoring features of the DS1670. The $\overline{RST}$ of the DS1670 is connected to the $\overline{RESET}$ pin of the microprocessor. The DS1670 will generate a reset signal if V_{CC} goes out of the nominal limits or if the watchdog timer is allowed to expire. Toggling the strobe ($\overline{ST}$) input before the watchdog timer expires will keep the watchdog from generating a reset. The watchdog timer can be disabled or programmed by

the user to be 250 ms, 500 ms, or 1000 ms. It is also possible to connect the $\overline{RST}$ pin to a monetary pushbutton switch to allow a manual reset of the system. In this type of application the $\overline{RST}$ pin will produce a 250 ms active reset after being momentarily grounded by an external momentary switch.

REAL TIME CLOCK

Another important feature of the DS1670 is the Real Time Clock (RTC). The RTC provides the system with seconds, minutes, hours, day, date, month, and year information. The accuracy of the clock is typically ±1 minute per month at room temperature. An especially powerful function of the RTC is the alarm. The alarm gives the user flexibility to program events at a specific time of the day. The DS1670 can be programmed to generate a single alarm or, through the use of its alarm mask bits, can generate an alarm once every second, minute, hour, or day. Added flexibility is provided in that the alarm is functional both when the device is powered by V_{CC} or when powered by V_{BAT}. Generating an alarm while powered by V_{BAT} is especially useful because this allows the DS1670 to "wake up" a sleeping system. For example, a portable datalogger may need to collect data once every hour. This system could be placed in a low power "sleep mode" when idle. The alarm could be used to "wake up" the system at the appropriate time to collect data. After the collection is completed, software could place the system back into sleep mode where it would remain until the next alarm interrupt.

ANALOG–TO–DIGITAL CONVERTER

The DS1670 also has a 3–channel 8–bit analog–to–digital converter. In our typical application example, one input of the ADC is used as a battery "gas gauge". This input samples the voltage on the primary power supply battery which is first reduced through a voltage divider circuit. The microprocessor monitors this voltage and generates a low battery warning if the voltage falls below a specified threshold, thus signaling the end user that the battery needs to be changed. The additional ADC inputs could be interfaced with environmental sensors or other analog signals.

LOW POWER

The functional features of the DS1670 are certainly a great benefit to many system designers, but equally important in portable systems is low power to insure long battery life. It is probable that the majority of the time the DS1670 will sit idle in a powered up system. In this type of standby operation, the DS1670 will consume a maximum of 100 µA (200 µA maximum if the analog–to–digital converter is enabled). During active operation, when the 3–wire interface is being driven, the DS1670 will typically consume 5 mA (10 mA maximum). The lowest power mode of the device is when the primary power supply is removed and the device is being powered by V_{BAT}. In this mode, the device typically consumes 300 nA (500 nA maximum).

SUMMARY

Overall, it has been shown that the DS1670 Portable System Controller provides many of the functions necessary in battery backed portable products. The integration of these features onto a single piece of silicon replaces as many as four separate integrated circuits and is available in an extremely small 20–pin TSSOP package. The DS1670 is also ideally suited for battery backed products because of its low power consumption, particularly when the device is in standby or battery backed mode.

DS1670 ADDRESS MAP Figure 3

	BIT7								BIT0
00	0		10 SECONDS			SECONDS			
01	0		10 MINUTES			MINUTES			
02	0	12/24	10 HR A/P	10 HR		HOURS			
03	0	0	0	0	0		DAY		
04	0	0		10 DATE			DATE		
05	0	0	0	10 MO.			MONTH		
06		10 YEAR					YEAR		
07	M		10 SEC ALARM			SECONDS ALARM			
08	M		10 MIN ALARM			MINUTES ALARM			
09	M	12/24	10 HA A/P	10 HA		HOUR ALARM			
0A	M	0	0	0		DAY ALARM			
0B		CONTROL REGISTER							
0C		STATUS REGISTER							
0D		WATCHDOG REGISTER							
0E		ADC REGISTER							
0F		RESERVED							
7F									

NONVOLATILE MEMORIES

DALLAS
SEMICONDUCTOR

For many memory systems requiring nonvolatile memory, Dallas Semiconductor NV SRAMs offer an SRAM's ease of implementation coupled with write protection circuitry and a 10 year information storage capability. NV SRAMs automatically write protect themselves when they detect an out–of–tolerance condition (usually at 10% of V_{CC}), making them a secure receptacle for data to be protected during a power failure.

One issue that is not addressed by the secure write protection strategy of the NV SRAM is this: what happens to the data currently being processed during a power failure? If the voltage has fallen to 10% of V_{CC}, time has already run out to perform any system housekeeping functions such as storing off data and storing the state of the microprocessor. What is required to truly address this need to "save data before write–protecting

memory" is a method to detect an impending power failure well before the power supply has fallen to 10% of V_{CC} so that a microprocessor can perform these housekeeping functions.

One way to accomplish this task is to use a second voltage monitoring device. Dallas Semiconductor manufactures the DS1233B, a 5V–5% voltage monitor in a 3–pin TO–92 size package. This 5% monitor drives an active low reset signal, $\overline{RST}$, as soon as it detects an out–of–tolerance condition. This active low signal can be used as an $\overline{IRQ}$ input to a microprocessor, providing the microprocessor with an advanced warning that the power supply is falling, and giving it time to service the interrupt before the system's nonvolatile memory has been write protected. The following diagram illustrates this concept.

5V – 5% $\overline{IRQ}$ GENERATION
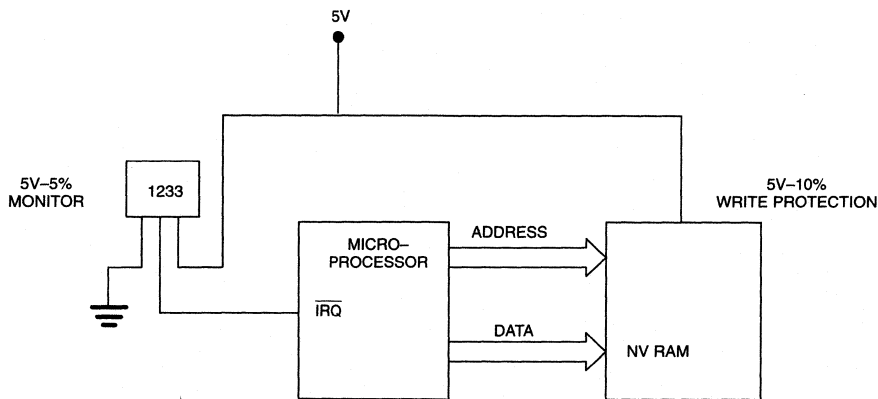
You might wonder of what use the time between a 5% and 10% drop in a 5V power supply could possibly be to a microprocessor. After all, don't power supplies fall rapidly when they do go through a hard failure? The answer is, yes, of course they do. But, fortunately, microprocessors can service interrupts and process information even faster. All that is required is that the system's interrupt servicing software be configured so

that it quickly identifies and services external interrupts. An example follows of how useful this time can be.

For the sake of this discussion, let's make several assumptions about the conditions that exist inside of the system in question. Let's assume:

---

1. That the power supply falls quickly, taking only 300 microseconds to fall from 4.75 to 4.0 volts.

2. That the microprocessor in question runs at a relatively moderate clock speed of 25 MHz.

3. That this microprocessor is a common 8–bit device, requiring on the order of six clocks to execute a single instruction.

With this set of givens, how many instructions should the processor be able to execute between the 5% and 10% trip points on a 5V power supply?

$$1/25 \text{ MHz} = 40 \text{ ns clocks}$$

six clocks/instruction = 240 ns per instruction

$$(4.75{-}4.00)/300 \; \mu s = .0025 \text{ V}/\mu s$$

5% – 10% drop = .25 V; hence 5%–10% drop = 100 μs

100 μs/ 240 ns per instruction = 416 instructions

Having 416 executable instructions at your disposal versus having none during a power down makes a big difference in saving 256 bytes of information or losing it, or in saving the state machine of the processor or losing it. In addition, the variables can be modified by design to give the processor even more time. The rate of fall of voltage of the power supply during a power failure can be slowed by adding capacitance. Processors requiring fewer than six clocks to execute an instruction can be used. In any case, using a DS1233B in conjunction with your NV SRAM requirements can give you the additional time you need to execute an orderly system shutdown, without corrupting your memory or allowing your microprocessor to run out of control.

ORDERING INFORMATION

DS1233B
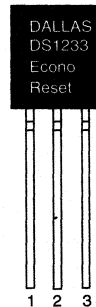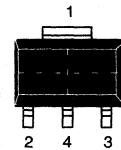
TO-92 Package   SOT-223 Package

NV SRAM PINOUTS

DS1220
2K X 8

Pin	Signal		Pin	Signal
1	A7		24	V_CC
2	A6		23	A8
3	A5		22	A9
4	A4		21	$\overline{WE}$
5	A3		20	$\overline{OE}$
6	A2		19	A10
7	A1		18	$\overline{CE}$
8	A0		17	DQ7
9	DQ0		16	DQ6
10	DQ1		15	DQ5
11	DQ2		14	DQ4
12	GND		13	DQ3

DS1225
8K X 8

Pin	Signal		Pin	Signal
1	NC		28	V_CC
2	A12		27	$\overline{WE}$
3	A7		26	NC
4	A6		25	A8
5	A5		24	A9
6	A4		23	A11
7	A3		22	$\overline{OE}$
8	A2		21	A10
9	A1		20	$\overline{CE}$
10	A0		19	DQ7
11	DQ0		18	DQ6
12	DQ1		17	DQ5
13	DQ2		16	DQ4
14	GND		15	DQ3

DS1230
DS1630
32K X 8

Pin	Signal		Pin	Signal
1	A14		28	V_CC
2	A12		27	$\overline{WE}$
3	A7		26	A13
4	A6		25	A8
5	A5		24	A9
6	A4		23	A11
7	A3		22	$\overline{OE}$
8	A2		21	A10
9	A1		20	$\overline{CE}$
10	A0		19	DQ7
11	DQ0		18	DQ6
12	DQ1		17	DQ5
13	DQ2		16	DQ4
14	GND		15	DQ3

DS1245
DS1645
128K X 8

Pin	Signal		Pin	Signal
1	NC		32	V_CC
2	A16		31	A15
3	A14		30	NC
4	A12		29	$\overline{WE}$
5	A7		28	A13
6	A6		27	A8
7	A5		26	A9
8	A4		25	A11
9	A3		24	$\overline{OE}$
10	A2		23	A10
11	A1		22	$\overline{CE}$
12	A0		21	DQ7
13	DQ0		20	DQ6
14	DQ1		19	DQ5
15	DQ2		18	DQ4
16	GND		17	DQ3

DS1250
DS1650
512K X 8

Pin	Signal		Pin	Signal
1	A18		32	V_CC
2	A16		31	A15
3	A14		30	A17
4	A12		29	$\overline{WE}$
5	A7		28	A13
6	A6		27	A8
7	A5		26	A9
8	A4		25	A11
9	A3		24	$\overline{OE}$
10	A2		23	A10
11	A1		22	$\overline{CE}$
12	A0		21	DQ7
13	DQ0		20	DQ6
14	DQ1		19	DQ5
15	DQ2		18	DQ4
16	GND		17	DQ3

DALLAS
SEMICONDUCTOR

Application Note 53
Designing a Nonvolatile 2M x 16 Memory Subsystem

Designing microprocessor cards and memory subsystem cards for backplane–based systems challenges system designers with problems not associated with single motherboard systems. Power supply stabilization, power supply monitoring, backup supply switching, and maintaining memory or microprocessor states when inserting or removing "hot" cards require that the system designer take special care to isolate system memory and microprocessor cards from transient conditions generated by the backplane or by other system cards. Dallas Semiconductor produces three product families, prefabricated memory products, CPU supervisory products, and integrated battery backup products, which are particularly well suited for subsystem card designs.

As an example let's say that you need to design a memory subsystem with the following minimum requirements:

1. Board must have single–sided assembly.

2. 2M x 16 of base memory required, with capabilities to support up to an additional 2M x 16 of extended memory, for a total of 4M x 16. Access times of 85 ns are required.

3. Memory must require no stand–alone refresh circuitry.

4. A backup power supply must be available to maintain the contents of the base memory and extended memory on the card.

5. Power supply (5V) must be monitored both on the subsystem card as well as on the backplane card.

6. Switching from V_{CC} to the secondary power supply must be automatically executed during a power–down or power failure.

7. Switching from the backup power supply to V_{CC} on power up must automatically occur once V_{CC} is in tolerance.

8. Switching system must be able to switch during a "hot card" situation without losing the contents of memory, requiring a smooth transition from a high current active memory state to a low current backup state.

9. The backup power supply must provide an advance warning of an impending out–of–tolerance condition on the backup supply.

10. Memory must be write protected during a power failure to prevent it from becoming corrupted.

While the list seems lengthy, these requirements are an essential minimum for many memory cards. Dallas Semiconductor provides condensed, ready–made solutions to these problems which simplify the card design, number of piece parts required, and real estate required for a solution. A block diagram of the required functions follows:

072996 1/4
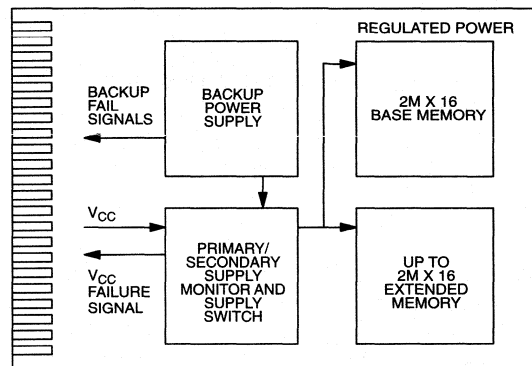
Each of these system functions can be realized using a condensed, off–the–shelf Dallas Semiconductor solution. Let's begin with the memory requirements.

In order to provide a low–voltage, backup–compatible memory, without the need for refresh circuitry, the best choice is to use SRAM memory. The 85 ns memory access requirement is available in SRAM as well. What we also need, however, is a way to provide system users with a convenient way of adding memory as they need it. Most likely, the end user does not want to have to add the memory SRAM by SRAM, or fumble with the individual parts themselves. Dallas Semiconductor's solution to this requirement is the DS2229. The DS2229 512K x 16–bit prefabricated memory Stik conforms to the JEDEC 80–pin connection standard. The high density word–wide configuration of the prefabricated memory Stik provides the memory card with more memory per surface area than surface–mounting the individual SRAMs directly onto the memory card. The modularity of a DS2229 makes it possible to provide the memory card's user with maximum flexibility in adding additional memory over the 2M x 16 minimum requirement. Four DS2229's satisfy the minimum requirement, with 80–pin JEDEC sockets on the card for up to four more DS2229's. Since the DS2229's are also removeable from the memory card, the card can be reconfigured as necessary, or DS2229's from one card can be used to upgrade the memory in other cards, or to provide base memory for other cards, reducing the overall cost of implementation for such a flexible memory subsystem.

512K X 16 80–PIN JEDEC STANDARD SIP STIK
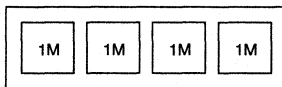
With items 1, 2, and 3 of the system requirements met, the next functions that can be easily consolidated into a one–chip solution are the power switching functions. The DS1336 is ideally suited for our system requirements (see Figure 2). The device can switch between a 3V or 5V, 1.5A supply to a backup 3V supply, with a maximum backup current of 4 mA. The DS1336 requires an external notification of a power failure. Once the external input is activated, the DS1336 immediately switches from the primary power supply to the backup power supply, in this case a lithium power source. Since the

DS1336 has two complemented power–fail inputs, $\overline{PF}$ and PF , one can be used for the on–card power monitor, and the other can be connected to a power–fail status indicator on the backplane which can provide an advanced warning of a power failure, before it reaches the memory card.

What about the primary and backup power supply management requirements? Again, Dallas Semiconductor offers several condensed solutions which are user–configurable to meet diverse system requirements. Let's begin with the power monitoring requirements.

If we want to monitor the status of our on–board power supply, backup supply, and also monitor the external connections to the backplane, we need to be able to monitor all of these supplies, and be able to use the output signals to drive high current switching circuitry. Dallas Semiconductor offers the DS1236 MicroManager, which condenses all of the required functions into one chip.

DS1236 MICROMANAGER Figure 1
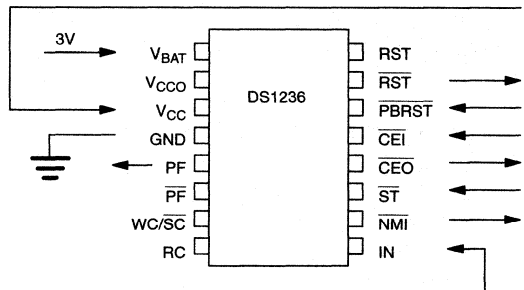
The DS1236 MicroManager monitors the primary power supply V_{CC} from the backplane, and the backup lithium source. If V_{CC} falls out of tolerance, the DS1236 provides two complemented power–fail signal outputs, PF and PF, either of which can be used to signal to a DS1336 to switch power from the primary supply to the backup lithium energy source. The DS1236 also has a user–definable voltage monitor, which in our memory card implementation can be used to monitor the status of the backup lithium source. In the implementation shown below, if the backup lithium source's voltage falls below 2.54V, the DS1236 will assert its $\overline{NMI}$ output to warn the memory controller or microprocessor that the backup power supply has failed. This feature satisfies requirement 9.

DS1336 AFTERBURNER 16–PIN DIP OR SOIC Figure 2
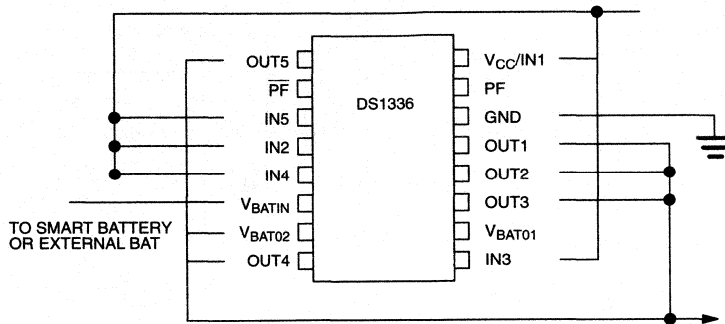
The last system requirement, that of write protecting memory, is accomplished by controlling the chip enable lines to the DS2229 prefabricated memory sticks. The chip enable input from the memory controller is fed to the DS1236, and is passed out as CEO during normal V_{CC} operating conditions. During an out–of–tolerance condition, the DS1236 holds CEO high so that a write to the memory with corrupted data can be prevented.

With the primary power supply monitoring requirements satisfied, we still require a method to switch from the higher active currents of the DS2229's to the very low data retention currents required for maintaining the memory in a standby state. The active current required for the system is:

$$8 \times (DS2229 \; I_{CC0}) = 8 \times 100 \; mA = 800 \; mA \; maximum$$

The data retention current required for the system is:

$$8 \times (DS2229 \; I_{CC} \; or \; standby) = 8 \times 8 \; \mu A = 64 \; \mu A$$

The power requirements for maintaining the memory contents in the absence of power is then:

2 years x 64 μA = 128 μA–years or 1122 mAh, so a minimum of a 1.2 Amp–Hour lithium source will be required.

To switch between the high current and low current requirements of the memory requires the ability to switch between the 5V high current supply and 3V backup supply, without dropping the current supply to the memory below the minimum required to keep it active. The DS1336 consolidates the required switching functions into a single 16–pin SOIC device capable of switching even more current than we require, up to 1.5A of active current. The DS1336 contains a series of five power FETS which can each handle 300 mA, for a combined capacity of 1.5A.

V_{CC} from the primary power supply is tied to the five IN pins of the DS1336, with the five OUT pins tied together and to the V_{BAT01} pin. The V_{BAT01} pin of the DS1336 has a series diode to prevent V_{CC} from charging the backup lithium supply, which is connected to V_{BATIN}. When PF becomes active from the DS1236, indicating a primary power failure on the memory card, the DS1336 switches power from the primary supply to the secondary lithium supply within 100 ns. In order to monitor power supplied by the backplane, the DS1336's other power fail input, $\overline{PF}$, can be connected to a $\overline{PF}$ signal on the backplane, so that a backplane failure can also cause the DS1336 to switch to the backup power source.
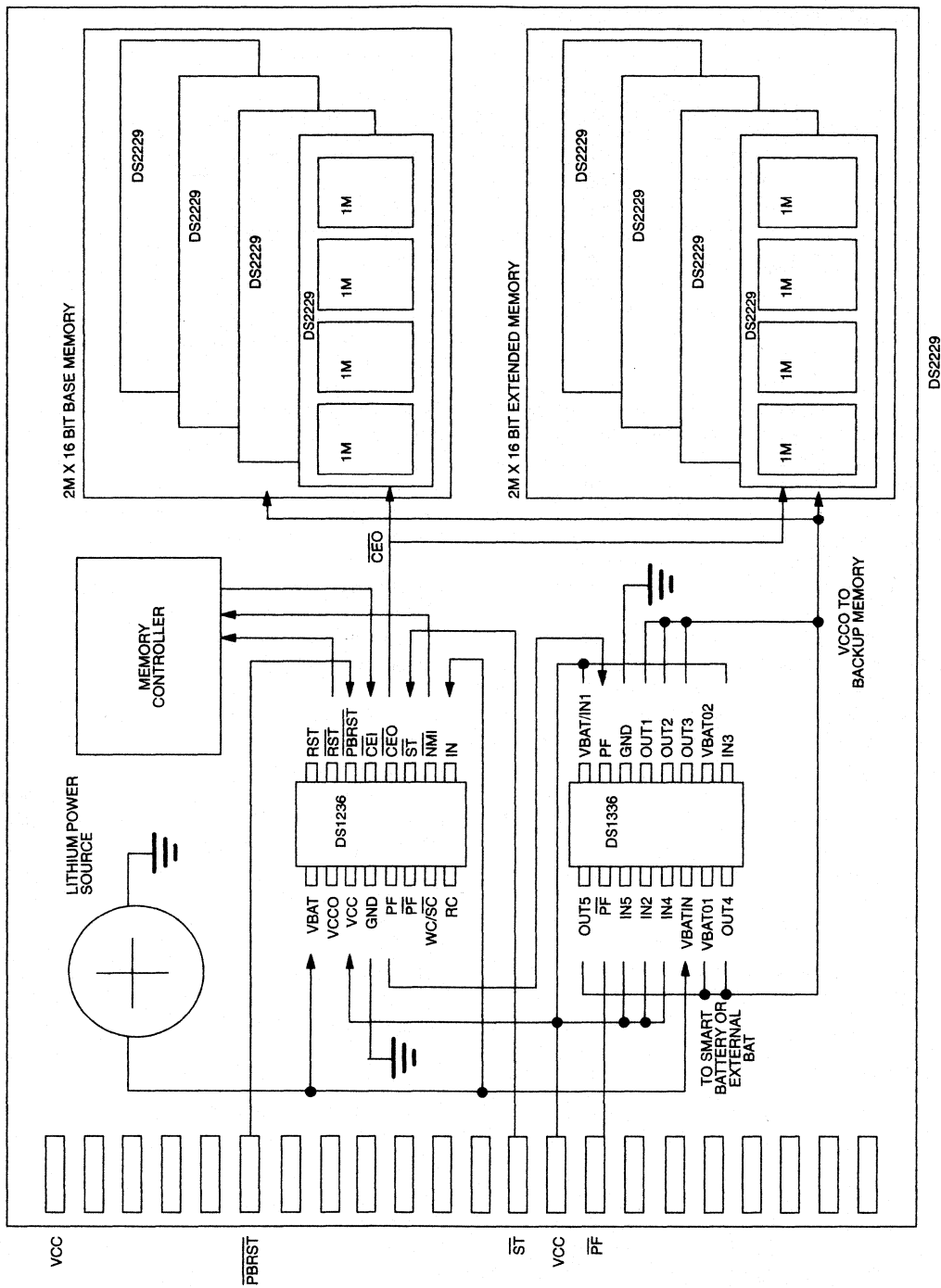
For more information on high density nonvolatile memory applications and related subsystems, please call the Dallas Semiconductor factory at (214) 450–0448.

DALLAS
SEMICONDUCTOR

Application Note 61
RAMport

072996 1/4
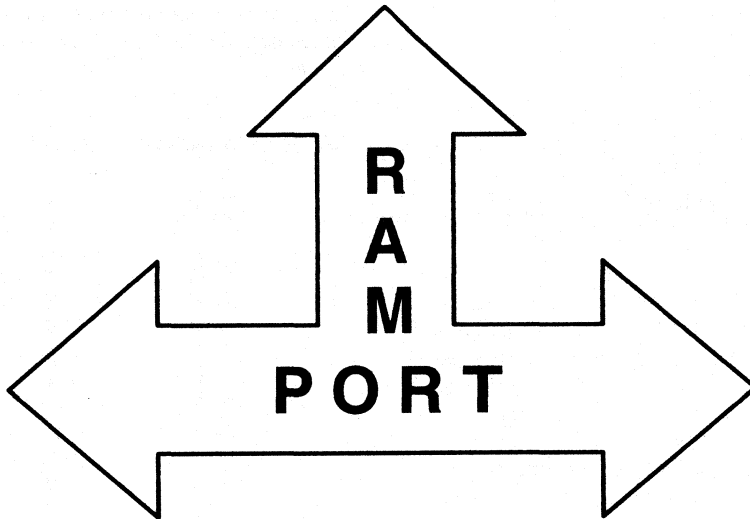
RAMport – 2K X 8 NV SRAM AND MORE

The RAMport memory developed by Dallas Semiconductor was designed to be connected to microcontrollers without robbing the device of valuable port pins. The name "RAMport" was chosen for the DS1381 to try to communicate this advantage to potential users (see Figure 1). Even if the name is effective in conveying this message, the overall capability of the device and the diversity of applications remains concealed.

DS1381 Figure 1
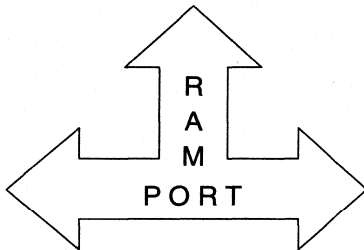
The DS1381 RAMport is a bytewide memory that uses a multiplexed address and data bus. The obvious disadvantage with this scheme is reduced performance because the address and data information is sent sequentially. The equally obvious advantage is reduced pin count (see Figure 2). The multiplexing scheme used with the DS1381 requires only 8 pins for address and data. The address is transferred to the device in two subsequent cycles. The first address transfer consists of A8 through A10 along with read or write command information. The second address transfer contains the low–order address bits and is followed by a third transfer which is data. Since the read or write command is transferred as part of the address, the need for read and write control signals is also eliminated, reducing control signal requirements to only two pins: memory and clock. The memory and clock signals direct data transfer to and from the memory and also command and control eight port input/outputs which are not found on conventional memories (see Figure 3). The total pin count for memory interface and port input/output plus V_{CC} and ground amounts to 20 pins. This leaves four pins for the special purpose of providing nonvolatility. Two pins provide for a direct connection to a data retention energy cell. These pins do not go outside the package, but are internally connected to a lithium coin cell. Since this connection is made with 2 of the 24 pins on a standard dual in line package, cost savings is achieved when compared to alternate interconnect systems or special lead frames. Finally, the DS1381 provides an output when power fail occurs which can be used to interrupt the processor, and a separate pin provides selection of the power fail detection point at 5% or 10% of the power supply voltage. The end result is a low–cost 24–pin 0.6" DIP which offers an I/O port and a power fail controller as a bonus.

DS1381 PIN INFORMATION Figure 2

TOL	1	24	V_{CC}
$\overline{PF}$	2	23	CLK
PI1	3	22	PO8
PO1	4	21	PI8
PI2	5	VBAT 20	N.C.
PO2	6	19	$\overline{MEM}$
PI3	7	18	PO7
PO3	8	17	PI7
PI4	9	GND 16	N.C
PO4	10	15	PO6
PI5	11	14	PI6
GND	12	13	PO5

PIN NAME	DESCRIPTION
$\overline{PF}$	POWER FAIL OUTPUT
PI1–PI8	PORT INPUTS (μP PORTS)
PO1–PO8	PORT OUTPUTS (EXTERNAL PORTS)
GND	GROUND
V_{CC}	+5 VOLTS
CLK	CLOCK
$\overline{MEM}$	MEMORY SELECT
N.C.	NO CONNECTION

DS1381 FUNCTIONAL BLOCK DIAGRAM Figure 3
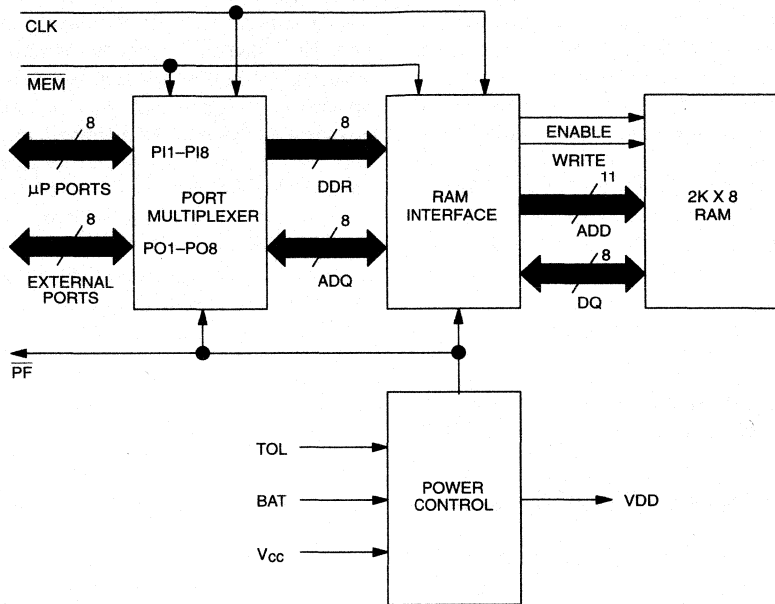
REDUCING TO PRACTICAL – MICROCONTROLLER INTERFACE

As mentioned, the DS1381 was designed to offer the microcontroller user some external nonvolatile memory that does not consume all of the valuable port pins. When using the DS1381 with a microcontroller, the interconnect system is simple and straightforward. Eight port pins of the microcontroller connect directly to the eight input port pins (PI1 – PI8) of the DS1381 (see Figure 3). These port pins are reproduced at the port output pins (PO1 – PO8). During operation, when no memory transfer is taking place, the DS1381 port output pins look exactly like the eight microprocessor port pins with the only addition being a small series impedance. Two other port pins of the microprocessor are required to control the DS1381, namely CLOCK and MEMORY. Since these port pins must be dedicated to control, they are not reproduced by the DS1381. The MEMORY and CLOCK signals must be generated using software within the microcontroller to establish the proper signal levels and timing relationships. The function of these two controls is to direct data to and from memory or to the data direction register within the DS1381. When data is being transferred to and from the DS1381, the port outputs are latched or become high impedance depending on their assigned function. More detailed information on the control signals is furnished in the DS1381 data sheet.

MICROPROCESSOR INTERFACE USING DS1381 MEMORY

While the hardware interface between the DS1381 and a microcontroller is straightforward, the interface to a microprocessor is also simple, although not as obvious. Only the 2K X 8 nonvolatile memory and power fail controller features are useful in this application (see Figure 5). The key to interfacing the DS1381 to a microprocessor is to use only the data bus for both address and data. Since only the memory is to be used, the MEMORY signal should be grounded. This connection fixes the PO1 – PO8 pins to remain in a status as dictated by the data bus and the DS1381 data direction register at the time that the MEMORY signal is grounded. Because it is unlikely that these pins contain any useful information under this condition (although some type of programming mode might be implemented to make these outputs useful) these pins should be left unconnected. With the MEMORY pin grounded, the clock input becomes analogous to the chip enable or chip select on a standard bytewide memory. The clock signal can be easily derived with a decoder from selected address lines which place the DS1381 properly in the memory map. To achieve proper timing (setup and hold times) the decode must be gated by a microprocessor control signal. The names and purpose of usable control signals vary with the type of microprocessor used. Howev-

072996 3/4

er, status signals, address latch enable, memory request, address strobe, and data strobe usually provide the proper timing relationship with respect to address and data for gating the decoder. The microprocessor and DS1381 data sheets should be consulted for exact timing details. Since only the data bus is used for both address and data, three separate memory cycles must be used to read or write each byte of data. However, this inconvenience is often a small price to pay for hiding a 2K X 8 of nonvolatile RAM in as small as one I/O or memory address space. An additional feature that standard memory does not provide is the power fail interrupt output.

IN CONCLUSION
The DS1381 is primarily tailored for use with microcontrollers. In microcontroller applications, the DS1381

provides an inexpensive nonvolatile RAM without using valuable port pins. In microprocessor applications, 2K X 8 of nonvolatile RAM is easily interfaced to a system with minimum impact on memory space by using the data bus for both address and data. Because of the multiplexed address/data bus, pin count and cost are kept to a minimum while additional and useful features are provided which are not found on standard memories. While the multiple uses of the RAMport are difficult to convey to potential users with only a name, closer inspection of the device reveals an economical, special-purpose nonvolatile memory with many microprocessor and microcontroller applications.

MICROCONTROLLER INTERFACE Figure 4
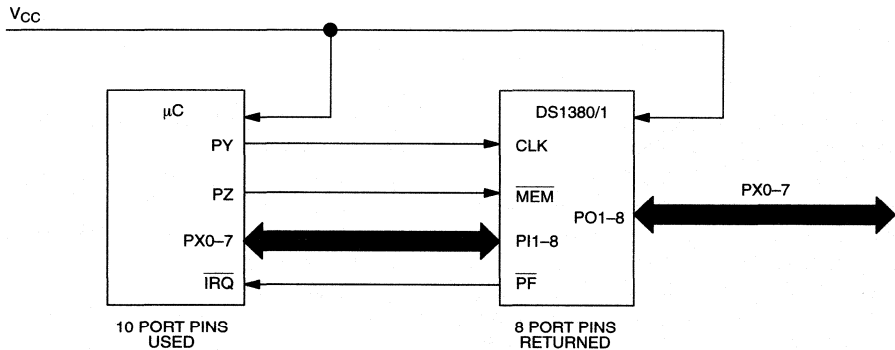
MICROPROCESSOR INTERFACE USING DS1380 2K X 8 NV RAM ONLY Figure 5

DALLAS
SEMICONDUCTOR

<div align="right">

Application Note 63
Using Nonvolatile Static RAMs

</div>
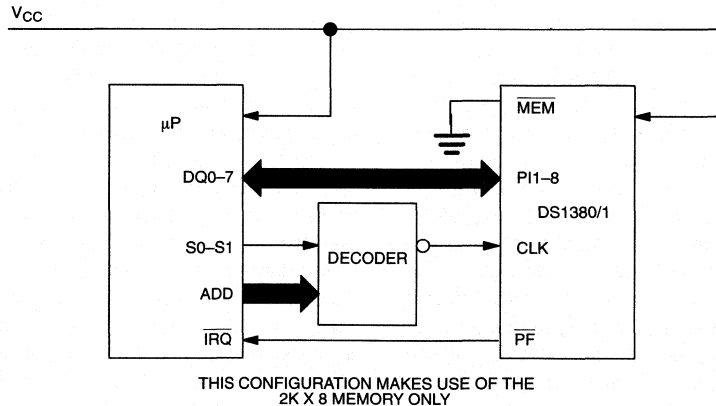
Vast resources have been expended by the semiconductor industry trying to build a nonvolatile random access read/write memory. The effort has been undertaken because nonvolatile RAM offers several advantages over other memory devices – DRAM, Static RAM, Shadow RAM, EEPROM, EPROM and ROM – which were developed to meet specific applications needs.

Characteristics of the ideal nonvolatile RAM are: low power consumption, high performance, high reliability,

high density, low cost, and the ability to be used in any semiconductor memory application.

While the various memory components designed to date do not meet the ideal memory scenario, each excels in meeting one or more of the sought after attributes (Figure 1).

MEMORY ATTRIBUTES Figure 1

	COST	EASE OF INTER- FACE	NONVOLA- TILE	DENSITY	PERFOR- MANCE	READ/ WRITE	DATA RETEN- TION
DRAM	+++			+++	++	+++	
STATIC RAM		+++		+	+++	+++	
NV SRAM		+++	++	+	+++	+++	++
PARTITION- ABLE NV SRAM		+++	++	+	+++	+++	++
PSEUDO STATIC	+	+		++	+	+++	
FLASH	++	++	++	++	++	+	++
EEPROM	+	++	+	+		+	+
EPROM	++	++	++	++	+		++
OTP EPROM	+++	+++	+++	+++	+		+++
ROM	+++	+++	+++	+++	+		+++

+ = Degree of excellence

TYPES OF MEMORY
Many types of memories have been devised to meet varying application needs. However, nonvolatile read/write random access memories can be substituted for all memory types independent of application, if cost is not a primary consideration.

DRAM: Dynamic Random Access Memory. A DRAM, similar to an SRAM, stores information as a 1 or a 0. In an SRAM, this information is stored in a four to six transistor flip–flop which is easy to address, but requires a relatively large memory cell. A DRAM, by comparison, stores its 1 or 0 as a charge on a small capacitor, requir-

ing much more current then an SRAM to maintain the stored data. The net memory cell size is smaller for the DRAM than for the SRAM, so the total cost per bit of memory is less. The DRAM's capacitors must be constantly refreshed so that they retain their charge. DRAMs require more sophisticated interface circuitry.

SRAM: Static Random Access Memory. An SRAM is essentially a stable DC flip–flop requiring no clock timing or refreshing. The contents of an SRAM memory are retained as long as power is supplied. SRAMs support extremely fast access times. SRAMs also have relatively few strict timing requirements and a parallel address structure, making them particularly suited for cache and other low–density, frequent–access applications.

NV SRAM: Nonvolatile Static Random Access Memory. An NV SRAM is a single package which contains a low–power SRAM, a nonvolatile memory controller, and a lithium type battery. When the power supply to this single modular package falls below the minimum requirement to maintain the contents of the SRAM, the memory controller in the module switches the power supply from the external source to the internal lithium battery and write protects the SRAM. These transitions to and from the external power source are transparent to the SRAM, making it a true nonvolatile memory. This unique construction combines the strategic advantages of SRAM–addressing structure, high–speed access, and timing requirements–with the non-volatility advantages of EEPROM technologies. Battery–backed SRAM modules from Dallas Semiconductor are pin–compatible with non–battery–backed SRAMs, making them ideal for any application where a traditional SRAM would be suitable.

PARTITIONABLE NV SRAM: A partitionable Dallas Semiconductor NV SRAM offers the same nonvolatility, addressing structure, and timing requirements of a regular Dallas Semiconductor NV SRAM product with the additional ability to write–protect selected blocks of memory, regardless of V_{CC}. This write protection feature requires no additional pins, and is instead controlled by a unique combination of read cycles (see DS1630, DS1645 and DS1650 data sheets). This feature allows a designer to configure a battery backed SRAM as both a RAM and a ROM–in one device. Because no additional pins are required for control, partitionable devices can be substituted for non–partitionables in existing designs, without making costly hardware changes.

PSEUDO STATIC RAM: Pseudo Static Random Access Memory. The advantages of using a Static RAM are the simplicity of the interface circuitry required, and the fact that the device is by nature "static," not requiring periodic refreshing to retain its data. A DRAM, however, provides lower cost–per–bit advantages and a higher memory density. A Pseudo–static RAM combines the advantages of the SRAM and DRAM by using dynamic storage cells to retain memory, and by placing all the required refresh logic on–chip so that the device functions similarly to an SRAM.

FLASH: A flash memory combines the electrical erase capability of an EEPROM with a cell that is similar to an EPROM. The result is that the modified cell may be block erased electrically instead of with UV light. This feature allows a Flash memory to accept new code updates or information while it is functioning in a system.

EEPROM: Electrically Erasable/Programmable Read Only Memory. A significant disadvantage of the EPROM memory is the fact that it cannot be reprogrammed while in a system. EPROM requires an external programming device to receive new code or data. An EEPROM eliminates this problem by providing a write function which can be used while the EEPROM is still in a circuit. A tradeoff for obtaining the write function while the EEPROM is still in a circuit is having to provide a high voltage (12.5V or above) source for the EEPROM when writing new data, or buying a more expensive EEPROM which has a charge pump in its package that allows it to be used with a standard 5 to 7V input. Although nonvolatile, EEPROM memory cells exhibit slow read/write access rates, making them most suitable for systems where performance is not an issue. The other read/write capable memories listed in Figure 1 provide the ability to frequently read and write data continuously over their entire lifetimes, in excess of 10 years, while EEPROM memory cells can rarely be rewritten more than 10,000 times. An EEPROM can be placed in a system and accessed as a standard RAM.

EPROM: Electrically Programmable Read Only Memory. An EPROM is a nonvolatile memory which offers the ability to both program and erase the contents of the memory multiple times. An EPROM must be programmed using a 12.5 volt (or higher) PROM programmer, and then transferred into the system in which it is intended to function. EPROMs can be erased by shining ultraviolet light into the window in the top of the IC package. The process of writing data into an EPROM and then erasing it may be repeated almost indefinitely.

EPROMs are usually used for product development, and later replaced with less expensive one–time programmable EPROMs.

OTP EPROM: One–Time Programmable EPROM. An EPROM which can only be written with code/data once instead of multiple times. Generally, OTP EPROMs are less expensive then erasable EPROMs.

ROM: Mask Programmable Read Only Memory. Mask programmable ROMs are the most durable form of memory storage. They are, however, "read only" and offer fairly slow performance. If a design has code/data that is very stable and will not need to be changed, a custom mask for the IC die can be made which will significantly reduce the cost of the ROM. A drawback to using a mask ROM is the significant cost penalty that must be incurred if an error in the code/data being stored forces a mask set change. The OTP EPROM fills the gap between ROM applications (no changes) and EPROMs (frequent changes).

MEETING APPLICATIONS NEEDS

NMOS DRAM memory provides performance and density, but, on the down side, must be constantly refreshed to retain data. At the opposite extreme are ROMs, offering nonvolatility and density, but lacking the ability to be updated with new data because information is programmed in only once. Between these two are a wide range of devices that fulfill some characteristics of the ideal memory.

Two popular devices, EEPROMs and Shadow RAMs, are designed to emulate a static RAM but also have the ability to retain data after a power loss. But despite their capability to retain data, both EEPROMs and Shadow RAMs fall short of meeting the industry's needs for several reasons.

Most notably, the EEPROM requires a special slow write cycle. The EEPROM's inability to support standard write cycle rates hinders performance in applications where memory is updated immediately as new data is available.

Another problem with EEPROMs is their wear–out mechanisms. These raise longevity concerns due to the limited number of write cycles allowed – sometimes as few as 10,000. If a static RAM with a 200 ns cycle time had this limitation, it would wear out in a mere 20 ms. In an application that requires constant updating, such as the buffer memory of a cashier's checkout terminal or a printer, the EEPROM's wear out mechanism is not acceptable.

Finally, because of the complexity of programming circuits, the cell structure and the special process technology required, the density of EEPROMs has not kept up with industry demands.

In systems requiring store–and–forward data, the memory must provide the desired fast write cycle as well as protection of data in the event of a power loss. Despite the promise of such a memory device and the effort invested by the industry, the ideal memory remains elusive.

To more nearly emulate the ideal memory, Dallas Semiconductor combines its intelligent CMOS control circuitry (DS1210), a lithium energy source, and a very low power SRAM to offer a high density, nonvolatile memory.

Many Dallas devices, including the DS1220 (2K x 8 bits), DS1225 (8K x 8 bits), DS1230 (32K x 8 bits), DS1245 (128K x 8 bits), and DS1250 (512K x 8), use this fusion of technologies to provide a nonvolatile random access memory solution in densities up to 4096K bits.

CMOS NV SRAMs currently available have read and write cycle times of 70 ns, which exceeds most system requirements. They are much more robust than EEPROM, because there is no wear–out mechanism or write cycle limitation.

NV SRAMs are also the easiest to use and interface because the pinout configurations are standard throughout the industry. In fact, X8 or bytewide NV SRAMs can be interfaced directly to microprocessors (Figure 2). In addition, CMOS NV SRAMs offer low power in both active and standby modes, a characteristic sought by many designers. In most designs, memories remain in standby much of the time, keeping power consumption negligible. In the standby mode, current drain consists only of leakage currents in the tens of nanoamperes.

BYTEWIDE RAM TO MICROPROCESSOR INTERFACE Figure 2
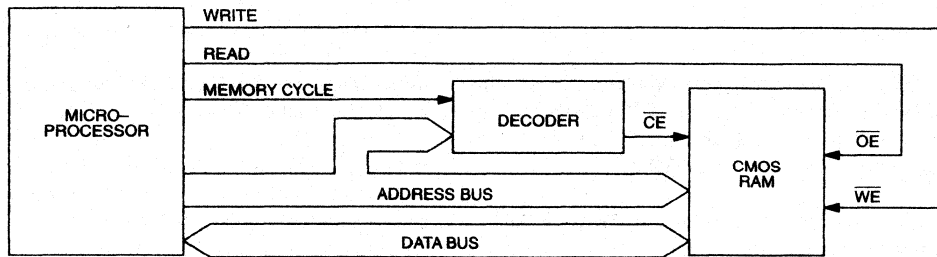
BYTEWIDE MEMORIES PROVIDE EASY INTERFACE TO MICROPROCESSORS BECAUSE OF THE X8 ORGANIZATION AND CONTROL SIGNAL DEFINITION.

PUTTING LITHIUM AND RAM TOGETHER

The minute leakage currents of modem CMOS SRAMs can be sustained with a backup energy source to yield a most attractive nonvolatile memory. However, the actual solution involves more than just a CMOS memory and back up energy source (see Figure 3).

Battery backup design schemes are many and varied. The increase in density and availability of low–power CMOS memories in recent years has made this approach even more attractive. Yet problems still exist with battery backup design due to battery packaging and a lack of appropriate standard components to implement the support circuitry. One problem is providing isolation between the battery and power supply (see Figure 4). Diodes can provide isolation but produce a voltage drop which requires nonstandard power supplies and also subtracts from the battery voltage. A second problem is that the circuitry must be powered from the battery. Unless these devices draw an extremely modest amount of current, battery selection changes drastically. In fact, a current drain of even a couple of

microamperes dictates the use of either rechargeable batteries or a replaceable battery scheme. If rechargeable batteries are selected, the recharging circuit can be costly and complex, and the best rechargeable battery cannot compare with the electrochemical stability of the lithium primary cell. Even worse, replaceable batteries add maintenance and cost to an in–service system. Battery packaging has also been a serious limitation, taking up valuable space and requiring special handling consideration to prevent discharge.

Dallas Semiconductor overcomes these obstacles by using high–capacity, non–rechargeable lithium batteries in its battery backed SRAMs.

ENERGY SOURCE

The energy source used to retain data in memory must be capable of outlasting the usefulness of the end product. Dallas Semiconductor NV SRAM products use an extremely stable electrochemical system with enough energy to guarantee a shelf life greater than 10 years.

BATTERY BACKUP CIRCUIT Figure 3
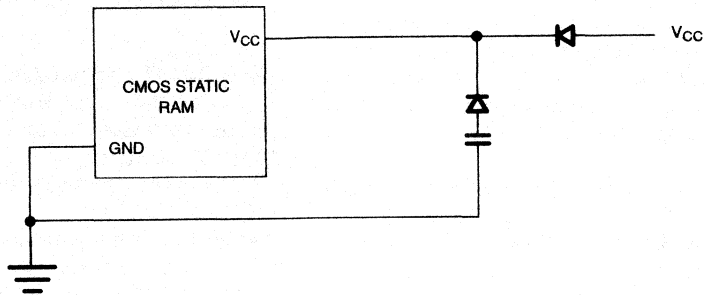
CMOS STATIC RAM REQUIRES MORE THAN JUST A BACKUP POWER SUPPLY. DATA MUST ALSO BE PROTECTED DURING POWER TRANSIENTS TO AVOID GARBLED DATA.

POWER SUPPLY AND BATTERY ISOLATION CIRCUITRY Figure 4
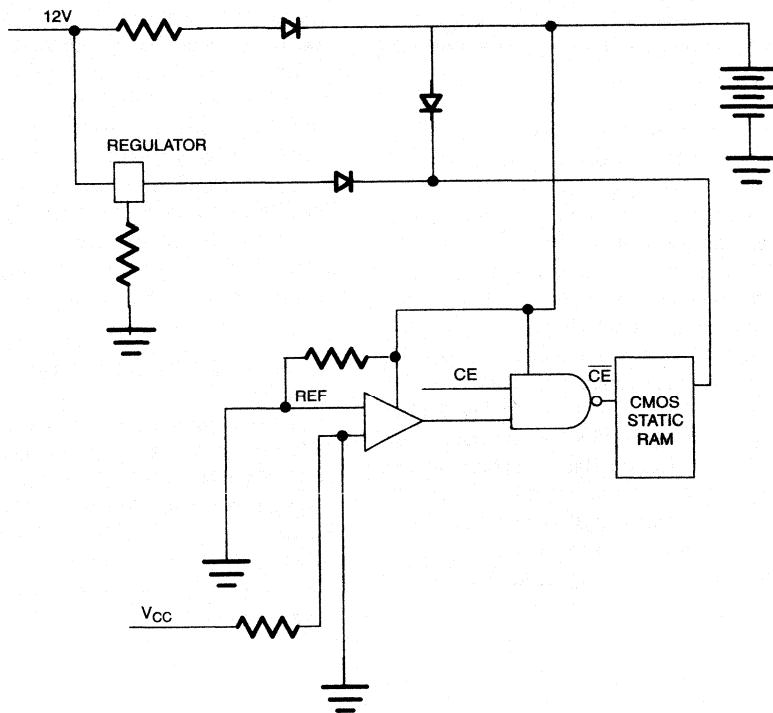
SUPPORT CIRCUITRY REQUIRED TO PRODUCE POWER FAIL DETECTION AND WRITE PROTECTION FORCES THE NEED FOR MULTICELL RECHARGEABLE BATTERY OR A LITHIUM BATTERY.

LITHIUM BATTERY BACKUP IS MORE RELIABLE

The lithium energy cell has raised concern about reliability and has been the object of much study. Data taken on the energy cell used in Dallas Semiconductor NV SRAMs indicates a cell failure rate less than 0.5% at 55°C over a 10 year period.

Additional life studies taken on the same lithium energy source encapsulated in Dallas Semiconductor's NV SRAMs have produced no failures in over 12 million device hours at 85°C. The lithium energy cell, then, is ideal for commercial and industrial semiconductor applications.

RETROFITTING EXISTING DESIGNS

The pinout of Dallas Semiconductor NV SRAMs is an established industry standard (Figure 9). The Joint Electronic Devices Engineering Council's Bytewide Version B Standard defines and upgrades from 2K x 8 in density to 128K x 8.

This standard accommodates RAM, ROM, UV EPROMs, and EEPROMs. Because of the flexibility and upgradeability of bytewide memories, the number of existing sockets is in the hundreds of millions. Therefore, many system designs can accommodate direct replacement of RAMs, EPROMs, ROMs, and EEPROMs with Dallas Semiconductor NV SRAMs. These solutions add real–time programmability and/or density upgrades to existing systems without redesign. Real–time programmability gives the system the ability to be personalized by the end user. In other words, NV SRAMs can be retrofitted into existing designs without making changes to existing hardware. This retrofitting offers a cost–effective, practical solution for companies who have invested in other memory devices that are less than ideal for their needs. For example, a design using conventional static RAM can be upgraded to nonvolatile memory by substituting a Dallas Semiconductor NV SRAM for the Static RAM memory.

IN–CIRCUIT PROGRAMMABILITY

The advantages of NV SRAM can be related to the capability of software. Modern systems seek customization for the cost of standard product. In this aspect, software can be adapted in a system to perform specialized functions. It is even possible to totally modify a system personality over the telephone. In–circuit programming also reduces maintenance cost by eliminating service calls to update software. Software stored in NV SRAM can be updated as often as necessary, depending on the configuration or application of the system.

PORTABLE APPLICATIONS

The advancement of high density, low–power portable computers is continuing to drive development requirements. Difficult interface circuitry and refresh requirements of DRAM memories make them unsuitable for such applications. SRAMs are not only easier to address and consume less power when operating, but also require very little power to maintain the contents of their memory. Even better, an NV SRAM can provide the high performance of a DRAM or SRAM and also guarantee that the memory is truly nonvolatile. When a portable PC needs to be in standby mode, the memory can be powered down altogether.

1 MBYTE MEMORY SUBSYSTEM USING NV SRAMs

Figure 6 shows a system block diagram with an Intel 386SL microprocessor with a 1 megabyte main memory of 128K x 8 NV SRAMs (DS1245). Figure 5, Portable Applications: Intel 386SL CPU/NV SRAM Timing, shows the requisite timing for the memory subsystem. The Intel 386SL is one of many microprocessors specially designed for low power, portable applications, and for addressing SRAM memory.

PORTABLE APPLICATIONS: INTEL 386SL CPU/NV SRAM TIMING Figure 5
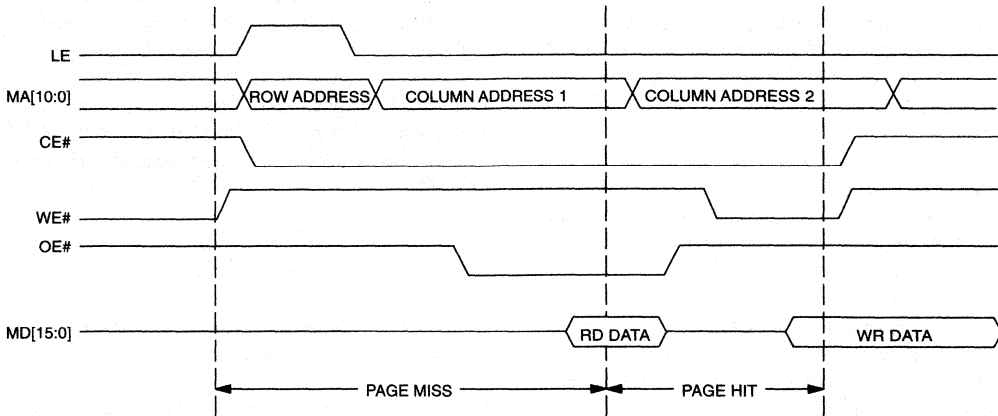
1 MBYTE MEMORY SUBSYSTEM USING NV SRAMS Figure 6
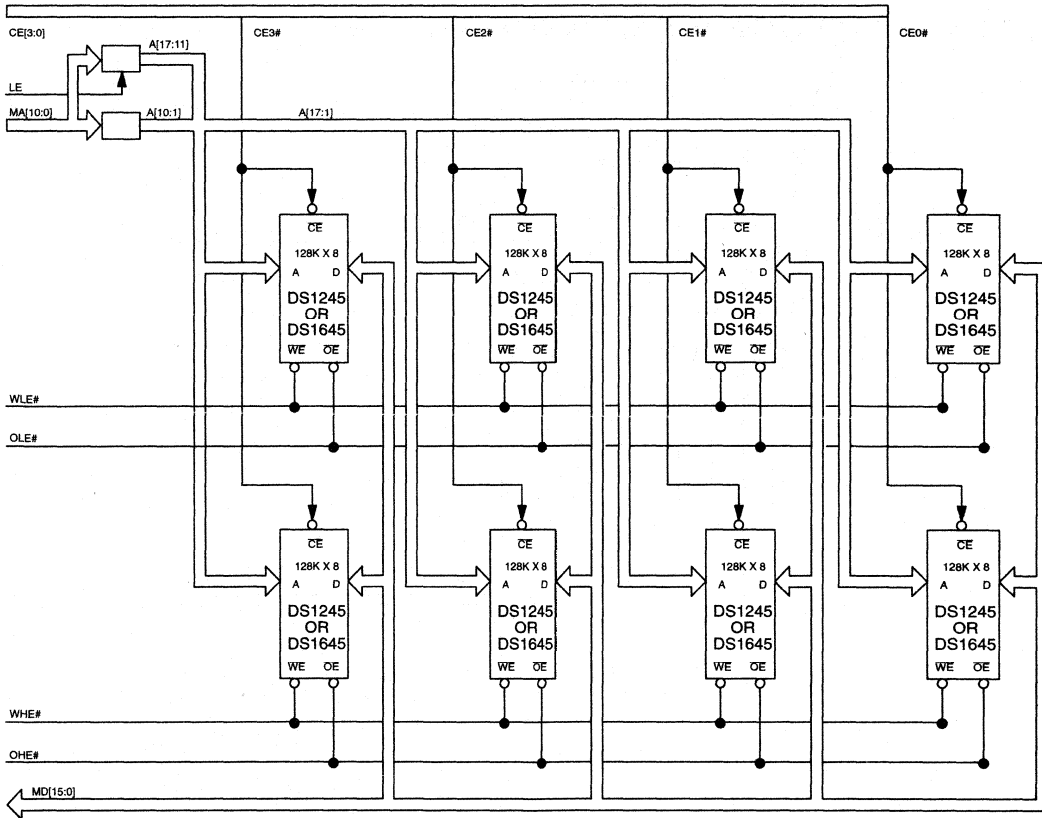
In Figure 6, eight DS1245 SRAMs are used to create a four–bank 1Mbyte SRAM memory subsystem. The following signals from the Intel 386SL CPU are required to address the SRAM module based system. (The 386SL memory controller must be configured in its SRAM addressing mode for this application.)

LE: Latch Enable. This signal is active high and serves to indicate that a row address is to be put on the address bus. A row address must be latched at this signal's falling edge. LE is connected to the latch enable input of the address latch.

MA[10:0]: Multiplexed Memory Address Bus. This bus provides address information for the Memory Controller Unit. The bus provides a 22 bit address in a multiplexed row/column sequence.

$\overline{CE}$[3:0]: Chip Enable outputs. These signals provide chip enable control for each SRAM bank.

$\overline{WLE}$: Write Low Enable. Indicates a write access to the lower byte of the 386SL CPU memory bus. The lower byte of data is put on the memory bus at the falling edge of $\overline{WLE}$.

$\overline{WHE}$: Write High Enable. Indicates a write access to the high byte of the 386SL CPU memory bus. The high byte of data is put on the memory bus at the falling edge of $\overline{WHE}$.

$\overline{OLE}$: Output Low Enable. Enables the lower byte output from the NV SRAM modules.

$\overline{OHE}$: Output High Enable. Enables the high byte output from the NV SRAM modules.

MD[15:0]: Memory Data Bus. This bus provides data information for the Memory Controller Unit. Accesses from the Memory Controller Unit to the NV SRAM memory modules take place through this bus.

16–BIT SINGLE–BANK NV SRAM BIOS CIRCUIT Figure 7
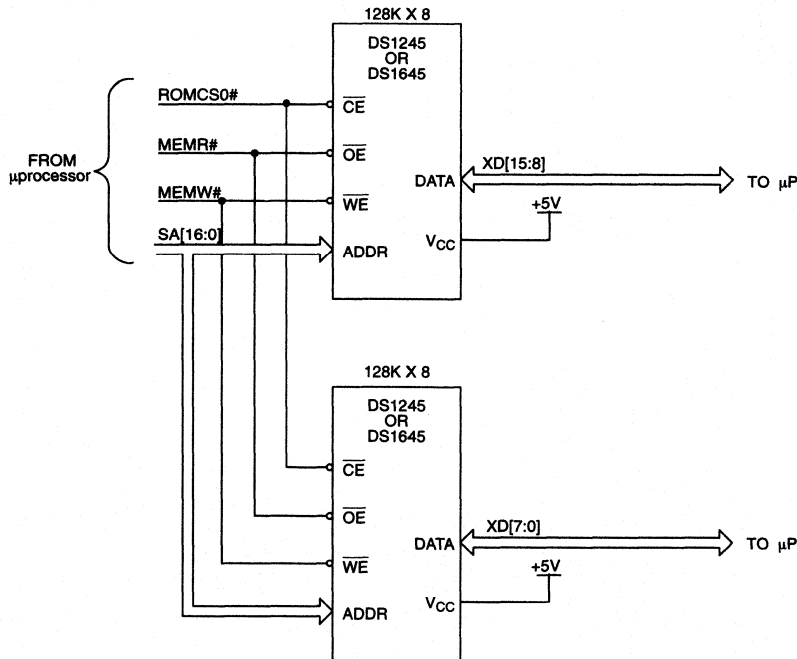
16–BIT SINGLE–BANK NV SRAM BIOS CIRCUIT

Figure 7 shows Dallas NV SRAMs providing BIOS memory storage for an Intel 386SL CPU. Using the DS1645 NV SRAMs provides several advantages over using either OTP EPROM or FLASH memories.

Flash memories require more operating current than NV SRAMs. Flash memories also require a high voltage source, 12V+, for any writes or updates that must be made to BIOS. NV SRAMs, on the other hand, require only their standard V_{CC} 5V input for both read and write access. Like Flash memories, a DS1645 NV SRAM maintains the contents of its memory in the absence of V_{CC}. A DS1645 has the additional feature that it can be easily programmed to write protect user–selected blocks of memory. In effect, individual memory blocks in the NV SRAM module can be configured to appear as ROM memory, without detracting from the DS1645's ability to receive BIOS updates in its non–write–protected blocks of memory.

Traditional OTP EPROMs, while nonvolatile and very low–power like the DS1245 and DS1645 NV SRAMs, are lacking in that they can only be programmed once, and usually require a special fixture to be programmed. DS1245 and DS1645 NV SRAMs provide the capability to update BIOS repeatedly without removing them from the system. DS1245 and DS1645 NV SRAMs also provide fast 70 ns access times, negating the need to insert additional wait states into BIOS access timing requirements.

The signals shown in Figure 7 are taken directly from the Intel 386SL CPU:

ROMCSO#: This signal is a dedicated ROM control signal provided by the 386SL CPU. It is active low and is used to enable the system BIOS.

MEMR#: Memory Read. This signal indicates when a memory read access is occurring on the ISA bus and is active low.

MEMW#: Memory Write. This signal indicates when a memory write access is occurring on the ISA bus and is active low.

XD[15:0]: X–bus Data. Buffered data lines from the system data bus. These signals are produced using an external transceiver (see Intel 386SL Superset System Design Guide).

SA[16:0]: System Address Bus. This bus is driven by the 386SL CPU for system I/O accesses.

DATA LOGGING Figure 8
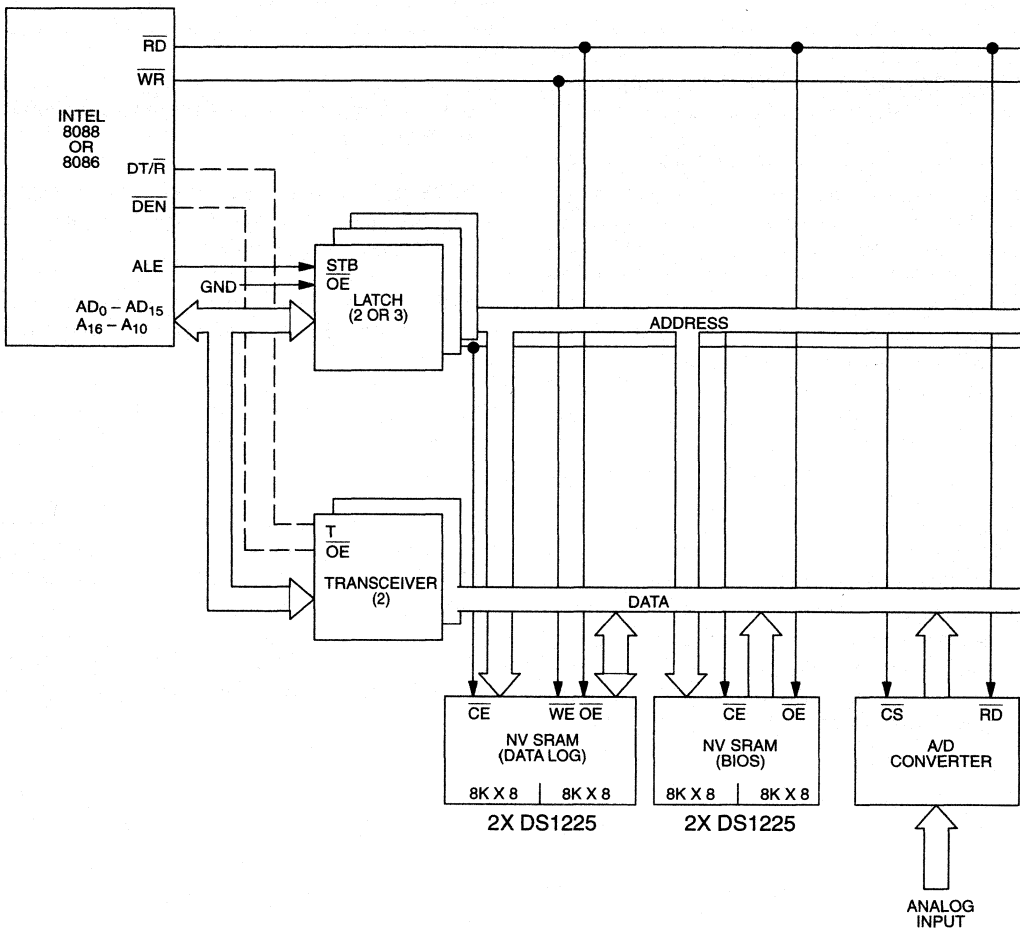
DATA LOGGING

Figure 8 shows how Dallas Semiconductor's NV SRAMs can provide a special advantage in environments where the power supply is not entirely reliable, or when power must periodically be shut down. Dallas Semiconductor NV SRAMs contain memory control circuitry which not only maintains the data in the SRAM in the absence of power, but also write protects the device if V_{CC} is out of tolerance. This feature ensures that an unstable power supply does not corrupt data which has been collected.

In this application, an Intel 8086 is shown in its minimal mode, connected to an address latch and bus transceiver to demultiplex the 8086's bus (see Figure 8). The resulting address and data busses may then be connected directly to two memory banks, one 8K x 16 BIOS memory consisting of two DS1225 NV SRAMs, the other an 8K x 16 memory bank consisting of two DS1225's acting as a data log. A data collecting device, such as an A/D converter, can be addressed as a read-only peripheral device to sample a value and write it to the DS1225 acting as a data log. The DS1225s acting as the data log can transmit their data on the data bus to another peripheral, or may be removed from the system and taken to another location to have the log extracted.

DALLAS SEMICONDUCTOR BATTERY BACKUP SRAM MODULES Figure 9

A7	1			24	V$_{CC}$
A6	2			23	A8
A5	3			22	A9
A4	4			21	$\overline{WE}$
A3	5		DS1220	20	$\overline{OE}$
A2	6		2K X 8	19	A10
A1	7			18	$\overline{CE}$
A0	8			17	DQ7
DQ0	9			16	DQ6
DQ1	10			15	DQ5
DQ2	11			14	DQ4
GND	12			13	DQ3

NC	1			28	V$_{CC}$
A12	2			27	$\overline{WE}$
A7	3			26	NC
A6	4			25	A8
A5	5			24	A9
A4	6		DS1225	23	A11
A3	7		8K X 8	22	$\overline{OE}$
A2	8			21	A10
A1	9			20	$\overline{CE}$
A0	10			19	DQ7
DQ0	11			18	DQ6
DQ1	12			17	DQ5
DQ2	13			16	DQ4
GND	14			15	DQ3

A14	1			28	V$_{CC}$
A12	2			27	$\overline{WE}$
A7	3			26	A13
A6	4			25	A8
A5	5			24	A9
A4	6		DS1230	23	A11
A3	7		DS1630	22	$\overline{OE}$
A2	8		32K X 8	21	A10
A1	9			20	$\overline{CE}$
A0	10			19	DQ7
DQ0	11			18	DQ6
DQ1	12			17	DQ5
DQ2	13			16	DQ4
GND	14			15	DQ3

NC	1			32	V$_{CC}$
A16	2			31	A15
A14	3			30	NC
A12	4			29	$\overline{WE}$
A7	5			28	A13
A6	6			27	A8
A5	7			26	A9
A4	8		DS1245	25	A11
A3	9		DS1645	24	$\overline{OE}$
A2	10		128K X 8	23	A10
A1	11			22	$\overline{CE}$
A0	12			21	DQ7
DQ0	13			20	DQ6
DQ1	14			19	DQ5
DQ2	15			18	DQ4
GND	16			17	DQ3

A18	1			32	V$_{CC}$
A16	2			31	A15
A14	3			30	A17
A12	4			29	$\overline{WE}$
A7	5			28	A13
A6	6			27	A8
A5	7		DS1250	26	A9
A4	8		DS1650	25	A11
A3	9		512K X 8	24	$\overline{OE}$
A2	10			23	A10
A1	11			22	$\overline{CE}$
A0	12			21	DQ7
DQ0	13			20	DQ6
DQ1	14			19	DQ5
DQ2	15			18	DQ4
GND	16			17	DQ3

SILICON TIMED CIRCUITS

DALLAS
SEMICONDUCTOR

<div align="right">

Application Note 107
DS1020/DS1021 8–Bit
Programmable Delay Lines

</div>

INTRODUCTION

This application note is designed to assist in the use of the DS1020/DS1021 programmable delay lines. The basic principles of device operation are covered in simplified form, but with sufficient detail to enable the user to understand what is happening within the device and how this affects its use in practical applications.

These flexible devices can be configured as traditional delay lines, as pulse width modulators or even as programmable oscillators. A variety of configurations are illustrated, the various features of which cover most applications.

Some of the key considerations which must be taken into account when designing in these products, based on the experience of previous users of the devices, are also covered.

The DS1020/DS1021 are similar devices, differing only in package and step size availability and response to power up conditions (see Page 107, Power Up).

KEY PRODUCT FEATURES

- Programmable over 256 steps in increments of 0.15 to 2 ns (DS1020), 0.25 or 0.5 ns (DS1021)

- Guaranteed monotonicity

- Serial (3–wire) or parallel (8–bit) programmability

- Cascadable

- DIP (DS1020 only) or SOIC packaging

PRODUCT SELECTION (all times in ns)			
PART NUMBER	STEP ZERO DELAY	DELAY PER STEP	MAXIMUM DELAY
DS1020–015	10	0.15	48.25
DS1020–025 DS1021–025	10	0.25	73.75
DS1020–050 DS1021–050	10	0.5	137.50
DS1020–100	10	1	265.00
DS1020–200	10	2	520.00

CIRCUIT CONFIGURATIONS

Programmable Delay Line

Programmable Pulse Width

TIMING WAVEFORMS Figure 1
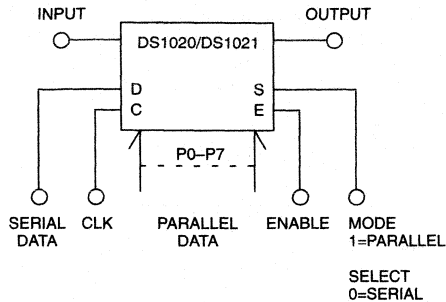
OUTPUT WAVEFORMS Figure 2

This is the "normal" mode of operation for the DS1020/DS1021. Input pulses applied to the device reappear at the output after a delay time set by the device programming. Both leading and trailing edges of the input waveform are delayed by the same amount.

The delay time can be programmed either by means of a serial data input, or can be loaded into an 8–bit parallel port. A Mode Select pin (S) determines which mode of operation is to be used. An Enable pin is available to latch in the serial data once it has been loaded, or to load parallel data and isolate the device from further changes to a shared parallel bus.

NOTE: In some of the following applications control and/or data input pins have been omitted for clarity. Unless reference is made to specific inputs, the same configuration can be used in either the serial or parallel mode.

The DS1020/DS1021 can be combined with some simple external logic to produce a programmable pulse width. In the example shown above the output pulse is triggered by the rising edge of the input waveform and can be adjusted in duration from 10 ns (the latent delay of the DS1020/DS1021) up to the maximum programmed delay value.

For correct operation over the full range of desired output pulse widths, the duration of both the high and low states of the input must be greater than the delay time of the DS1020/DS1021 which corresponds to the maximum output pulse width (see Page 108).

The rising edge of the output will be delayed with respect to the input by the propagation delay through the two gates. The falling edge will be dependent on the programmed delay of the DS1020/DS1021 and the propagation delay of the output gate (see diagram next page).

072996 2/21
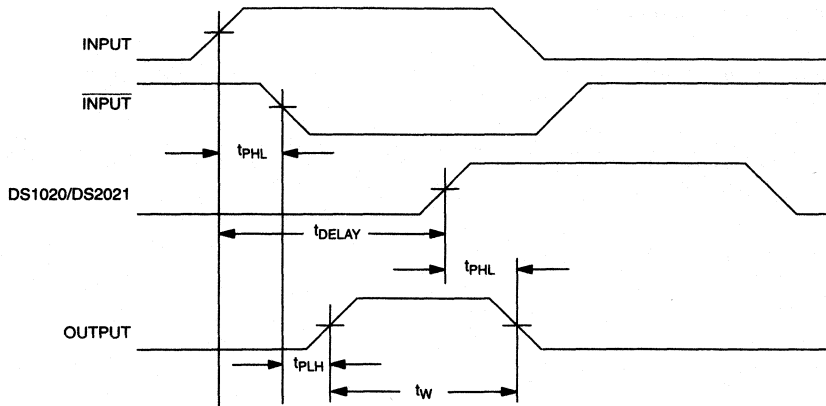
Therefore the output pulse width is given by:

t_w = (input to falling edge of output) − (input to rising edge of output)

= $(t_{DELAY} + t_{PHL}) - (t_{PHL} + t_{PLH})$

= $t_{DELAY} - t_{PLH}$

PULSE WIDTH MODULATOR Figure 3
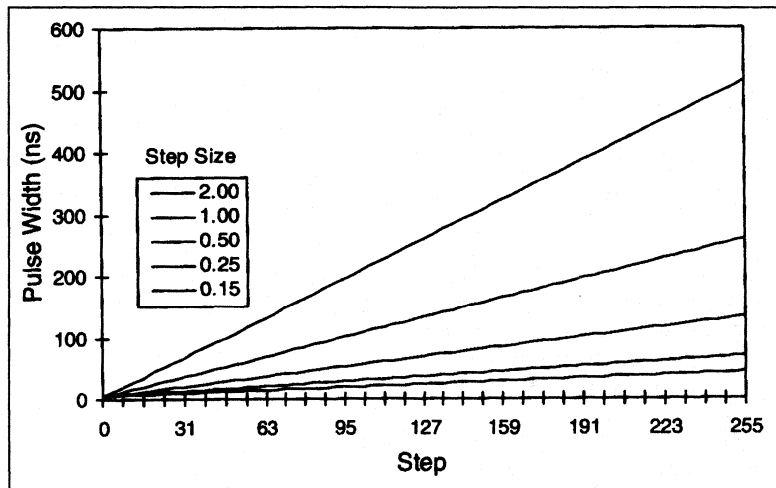
Figure 3 shows the range of pulse widths available for the various members of the DS1020/DS1021 family.

NOTE: Using HCMOS gates the minimum pulse width will be approximately 5 ns.

Programmable Oscillator

If the output of the DS1020/DS1021 is inverted and fed back to the input a free–running oscillator is produced. The oscillator can be gated if desired by replacing the inverter with a NAND or NOR function and using the additional input as an enable.

The period of the output signal will equal approximately twice the sum of the programmed delay and the propagation delay through the inverter, or more accurately:

$$f_O = 1/\{2(t_{DELAY})_{DS1020} + (t_{PLH} + t_{PHL})INV\}$$
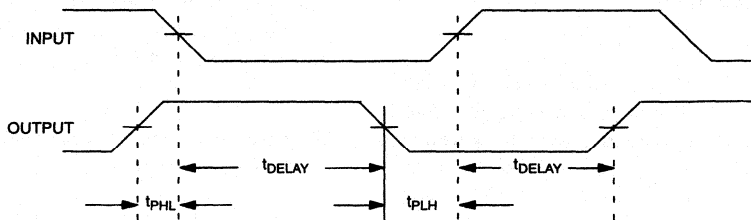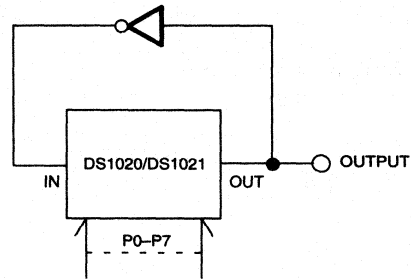
The minimum frequency is determined by the maximum achievable delay from the DS1020/DS1021, the maximum is determined by the propagation delay of the inverter and the step zero delay time of the DS1020/DS1021.

The following table summarizes some bench measurements on three of the available speed options:

DEVICE	STEP SIZE	STEP NUMBER	FREQUENCY	JITTER (ns)
		255	5.8	5.0
		128	9.7	2.0
DS1020–025	0.25 ns	0	33.0	1.0
		255	1.8	10.0
		128	3.2	5.0
DS1020–100	1 ns	0	33.0	1.5
		255	0.9	22.0
		128	1.7	14.0
DS1020–200	2 ns	0	30.0	1.5

In practice the speeds tend to be higher than suggested by the data sheet values for the inverter propagation delays because the devices are more lightly loaded.

The maximum frequency can be increased by using a faster inverter:

DEVICE: DS1020–025			STEP SIZE: 0.25 ns	
	'HC04		'F04	
STEP NUMBER	FREQUENCY (MHz)	JITTER (ns)	FREQUENCY (MHz)	JITTER (ns)
255	5.8	5.0	6.5	2.5
128	9.7	2.0	10.8	1.2
0	33.0	1.0	47.0	0.3

The jitter values shown in these charts are approximate values for the peak to peak jitter on the output signal. The effect of jitter increases as the operating frequency is increased, but can be minimized by device decoupling (see also Page 101).

The increased jitter of the oscillator using an HCMOS inverter versus an F–TTL inverter can be attributed to the difference in noise coupled to the supply when the inverter output changes state. The HCMOS "through current" results in a larger glitch on the supply than the bipolar totem–pole output stage.

Using this bench data we can project the performance for each member of the family across its entire programming range. Figures 4 and 5 show the theoretical frequencies obtainable for given programmed delay values. The first chart assumes an HCMOS type inverter or gate, the second achieves a greater maximum frequency by using an F–TTL device.

To derive these charts the following values have been used for the propagation delays of the inverters:

'HC04: $t_{PLH} = t_{PHL} = 6$ ns   'F04: $t_{PLH} = t_{PHL} = 3$ ns

FREQUENCY VS. PROGRAMMED STEP (HCMOS INVERTER) Figure 4
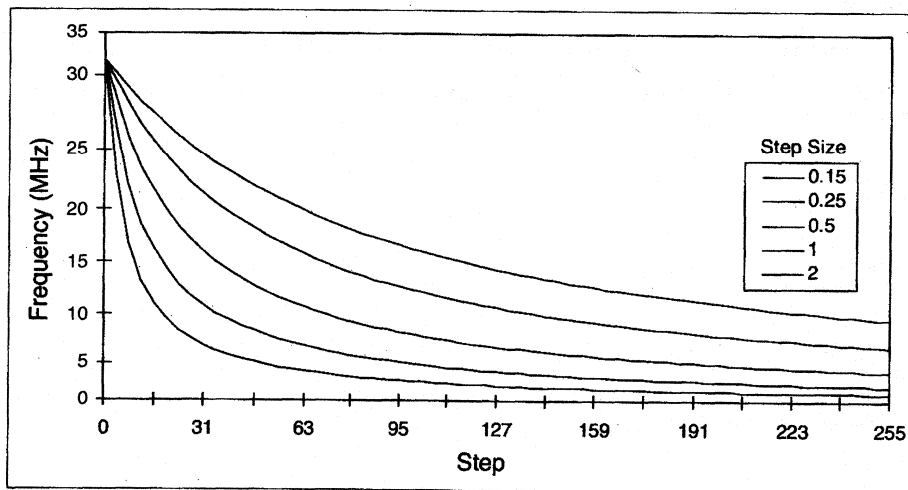
FREQUENCY VS. PROGRAMMED STEP (F–TTL INVERTER) Figure 5
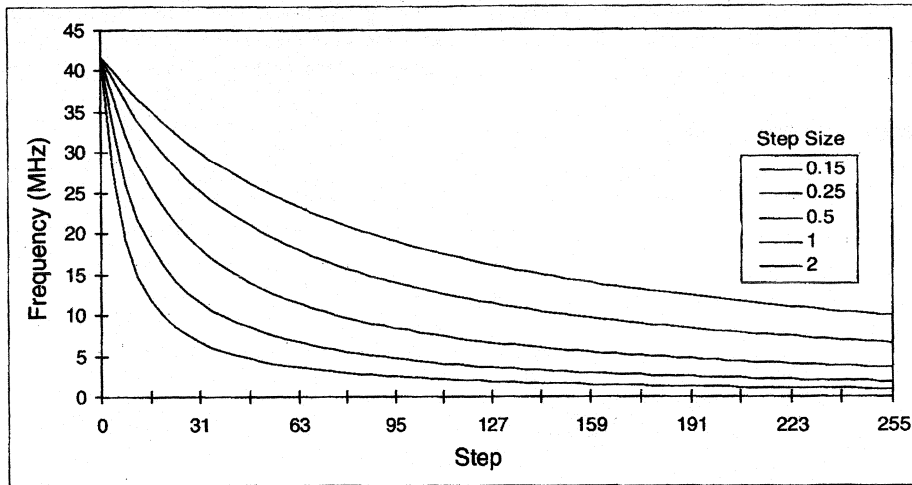
Closed Loop Operation

Any of the preceding configurations can also be set up for closed loop operation. In this mode some sort of feedback loop is set up such that the programming pins are modified until the output delay, or some other parameter dependent on the output delay, reaches a desired level or state. The advantage of this arrangement is that the negative feedback eliminates any effects caused by device error (deviation between actual delay and programmed delay).

ULTRASONIC RANGEFINDER APPLICATION Figure 6
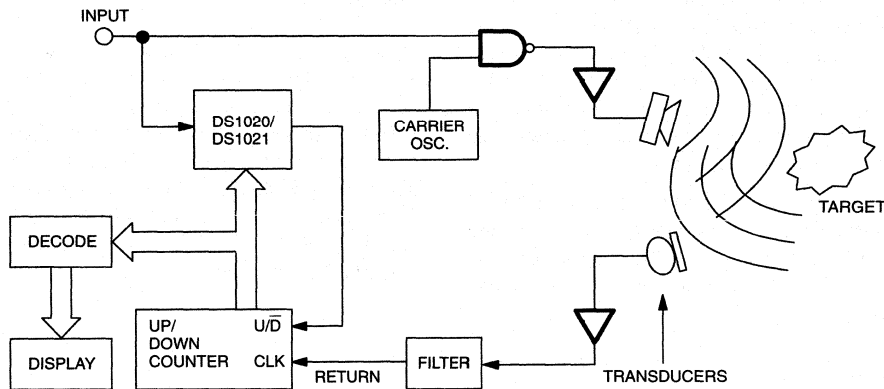
A digital ultrasonic rangefinder is shown to illustrate the use of the DS1020/DS1021 in a closed loop mode. These devices can be used in a variety of ultrasound, laser and video applications and with a variety of feedback mechanisms. For example in laser applications an analog feedback loop with an AID could be used to control pulse energy or power, for mouse pointer application the feedback is provided by the operator moving a mouse until the pointer is located at the desired location.

In the example shown an input waveform is used to generate a series of ultrasonic bursts which are emitted by a transducer.

This signal gates an oscillator (at the appropriate carrier frequency) which results in a burst duration equal to the high level pulse width of the input and a burst frequency dependent on the period of the carrier oscillator.

If a target is present the signal will be reflected to the receiver, which can be amplified and have the carrier removed to leave a return pulse which will be delayed from the input signal by an amount dependent on the distance to the target.

TIMING WAVEFORMS Figure 7
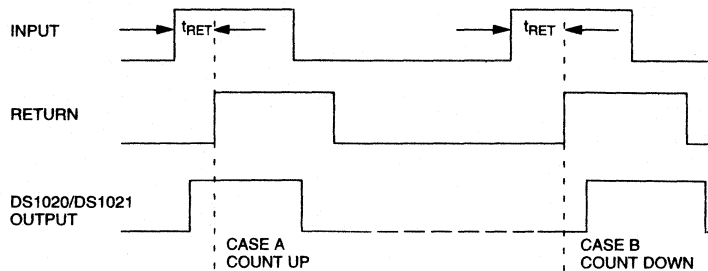
To measure this distance the programmed delay of the DS1020/DS1021 is adjusted until it matches the delay of the RETURN pulse. This is done by using the rising edge of the return pulse to increment or decrement an up/down counter, while the direction of count is governed by the state of the DS1020/DS1021 output. In the diagram above Case A shows too small a delay from the DS1020/DS1021, the output is high so the clock pulse from the return signal will cause the counter to increment. In Case B the delay is too large, so in this case the output is low and the clock pulse will cause the counter to decrement. Eventually, when the DS1020/DS1021 delay time approximately equals the return delay, a state will be reached where the counter alternately decrements and increments on each cycle.

The resulting programmed value can then be decoded and displayed (including conversion from programmed value to distance dimensions and, if greater accuracy is wanted, corrections for device error – see pages 98 and 102).

Performance Limitations

While this example serves to illustrate the principle it does show some performance limitations (which render it unsuitable for a practical application).

When the programmed delay is approximately equal to the return delay the LSB of the counter will toggle, therefore it is probably advisable to discard the LSB from the displayed value to keep it stable.

The response time of the system is also somewhat slow as the counter is only incremented or decremented by one count each cycle, so many cycles may be needed before the delays are matched. Some additional gating of the return signal, the DS1020/DS1021 output and a high frequency clock could be used to allow the counter to count up or down multiple clock cycles depending on the timing difference between the return signal and the DS1020/DS1021 output.

Cascading Multiple Devices

SERIAL OPERATION Figure 8
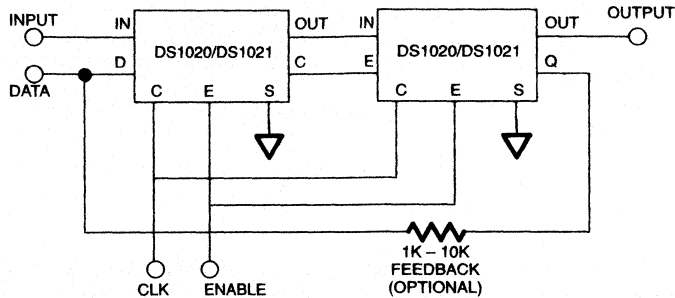
Multiple devices can be simply cascaded together. When this is done the input to output delay will be equal to twice (or n times for n devices) the step zero delay (20 ns) plus the sum of the programmed delays for each device. This is equivalent to doubling the number of steps, or range, of a single device.

Figure 8 shows the arrangement for serial programming. The serial data output is daisy chained to the serial input of the following device, the clock and enable pins are connected, and driven, in parallel. The Mode Select pin (S) is tied low for serial operation. (The unused parallel data pins, PI – P7, must be tied to a well defined logic level and not allowed to float.)

Also shown is a feedback resistor to allow the contents of the internal registers to be read back. If this option is used the source driving the serial data input must be in a high impedance state during readback.

PARALLEL OPERATION – 16 BIT Figure 9
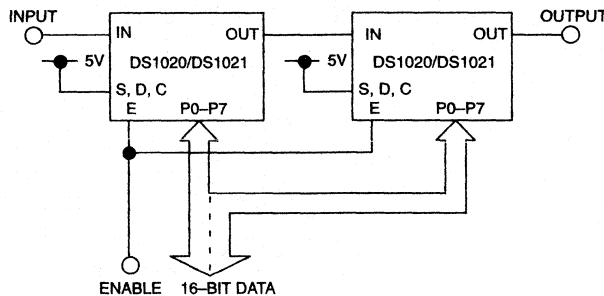
If parallel programming is preferred the parallel data inputs, P0 – P7, are driven, the Mode Select pin, S, is held high and the unused serial data in (D) and clock (C) pins are tied to well defined logic levels. In this mode the Enable pin can be permanently tied high, or driven to a low level to latch in the programmed data and allow further changes on the data inputs to be ignored (e.g. for use on a data bus).

In the parallel mode the data could be 16–bits wide, or two 8–bit bytes could be used with the Enable pins on each device being driven separately to steer each byte of data to the appropriate device.

PARALLEL OPERATION – 8 BIT Figure 10
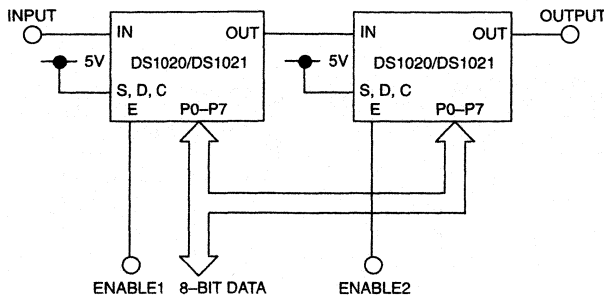
If the data lines for each device are driven in parallel (both Enables high) it is functionally equivalent to having a single DS1020/DS1021 with twice the delay step size and total delay. The same effect can be realized in the serial mode either by driving the data inputs, D, in parallel (do not connect the Q output from the first device to the D input of the second !) or by simply repeating the same 8–bit sequence twice in the daisy chain mode (see Figure 10).

ENABLE1	ENABLE2	OPERATION
0	0	Data latched into both devices. Changes on data bus have no effect.
0	1	Data latched in device 1. Device 2 follows data bus.
1	0	Data latched in device 2. Device 1 follows data bus.
1	1	Both devices follow data bus. Equivalent to single device with 2x step size and range.

PRINCIPLES OF OPERATION

This section gives some insight into what is happening internally to the device. While it is not necessary to review this to use the device it is beneficial when it comes to understanding some of the design considerations discussed in this brief.

Basic Delay Element
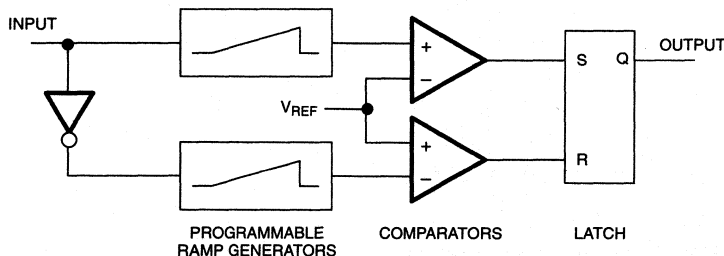
The diagram above shows the basic delay line timing element. The input signal is effectively integrated by ramp generators which must exceed the comparator threshold voltage before the input pulse is transferred to the output via an S–R latch. The positive–going edge of the input signal is used to initiate a timing ramp which ultimately sets the output latch, the negative–going edge of the input initiates a second ramp which ultimately will reset the latch.

TIMING WAVEFORMS Figure 11
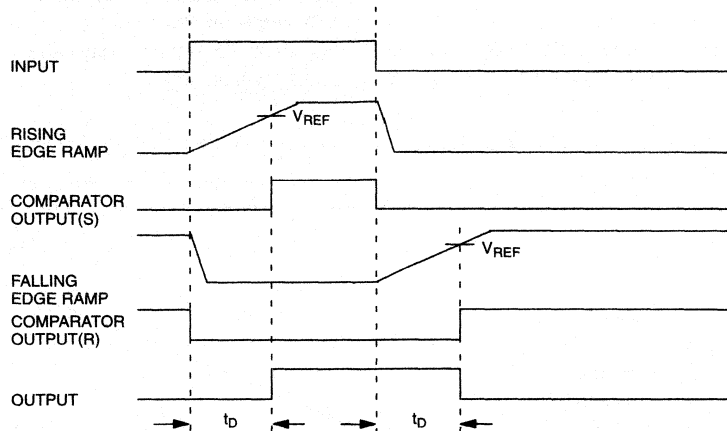
Therefore, as can be seen from the timing diagrams, the delay time is dictated by the slope of the rising and falling edge ramp generators (assuming a fixed threshold voltage for the comparators).

Ramp Generators
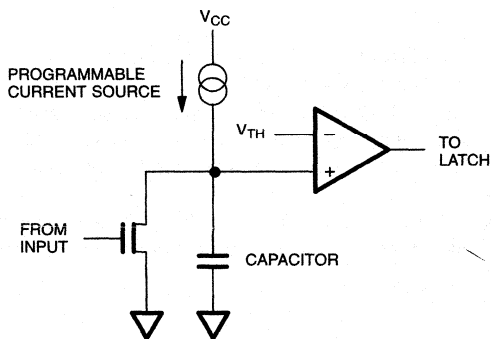
The basic elements of the ramp generators are capacitors which are charged at a constant rate from programmable current generators. Prior to a timing period the capacitor is kept discharged by the input transistor. When an input transition occurs the input transistor is turned off allowing the capacitor to charge up to the threshold voltage of the comparator, which in turn sets (or resets) the output latch. When the input returns to its

original state the input transistor discharges the capacitor ready for the next cycle.

NOTE: Inverter stages are inserted between the device input pin and the input transistor as appropriate for either the rising edge delay or the falling edge delay portions of the circuit.

Programmable Current Source
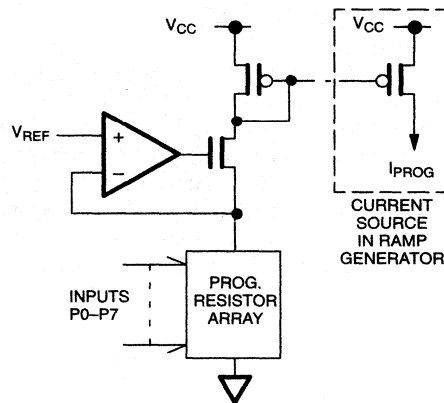
A representation of the programmable current source is shown above. An internally derived reference voltage is applied to an array of resistors. The resistance of this array will therefore set the value of the current flowing through the output transistors. This current is in turn

mirrored by another transistor in the ramp generator circuit.

The value of the resistor array is determined by the binary code applied to the digital inputs. This binary number is decoded and used to select the appropriate resistor combination to set the desired current. Laser trimming is employed during manufacture to minimize the effect of normal process variations on the accuracy of the resistor values.

DS1020/DS1021 "IDEAL" RESPONSE Figure 12
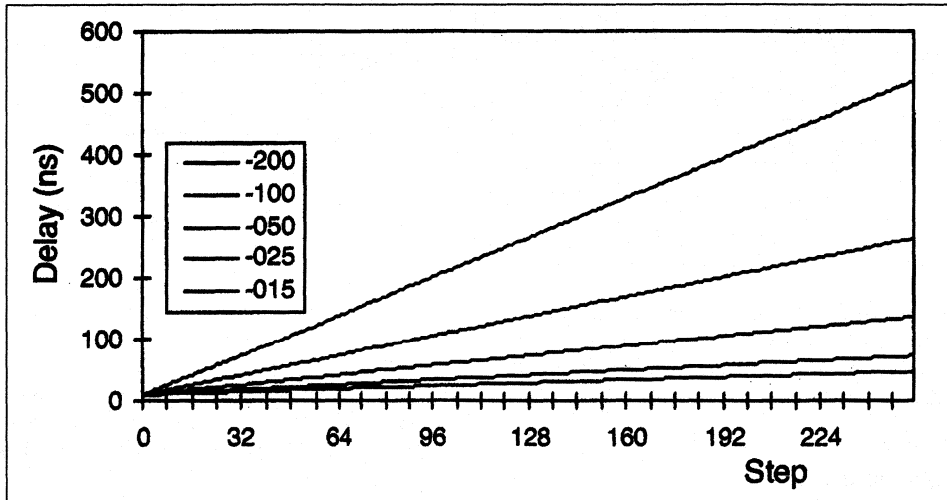
DESIGN CONSIDERATIONS
Delay Tolerance
An "ideal" programmable delay line would show a linear response between the programmed delay value and the measured output delay (see Figure 12). In the real world manufacturing variances, e.g. normal process tolerances etc., result in a deviation from this ideal response. The charts on the following pages show typical response curves for members of the DS1020/DS1021 family. This data is the average for five devices of each speed grade, and was taken at nominal conditions (5V supply and 25°C). The scales on the charts have been chosen such that the maximum and minimum values correspond to the data sheet limits for deviation from the ideal response for each device.

NOTE: There is an additional (tighter) tolerance specification for the step 0, or inherent, delay of the device.

DEVIATION (ns) Figure 13
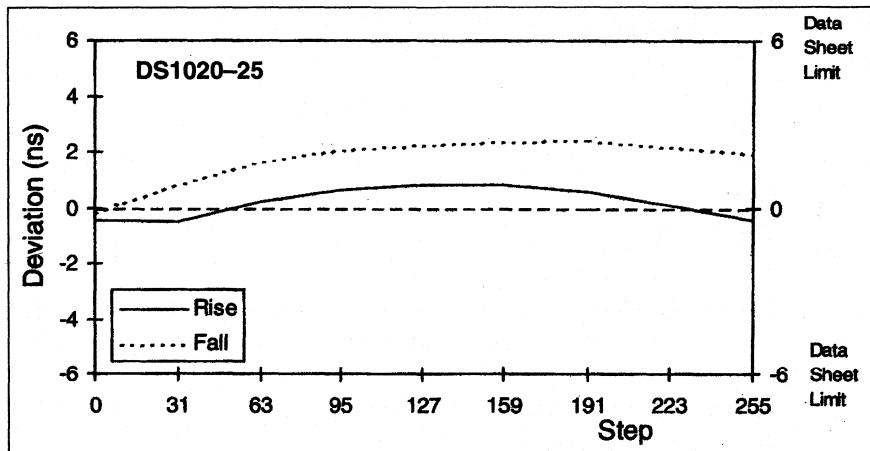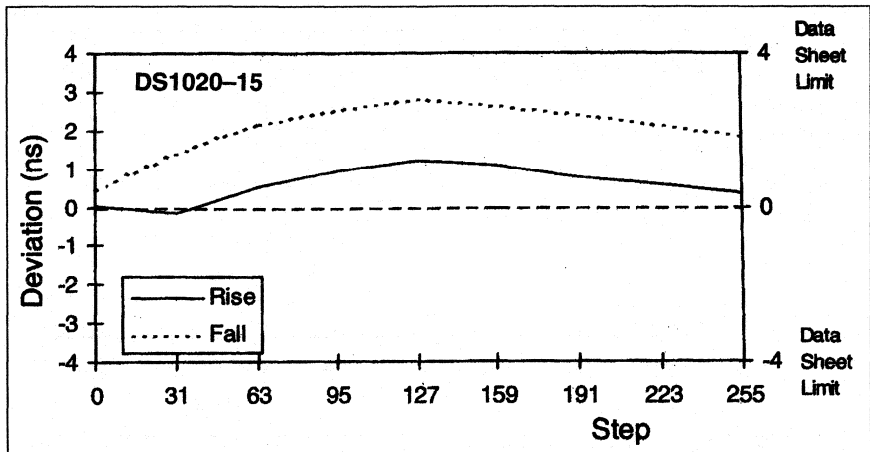
DEVIATION (ns) (cont'd) Figure 13
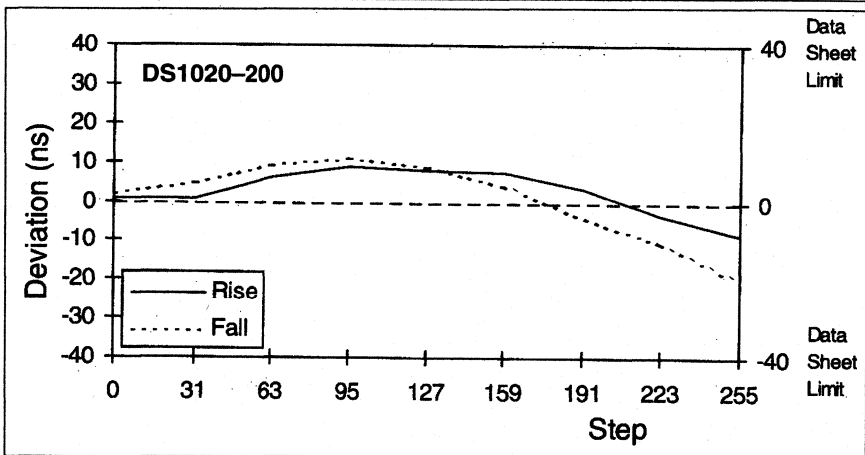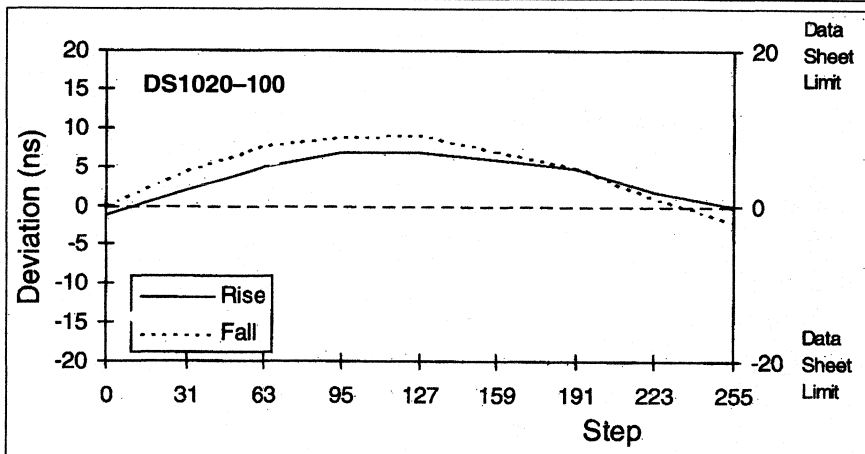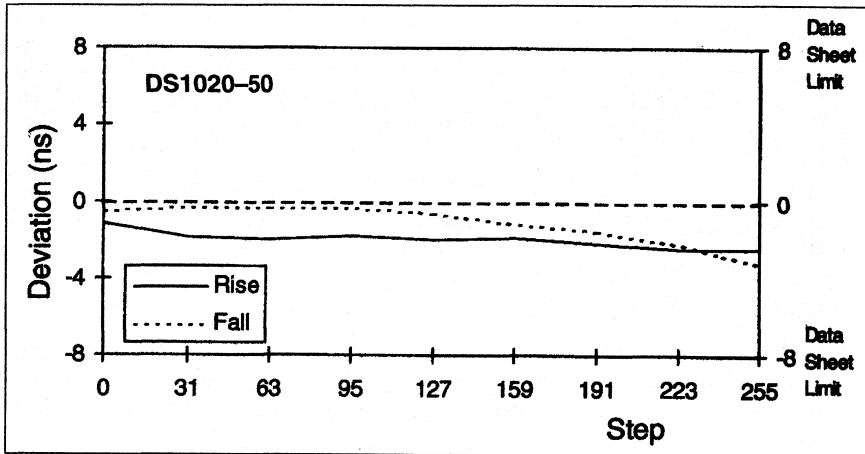
The significance of this variation will depend on the application. In closed loop systems, as long as the programmed delay versus actual delay response is monotonic, the loop will settle to the correct value.

In systems where the delay is not altered during normal operation (only being changed during a production calibration step, for example), the programmed value can be varied until the desired measured delay is reached. This value can then be hardwired on the board, typically using DIP switches or jumpers.

Jitter

The output of the device is subject to some random fluctuations (jitter) caused by noise. This effect can be mini-

mized by good supply voltage decoupling, but some jitter will remain due to internally generated noise. This effect is most noticeable at the longer programmed delays (when the internal charge currents are smallest), when a small amount of noise can cause some variation in the internal trigger points.

The following charts show some typical variations in peak–to–peak jitter with output delay for the various members of the family. When considered as a percentage of the programmed output delay the peak–to peak jitter will usually be less than 3%.

PEAK–TO–PEAK JITTER (RISING EDGE) Figure 14
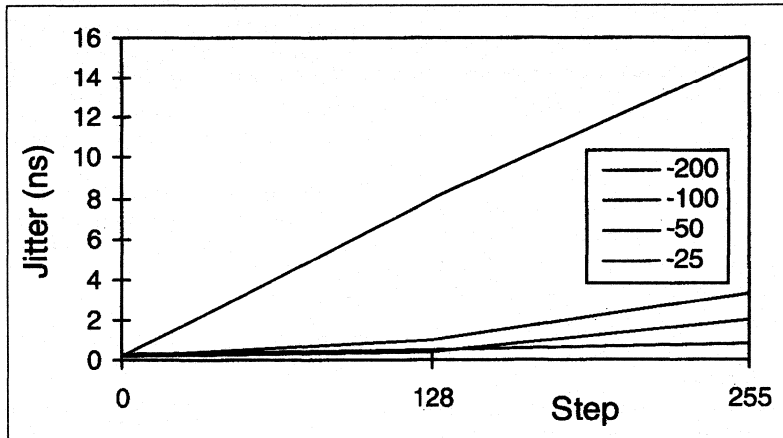
PEAK–TO–PEAK JITTER (FALLING EDGE) Figure 15
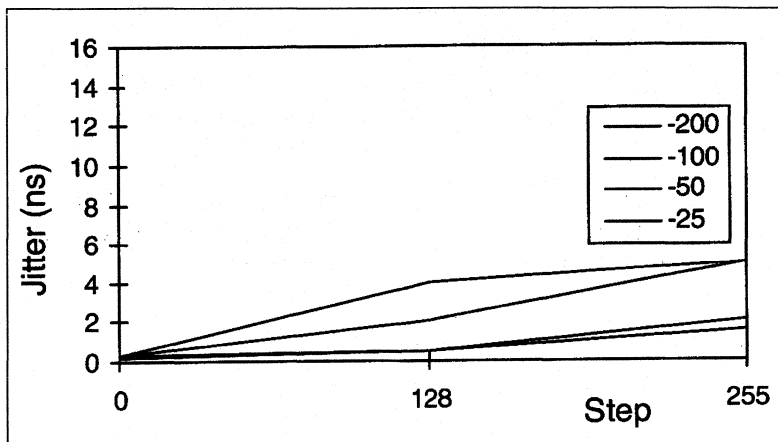
The actual measured peak–to–peak jitter will be very dependent on the actual circuit configuration (e.g., noise on the supply, decoupling, other noise sources in close proximity, etc.). This probably accounts for the somewhat anomalous readings for the –200 device rising edge. An rms measurement of jitter will yield somewhat smaller results. The main consideration is that output jitter will exist and that it will increase with longer programmed delays.

In the extreme situation when the input pulse width is close to the programmed delay value there can be a substantial increase in jitter. This is caused by noise being introduced to the supply line when the output changes state. If this noise occurs close to the next transition on the input, the trigger point becomes less well defined and output jitter will increase. Although the device should not normally be operated under these conditions (see Page 108), it is of interest when using configurations such as the oscillator described on pages 91–93, and explains why the jitter of the oscillator exceeds the values shown above.

NOTE: If you wish to attempt to make jitter measurements it is unwise to simply measure the delay time between successive output edges. Most test generators also generate jitter. Therefore timing measurements should be made between an input edge and the corresponding output edge.

Improving Absolute Accuracy
Systems in which the delay is changed during operation and have no feedback applied may require a greater absolute accuracy than the data sheet tolerance, perhaps approaching the same magnitude as the individual step size. In this situation each device will need to be individually calibrated to the desired accuracy. Typically this would be done by measuring actual delay values in the application and generating a look–up table in EPROM for use in normal operation.

Alternatively the device can be operated in a closed loop mode to eliminate the effects of any inaccuracies (see Page 93).
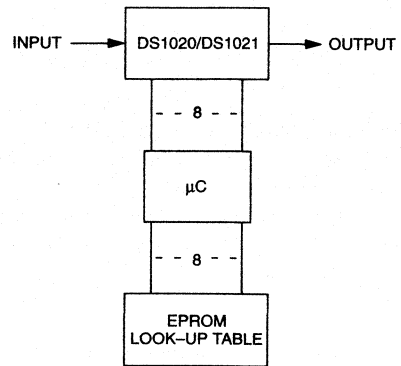
Output Loading
As with any type of delay line the output delay will be dependent on the output loading. The DS1020/DS1021 is tested and characterized with a load of 15 pF. Different values of load capacitance will change the slope of the output rise and fall waveforms and produce a resultant change in measured delay time.

The effect of various output loads can be approximated by assuming the output consists of a voltage source, switch and resistors as shown in this diagram:

SIMPLIFIED OUTPUT MODEL Figure 16
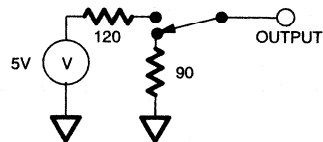
Using this model the following charts were generated to show the predicted change in output transition with load capacitance.

NOTE: The resistor values used were chosen so that the charts approximate to actual (dynamic) in–circuit performance and do not necessarily apply for DC analysis.

OUTPUT LOADING (LOW–TO–HIGH) Figure 17
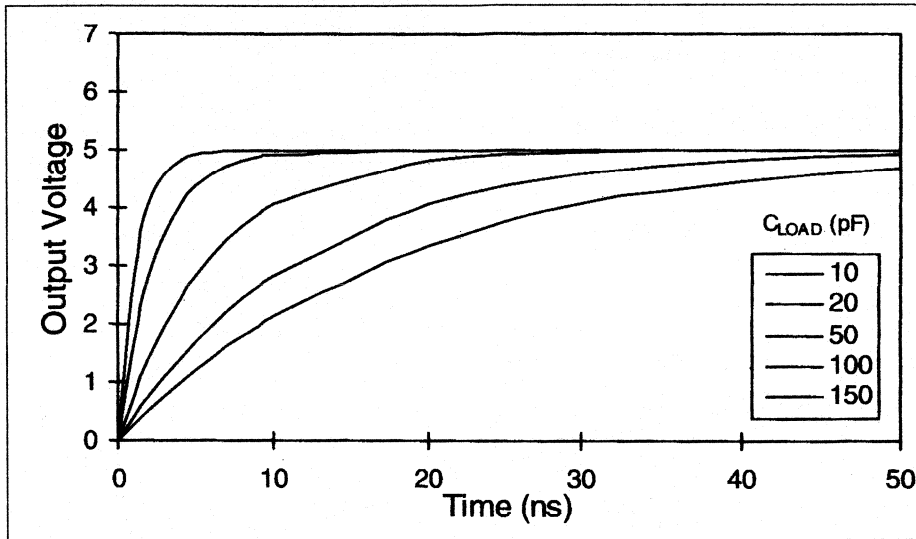
OUTPUT LOADING (HIGH–TO–LOW) Figure 18
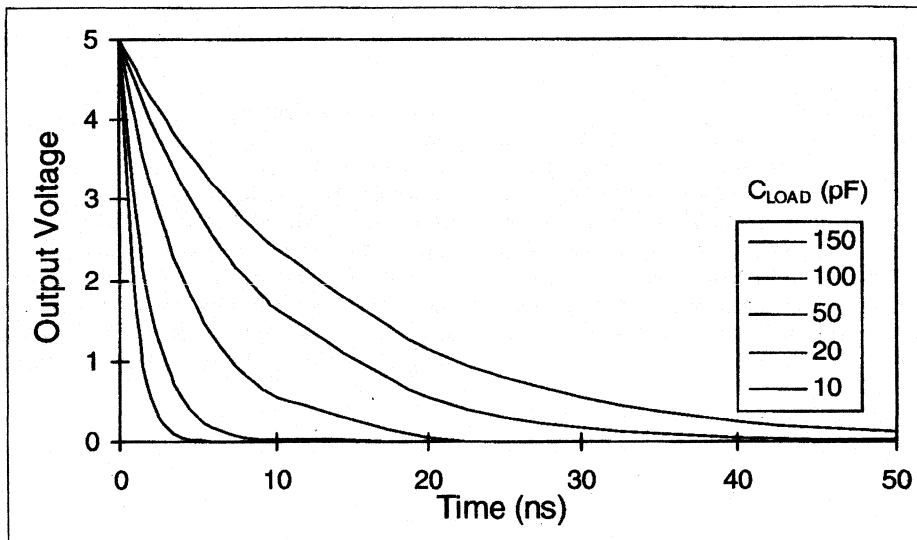
Voltage and Temperature Variations

The data sheet specifications on delay tolerance apply at 25°C and with a 5V supply. The DS1020/DS1021 feature on–chip circuitry to minimize the effect of temperature changes, but there will still be some change in the resultant delays if either the temperature or supply voltage changes from these nominal values.

SUPPLY VOLTAGE VARIATION Figure 19
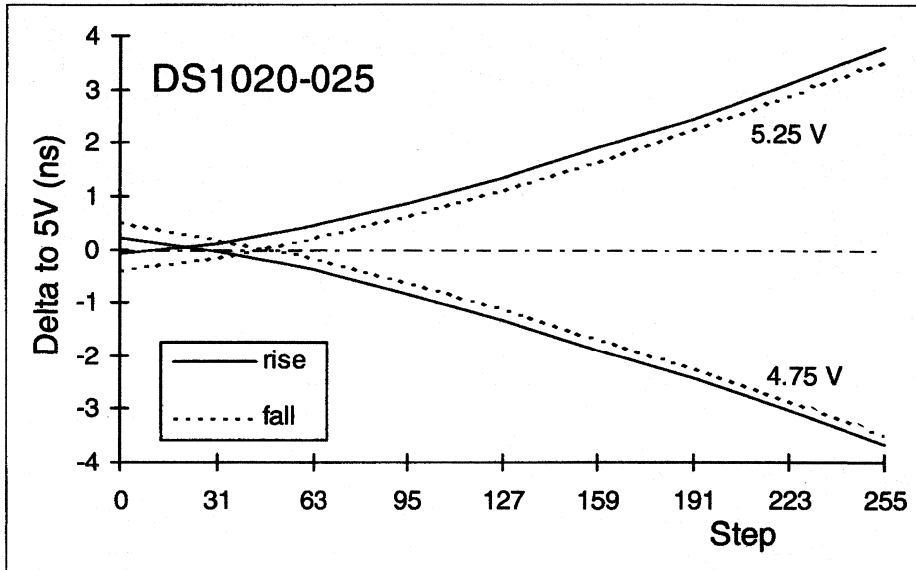
Supply Voltage Variation

The chart above shows the variation in delay time with supply voltage. It can be seen that the variation is almost directly proportional to the programmed value. The maximum variation is about 3.5 ns for this device, which is approximately 5% of the programmed delay. This results from a 5% change in supply voltage, so a good approximation to the percentage change in delay can be obtained by assuming that it is equal to the percentage change in supply voltage.

Even though the data shown pertains to the –025 device, all the members of the family exhibit similar characteristics.

Temperature Variation

Some typical variations with temperature for the DS1020/DS1021 family members are shown in the following charts. This data is based on a sample of five devices of each speed option, operating from a 5V supply. In each case the change in delay versus 25°C operation is indicated, where positive values indicate an increased delay and negative numbers a decreased delay. Data was taken for both rising and falling edges and at the extremes of the operating temperature range.

DELTA TO 25C (ns) Figure 20
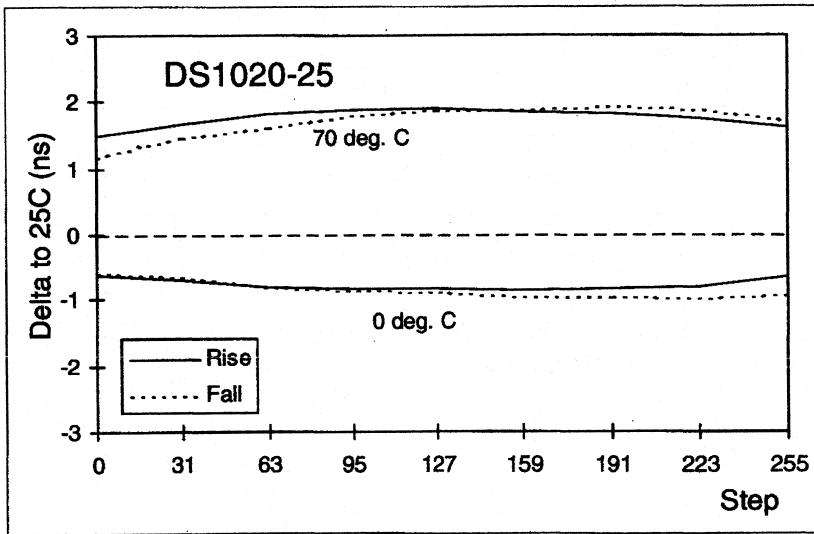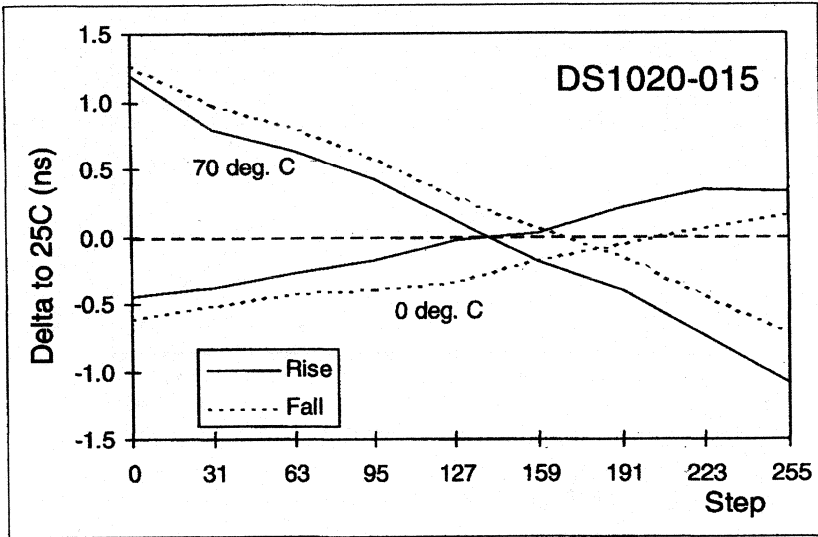
DELTA TO 25C (ns) Figure 20 (cont'd)
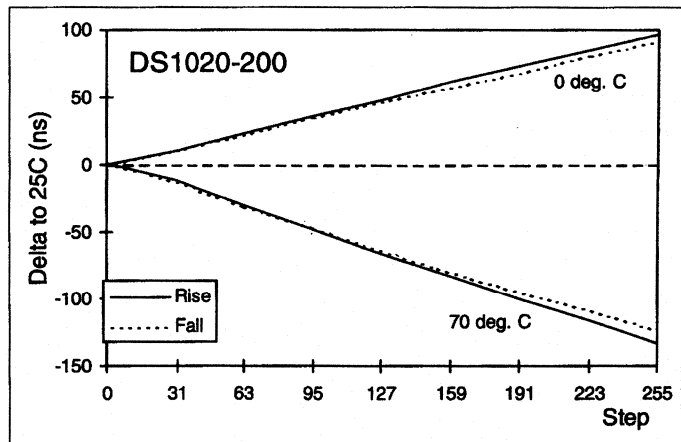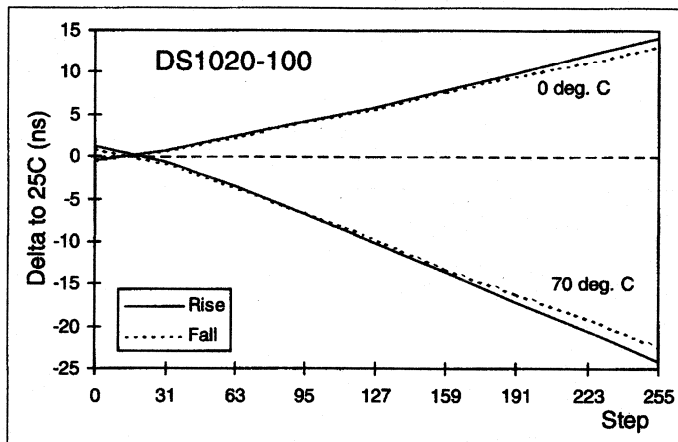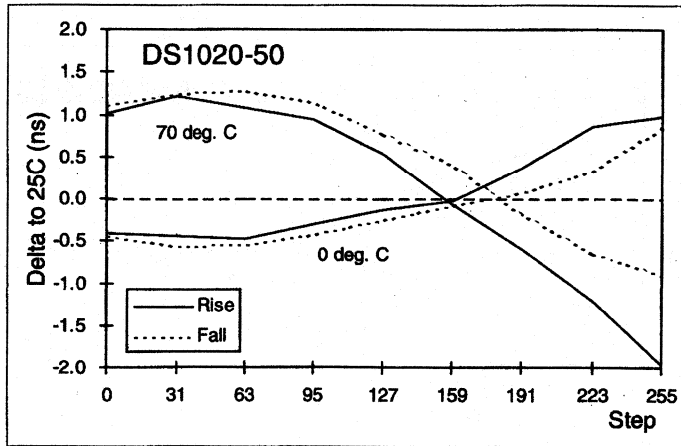
Power Up
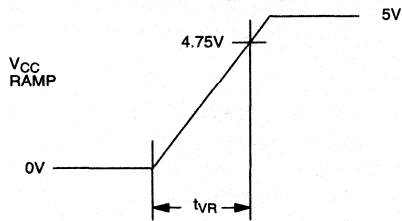
The DS1020/DS1021 features internal Power On Reset circuitry to ensure that all the internal functions are brought up in the correct state. However, if the supply voltage is brought up too rapidly, a situation can occur where the device does not have enough time to complete these reset activities before attempting to go into normal operation.

Most system power supplies come up relatively slowly and have no adverse effect on normal device operation. The DS1020/DS1021 differ in their tolerance to rapidly rising supplies. The DS1021 will operate correctly when the rise time (t_{VR}) is greater than 20 ms. An additional test screen for the DS1020 allows the use of faster rise times, down to approximately 2 ms.

If it becomes necessary to use either device with a power supply which rises too quickly to meet the respective rise time requirements it may be possible to use a series resistance from the system supply to the device supply pin in conjunction with additional decoupling capacitance at the device. Care should be taken to ensure that the voltage applied to the device remains within the data sheet limits when the device supply and load currents are taken into account.

Changing the Programmed Delay
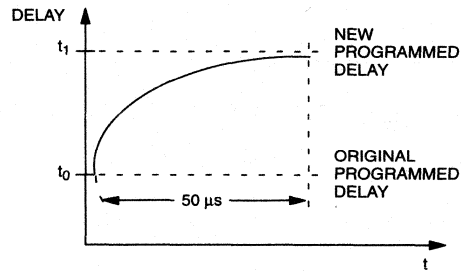
The data sheet contains two parameters, t_{VPD} and t_{EDV} which relate to the time taken after a programming change before the output delay is valid. Some commonly asked questions are:
Why is this delay so long (50 μs)?
What happens in the time until this condition is met?

To understand the reason for this delay it is necessary to refer to the means by which the delay is programmed (see Page 97). The main reason is related to the programmable current source. When the programming is changed there is a finite settling time associated with the op amp configuration of the current source. Therefore the 50 μs is derived from the time taken for this current source to become stable at the new value.

While the current source is settling the device continues to operate normally – input pulses continue to be delayed before reappearing at the output. However, the exact duration of the delay will be uncertain until the settling time has passed. In practice, pulses arriving during this 50 μs time interval will typically be delayed by an amount somewhere between the old programmed value and the new programmed value. The current source settles in an exponential fashion and this is reflected by an exponential change in delay time until the new, stable value is reached.

Minimum Input Pulse Width/Period
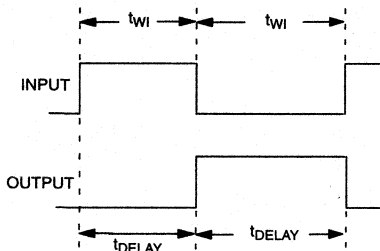
For the device to operate correctly the input pulse width must be greater than the output delay time (see diagram above). This applies to both the high level pulse width and the low level pulse width. The reason for this is that the input pulse is integrated to provide the output delay (see Page 97).

For example, consider the case when the input is in a high state. The leading edge ramp generator must be allowed to reach the comparator threshold, and set the output latch, before the input changes state. Otherwise the output will not go high, the leading edge ramp generator will be reset, and the trailing edge ramp will be initiated. In effect, the input pulse has been "swallowed".

NOTE: This phenomenon can be used to good effect as a pulse width discriminator. By suitably altering the programmed value only pulses widths greater than a certain value will be output, narrow pulses (and/or noise) will be rejected.

The tolerance of the output delay must also be taken into account and the input pulse widths set to exceed the maximum actual delay rather than the ideal programmed value.

From the preceding paragraph it can be seen that the minimum input period will be equal to twice the minimum pulse width (assuming a square wave input). The data sheet limit of three times the input pulse width is therefore somewhat conservative. As long as adequate margins are included to allow for device tolerances correct operation is assured.

NOTES:

DALLAS
SEMICONDUCTOR

Application Note 1
Pulse–Doublers Using the DS1012 Delay Line with Logic

FEATURES

- All Silicon Delay Line with logic

- Low Power Operation: 53 µW max quiescent mode

- Two input device/four independent buffered delays

- TTL/CMOS compatible

- Quick turn prototypes available through LPP

- Custom delays and logic options available

- Logic Options: AND, NAND, OR, NOR, XOR, XNOR, HALF–XOR, HALF–XNOR

- Delay tolerance: ±1.5 ns (delays: 3–10 ns)
                     ±2.0 ns (delays: 11–40 ns)

- Vapor phase, IR and Wave Solderability

- Small Outline Packaging Available (150 mil Mini–SOIC)

APPLICATIONS

- Pulse Doublers

- Voltage Control Oscillator (VCO)

- Waveform Generation

- Manchester Encoder

DS1012 BACKGROUND

Description
The DS1012 delay line is probably one of the most unique product offerings in the Dallas Semiconductor delay line family. Not only can simple delays be obtained from the device, but also delays combined with logic functions. Figure 1 presents the general logic diagram for the DS1012 Delay Line. As shown, the device is provided with two inputs and four outputs. Outputs 1 and 2 are straight delays of inputs 1 and 2, respectively.

These outputs can also be configured to provide delayed complements of the inputs if desired. Outputs 3 and 4 are combinations of inputs 1 and 2 having both a delayed and logic relationship. Function F3 can provide the functions AND, HALF–XOR, a straight delay of D3 or a complement of these functions. Similarly, function F4 can provide the functions OR, XOR, a straight delay or their complements.

Specifying a Custom DS1012
As a product, the DS1012 has seen most use in custom applications where users of this device wish to develop custom waveforms by specifying both delay and logic combinations. Dallas Semiconductor, through late package programming (LPP), has the ability of providing (to customer specifications) customized parts for evaluation and testing usually within a 10–day period.

The DS1012, as shown in Figure 1, can be configured with numerous options ranging from simple delayed outputs to delayed outputs combined with logic. When determining new applications for the DS1012, delay and/or logic specifications must adhere to rules provided in Table 1 and/or Device Notes. There exist two major considerations which must be made when selecting the device for a particular application; namely, operating frequency and delay/logic available. The operating frequency (input signal, duty cycles, etc.) is in large part dependent of delay values chosen. Additionally, delay values D1, D3 and D2, D4 are related in terms of what values can be selected for the pairs. In any configuration D1 and D2 can be specified between the range of 4 ns and 10 ns, and D3 and D4 can be specified between the range of 4 ns and 40 ns. However, if D1 is greater than 10 ns, then D3 must be equal to D1. Similarly, if D2 is greater than 10 ns then D4 must be equal to D2. Table 1 and the Device Notes should be used to check the validity for values of delay and logic chosen.

102594 1/6

Once the values of D1, D2, D3, and D4 have been made, determination of input pulse width and period can be made. For example, the DS1012–001 is configured with delays D1=5 ns, D2=5 ns, D3=10 ns and D4=10 ns. The minimum input pulse width, t_{WI}, for the device is specified by three times the longer of D1, D2, D3, or D4 or in this case 30 ns. The minimum period, T, will be equal to twice t_{WI} or 60 ns. Pulse doublers are configured to exceed the limitations of minimum period and input pulse width as shown in the Device Notes. Here the minimum input pulse width approximates the longer of the delays D1, D2, D3 or D4.

DS1012 LOGIC DIAGRAM Figure 1
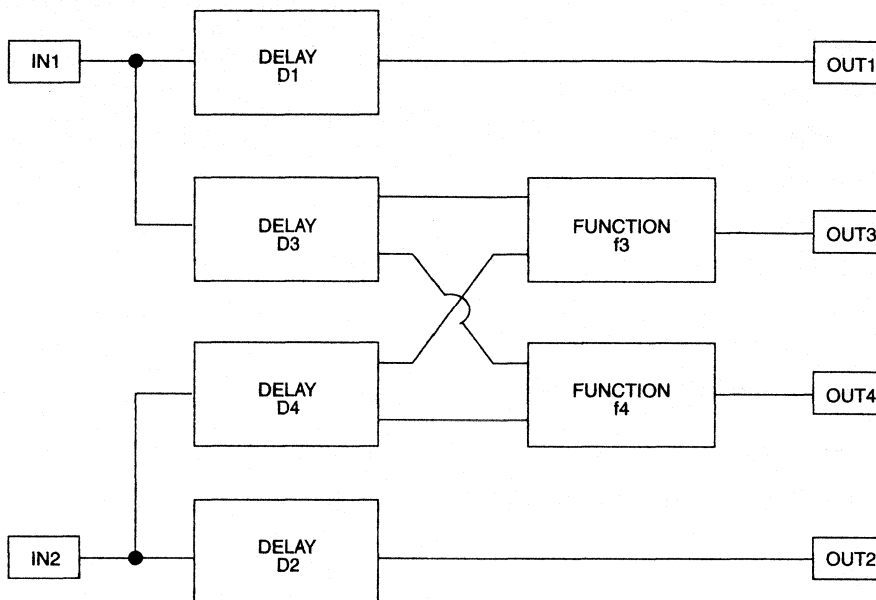
AC ELECTRICAL CHARACTERISTICS Table 1

PARAMETER	SYMBOL	MIN	TYP	MAX	UNITS	NOTES
Input Pulse Width	t_{WI}				ns	1
Input to Output (leading edge)	$t_{D1}, t_{D2}, t_{D3}, t_{D4}$				ns	2
Power–up Time	t_{PU}			0	ns	3
Period	T	$2(t_{WI})$			ns	

Device Notes:

1. For specified accuracy, t_{WI} (min) is the longer of $3(t_{D1})$, $3(t_{D2})$, $3(t_{D3})$, or $3(t_{D4})$. Pulse doublers designed for single frequency use will meet specified accuracies at 50% duty cycle; i.e., $2(t_{WI}) = 1/FREQ = PERIOD$. Customs will be adjusted to be accurate at customer input width specifications when t_{WI} is longer than t_{D1}, t_{D2}, t_{D3}, and t_{D4}.

2. $V_{CC} = 5V$ @ 25°C. Delays referenced to leading (input rising) edges are accurate within ±1.5 ns for values between 3 to 10 ns and ±2 ns for values between 11 to 40 ns. Delays referenced to trailing (input falling) edges will typically equal the corresponding leading edge delay within ±1 ns.

3. On power–up, the DS1012 will supply timing and logic functions with specified accuracy as soon as V_{CC} achieves nominal value.

Power Supply Conditioning

The DS1012 is recommended for operation with supply voltages between 4.75V (min) and 5.25V (max). Output delay is typically affected by supply voltage and can vary 15% over the 4.75V to 5.25V range. In most applications it is recommended that the power supply be decoupled as shown in Figure 2.

POWER SUPPLY CONDITIONING Figure 2
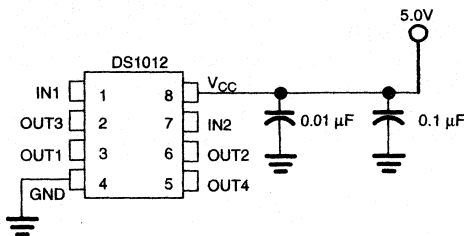
Package Options

The DS1012 is available in two different package options, the 8–pin DIP (300 mil, DS1012M–xxx) and the 8–pin mini–SOIC (150 mil, DS1012Z–xxx). Package dimensions can be found in the Dallas Semiconductor System Extension data book.

PULSE–DOUBLER APPLICATIONS

The function of a pulse doubler is to produce two output pulses from a single input pulse. Pulse and frequency doublers are implemented in a variety of ways. Some devices use PLL technology to double the frequency of the input. These devices require synchronization and lock times in the tens to 100s of milliseconds to generate the doubled output. Some designs employ the use of discrete delays, buffers, and logic to generate the desired output. Discretes provide the feature of no synchronization and lock times, however, multiple components, and additional board space are required. The DS1012 can be configured to provide, through a combination of logic and delay, a pulse doubler for a specified input waveform. Unlike the discrete approach, all device technology resides in one package (see Figure 1). Additionally, the device does not use PLL technology and thus requires no synchronization or lock times for generating the pulse doubler output.

A logic diagram of the DS1012 configured as a pulse–doubler is presented in Figure 3. Inputs 1 and 2 are driven by the same input signal. Delays D3 and D4 are typically set so that the time difference between the two is equal to a quarter period of the input wave to be

doubled. This setup gives an output waveform of 50% duty cycle as shown in Figure 4a. Figures 4b and 4c show waveform outputs as the time difference between delays D3 and D4 change with respect to the period of the input waveform. These figures illustrate how output waveforms can be shaped in addition to doubling. The doubled output is taken from OUT4. If delay D1 and D3 are equal in value, the OUT1 and the doubled output, OUT4, will be in–phase. OUT3 in the doubler application is most often a don't care. A circuit diagram of this configuration is shown in Figure 5.

The D–series of DS1012s (DS1012–D16, DS1012–D20, DS1012–D25, and DS1012–D33) have been configured through delay and logic selection to operate as pulse doublers; a function that is finding considerable use in the PC upgrade market. The typical requirement for the PC upgrade market is to make use of the existing system clock by using some combination of delay and logic to double the clock for use with a faster processor. The DS1012 is an ideal platform to provide the doubler function since a combination of delay and logic is present in the same package. Additional board space and cost savings can be realized by using the tiny 150 mil mini–SOIC package.

DS1012 PULSE–DOUBLER LOGIC DIAGRAM Figure 3
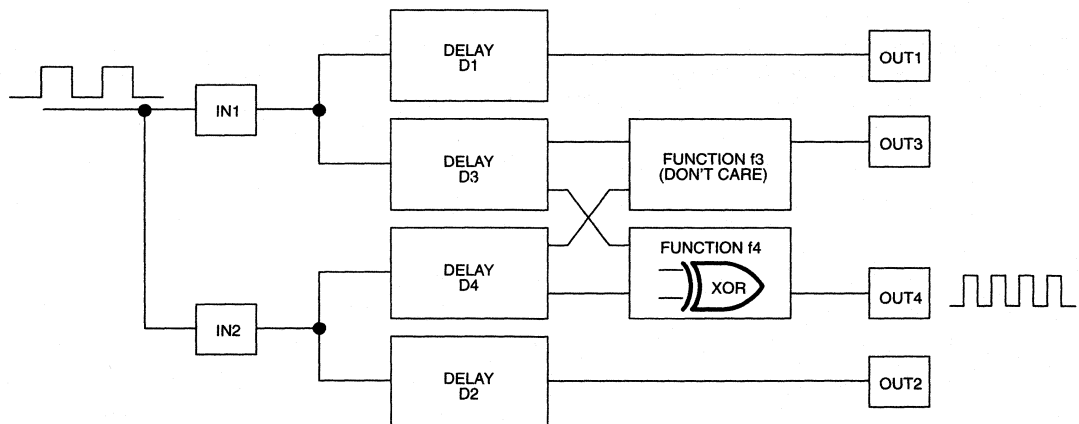
|D3–D4| = T/4, where T = period of the input pulse.

DELAY TIMING DIAGRAM Figure 4

(a)

|D3–D4| = T/4 where T = period

(b)

|D3–D4| < T/4 where T = period

(c)

|D3–D4| > T/4 where T = period
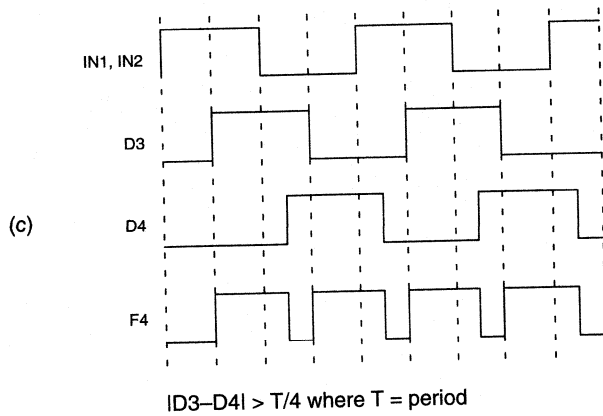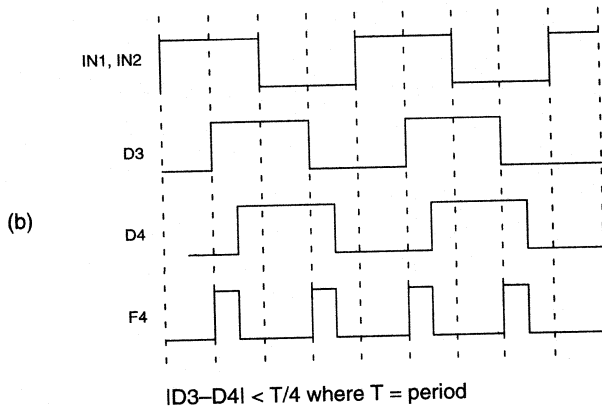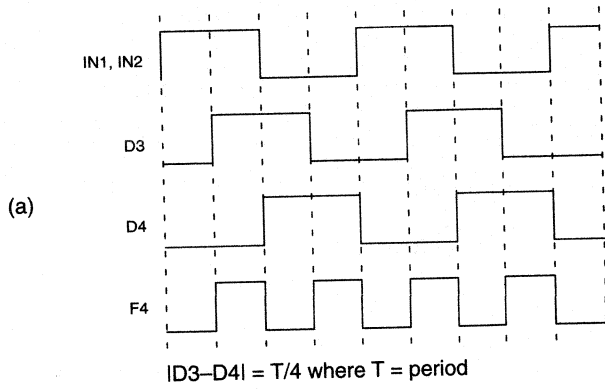
DS1012 PULSE–DOUBLER CIRCUIT DIAGRAM Figure 5

DALLAS
SEMICONDUCTOR

<div align="right">
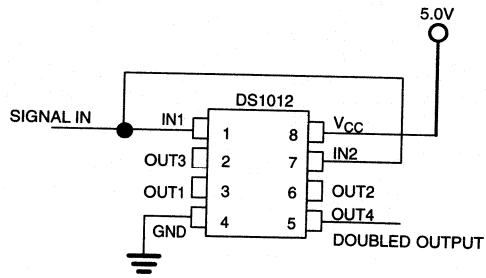
Application Note 3
Device Characteristics of the DS1045
Dual 4-Bit Programmable Delay Line

</div>

PRODUCT DESCRIPTION

The DS1045 is a 4-Bit Dual Programmable Delay Line that supports two programmable outputs from a single input. This CMOS device is capable of producing outputs in binary steps for maximum delays of up to 84 ns. The selection of one of four standard devices will allow steps of 2, 3, 4, or 5 ns. Table 1 indicates the standard product part for each delay and the maximum delay that can be obtained by each part.

Each half of the device can be programmed through separate input ports. Inputs A0 through A3 control side A output while inputs B0 through B3 control side B output. The inputs can either be held in a static mode or changed dynamically. In the dynamic mode the data enable, setup and hold times must be met. Typically the delay to a valid output is 15 ns. During the transition period the outputs are in an undefined state. The pulse widths of the output will be a reproduction of the input delayed by the selected input delay value. Typical applications are included in the following section.

Each of the outputs is capable of driving 10 standard 74LS type loads. The device is compatible with both TTL and CMOS and is specified driving a 15 pF load. Input capacitance is 10 pF. All timing measurements are measured at 1.5 volts with the exception of rise and fall times. Input and output rise and fall times are measured between 0.6 volts and 2.4 volts.

DELAY VS. PROGRAMMED VALUE Table 1

PART NUMBER	OUTPUT DELAY VALUE															
DS1045-2	9	11	13	15	17	19	21	23	25	27	29	31	33	35	37	39
DS1045-3	9	12	15	18	21	24	27	30	33	36	39	42	45	48	51	54
DS1045-4	9	13	17	21	25	29	33	37	41	45	49	53	57	61	65	69
DS1045-5	9	14	19	24	29	34	39	44	49	54	59	64	69	74	79	84
PROGRAM VALUES FOR EACH DELAY VALUE																
A0 or B0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
A1 or B1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
A2 or B2	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
A3 or B3	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

072996 1/18

MAXIMUM OPERATIONAL CHARACTERISTICS

The DS1045 is capable of operation at a very high speed. Consideration should be given with respect to the minimum pulse width of the input signal when the maximum delay step is selected. For example, if a delay of a DS1045-5 of 84 ns is selected, then the input pulse width must not be shorter than 9 ns. Table 2 summarizes the minimum pulse widths (step zero delay), maximum delay times and the delay tolerance for each part number.

PART NUMBER TABLE Table 2

PART NUMBER	STEP ZERO DELAY	MAX DELAY TIME	MAX DELAY TOLERANCE
DS1045-2	9 ± 1 ns	39 ns	±1.8 ns
DS1045-3	9 ± 1 ns	54 ns	±2.5 ns
DS1045-4	9 ± 1 ns	69 ns	±3.3 ns
DS1045-5	9 ± 1 ns	84 ns	±4.1 ns

BLOCK DIAGRAM

The DS1045 is composed of a 16 stage delay line and two sets of digital multiplexers. Each side of the device can select an appropriate output delay stage by providing an input to the control code specified in Table 1. The binary value selects the individual stage of delay that becomes the output.

DS1045 BLOCK DIAGRAM Figure 1
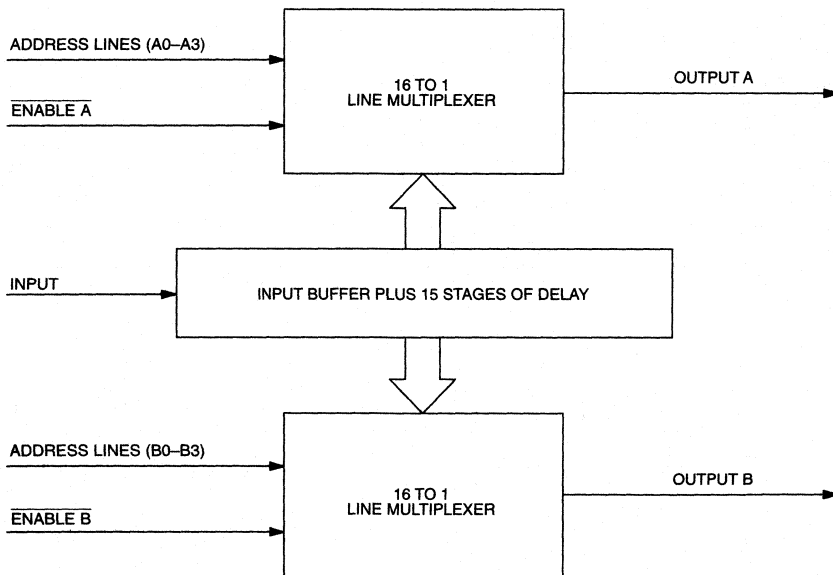
CIRCUIT ANALYSIS

Individual stages may be thought of as RS flip–flops that have a variable capacitive load. Increasing the capacitive load increases the delay. Similarly, a decrease in the load capacitance decreases the delay. The operation of the variable capacitive load is similar to the operation of a varistor. The effect of this variable capacitive loading is that the individual stage set and reset times are precisely controlled. This process establishes controlled rise and fall times and improve the input to output waveform signal integrity of the device. The exact capacitive load is established at the factory in the final stages of the manufacturing process. A simplified schematic of the circuit is shown in Figure 2.

DS1045 CIRCUIT DIAGRAM Figure 2
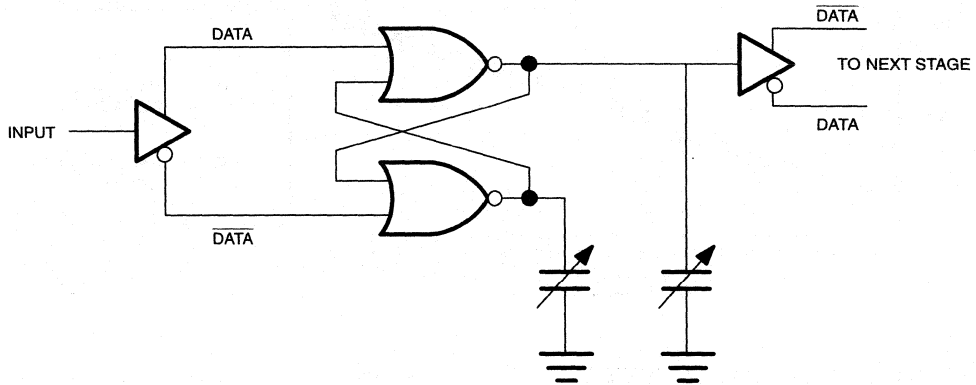
FACTORY SELECTED VALUE

FAMILY STEP CHARACTERISTICS

Figure 3 indicates the general characteristic of each of the DS1045 devices in this family as a function of Binary Step Value vs. Delay Time.

DS1045 FAMILY DELAY BY PART TYPE Figure 3
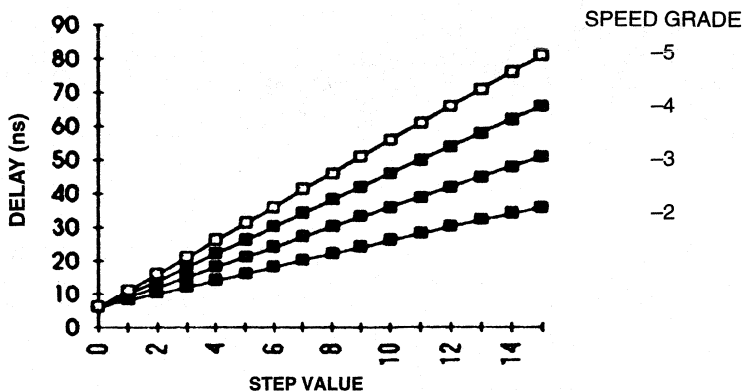
STEP LINEARITY CHARACTERISTICS

Figure 4 indicates step linearity for rising and falling edge signals. Each of the devices exhibits a virtual straight line in step linearity. The device specification limits indicate that the starting delay of the zero position on the chart is the same for each of the devices (9 ns).

This is due to the inherent delay of the first stage. The maximum deviation for any given step is indicated in Table 2. The maximum step tolerance is specified as ±2.5 ns for a DS1045–3 over the full binary range. The delay time at step zero is the initial buffer delay of 9 ns ±1 ns.

DS1045–2 RISING AND FALLING EDGE DELAY TIMES VS. STEP SIZE Figure 4
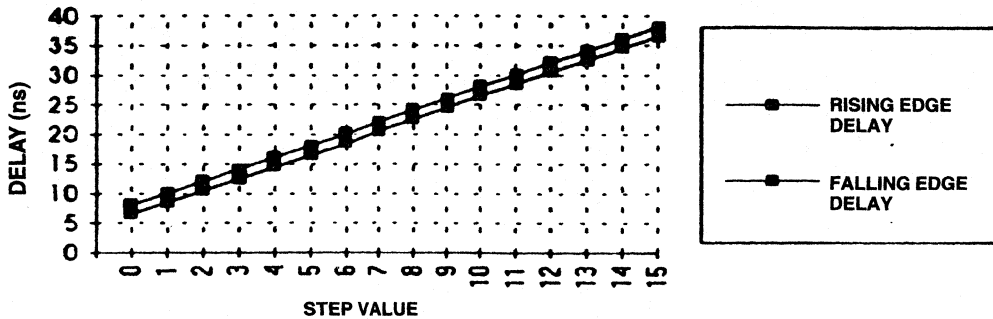
TEMPERATURE CHARACTERISTICS

The DS1045 exhibits excellent temperature characteristics. Figure 5 indicates how the change in delay times for each step is effected by temperature. The data was taken with a 5.0 volt supply voltage. A maximum total excursion (delta) of less than ±1 ns for both rising and falling edge signal are indicated.

TEMPERATURE CHANGES (0°C TO 70°C) Figure 5
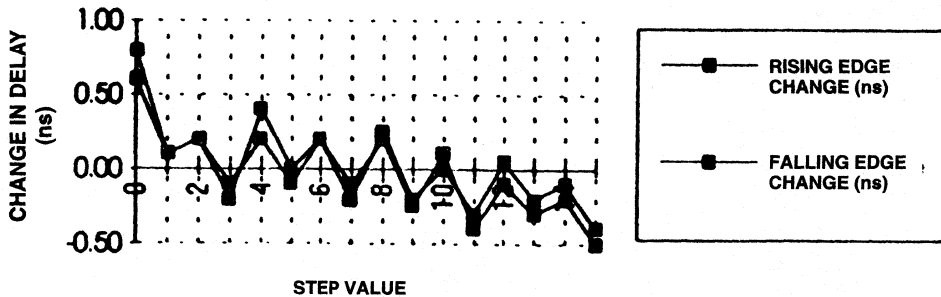
Step Value	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Rising Edge Change (ns)	0.80	0.10	0.20	–0.20	0.40	0.00	0.20	–0.20	0.25	–0.20	0.00	–0.30	0.05	–0.20	–0.10	–0.40
Falling Edge Change (ns)	0.60	0.10	0.20	–0.10	0.20	–0.10	0.20	–0.10	0.20	–0.25	0.10	–0.40	–0.10	–0.30	–0.20	–0.50

VOLTAGE CHARACTERISTICS

The DS1045 is a voltage compensated device whose outputs, both rising and falling edges, vary less than 300 picoseconds with voltage excursions from 4.75 to 5.25 volts. Figure 6 indicates the values for each of the 16 delay steps.

DS1045 VOLTAGE COMPENSATION Figure 6
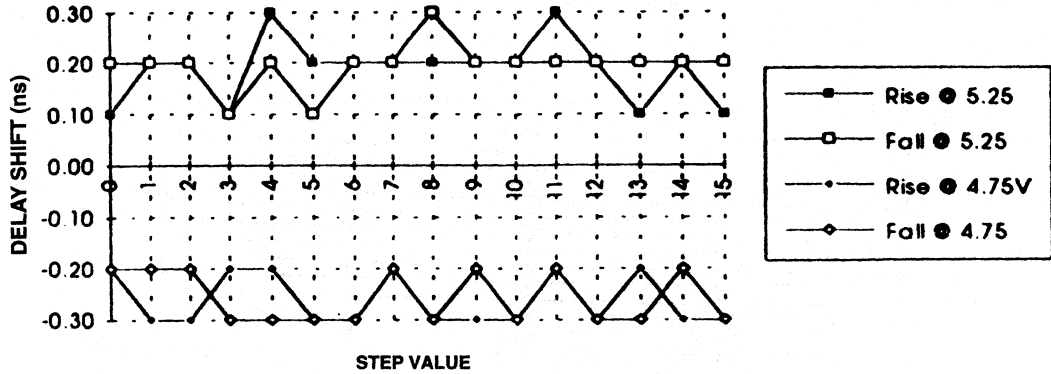
DELAY VS. TEMPERATURE

DS1045–3 SIDE A RISING EDGE SIGNAL
@ VOLTAGE = 4.75 VOLTS
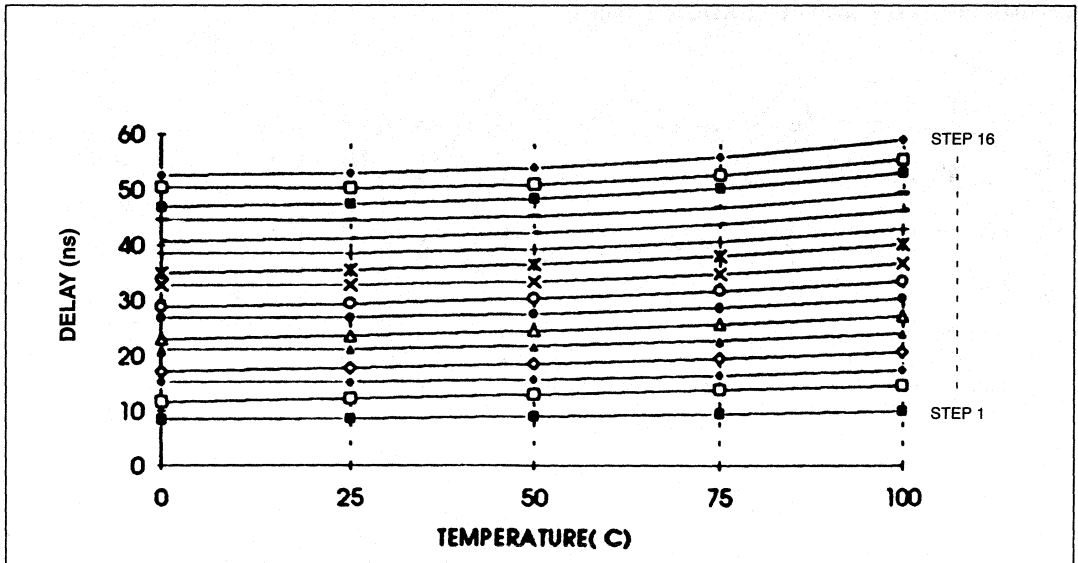
PROGRAM STEP	DELAY (ns)				
	0°C	25°C	50°C	75°C	100°C
1	8.474	8.629	9.005	9.506	10.063
2	11.587	12.223	12.966	13.769	14.602
3	15.260	15.205	15.667	16.453	17.453
4	17.075	17.678	18.491	19.468	20.632
5	21.084	21.150	21.703	22.675	24.042
6	22.963	23.576	24.496	25.674	27.182
7	26.874	26.910	27.504	28.661	30.333
8	28.820	29.393	30.340	31.674	33.528
9	32.791	32.774	33.409	34.722	36.744
10	34.957	35.502	36.512	38.035	40.193
11	38.597	38.529	39.189	40.651	42.948
12	40.635	41.125	42.133	43.787	46.259
13	44.614	44.486	45.150	46.728	49.368
14	46.891	47.321	48.367	50.188	53.020
15	50.397	50.197	50.868	52.569	55.463
16	52.545	52.878	53.918	55.857	58.977

DELAY VS. TEMPERATURE

DS1045–3 SIDE A RISING EDGE SIGNAL
@ VOLTAGE = 5.00 VOLTS
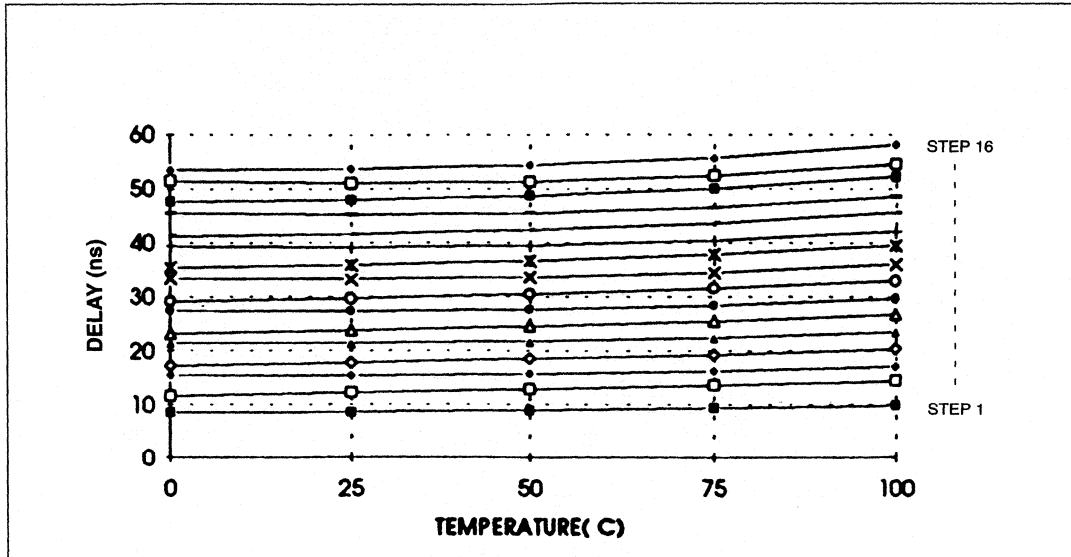
PROGRAM STEP	DELAY (ns)				
	0°C	25°C	50°C	75°C	100°C
1	8.546	8.594	8.898	9.33	9.835
2	11.539	12.162	12.869	13.619	14.388
3	15.459	15.341	15.622	16.217	17.056
4	17.179	17.716	18.449	19.293	20.333
5	21.45	21.378	21.698	22.413	23.515
6	23.168	23.713	24.491	25.46	26.761
7	27.362	27.241	27.567	28.381	29.717
8	29.151	29.638	30.405	31.46	33.01
9	33.405	33.22	33.535	34.442	36.035
10	35.399	35.852	36.646	37.805	39.591
11	39.357	39.107	39.402	40.381	42.179
12	41.217	41.592	42.342	43.57	45.603
13	45.492	45.185	45.442	46.491	48.56
14	47.597	47.914	48.667	49.989	52.243
15	51.413	51.04	51.272	52.368	54.579
16	53.398	53.622	54.304	55.676	58.13

DELAY VS. TEMPERATURE

DS1045–3 SIDE A RISING EDGE SIGNAL
@ VOLTAGE = 5.25 VOLTS
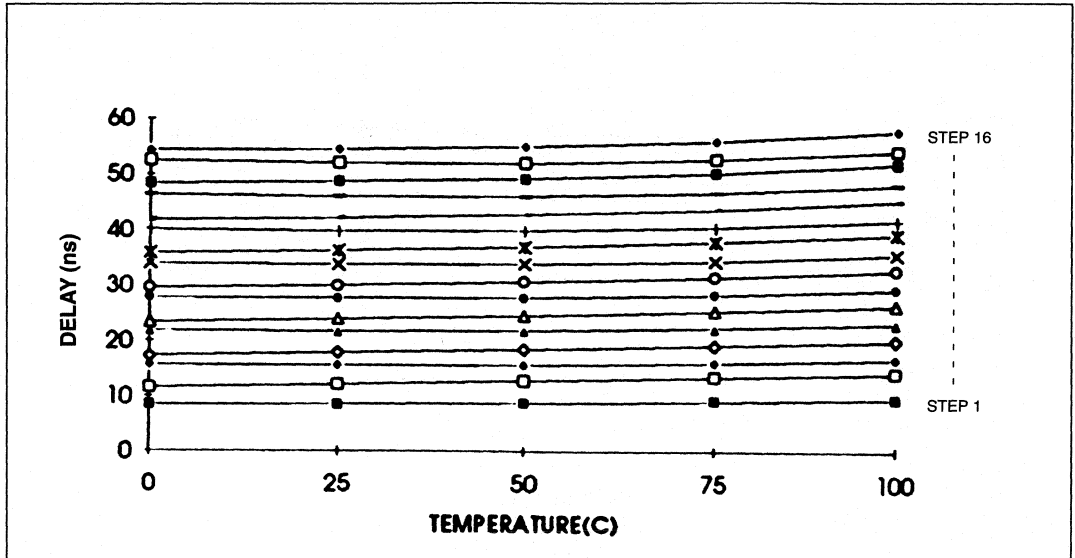
PROGRAM STEP	DELAY (ns)				
	0°C	25°C	50°C	75°C	100°C
1	8.545	8.579	8.829	9.202	9.658
2	11.542	12.114	12.801	13.515	14.257
3	15.695	15.531	15.681	16.134	16.811
4	17.264	17.814	18.478	19.234	20.153
5	21.815	21.674	21.834	22.348	23.227
6	23.400	23.911	24.598	25.428	26.555
7	27.856	27.666	27.799	28.37	29.406
8	29.512	29.956	30.612	31.476	32.787
9	34.033	33.764	33.863	34.482	35.715
10	35.881	36.304	36.943	37.895	39.354
11	40.136	39.793	39.848	40.509	41.879
12	41.824	42.154	42.759	43.715	45.358
13	46.388	45.981	46.010	46.679	48.235
14	48.349	48.619	49.181	50.196	51.990
15	52.467	51.979	51.963	52.651	54.331
16	54.285	54.453	54.963	55.975	57.906

DELAY VS. TEMPERATURE

DS1045–3 SIDE A FALLING EDGE SIGNAL
@ VOLTAGE = 4.75 VOLTS
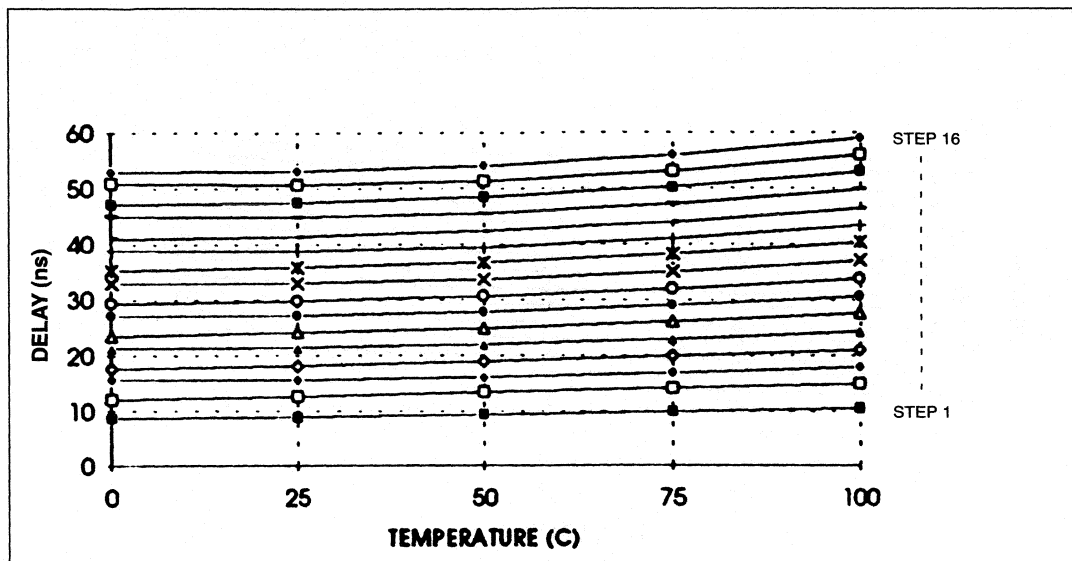
PROGRAM STEP	DELAY (ns)				
	0°C	25°C	50°C	75°C	100°C
1	8.741	8.901	9.368	9.941	10.558
2	12.066	12.637	13.352	14.115	14.922
3	15.645	15.565	16.071	16.874	17.899
4	17.586	18.14	18.924	19.864	21.012
5	21.306	21.378	21.949	22.954	24.33
6	23.517	24.096	24.944	26.075	27.575
7	27.156	27.195	27.814	29.006	30.682
8	29.296	29.806	30.697	31.968	33.814
9	33.025	33.018	33.677	35.03	37.076
10	35.355	35.81	36.736	38.201	40.341
11	38.887	38.816	39.464	40.95	43.304
12	41.014	41.387	42.317	43.916	46.385
13	44.944	44.822	45.493	47.119	49.784
14	47.186	47.499	48.439	50.169	52.938
15	50.828	50.618	51.28	53.015	55.946
16	52.914	53.122	54.036	55.895	58.934

DELAY VS. TEMPERATURE

DS1045–3 SIDE A FALLING EDGE SIGNAL
@ VOLTAGE = 5.00 VOLTS
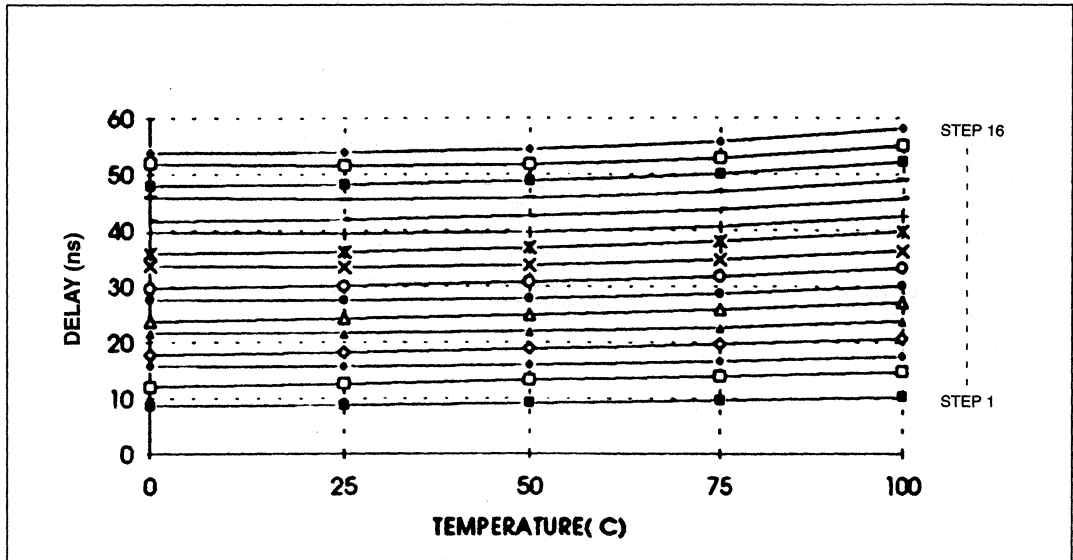
PROGRAM STEP	DELAY (ns)				
	0°C	25°C	50°C	75°C	100°C
1	8.721	8.817	9.210	9.724	10.300
2	12.031	12.565	13.245	13.967	14.722
3	15.825	15.689	15.994	16.616	17.485
4	17.687	18.185	18.874	19.691	20.699
5	21.681	21.591	21.926	22.667	23.797
6	23.736	24.242	24.943	25.881	27.152
7	27.649	27.514	27.851	28.704	30.083
8	29.638	30.065	30.766	31.783	33.299
9	33.651	33.451	33.787	34.721	36.371
10	35.812	36.206	36.887	37.994	39.729
11	39.638	39.384	39.683	40.678	42.51
12	41.601	41.904	42.547	43.717	45.677
13	45.846	45.512	45.789	46.832	48.914
14	47.913	48.146	48.756	50.001	52.144
15	51.867	51.459	51.682	52.79	55.052
16	53.801	53.903	54.460	55.745	58.141

DELAY VS. TEMPERATURE

DS1045–3 SIDE A FALLING EDGE SIGNAL
@ VOLTAGE = 5.25 VOLTS
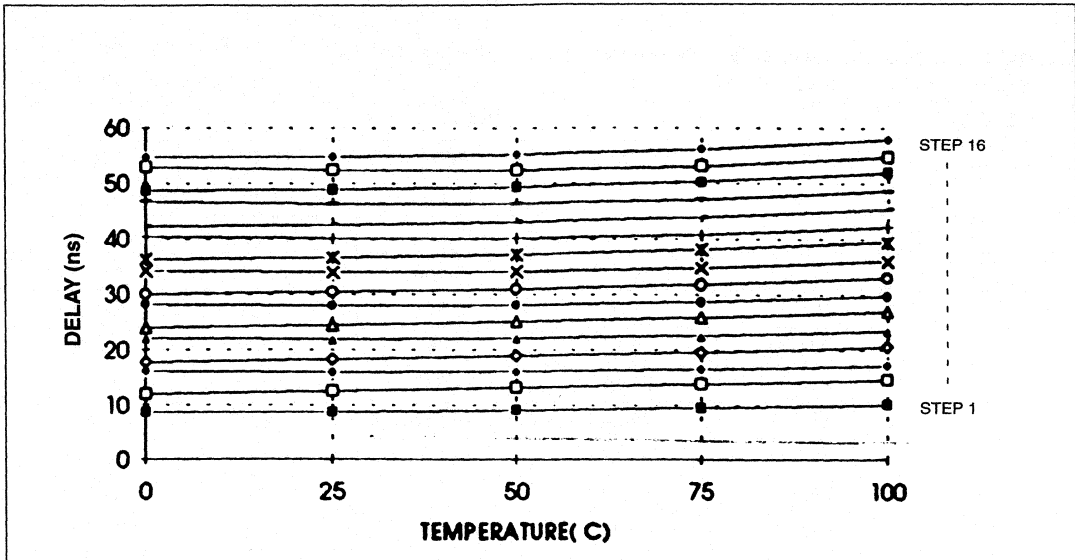
PROGRAM STEP	DELAY (ns)				
	0°C	25°C	50°C	75°C	100°C
1	8.713	8.76	9.091	9.547	10.079
2	11.987	12.505	13.173	13.856	14.563
3	16.143	15.884	16.05	16.494	17.216
4	17.767	18.269	18.901	19.621	20.528
5	22.07	21.893	22.052	22.578	23.501
6	23.963	24.438	25.058	25.85	26.94
7	28.182	27.936	28.091	28.67	29.769
8	29.992	30.392	30.986	31.807	33.08
9	34.287	33.999	34.111	34.749	36.022
10	36.31	36.652	37.211	38.093	39.511
11	40.423	40.067	40.128	40.775	42.204
12	42.224	42.477	42.985	43.875	45.467
13	46.739	46.318	46.346	47.033	48.62
14	48.669	48.854	49.31	50.238	51.973
15	52.909	52.401	52.379	53.062	54.774
16	54.702	54.754	55.151	56.081	57.944

DELAY VS. TEMPERATURE

DS1045-4

The following table indicates the performance of the DS1045-4 Programmable Delay. The plotted data is similar to the plots obtained for the DS1045-3. All steps are monotonic and show consistent variations with voltage and temperature.

DS1045-4 LOT#: 22804, DATE CODE: 4292A1
V_{CC}=4.75 VOLTS RISING EDGE

PROGRAM STEP	DELAY (ns)				
	0°C	25°C	50°C	75°C	100°C
1	7.775	7.923	8.361	8.880	9.450
2	11.547	11.545	11.933	12.640	13.55
3	15.751	16.140	16.656	17.281	18.041
4	19.229	19.264	19.632	20.328	21.270
5	23.533	23.973	24.485	25.097	25.931
6	27.207	27.283	27.637	28.332	29.353
7	31.417	31.844	32.287	32.850	33.697
8	35.129	35.202	35.479	36.124	37.175
9	39.374	39.817	40.226	40.763	41.667
10	43.061	43.172	43.449	44.118	45.229
11	47.382	47.811	48.136	48.597	49.490
12	50.877	51.000	51.213	51.823	52.970
13	55.357	55.797	56.087	56.518	57.462
14	59.099	59.227	59.417	60.014	61.238
15	63.287	63.690	63.842	64.194	65.131
16	67.211	67.337	67.442	67.943	69.168

DS1045–4 LOT#: 22804, DATE CODE: 4292A1
V_{CC}=5.0 VOLTS RISING EDGE

PROGRAM STEP	DELAY (ns)				
	0°C	25°C	50°C	75°C	100°C
1	7.622	7.784	8.169	8.642	9.184
2	11.941	11.878	12.076	12.561	13.347
3	15.584	15.953	16.451	17.022	17.714
4	19.708	19.654	19.791	20.278	21.075
5	23.363	23.816	24.304	24.858	25.596
6	27.671	27.677	27.817	28.3	29.164
7	31.23	31.699	32.142	32.649	33.39
8	35.585	35.626	35.705	36.149	37.005
9	39.171	39.691	40.115	40.599	41.361
10	43.503	43.609	43.698	44.143	45.058
11	47.187	47.737	48.083	48.486	49.233
12	51.306	51.455	51.501	51.89	52.834
13	55.136	55.727	56.058	56.437	57.202
14	59.525	59.706	59.745	60.123	61.101
15	63.073	63.659	63.884	64.182	64.928
16	67.635	67.843	67.822	68.131	69.081

V_{CC}=5.25 VOLTS RISING EDGE

PROGRAM STEP	DELAY (ns)				
	0°C	25°C	50°C	75°C	100°C
1	7.510	7.668	8.015	8.453	8.957
2	12.359	12.304	12.378	12.696	13.329
3	15.445	15.82	16.297	16.824	17.465
4	20.203	20.156	20.178	20.452	21.096
5	23.215	23.69	24.175	24.683	25.355
6	28.143	28.19	28.225	28.507	29.174
7	31.064	31.596	32.056	32.529	33.178
8	36.051	36.165	36.159	36.404	37.067
9	39.010	39.614	40.062	40.517	41.17
10	43.933	44.152	44.177	44.420	45.134
11	47.004	47.672	48.098	48.475	49.104
12	51.734	52.003	52.015	52.241	52.949
13	54.945	55.683	56.095	56.447	57.101
14	59.929	60.281	60.306	60.493	61.23
15	62.861	63.646	63.984	64.276	64.886
16	68.064	68.472	68.446	68.577	69.271

DS1045–4 LOT#: 22804, DATE CODE: 4292A1
V_{CC}=4.75 VOLTS FALLING EDGE

PROGRAM STEP	DELAY (ns)				
	0°C	25°C	50°C	75°C	100°C
1	7.996	8.349	8.944	9.58	10.236
2	11.924	11.926	12.359	13.069	13.979
3	16.287	16.674	17.213	17.859	18.638
4	19.568	19.611	20.008	20.705	21.657
5	24.109	24.565	25.077	25.705	26.576
6	27.496	27.529	27.863	28.545	29.537
7	32.092	32.543	32.986	33.566	34.432
8	35.557	35.564	35.813	36.436	37.431
9	39.881	40.374	40.818	41.422	42.346
10	43.588	43.616	43.831	44.436	45.47
11	47.817	48.307	48.66	49.203	50.163
12	51.395	51.414	51.549	52.099	53.154
13	56.12	56.603	56.917	57.417	58.412
14	59.297	59.33	59.456	59.977	61.117
15	63.943	64.415	64.64	65.068	66.065
16	67.238	67.244	67.285	67.73	68.877

V_{CC}=5.00 VOLTS FALLING EDGE

PROGRAM STEP	DELAY (ns)				
	0°C	25°C	50°C	75°C	100°C
1	7.794	8.145	8.698	9.297	9.912
2	12.311	12.233	12.432	12.945	13.732
3	16.093	16.491	16.994	17.587	18.297
4	20.018	19.957	20.127	20.622	21.442
5	23.88	24.383	24.876	25.46	26.225
6	27.944	27.904	28.026	28.49	29.329
7	31.854	32.387	32.826	33.354	34.114
8	36.017	35.986	36.029	36.429	37.259
9	39.641	40.226	40.672	41.216	42.045
10	44.04	44.051	44.068	44.455	45.315
11	47.564	48.164	48.559	49.044	49.861
12	51.824	51.857	51.821	52.172	53.021
13	55.838	56.478	56.848	57.291	58.132
14	59.709	59.782	59.757	60.099	60.973
15	63.662	64.338	64.627	64.997	65.83
16	67.644	67.735	67.628	67.891	68.808

V$_{CC}$=5.25 VOLTS FALLING EDGE

PROGRAM STEP	DELAY (ns)				
	0°C	25°C	50°C	75°C	100°C
1	7.622	7.94	8.481	9.05	9.649
2	12.668	12.594	12.683	13.016	13.659
3	15.9	16.335	16.822	17.368	18.023
4	20.457	20.417	20.446	20.749	21.411
5	23.694	24.23	24.723	25.263	25.961
6	28.397	28.392	28.393	28.664	29.308
7	31.634	32.24	32.711	33.202	33.885
8	36.456	36.496	36.443	36.661	37.305
9	39.416	40.091	40.579	41.084	41.821
10	44.454	44.558	44.502	44.717	45.384
11	47.314	48.057	48.506	48.966	49.688
12	52.26	52.417	52.335	52.496	53.141
13	55.586	56.39	56.844	57.261	57.985
14	60.119	60.348	60.284	60.437	61.135
15	63.412	64.272	64.682	65.038	65.741
16	68.048	68.321	68.216	68.314	68.996

DS1045–4 DELAY VS. VOLTAGE CHARACTERISTICS

The following chart indicates how the DS1045–4 delay changes for the positive and negative edges of step 1 and step 16 over voltage variations of 5 volts ±5%. For step 1, the falling edge starts at 7.923 ns and declines to 7.579 ns over an increasing voltage. The rising edge of step 1 follows a similar curve. For step 16, the slope is positive for increasing voltage. The exact values for each device are indicated in the following tables. Characteristics for the DS1045–3 and 5 are similar to the DS1045–4.

DS1045–4 GRAPH
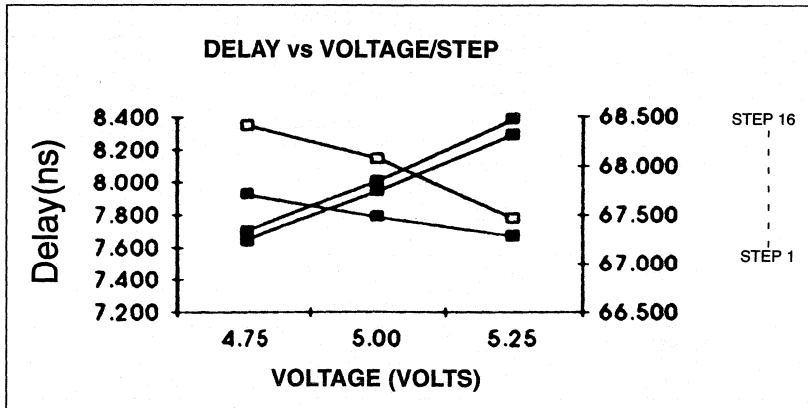
DS1045–3 STEP DELAY VARIATION VS. VOLTAGE

VOLTAGE (volts)	4.75V	5.00V	5.25V
Programming Step 1 Falling Edge	8.629	8.594	8.579
Programming Step 1 Rising Edge	8.901	8.817	8.760
Programming Step 16 Falling Edge	52.878	53.622	55.346
Programming Step 16 Rising Edge	53.122	53.903	55.675

DS1045–4 STEP DELAY VARIATION VS. VOLTAGE

VOLTAGE (volts)	4.75V	5.00V	5.25V
Programming Step 1 Falling Edge	7.923	7.784	7.579
Programming Step 1 Rising Edge	8.349	8.145	7.940
Programming Step 16 Falling Edge	67.337	67.843	68.472
Programming Step 16 Rising Edge	67.244	67.735	68.321

DS1045–5 STEP DELAY VARIATION VS. VOLTAGE

VOLTAGE (volts)	4.75V	5.00V	5.25V
Programming Step 1 Falling Edge	7.817	7.681	7.574
Programming Step 1 Rising Edge	8.213	8.005	7.815
Programming Step 16 Falling Edge	82.506	82.961	83.395
Programming Step 16 Rising Edge	82.958	83.363	83.720

OPERATING CURRENT VS. FREQUENCY
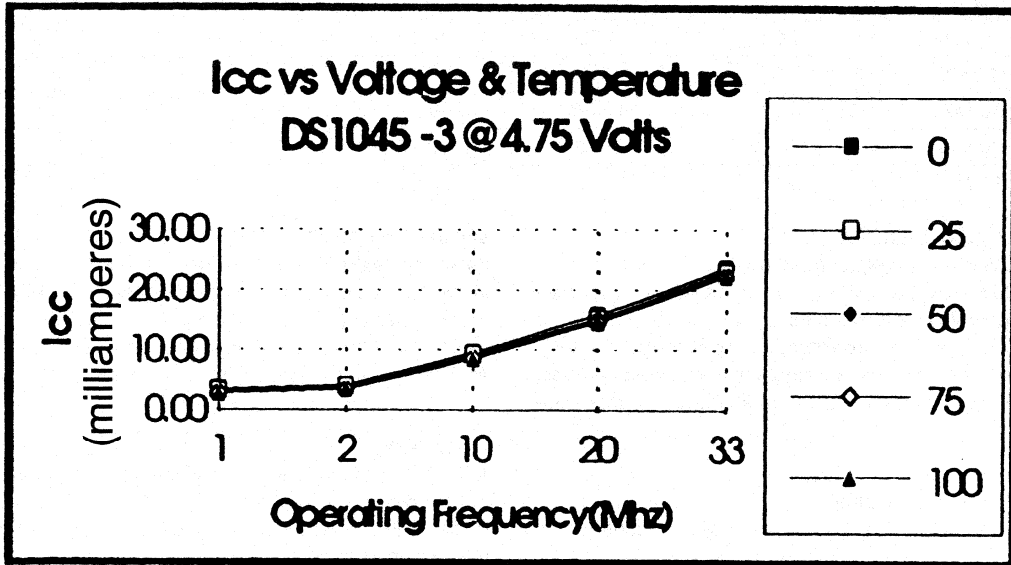
DS1045–3 @ 4.75 VOLTS

FREQUENCY	I_{CC} (MA) TEMPERATURE				
	0°C	**25°C**	**50°C**	**75°C**	**100°C**
1 MHz	3.41	3.22	3.02	2.88	2.75
2 MHz	4.11	3.89	3.68	3.54	3.41
10 MHz	9.59	9.15	8.83	8.66	8.52
20 MHz	15.83	15.18	14.86	14.75	14.69
33 MHz	23.65	23.02	22.63	22.34	22.32

DS1045–3 @ 5.00 VOLTS

FREQUENCY	I_{CC} (MA) TEMPERATURE				
	0°C	**25°C**	**50°C**	**75°C**	**100°C**
1 MHz	3.69	3.48	3.26	3.11	2.97
2 MHz	4.45	4.20	3.98	3.82	3.68
10 MHz	10.37	9.89	9.56	9.35	9.18
20 MHz	17.10	16.37	15.98	15.81	15.77
33 MHz	25.41	24.63	24.34	24.07	23.81

DS1045–3 @ 5.25 VOLTS

FREQUENCY	I_{CC} (MA) TEMPERATURE				
	0°C	25°C	50°C	75°C	100°C
1 MHz	4.02	3.78	3.53	3.37	3.22
2 MHz	4.83	4.55	4.30	4.13	3.98
10 MHz	11.21	10.68	10.31	10.01	9.89
20 MHz	18.44	17.63	17.17	16.96	16.83
33 MHz	27.26	26.29	26.03	25.80	25.53

TEST CONSIDERATIONS

The DS1045 is a high speed device and as such care should be exercised when testing. Good ground planes and power supply decoupling techniques should be used. It is also required that pin 2 be connected to V_{CC}. A precision time interval counter is required with test resolution ten times better than the required data measurement.

TEST CONDITIONS

Input source: 50 ohms maximum with measurements taken at the 1.5 volt level.
Input signal pulse of 250 ns and a period of 1 ms.
Rise and Fall times of 3 ns between 0.6 and 3.0 volts.
Load Capacitance = 15 pF.

APPLICATIONS

The ability to use the DS1045 Dual Programmable Delay Line in multiple delay line applications makes it unique in the industry. It not only allows you to reduce inventory cost by having a lower number of parts in stock, but it also affords the designer the ability to improve system performance much later in the design cycle. All of the usual applications for delay lines are applicable to this part, but with much greater flexibility.

CAUTION

A word of caution is in order when loading the input delay registers with the binary value of the desired delay function. In order to insure that the input signal integrity is maintained after the register has been loaded, the input signal must be allowed to propagate through the DS1045 for at least twice the selected delay value. For example, if you were using the DS1045–3, and selected a binary 8, then you should allow at least 32 ns times 2 or 64 ns before the input integrity is established.

DALLAS
SEMICONDUCTOR

<div align="right">

Application Note 14
Design Considerations for
All–Silicon Delay Lines

</div>

SILICON DELAY LINES VS. HYBRID L–C NETWORKS

Figure 1 shows internal views of a typical 5–tap hybrid delay line and its silicon counterpart. A hybrid is manufactured using a commercially available hex inverter DIP (e.g., 74LS04) with a small PC board placed on top to supply a ground plane. Next, several leads are bent up and over the top of the PC board. Five chip capacitors and a terminating resistor are soldered to the ground plane and a 5–tap ferrite inductor is positioned above. Note that nearly two dozen solder joints are required to electrically connect the various components. Finally, the entire assembly is placed into an oversized plastic tub and filled with a potting material.

INTERNAL VIEWS Figure 1

HYBRID

ORDINARY
VIEW BEFORE
"POTTING"

DALLAS SEMICONDUCTOR
SILICON

CUT–AWAY
VIEW
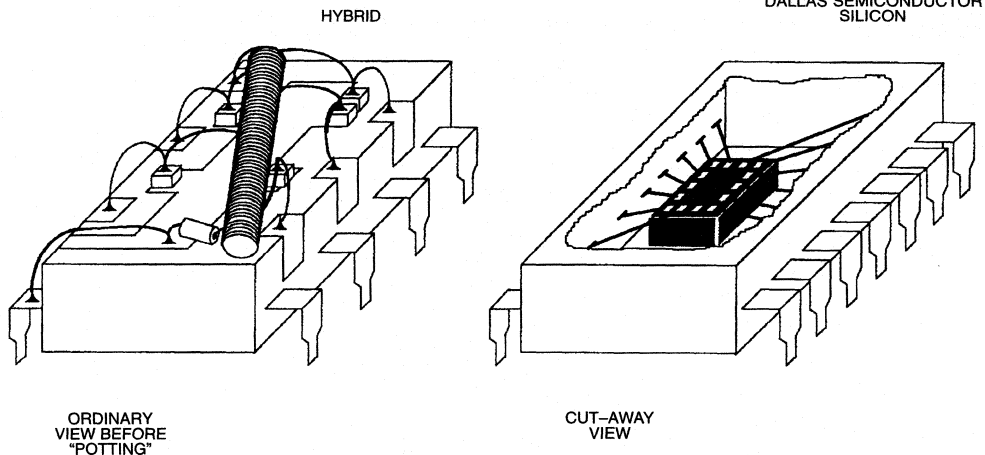
By comparison, the silicon delay line consists of a laser–optimized die bonded to a conventional lead frame molded into an auto–insertable industry standard DIP or surface mount SOIC package. The die is a low–power CMOS design fabricated on six–inch wafers by Dallas Semiconductor's Class 1 facility. Using lasers for late definition of finished wafers provides both economy and maximum flexibility; both rising and falling edges can be programmed to standard or custom delays over a wide range of values. A post–laser final passivation step protects against contamination by covering the laser fuse windows before packaging.

The basic building block of a silicon delay line consists of a ramp generator with associated logic (Figure 2). The input signal triggers a ramp generator that supplies a laser–adjusted voltage–to–time relationship (Figures 3 and 4). A comparator is used to detect the ramp reaching the reference voltage (V_{REF}); this sets or resets the output latch. The DS1013 family has three independent blocks in parallel while the DS1000, DS1005, and DS1010 families have the blocks connected in series with a single external input (Figure 5). All silicon delay lines, unlike most TTL–based hybrids, have true CMOS output levels.

072996 1/11

BASIC BUILDING BLOCK Figure 2

```
[INPUT]  →  [RAMP]  →  [J  Q / K  LOGIC]  →  [OUTPUT]
```

INPUT

RAMP

LOGIC

OUTPUT
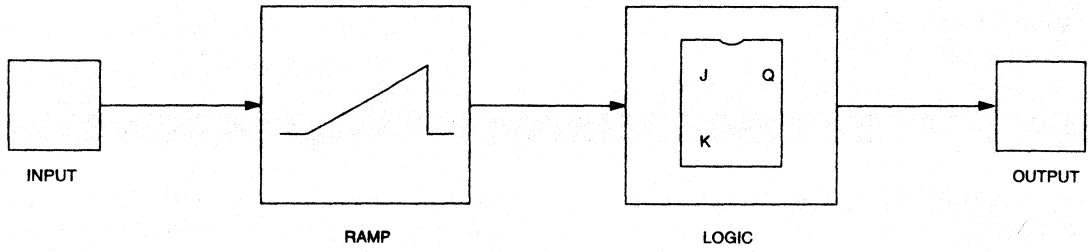
VOLTAGE TO TIME CONVERSION Figure 3

VOLTAGE ON
CAPACITORS

V_{REF}

TIME

VOLTAGE OUT
OF COMPARATOR

DELAY

TIME
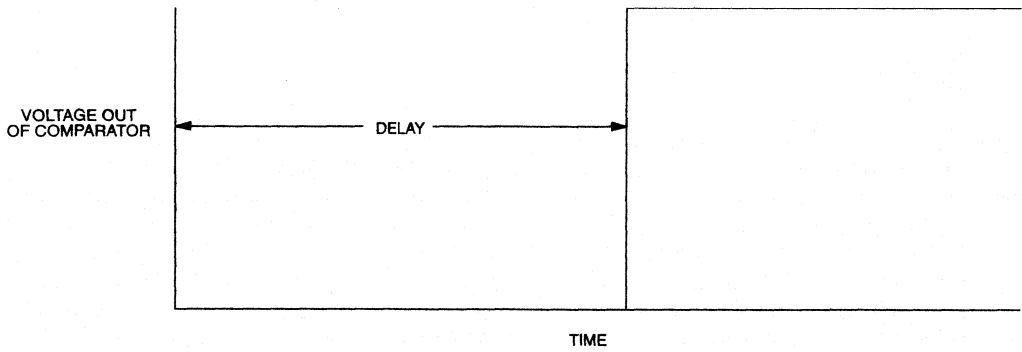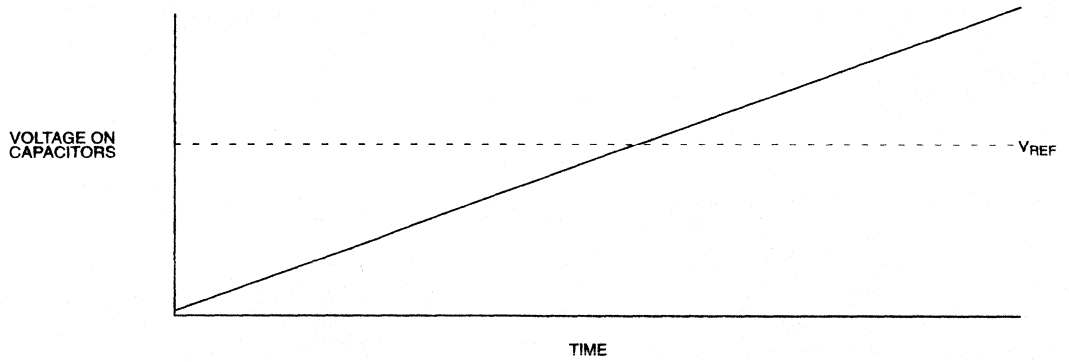
EXPANDED BASIC BLOCK Figure 4
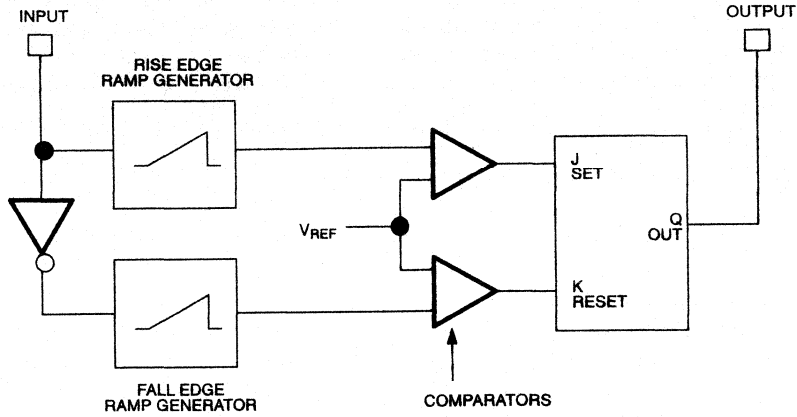
DELAY LINE FAMILIES Figure 5

Tapped Delays (e.g., DS1000 and DS1010)
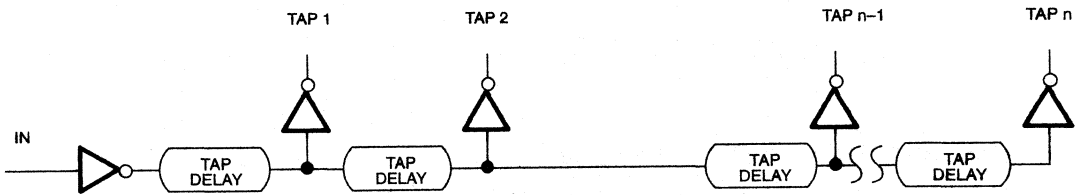
Multiple Independent Delays (e.g., DS1013, DS1007, and DS1044)

TD = TOTAL DELAY
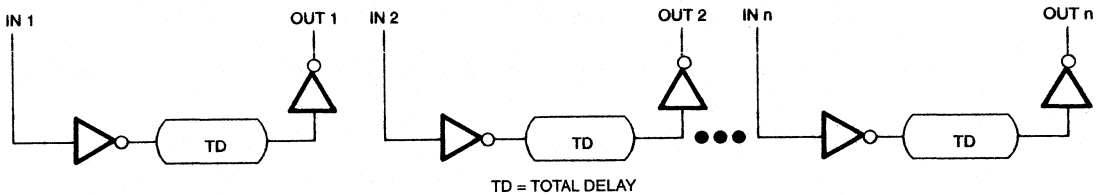
The linear ramp generator is implemented with constant current sources charging capacitors (Figure 6). By using a combination of several large current sources and capacitors and binary weighted smaller current sources and capacitors, maximum flexibility with subnanosecond adjustment is obtained on a single silicon die (Figure 7). Under the direction of a computer–controlled tester with 20 picosecond resolution, the proper slope of the ramp (Figure 8) is obtained by directing a laser to remove the unnecessary current sources and capacitors. This is accomplished by opening polysilicon fuses (Figure 9).

BASIC RAMP GENERATOR Figure 6
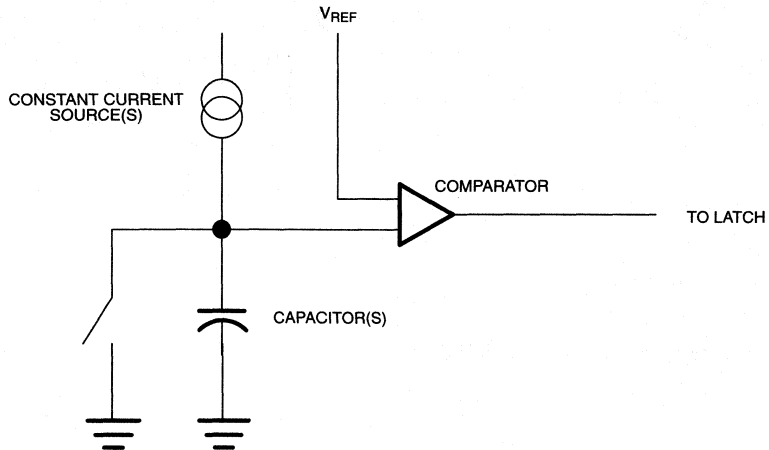
LASER PROGRAMMABLE ELEMENTS Figure 7
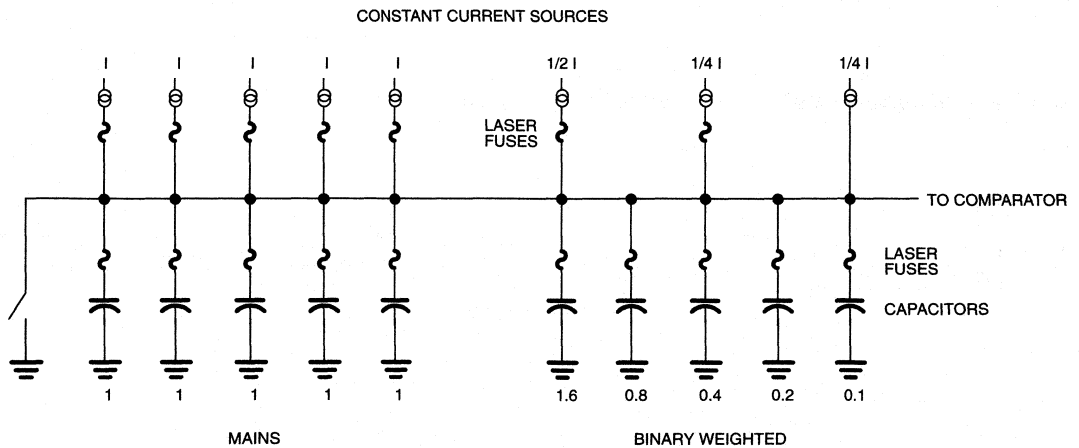
RANGE OF ADJUSTMENT Figure 8

FEW CAPACITORS
MANY CURRENT
SOURCES

MANY CAPACITORS
FEW CURRENT
SOURCES

V_{CC}

VOLTAGE
ON CAPS

V_{REF}

0
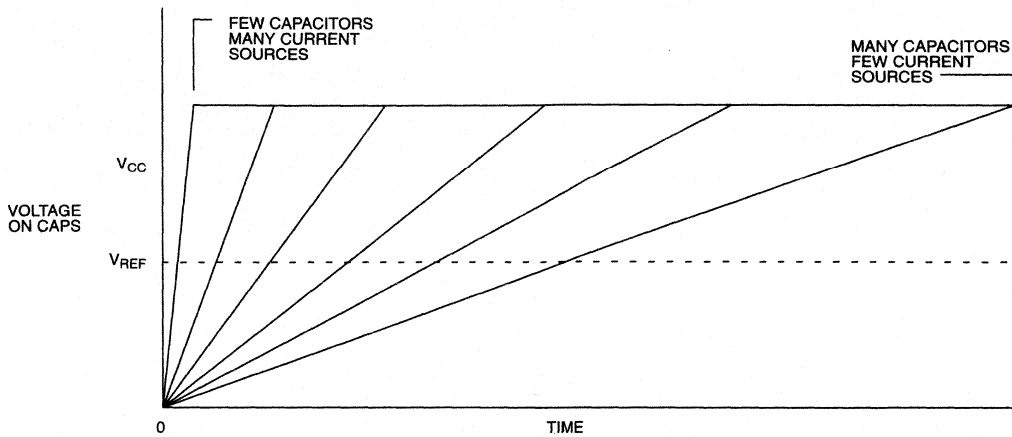
TIME

LASER BLOWING POLYSILICON FUSE Figure 9

TO
CAPACITORS

LASER
BEAM

LENS

GLASS

FROM CONSTANT
CURRENT SOURCES

137

DELAY VS. TEMPERATURE Figure 10

SILICON LOGIC DELAY (ONLY)

SILICON RAMP DELAY (ONLY)

SILICON DELAY LINE
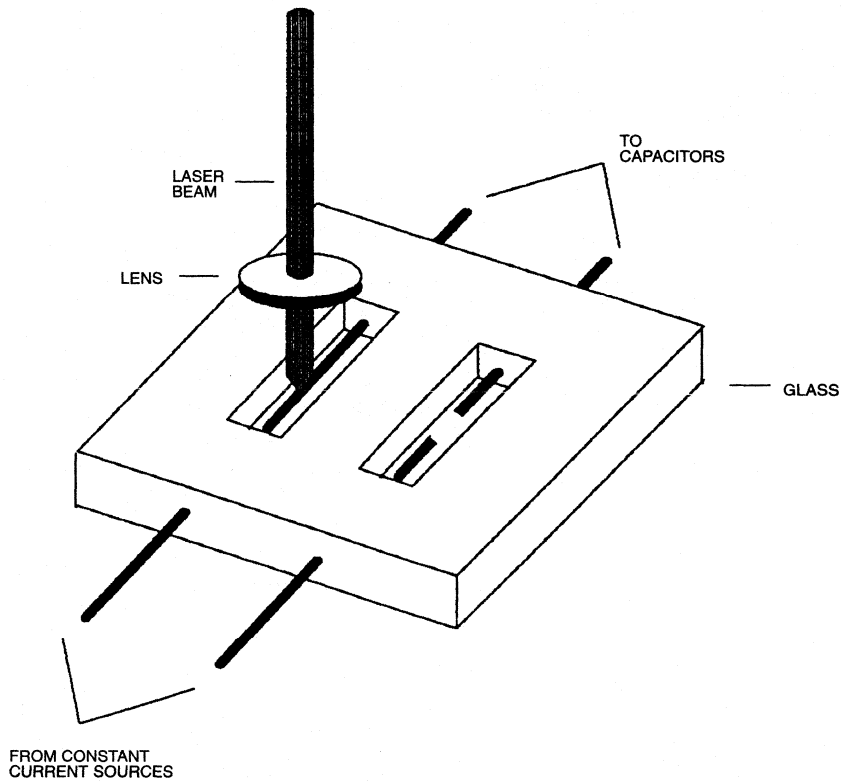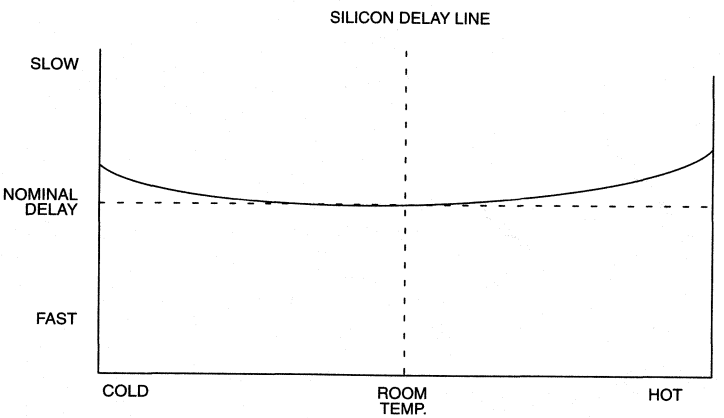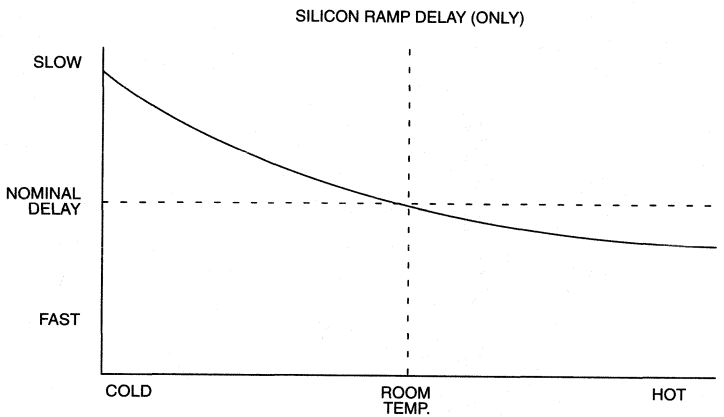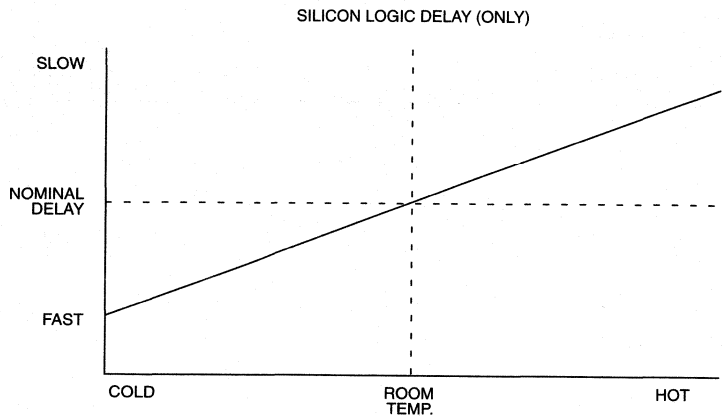
Timing on hybrid lines is determined by coil winding and/or capacitor selection or trimming. Achieving both rising and falling edge accuracy at the same time is difficult to achieve and comes at a premium price. Furthermore, timing is subject to the variations of the 74LS04 devices procured from other manufacturers.

On silicon delay lines, temperature compensation is implemented by balancing the positive temperature coefficient of the CMOS logic against the negative coefficient of the ramp generator. Figure 10 shows that the logic portion of the circuit, like hybrid delay lines in general, slows linearly with increasing temperature. Since the ramp speeds up non–linearly, the two effects tend to cancel, minimizing the effects of temperature. Because the net result resembles a parabolic shape, temperature

coefficients specified in parts per million (ppm) are inappropriate to describe the behavior of silicon delay lines. A more meaningful parameter is maximum shift from nominal, anywhere over the rated temperature range. Figure 11 compares the two technologies over temperature.

While some silicon delay line families (DS1000, DS1010) provide minimal voltage independence (4% delay change for 5% supply variation), the newer designs (DS1005 and DS1013) provide a higher degree of supply isolation ($\leq$1% delay change for 5% supply variation). The newer designs achieve supply voltage independence by employing positive ramps referenced to ground rather than negative ramps referenced to V_{CC} (Figure 12).

HYBRID VS. SILICON OVER TEMPERATURE Figure 11
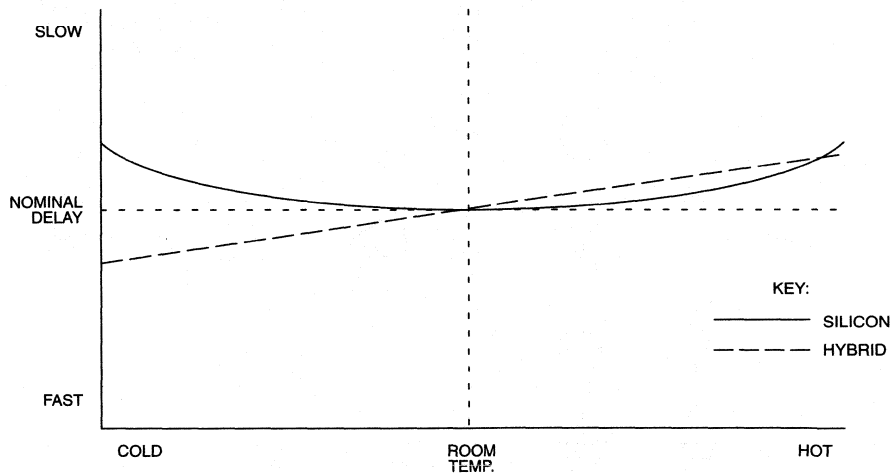
COMPARISON OF DELAY DESIGNS Figure 12

V_CC   V_REF

DS1000

TO OUTPUT

NOT V_CC INDEPENDENT

V_CC   V_REF

DS1005

TO OUTPUT

NEARLY V_CC INDEPENDENT
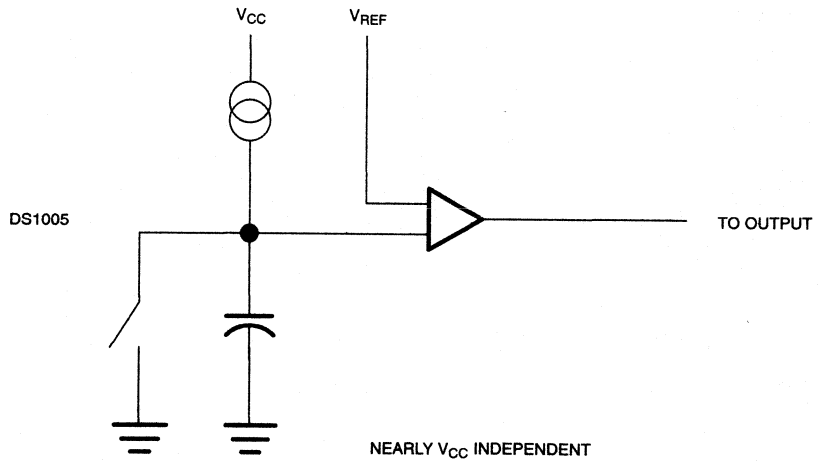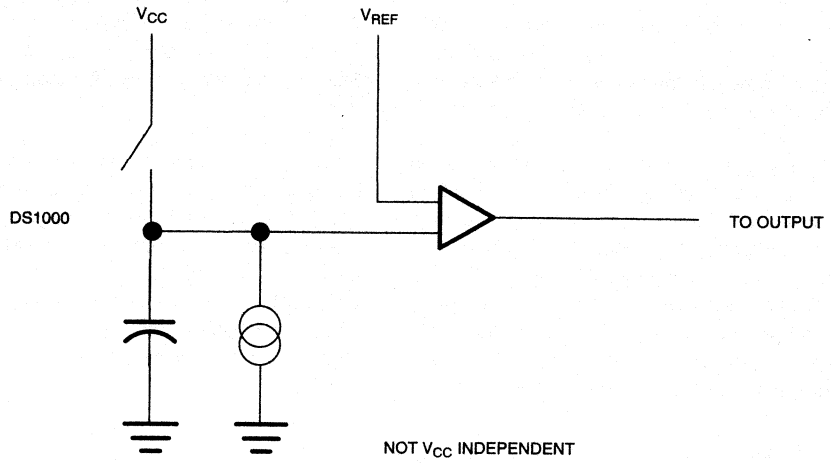
While hybrids offer little flexibility in packaging, silicon lines are available in a variety of industry standard DIP and SOIC packages (Figure 13). To maintain compatibility with existing designs based on hybrids having missing pins, a clipped lead version is offered. Finally, for surface mount applications, two solutions are available: 300 mil DIPs with the leads trimmed and "gull winged," and industry standard SOIC packages.

Table 1 summarizes some of the disadvantages of the hybrid design and some of the advantages of the silicon solution.

TABLE 1

Disadvantages of Hybrids:
• Unreliable solder joints
• Difficult to control falling edge accuracy
• TTL output levels
• Large, non–standard package dimensions

Advantages of Silicon
• Reliable all–silicon design
• Accurate rise and fall edges
• Easily customized
• CMOS output levels
• No inductors
• Industry standard DIP and SOIC packaging
• Standard IC handling including reflow soldering

PACKAGING OPTIONS Figure 13

DUAL–IN–LINE (DIP)
DIP packages are available in pin counts of 8, 14 and 16 pins. Three lead–forming options are available:

Straight lead. This is the conventional package and is used for through–hole mounting in PCBs.

Gullwing. The leads are formed to sit flat on the surface of a PCB for surface mount applications.

Sheared NC. All "no connect" (NC) leads are sheared off the package. This package is commonly used for hybrid replacement applications.

NOTE: The use of gullwing or sheared NC packages is not encouraged for new designs; however, these packages will continue to be made available for existing designs.

SMALL OUTLINE (SOIC)
Small outline surface mount packages are available in pin counts of 8, 14 and 16 pins. Two package widths are also available: 150 mil and 300 mil.

PACKAGE AVAILABILITY/LETTER DESIGNATIONS					
	DIP			SOIC	
# PINS	STD	GULLWING	SHEARED	150 mil	300 mil
8	M	H		Z	
14	none*	G	K	R	
16					S

* Default package, no letter designator required.
  Tape and reel packaging is also available; contact the factory for more information

QUESTIONS AND ANSWERS

QUESTION: How do silicon timed delays differ from hybrid devices?

ANSWER: A typical hybrid consists of a hex inverter DIP, a PC board acting as a ground plane plus chip capacitors, a terminating resistor and a multiple tap ferrite inductor. Timing is determined by coil winding and/or capacitor selection or trimming. The Dallas Semiconductor Silicon Timed Circuit (STC) design uses a laser–optimized die bonded to a conventional lead frame molded into an auto–insertable DIP or SOIC package.

QUESTION: What are the advantages of silicon timed circuits over hybrid devices?

ANSWER: STC (Silicon Timed Circuits) offer advantages in design, packaging and manufacture over hybrids. Specifically, STCs offer the increased reliability of silicon and greater accuracy on both rise and fall edges. Unlike TTL devices, STCs are CMOS devices and offer true output levels. Using standard DIP and SOIC packaging, STCs are well suited to standard IC handling including reflow soldering.

QUESTION: Whats the basic operation of a silicon delay line device?

ANSWER: The basic building block of the silicon delay line is a ramp generator and associated logic. The input signal triggers a ramp generator that supplies a laser–adjusted voltage–to–time relationship.

QUESTION: Is there a way to correlate the performance of silicon time delays against hybrids?

ANSWER: Following this procedure will provide a correlation. At a fast (3 ns) rise/fall input pulses with 0/3 volt logic levels, low inductance decoupling techniques using 0.01 to 0.1 μF capacitors, low capacitance measurement probes placed as close as possible to the package, and relaxed timing (500 ns pulse width, μ1 s period) will aid in timing correlation when measured at 1.5 volt levels.

QUESTION: How does laser–adjustment time the delay?

ANSWER: Under the direction of a computer–controlled tester with 20 picosecond resolution, the proper slope of the ramp is obtained by directing a laser to open polysilicon fuses removing unnecessary current sources and capacitors.

QUESTION: Can I get any delay time I need?

ANSWER: Yes, silicon delay lines can be customized to meet your design timing requirements. We can even supply sample quantities in ceramic packages for evaluation.

QUESTION: Do STCs need to compensate for temperature like hybrids?

ANSWER: On silicon delay lines, temperature compensation is implemented by balancing the positive temperature coefficient of the CMOS logic against the negative coefficient of the ramp generator. The logic portion of the circuit speeds up linearly with increasing temperature while the ramp speeds up non–linearly. These two effects tend to cancel, minimizing the effects of temperature. Rather than measuring the coefficients in parts per million, silicon delay lines are meaningfully measured as the maximum shift from nominal anywhere along the rated temperature range.

QUESTION: Do silicon timed delays require decoupling?

ANSWER: STCs contain noise sensitive voltage detection circuits and fast rise time output circuits so decoupling is indicated. A 0.01 to 0.1 μF low inductance capacitor should be used in close proximity to the delay line to assure the highest performance.

QUESTION: Do silicon delay lines also function as glitch discriminators?

ANSWER: Yes, if a pulse or glitch is less than about 60% of the first stage delay of a 5– or 10–tap serial device (DS1000, DS1005 or DS 1010) or less than about 60% of any tap on a 3–in–1 parallel device (DS1013) there will be no output.

QUESTION: Can a silicon delay line be used to time–shift a square wave?

ANSWER: Yes, it s a good solution. Just be certain the period specification is met.

QUESTION: What temperature parameters do silicon timed delays meet?

ANSWER: STCs function over the full military (–55°C to +125°C) temperature range, yet have been optimized for the commercial (0°C to +70°C) range. One product family the DS1000–Ind is available in the industrial temperature range. Other parts may be specified for the industrial temperature range on a custom basis.

QUESTION: Can I add or daisy chain several devices to achieve a longer delay?

ANSWER: Daisy chaining silicon delay lines presents a problem that is increased with the number of packages in a chain. A voltage chopped stabilization circuit in the design causes a slight time jitter (well within specifications and almost transparent to the circuit) on the output. When packages are daisy chained, the time jitter is cumulative and degrades the accuracy of downline stages. If four to six packages are chained in a series, the results are generally unacceptable.

QUESTION: What about daisy chaining the stages of a parallel STC?

ANSWER: With the exception of the fast taps on the DS1007, some jitter will be added with each stage when added in a series. This will not be a problem in most cases. Users report that daisy chaining this way is a convenient method of implementing non standard series delays.

THERMAL AND BATTERY MANAGEMENT

DALLAS
SEMICONDUCTOR

Application Note 54
Increasing Charging Current and
Voltage for the DS1633 Battery Recharger

INTRODUCTION

The DS1633 Battery Recharger is designed to be a complete battery charging system for standard charge or trickle charge applications. The device is flexible enough to be used with a variety of battery chemistries and cell capacities. It provides timer termination of standard charge and automatically shifts into trickle charge. Battery voltage may be monitored and charging terminated if it exceeds a preset maximum as a safety feature. The output load line may be specified as the usual constant current recharge with a voltage limit or it may be configured to approximate any practical load line. All parameters, such as power supply range, charge current load line, trickle charge rate, and timer setting, are programmed into nonvolatile memory using the battery pin as a one–wire communication port. This functionality is provided in a small, 3–pin TO–220 package.

The DS1633's functionality is a result of its unique architecture. The device monitors the battery voltage and adjusts the values of the output impedance (R_{TH}) and the open circuit voltage (V_{OC}) it presents to the battery. These values may be adjusted at 32 user–definable points (breakpoints) that occur roughly every 37 mV. This allows the device to approximate a wide range of charging lines and is not limited to constant current or even monotonically decreasing functions.

Using only the DS1633, supply voltages of either 5V or 6V may be used, and battery voltages as high as 4.7V (3.7V for 5V supplies) are allowed. The DS1633 is available in preprogrammed versions, with charging currents up to 100 mA available. These provide for a simple component solution to several battery charging tasks, which users may stock and use when needed.

For typical NiCd batteries, the 4.7V limit on battery voltage limits the battery stack to three cells. Many battery packs today use five cells. In addition, with higher capacity battery packs available, there are situations which require charging currents higher than 100 mA. This application note examines some circuit alternatives that allow the DS1633 to be used with battery packs that have more than three cells, or in applications which require higher charging currents. Other circuit features are presented which show how easily the DS1633 can be made into a full–featured charging system.

DEFINITIONS

The DS1633 has only three pins, as shown in Figure 1. The voltages which will be referred to throughout this application note are defined in Figure 1 and in the table below.

VOLTAGE	DESCRIPTION	LIMITS
V_{CC}	DS1633 Supply Voltage	5V mode: 4.75V<V_{CC}<6.5V referred to V_{GND} 6V mode: 5.7V<V_{CC}<6.5V referred to V_{GND}
V_{BAT}	Battery Voltage	5V mode: 0<V_{BAT}<3.7V referred to V_{GND} 6V mode: 0<V_{BAT}<4.7V referred to V_{GND}
V_{GND}	Ground. All DS1633 voltages are referred to this potential.	None

DS1633 VOLTAGE DEFINITIONS Figure 1
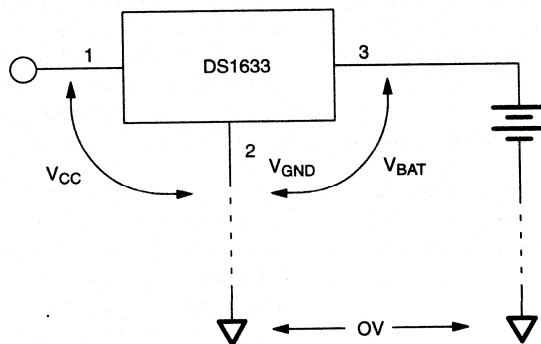
ADDING A CHARGE INDICATION LED Figure 2
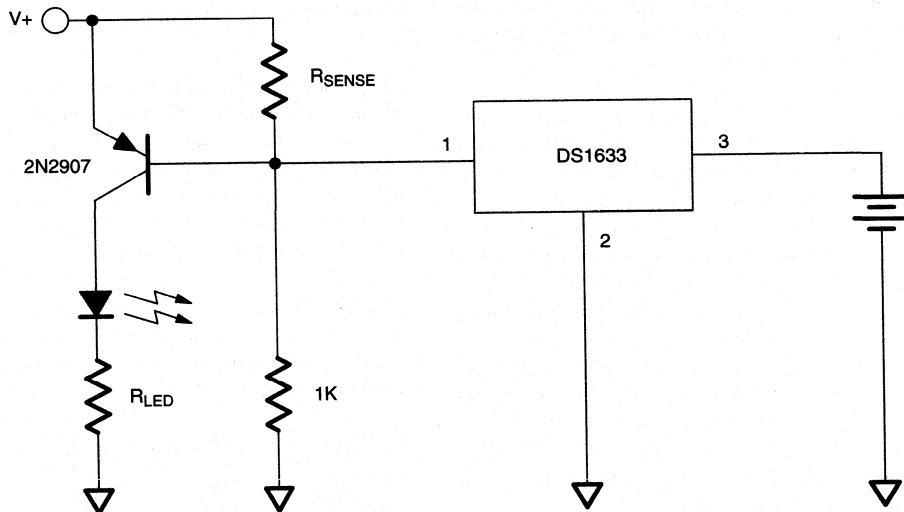
$$R_{SENSE} = \frac{0.7V}{I_{CHG}}$$

ADDING AN LED FOR CHARGING INDICATION

Typically, battery chargers have some visual indication that standard charging is taking place and may also indicate trickle charge mode. This feature is easy to add to a DS1633, using the circuit shown in Figure 2.

The DS1633 draws only 1 mA of quiescent current itself. The sense resistor, R_{SENSE}, is selected so that when the charging current begins to be drawn from the V+ supply, 0.7V is dropped across it. This forward–biases the transistor, allowing current to flow out of the transistor's collector to drive the LED. The LED current is limited by R_{LED}.

The values for R_{SENSE} and R_{LED} depend upon the charger application. To turn on the transistor, the voltage drop across R_{SENSE} must be greater than 0.7V. The value for R_{SENSE} may be found by where I_{CHG} is the magnitude of the charging current. The 1K resistor to ground is optional; it may be needed to set a proper operating point for the transistor to switch properly when the DS1633 is in trickle mode.

It is important to note that the voltage drop across R_{SENSE} must be accounted for in the overall charger design; the DS1633's V_{CC} limits must be observed. For example, with a DS1633 in 6V mode, and a charge current of 50 mA, R_{SENSE} would be 14 ohms; a 15 ohm resistor would do fine as the closest available 10% resistor value. This means that 0.75V would be dropped across R_{SENSE}; and so V+ must be in the range $6.45V < V+ < 6.5V$, so that the DS1633's V_{CC} limits are met. The upper limit is not changed by the addition of the resistor, since when the DS1633 goes into trickle mode, the output current will at times drop to zero, so only the quiescent current of the DS1633 is flowing. This would cause the full input voltage to be seen at pin 1. A regulated V+ is therefore necessary.

The value for R_{LED} depends upon the type of LED used, and its desired brightness. Typically, LEDs drop approximately 2V across them, and require 20 mA of drive current for full brightness. Using this information, and assuming that the V_{CESAT} of the 2N2907 is negligible, R_{LED} may be found by

$$R_{LED} = \frac{(V+) - 2V}{20 \text{ mA}}$$

For example, using a V+ of 7V as found above, R_{LED} would be 250 ohms for full brightness; a 510 ohm resistor could be used for a slightly dimmer LED. Some LEDs can be driven with as little as 1 or 2 mA; use the lowest current possible to drive the LED in order to reduce the current drive requirements of the power supply for the charger.

One of the interesting aspects of this circuit is its operation in trickle mode. In standard charge mode, the LED will be on all the time. When the DS1633 moves into trickle mode, however, it does so by pulsing the standard charge current at some specified duty cycle. This means that trickle mode will be indicated by the LED blinking at a slow rate. When full charge is achieved, (i.e. the battery voltage limits have been met), current is turned off completely, and the LED will be extinguished. Thus, this indicator can show when the DS1633 is in standard charge mode, in trickle charge mode, and when charging is complete.

INCREASING OUTPUT CURRENT

While the DS1633 is capable of supplying up to 100 mA of charging current, there are situations where more charging current is desired. For example, an 800 mAh battery pack would require almost 13 hours of charging at 100 mA. If the charging current could be increased to 160 mA, this battery could be fully charged in 8 hours.

Preprogrammed DS1633's come in several different charge currents, all with 8 hour timer cutoff. The products that are available preprogrammed, requiring no further programming by the user, are shown in Table 1.

PRODUCT SELECTION GUIDE Table 1

PART NUMBER	I_{MAX} (mA)	V_{MAX} (V)	TIMER (Hours)
DS1633–A	100	4.65	8
DS1633–B	80	4.65	8
DS1633–C	60	4.65	8
DS1633–D	40	4.65	8
DS1633–E	20	4.65	8

Since the DS1633 is essentially a voltage source with an adjustable resistor, it is capable only of sourcing current; it cannot sink current. This fact makes it possible to place any number of DS1633's in parallel, with no need for any external components. It is generally wise, however, to keep the timer lengths of the paralleled parts the same to avoid one going into trickle much before its counterpart does.

Using this approach, the 800 mAh battery may be charged using a charger as shown in Figure 3.

PARALLEL DS1633's Figure 3
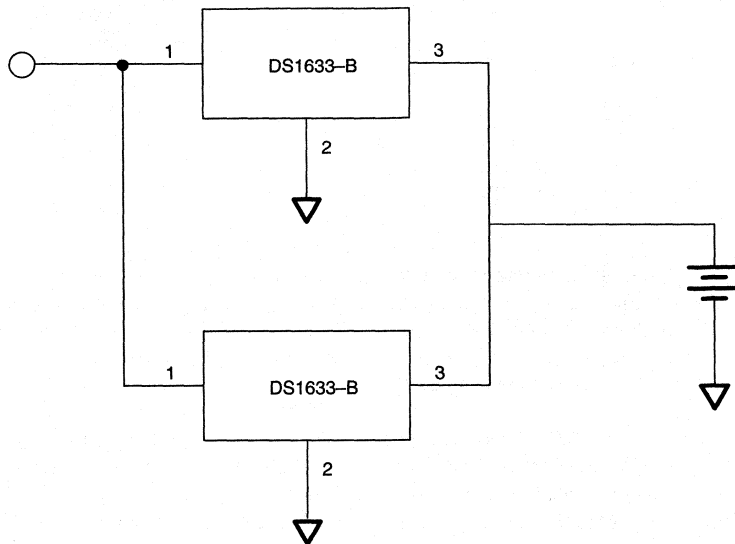
The charge indication circuit of Figure 2 may be used with this circuit to provide an indication of charging status.

INCREASING BATTERY VOLTAGE RANGE

The battery voltage limits on the DS1633 are suitable for NiCd battery packs with up to three cells. With five cell battery packs increasing in usage, a method of charging these battery packs using the DS1633 is desirable.

Since the limit on the battery voltage is 4.7V referred to V_{GND}, it is possible to raise the potential of V_{GND} to keep V_{BAT} within limits. This method allows the DS1633 to charge any number of NiCd cells, with certain constraints.

The circuit of Figure 4 provides the DS1633 with the ability to charge a five cell NiCd battery pack. Typically, a NiCd cell will be considered fully discharged when the cell voltage goes to 0.9V. This means the total voltage across the battery pack at a minimum will be 4.5V. By offsetting the DS1633 GND pin to 4.3V using the zener diode shown, the V_{BAT} referred to ground will range from 0.2V (when the battery potential is 4.5V) to 3.7V (when the battery potential is 8V, in a fully charged condition).

Note that by offsetting the GND pin, the V+ voltage required for the DS1633 is now between 10V and 11.5V (these levels keep V_{CC} between 5.7V and 6.5V referred to V_{GND}).

INCREASING BATTERY VOLTAGE RANGE Figure 4
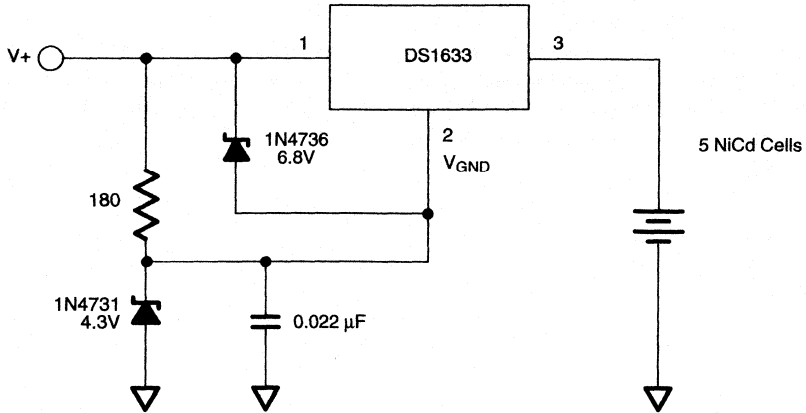
It is possible that the V+ potential may rise faster than the V_{GND} potential from the zener upon power up. If this should happen, the DS1633 may be damaged, or may be placed in a test/programming mode. The 6.8V zener from V_{CC} to V_{GND} assures that this will never happen.

The circuit of Figure 4 will work reliably as long as it is never connected to a battery pack which is discharged below 4.5V. Should the V_{BAT} potential ever fall below that of V_{GND}, substrate diodes will be forward biased and large currents will flow; these may damage the DS1633. Therefore, it is advised that this circuit be used with caution, and only if the battery potentials are well known and controlled.

A circuit which allows battery voltages to go below 4.5V and still will charge batteries with more than three cells is shown in Figure 5. This circuit has several features which will be discussed below.

The first feature of this circuit consists of Q1, R2, and D1; this simple pass voltage regulator supplies the DS1633 with V_{CC} of 6.2V referred to V_{GND} at all times. This serves two purposes, the first of which is to allow the V+ supply for the charger to be a convenient value, such as +12V, rather than requiring a precise 10V or 11V, as with the circuit in Figure 4. The second purpose will become clear in a moment.

BATTERY CHARGER FOR ONE TO FIVE CELLS Figure 5
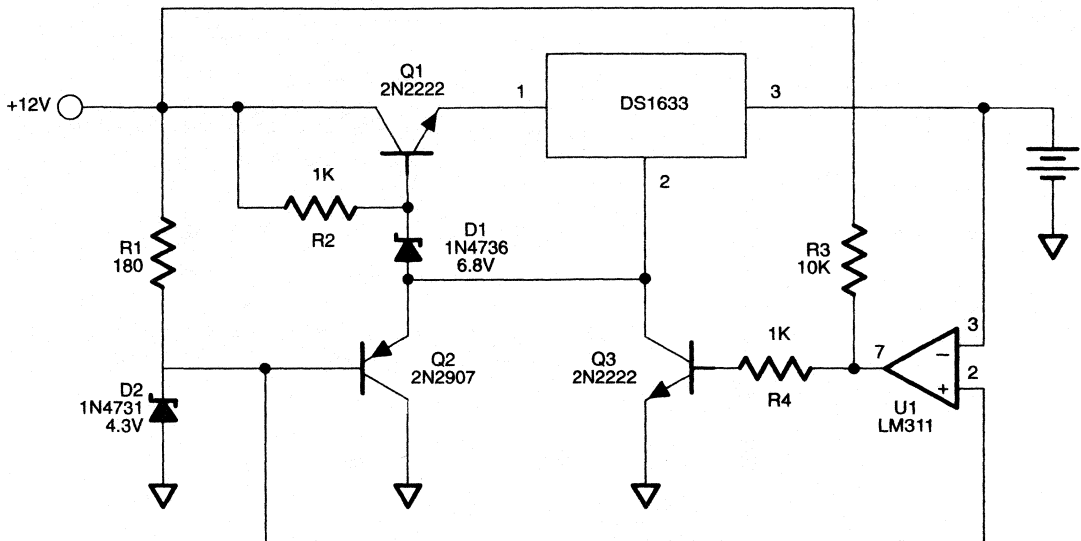
R1 and D2 set up the offset ground reference voltage, as was done in the circuit of Figure 4. The reference voltage is connected to V_{GND} through Q2, which is configured as an emitter follower. Note that this will place V_{GND} at approximately 0.7V above the zener voltage of D2.

The reference voltage is also fed to a comparator, consisting of U1 and R3. The comparator compares the battery voltage with the reference supplied by R1 and D2.

If the battery voltage is below 4.3V, the comparator's output will go high. This will turn on Q3, which will effectively pull V_{GND} down to within a few millivolts of the ground potential. This is good enough to make the DS1633 operate at battery voltages between 0V and 4.3V.

If the battery voltage is above 4.3V, Q3 is turned off, and V_{GND} goes to the reference voltage as supplied through Q2. The change in ground reference voltage is automatic, and will occur during charging if necessary.

It is this ability to dynamically shift the V_{GND} potential which requires the regulator circuit initially discussed. This regulator "floats" with the V_{GND} voltage, and will maintain the proper V_{CC} voltage for the charger for either V_{GND} potential available.

A full–featured charger which provides this automatic battery voltage sensing and charge indication is shown in Figure 6.

FULL FEATURED BATTERY CHARGER Figure 6
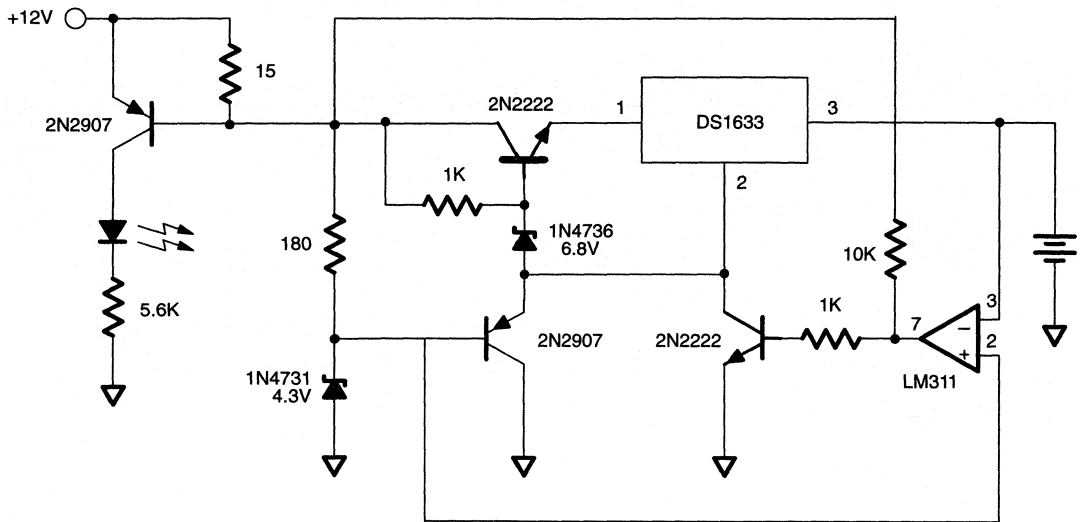
Another method of charging batteries with more than three cells is an alternative to the method presented above. Instead of offsetting the DS1633's V_{GND}, it is possible to change the battery's reference point.

This is accomplished by using a bipolar supply to drive the DS1633. The positive side drives the DS1633, while the negative supply is used as the battery reference, as shown in Figure 7.

As with the circuit of Figure 4, this approach works well provided that the battery voltage is greater than 5V at all times. Other negative potentials may be used to adjust the circuit to a specific application.

USING A BIPOLAR SUPPLY TO INCREASE BATTERY VOLTAGE RANGE Figure 7
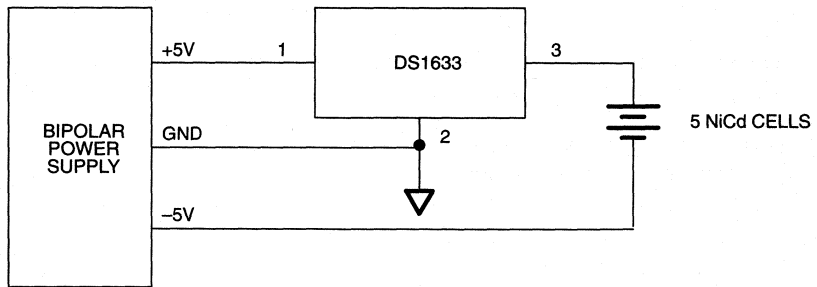
DALLAS
SEMICONDUCTOR

INTRODUCTION

The DS1620 Digital Thermometer and Thermostat provides 9–bit temperature readings which indicate the temperature of the device. With three thermal alarm outputs, the DS1620 can also act as a thermostat. T_{HIGH} is driven high if the DS1620's temperature exceeds a user defined temperature TH. T_{LOW} is driven high if the DS1620's temperature falls below a user defined temperature TL. T_{COM} is driven high when the temperature exceeds TH, and stays high until the temperature falls below that of TL. This is shown in Figure 1.

User defined temperature settings are stored in non–volatile memory, so parts may be programmed prior to insertion in a system, as well as used in stand–alone applications without a CPU. Temperature settings, and temperature readings are all communicated to/from the DS1620 over a simple 3–wire interface.

TEMPERATURE CONTROL WITH THE DS1620

The thermostat outputs of the DS1620 allow it to directly control heating and cooling devices. For example, the T_{HIGH} output could be used with an external latch to turn on a fan as soon as the measured temperature exceed the TH threshold, as shown in Figure 2. This is one possible use of the thermostat outputs, but it isn't the most efficient way to control a fan, since once the fan turned on, there is no way to turn it off.

Conversely, the T_{LOW} output could be used in a similar fashion to turn on a heating device, as shown in Figure

3. Once again, however, there is no way to turn the heater off once the desired temperature range is reached. Using a self–regulating heater prevents overheating, while the DS1620 assures that it initially turns on only if the temperature is cold enough to warrant it.

It would be possible to use the T_{HIGH} and T_{LOW} outputs to set and reset an external latch, so that once the fan came on, it would stay on until the lower temperature threshold is reached, as shown in Figure 4. This same principle could be used with the heating example, provided T_{LOW} set the latch and T_{HIGH} reset it.

The T_{COM} output is provided on the DS1620 to allow a fan to be directly controlled from the DS1620, without the need for external latches. An example of this is shown in Figure 5. This allows the fan to turn on only when the temperature exceeds the TH trip point, and will stay on until the temperature drops below the TL trip point. After that, the fan is turned off. This is ideal for systems where power is important, as the fan is used only when necessary and only for as long as needed.

The previous examples show the DS1620 being used in a stand–alone mode, without a connection to a microprocessor. In order to use it this way, the temperature limits must first be programmed into the DS1620; since the settings are nonvolatile, the parts can be programmed prior to insertion in the end system application. While these examples show the DS1620 in stand–alone mode, nothing prevents them from being used in an application where the DS1620 is interfaced to a microprocessor.

072996 1/4

THERMOSTAT OUTPUT OPERATION Figure 1
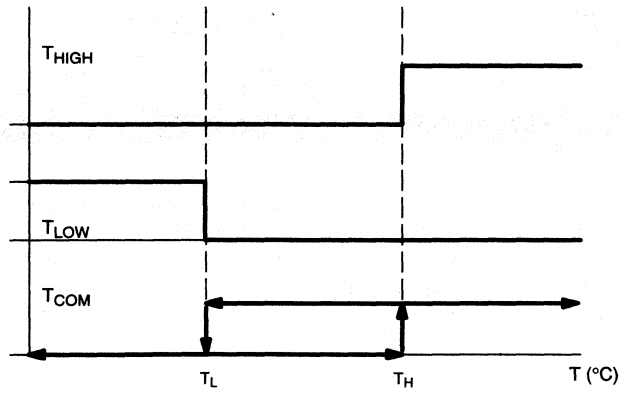
USING T_{HIGH} TO DRIVE A FAN Figure 2
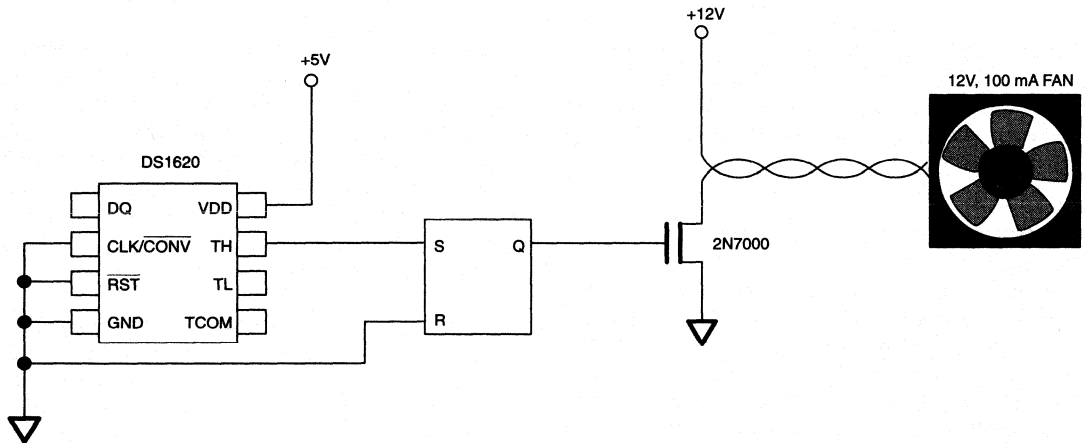
USING T$_{LOW}$ TO TURN ON A HEATING DEVICE Figure 3
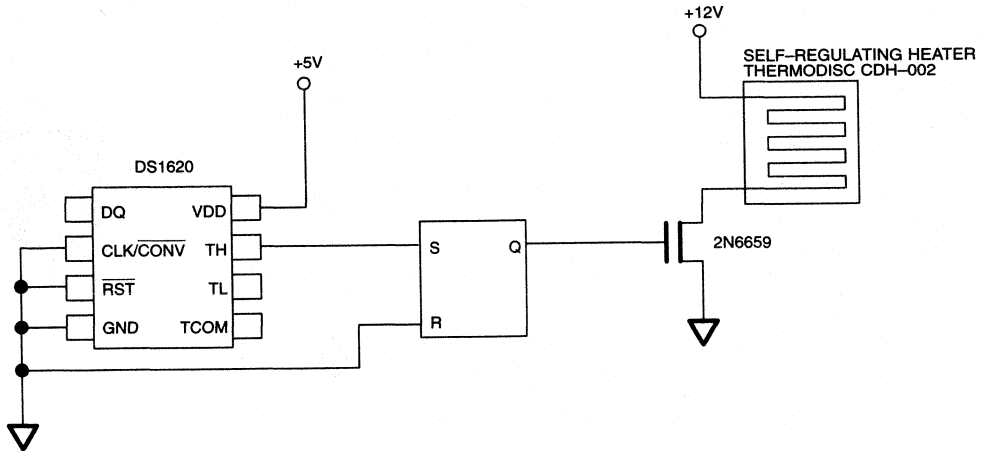
FAN CONTROL WITH EXTERNAL LATCH Figure 4
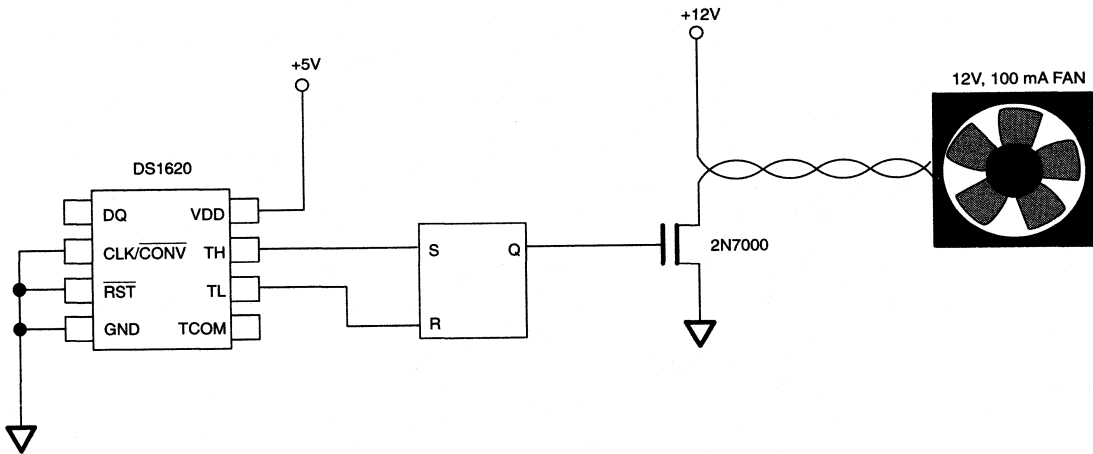
USING THE T$_{COM}$ OUTPUT TO DRIVE A FAN Figure 5
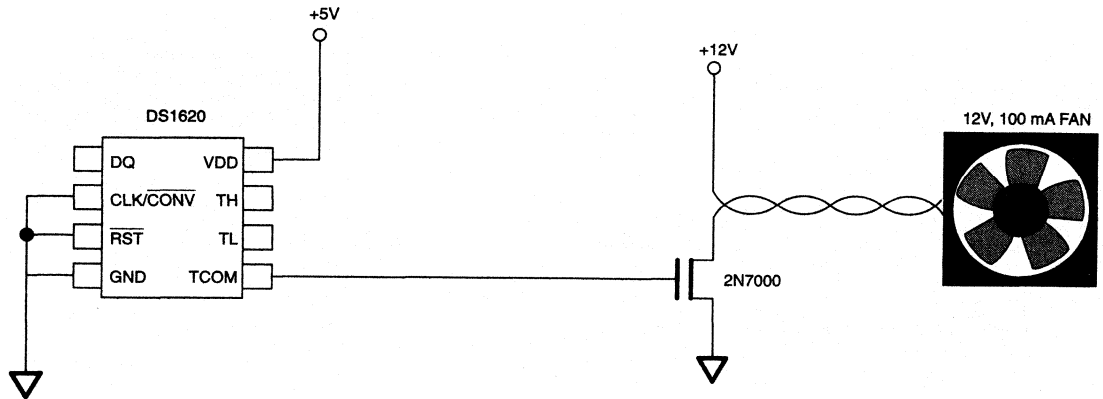
RELATED PRODUCTS:
DS1620, DS1621, DS1623, DS1625, DS1821

DALLAS
SEMICONDUCTOR

<div align="right">

Application Note 68
Increasing Temperature
Resolution on the DS1620

</div>

INTRODUCTION

The DS1620 Digital Thermometer and Thermostat provides 9 bit temperature readings which indicate the temperature of the device. This temperature is expressed in 0.5°C increments, providing 0.5°C resolution. In the 0°C to +70°C range, the part is accurate to 0.5°C as well. In the −40°C to 0°C range, as well as from +70°C to +85°C, the DS1620's accuracy is within 1°C. Extending the temperature range even further, from −55°C to −40°C and from +85°C to +125°C, the device is accurate to within 2°C.

These accuracies are very good as far as temperature sensors go, but there are situations where the 0.5°C resolution is not adequate. In proportional control systems where the absolute temperature is not as critical as the trend in temperature, greater resolution may be required.

Using some undocumented test modes for the DS1620 allows the user to provide 0.1°C resolution with the DS1620 and some overhead software. While this does not increase the accuracy of the device, it may provide better control of some systems by having finer resolution of temperature.

OPERATION MEASURING TEMPERATURE

The DS1620 measures temperatures through the use of an on–board proprietary temperature measurement technique. A block diagram of the temperature measurement circuitry is shown in Figure 1.

The DS1620 measures temperature by counting the number of clock cycles that an oscillator with a low temperature coefficient goes through during a gate period determined by a high temperature coefficient oscillator. The counter is preset with a base count that corresponds to −55°C. If the counter reaches zero before the gate period is over, the temperature register, which is also preset to the −55°C value, is incremented, indicating that the temperature is higher than −55°C.

At the same time, the counter is then preset with a value determined by the slope accumulator circuitry. This circuitry is needed to compensate for the parabolic behavior of the oscillators over temperature. The counter is then clocked again until it reaches zero. If the gate period is still not finished, then this process repeats.

The slope accumulator is used to compensate for the nonlinear behavior of the oscillators over temperature, yielding a high resolution temperature measurement. This is done by changing the number of counts necessary for the counter to go through for each incremental degree in temperature. To obtain the desired resolution, therefore, both the value of the counter and the number of counts per degree C (the value of the slope accumulator) at a given temperature must be known.

Internally, this calculation is done inside the DS1620 to provide 0.5°C resolution. Note that temperature is represented in the DS1620 in terms of a $1/2$°C LSB, yielding the following 9–bit format:

1		1	1	0	0	1	1	1	0
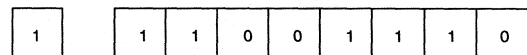
= −25°C

Higher resolutions may be obtained by reading the temperature and truncating the 0.5°C bit (the LSB) from the read value. This value is TEMP_READ. The value left in the counter may then be read. This value is the count remaining (COUNT_REMAIN) after the gate period has ceased. By loading the value of the slope accumulator into the count register, this value may then be read, yielding the number of counts per degree C (COUNT_PER_C) at that temperature. The actual temperature may be then be calculated by the user using the following:

072996 1/3

$$\text{TEMPERATURE} = \text{TEMP\_READ} - 0.25 + \frac{(\text{COUNT\_PER\_C} - \text{COUNT\_REMAIN})}{\text{COUNT\_PER\_C}}$$

TEMPERATURE MEASURING CIRCUITRY Figure 1
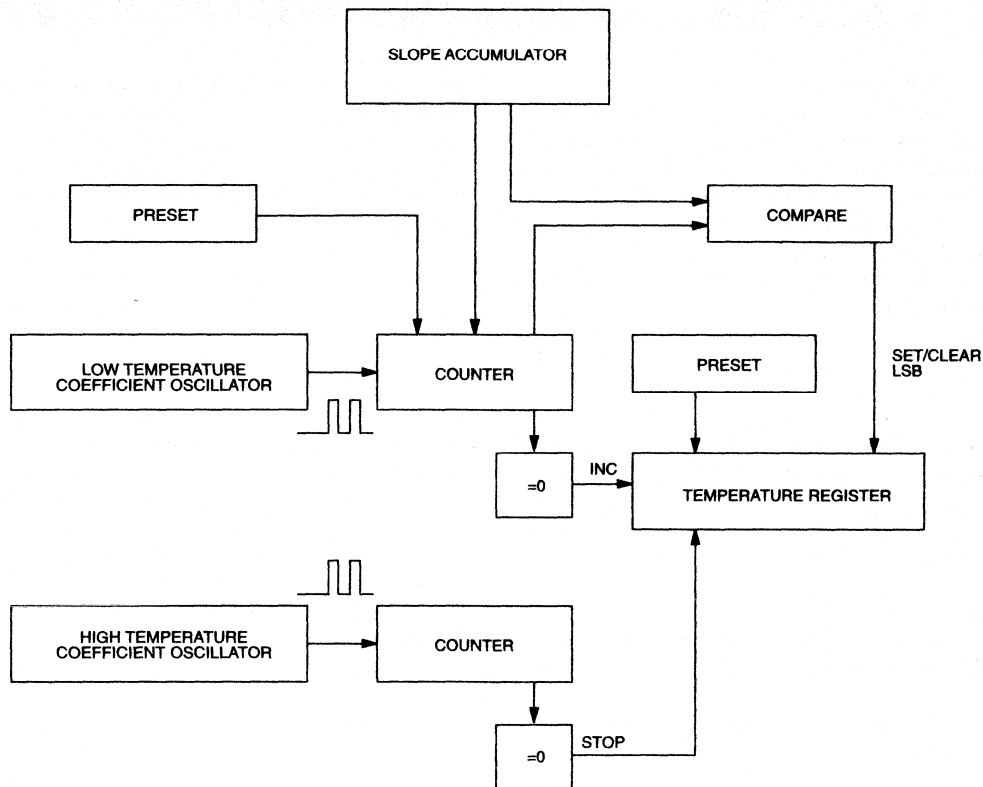
OBTAINING 0.1°C RESOLUTION WITH THE DS1620

The following steps describe the process necessary to obtain 0.1°C resolution from the DS1620:

1. If the DS1620 is not already in ONESHOT mode, do so by sending the WRITE CONFIG protocol with the appropriate write data. For more information on this protocol and the mode settings, see the DS1620 data sheet.

   Protocol to set both ONESHOT and CPU modes:
       0C   03   (hex)

   Protocol to set only ONESHOT mode:
       0C   01   (hex)

The first protocol listed above is usually the desired mode setting when using a computer to control the DS1620.

2. Issue the START CONVERT command to the part (EE hex).

3. Issue the READ CONFIG command (AC hex) to read the configuration register data from the DS1620. Repeat this command until the most significant bit (i.e., the DONE bit) of the register is a '1'. This means that the temperature conversion has completed. The logical description for that condition using C language nomenclature is:

       (config & 0x80) = = 0x80

4. Issue the READ TEMPERATURE command, as described in the data sheet, which is AA hex. Truncate the 1/2 degree bit from the read value, and convert it to a signed integer. That modified value is TEMP_READ.

5. Issue the READ COUNTER command, which is not documented in the data sheet. That protocol works the same way as those that read the thermostat settings. First, send the protocol, which is A0 hex. Then read back a 9-bit value from the part. This value is COUNT_REMAIN.

6. Issue the LOAD SLOPE command, which also is not documented in the data sheet. The protocol is 41 hex, and requires no data to read or write. This command loads the slope value into the counter.

7. Issue the READ COUNTER command again, as described in step 5 above. The value read is COUNT_PER_C.

8. Calculate the actual temperature using the following formula, again using C nomenclature:

TEMPERATURE=TEMP_READ − 0.25 + (COUNT_PER_C − COUNT_REMAIN) / COUNT_PER_C

9. Repeat steps 2 through 8 as desired.

RELATED PRODUCT:
DS1623

NOTE:
See Application Note 105 for sample C code. The code can be downloaded from Dallas Semiconductor's anonymous FTP site.

DALLAS
SEMICONDUCTOR

This application note is a courtesy of Michel St–Hilaire and Marc Desjardins from XyryX Technologies, Quebec city, Province of Quebec, Canada.

INTRODUCTION

The DS1620 Digital Thermometer and Thermostat provides 9–bit temperature readings which indicate the temperature of the device. With three thermal alarm outputs, the DS1620 can also act as a thermostat. Temperature settings and temperature readings are all communicated to/from the DS1620 over a simple 3–wire interface.

However, the SPI interface found on many Motorola processors cannot directly communicate with the 3–wire interface found on the DS1620. First, the data flow to and from the DS1620 is multiplexed on only one pin (DQ) while SPI needs two separate signals (MOSI, MISO).

Second, most SPI interfaces are limited to 8–bit data transfer, complicating sending and receiving the 9–bit temperature readings to and from the DS1620. In addition, the DS1620's interface transfers LSB first, while SPI is an MSB-first communication protocol.

Lastly, the $\overline{RST}$ is unlike a $\overline{CS}$ (chip select) signal in that $\overline{RST}$ must be high from the beginning of a transfer (protocol) to the end of all transfer of data (e.g. 9th bit transferred when reading temperature value).

Despite all these constraints, a fairly simple solution can be found which allows an SPI interface to communicate with a DS1620. This technique is described in this application note.

SPI INTERFACE

The circuit shown in Figure 1 can be used to control data flow direction with an SPI bus interfaced to a DS1620. This circuit could be integrated into a small PAL if desired.

The purpose of the DIR signal is to select between sending data to or receiving data from the DS1620. When DIR is low, the DS1620 is receiving data; if DIR is high, data is being read by the SPI controller.

The resistor is necessary to prevent contention between the output of the tri–state buffer on the MOSI line and the DQ pin of the DS1620, because after a READ command protocol has been received by the DS1620, its DQ pin changes direction from input to output in a few hundred nanoseconds. This time is much too short for the microprocessor controlling the DIR signal to take action.

When connecting multiple peripherals on the same SPI bus, the MISO signal must be tri–stated when the DS1620 is not accessed to prevent contention with the MISO signal of other peripherals. That is why the $\overline{RST}$ signal is necessary in the logic which determines the data direction.

Note that the SPI clock is wired directly to the CLK pin of the DS1620. The software has to take care of the polarity and phase of the SPI clock to be compatible with the CLK timing requirements of the DS1620.

SPI TO DS1620 INTERFACE CIRCUIT Figure 1
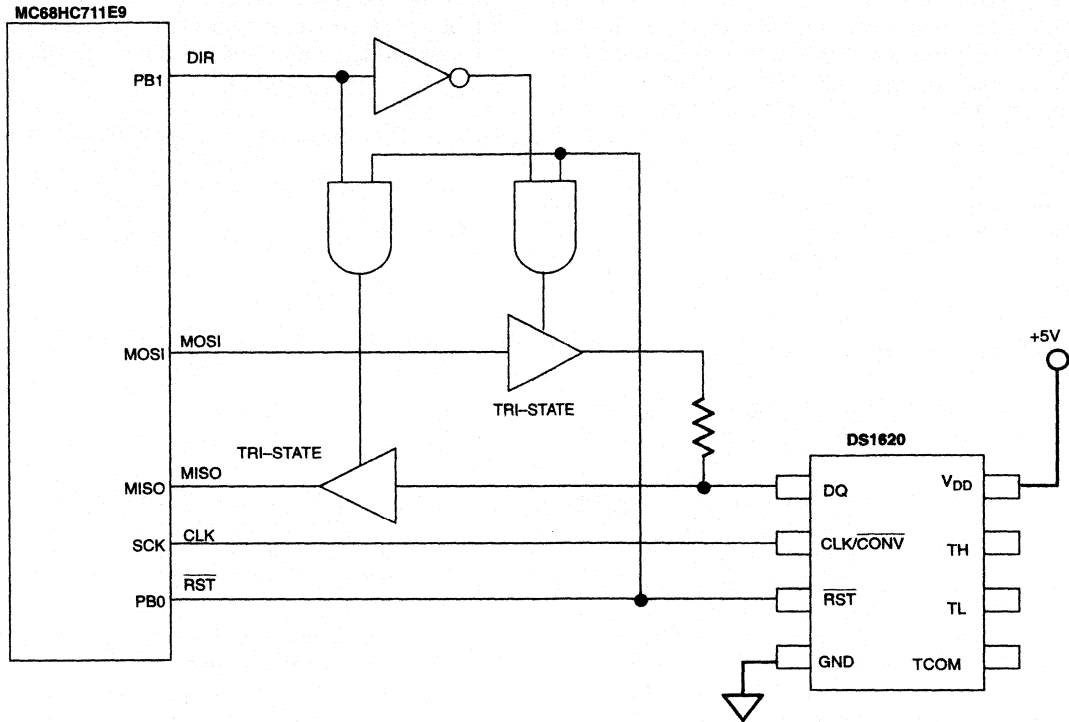
SOFTWARE FOR THE INTERFACE

While the hardware for the interface is relatively straightforward, the rest of the SPI/DS1620 interface must be handled by software. The following example shows a way to do this in the case of reading the temperature from the DS1620. This code fragment assumes that the DS1620 has already been initialized, that the configuration register is set up properly, and that temperature conversions have been initiated. See the DS1620 data sheet for details on these operating modes.

Before accessing the DS1620, the DIR signal must be asserted low for a WRITE transfer to occur. $\overline{RST}$ must be driven high to enable the DS1620. The SPI controller sends out the protocol (eight bits long) to the DS1620. Again, note that SPI sends information MSB first, while the DS1620 communicates LSB first. In order to accomplish this, a software "mirror" should be used to reverse the bit order. An example of such a function is given by:

```
unsigned char mirror(unsigned char value)
{
     unsigned char i;
     unsigned char value_mirrored = 0x00;

     for (i=0;i<=7;i++)
     {
        value_mirrored = value_mirrored | (((value>>i)&0x01)<<(7-i));
}
return (value_mirrored);
}
```

With the protocol sent, the DIR is changed from low to high (indicating now a READ transfer) because the DS1620 is ready to send out the 9–bit value. Note that RST is still high. The SPI controller reads the first eight bits of the 9–bit value (LSB first). The software has to "mirror" the received byte. The 9th bit (followed by seven dummy bits) is pulled out by making another READ transfer and keeping DIR and RST as they are. When the second byte is received, the software again mirrors it and pulls RST low, terminating communication with the DS1620.

```
#define         RST_bit             0       /* PB0 */
#define         RST_port            PORTB
#define         DIR_bit             1       /* PB1 */
#define         DIR_port            PORTB
#define         READ_TEMP_CMD       0xAA

unsigned int read_temp(void)
{
    unsigned char      temp_value_lo;
    unsigned char      temp_value_hi;

    DIR_port = DIR_port & ~(1<<DIR_bit);            /* DIR = LO: WRITE mode */
    RST_port = RST_port | (1<<RST_bit);            /* RST = HI: DS1620 enabled */
    SPDR = mirror(READ_TEMP_CMD);                  /* Send protocol to DS1620 */
    DIR_port = DIR_port | (1<<DIR_bit);            /* DIR = HI: READ mode */
    while ((SPSR & (1<<SPIF_bit)) == 0);           /* Wait for SPI flag = ready */
    temp_value_lo = mirror(SPDR);                  /* Receive 8 lowest bits */
    temp_value_hi = mirror(SPDR);                  /* Receive 8 highest bits */
    RST_port = RST_port & ~(1<<RST_bit);           /* RST = LO: Temp. reading done */
    return ((temp_value_hi<<8)+temp_value_lo);     /* Return the 9-bit value */
}
```

RELATED PRODUCT:
DS1623

DALLAS
SEMICONDUCTOR

Application Note 105
High Resolution Temperature Measurement with Dallas Direct–to–Digital Temperature Sensors

INTRODUCTION
This application note describes the principle of operation of Dallas Semiconductor's line of direct–to–digital temperature sensors, and outlines a method of achieving high (<0.05°C) resolution with these devices. An example C code listing is given for use with the DS1620.

DIRECT–TO–DIGITAL TEMPERATURE SENSOR PRINCIPLE OF OPERATION
The Dallas Direct–to–Digital Temperature Sensors measure temperature through the use of an onboard proprietary temperature measurement technique. A block diagram of the temperature measurement circuitry is shown in Figure 1.

Each temperature sensor measures temperature by counting the number of clock cycles that an oscillator with a low temperature coefficient goes through during a gate period determined by a high temperature coefficient oscillator. The counter is preset with a base count that corresponds to –55°C. If the counter reaches zero

before the gate period is over, the temperature register, which is also preset to the –55°C value, is incremented, indicating that the temperature is higher than –55°C.

At the same time, the counter is then preset with a value determined by the slope accumulator circuitry. This circuitry is needed to compensate for the parabolic behavior of the oscillators over temperature. The counter is then clocked again until it reaches zero. If the gate period is still not finished, then this process repeats.

The slope accumulator is used to compensate for the nonlinear behavior of the oscillators over temperature, yielding a high resolution temperature measurement (0.5°C for almost all the products). This is done by changing the number of counts necessary for the counter to go through for each incremental degree in temperature. To obtain the desired resolution, therefore, both the value of the counter and the number of counts per degree C (the value of the slope accumulator) at a given temperature must be known.

TEMPERATURE MEASUREMENT CIRCUITRY Figure 1

072996 1/19
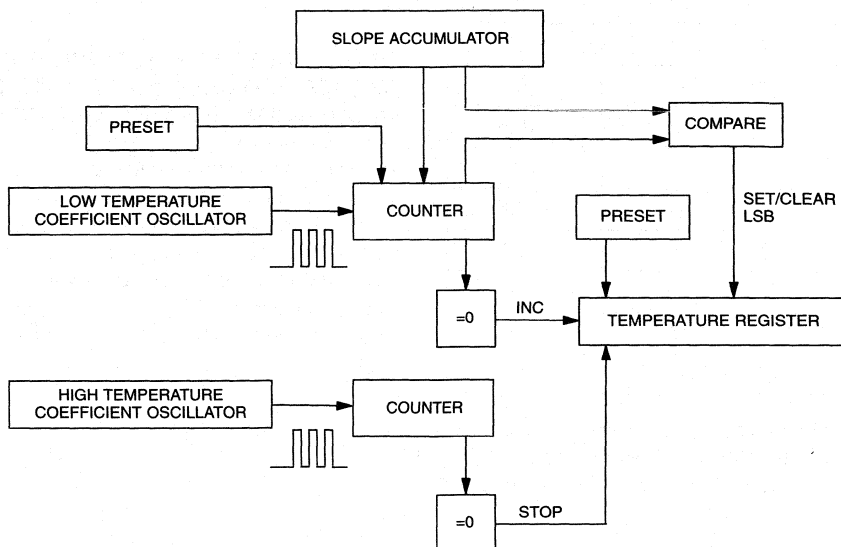
INCREASING TEMPERATURE RESOLUTION

Most of Dallas' direct–to–digital temperature sensors provide 0.5°C resolution directly. This is accomplished by the device determining whether to set or clear the least significant bit (LSB), based on the actual temperature. The device attempts to keep errors within $1/2$ LSB, by quantizing different readings into the LSB step size. For example, a part which is ramping up in temperature from 25°C to +26°C, or down in temperature from –10°C to –11°C, would exhibit this behavior:

ACTUAL TEMPERATURE	SET/CLEAR LSB	DIRECT READING
25	Clear	25
25.1	Clear	25
25.2	Clear	25
25.3	Set	25.5
25.4	Set	25.5
25.5	Set	25.5
25.6	Set	25.5
25.7	Set	25.5
25.8	Clear	26
25.9	Clear	26
26	Clear	26
–10	Clear	–10
–10.1	Clear	–10
–10.2	Clear	–10
–10.3	Set	–10.5
–10.4	Set	–10.5
–10.5	Set	–10.5
–10.6	Set	–10.5
–10.7	Set	–10.5
–10.8	Clear	–11
–10.9	Clear	–11
–11	Clear	–11

This example shows then that every reading is rounded up (in absolute value) by $1/2$ LSB. For most parts, this is 0.25°C. This is important to remember, since in doing calculations to achieve higher resolutions, this rounding factor must be taken into account.

With the exception of devices intended for battery management (DS2434 and DS2435), the temperature sensors can measure temperature over the range of –55°C to +125°C in 0.5°C increments. For Fahrenheit usage, a lookup table or conversion factor must be used.

Higher resolutions may be obtained by reading the temperature and truncating the least significant bit from the read value. From the example above, it should be apparent that this must be done on the raw, 9–bit number, in two's–complement arithmetic, in order for the correct reading to result. After the truncation, the number can then be converted into a signed integer. This value is referred to below as temp_read. The value left in the counter can then be read by issuing a special command protocol to the sensor. This value is the count remaining (count_remain) after the gate period has ceased. Reading the value of the slope accumulator (by using another command protocol or set of protocols, as outlined below) yields the number of counts per degree C (count_per_degree) at that temperature. Once these parameters are all known, the actual temperature can be calculated from the following equation:

$$TEMPERATURE = temp\_read - 1/2 LSB + \frac{(count\_per\_degree - count\_remain)}{count\_per\_degree}$$

A simple routine in C, called CalcHiResTemp, is given in the listing of Figure 7.

PROCEDURES FOR READING COUNTER VALUES AND CALCULATING HIGH RESOLUTION TEMPERATURE READINGS

The following is a list of procedures for performing high resolution temperature readings from various digital thermometers. For all parts, conversions must be done in oneshot mode (if applicable). XXh refers to the protocol to send to the part; if two protocols are listed together (e.g., 84h A0h), both protocols should be sent in the given order, without resetting the part between protocols. A variable name XXX is designated as "named XXX."

NOTE:

The high–resolution temperature equation is slightly different for the DS1821 as compared to the DS1620, DS1623, DS1625, DS2434, and DS2435.

DS1620, DS1623, and DS1625:

1. Issue Start Convert protocol (EEh).

2. When conversion is finished, read 9–bit temperature value (AAh).

3. Truncate half–degree bit from reading.

4. Convert truncated value from 2's complement to signed integer (named temp_read).

5. Read 9–bit counter value (A0h; named count_remain).

6. Send undocumented Load Counter protocol (41h).

7. Read 9–bit counter value (A0h; named count_per_degree).

8. Calculate high–resolution temperature using the high resolution temperature equation given in the previous section. $1/2$ LSB = 0.25.

DS1621, DS1624, and DS1820:

The procedure to find the high resolution temperature parameters and the calculation to use are given in the respective product data sheets for these products. Note that since the DS1624 already provides a 13–bit number with 0.03125°C resolution, no further processing is possible to achieve any higher temperature resolution.

DS1821:

1. Issue Start Convert protocol (EEh).

2. When conversion is finished, read 8–bit temperature value (AAh).

3. Convert value from 2's complement to signed integer (named temp_read).

4. Read 9–bit counter value (A0h; named count_remain).

5. Send undocumented Load Counter protocol (41h).

6. Read 9–bit counter value (A0h; named count_per_degree).

7. Calculate high–resolution temperature using the high resolution temperature equation given in the previous section. Note that for the DS1821, 1/2LSB = 0.5.

DS2434 and DS2435:

1. Issue Start Convert protocol (D2h).

2. When conversion is finished, read 8–bit temperature value (B2h 61h).

3. Convert temperature value from 2's complement to signed integer (named temp_read).

4. Read 9–bit counter value (84h A0h; named count_remain).

5. Send undocumented Load Counter protocol (84h 41h).

6. Read 9–bit counter value (84h A0h; named count_per_degree).

7. Calculate high–resolution temperature using the high resolution temperature equation given in the previous section. $1/2$LSB = 0.25.

EXAMPLE C CODE

The following example is a simple exerciser for the DS1620 from an IBM–PC compatible computer. It reads temperature and displays it in both the normal 0.5°C resolution and in high resolution. The DS1620 is interfaced to the parallel port of the PC using the circuit shown in Figure 2, which is the schematic of the DS1620K demo kit.

The hardware for the DS1620K demo kit "steals" power from a parallel port using D1, D2, D3, and C3. Not all PC parallel ports are able to supply sufficient current to make this hardware work, so be advised that if this circuit does not work, try connecting a +5V power source to the +5V line in this circuit and try again. R1 and C1 serve to filter the CLK line and prevent negative undershoots. Likewise, C2 helps prevent negative undershoot on the DQ line.

The code is written in an attempt to make it easy to adapt to any of the digital temperature sensors. All that need be done to use it with other devices is to change the command header file to account for the protocols and resolution of the device being used, and change the files which manipulate the hardware interface to the device to account for either 1, 2 or 3–wire interfaces.

DS1620K HARDWARE SCHEMATIC Figure 2
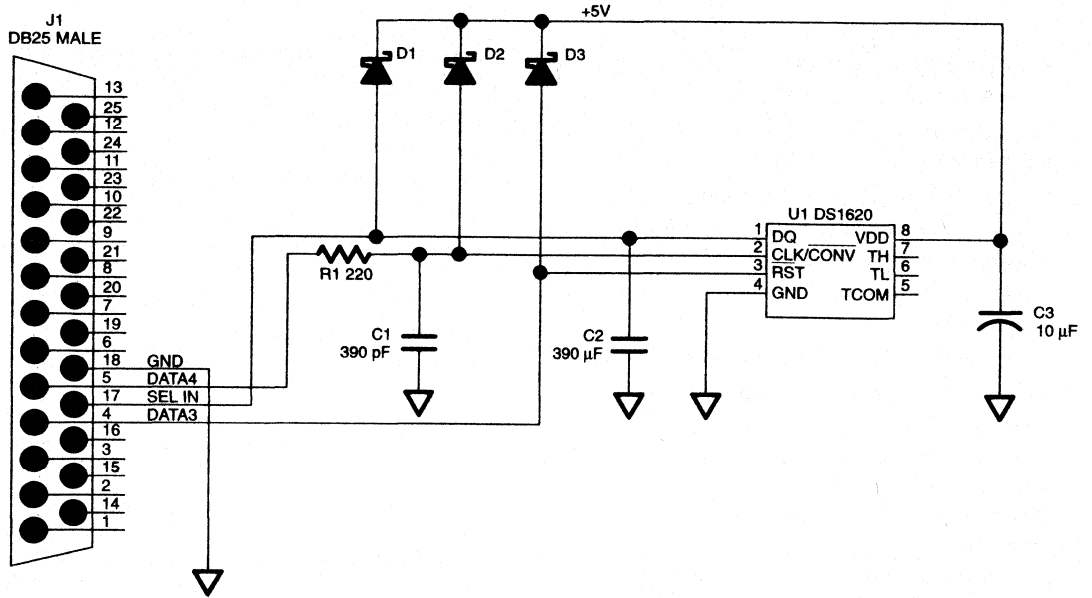
THREEWIR.H C CODE LISTING Figure 3

```
/* THREEWIR.H
This file defines the method for setting the 3-wire interface lines from an IBM-PC
compatible computer. The interface is through the parallel port.  The parallel
port lines used are as follows:


/SELIN          pin 17 DQ
Data4           pin 5   CLK
Data3           pin 4   /RST
GND             pin 18 GND


The inportb, outportb, and delay functions are defined in most PC-based C compil-
ers in dos.h
*/


#include <dos.h>


#define CLK_HI          p_data|= 0x08; outportb(p_addr,p_data); delay(1);
#define CLK_LOW         p_data&=~0x08; outportb(p_addr,p_data); delay(1);
#define RSTB_HI         p_data|= 0x04; outportb(p_addr,p_data); delay(1);
#define RSTB_LOW        p_data&=~0x04; outportb(p_addr,p_data); delay(1);
#define DQ_HI           outportb(p_addr+2,0x02); delay(1);
#define DQ_LOW          outportb(p_addr+2,0x0A); delay(1);
#define READ_BACK       -((inportb(p_addr+2) & 0x08) >> 3)+1


#define p_addr peek(0,0x0408) /* Finds the address of the parallel port     */


char    p_data = 0xFF;
```

DS3WIRE.C C CODE LISTING Figure 4

```
/* DS3wire.C
These routines handle a Dallas 3-wire interface. Methods of changing the individ-
ual bit states in the hardware need to be defined in "threewir.h".
*/


#include "threewir.h"


/*
Bit level drivers, used in routines for handling the protocol and data interface
to a 3-wire device.
*/


/* Drives RSTB signal high (state=1) or low (state=0) */


void rb3w( int state )
    {
      if (state==0) {RSTB_LOW;}
      else {RSTB_HI}
    }


/* Drives CLK signal high (state=1) or low (state=0) */


void c3w( int state )
    {
      if (state==0) {CLK_LOW}
      else {CLK_HI}
    }


/* Drives DQ signal high (state=1) or low (state=0)  */


void dq3w( int state )
    {
      if (state==0) {DQ_LOW}
      else {DQ_HI}
    }


/* Reads data back from port */


int rd_back()
    {
    char i;
    i=READ_BACK;
    return( i );
    }
```

```
/* Writes a 0 (w_bit=0) or a 1 (w_bit=1) to a three wire device. */

void write_bit( int w_bit )
    {
    if ( w_bit == 0 ) dq3w( 0 );
    else dq3w( 1 );
    c3w( 0 );
    c3w( 1 );
    dq3w( 1 );
    }


/* Reads a 0 (return 0) or a 1 (return 1) from a three wire device. */

    int read_bit()
    {
    int i;

    c3w( 0 );
    i=rd_back();
    c3w( 1 );

    return(i);
    }


/*
Routines to read and write a Dallas 3-wire part. The three wire interface typi-
cally requires that the part be sent an eight-bit protocol, followed by data. The
exception to this are the digital potentiometers, which use no protocols but send
only data. All transfers are LSB first.

If writing, the data must follow the protocol immediately, but there is no fixed
number of bits that may be data.

If reading, the part must be written with an eight bit protocol; the DQ pin must
then be changed to READ back data. As with writing, there is no fixed number of
bits that may be data.

These routines are set up to handle an arbitrary data length UP TO the size of an
int (which is typically 16 bits for most C compilers).
*/
```

```
/* write_part
Write a command and data to a 3-wire part. param is the integer value of the data
to be written after the write protocol. n_bits is the number of bits to transfer
in param.
*/

void write_part(int protocol, int param, int n_bits)
{
int  index,data;

rb3w(1);                                        /* Raise the /RST line */
for (index = 0; index<8; index++)               /* protocol is 8 bits   */
    {
    data = protocol>>index;
    data &= 0x01;
    write_bit(data);
    }
for (index = 0; index<n_bits; index++)          /* Write out data       */
    {
    data = param>>index;
    data &= 0x01;
    write_bit(data);
    }
rb3w(0);                                         /* Drop /RST line       */
}


/* read_part
Reads data from a 3-wire part. protocol is command to be written to the part,
and n_bits is number of bits to read after the protocol is sent. Returns int value
of reading.
*/

int read_part(int protocol, int n_bits)
{
int  index,data;
int  r_data = 0;

rb3w(1);                                        /* Raise the /RST line */
for (index = 0; index<8; index++)               /* protocol is 8 bits   */
    {
    data = protocol>>index;
    data &= 0x01;
    write_bit(data);
    }

for (index = 0; index<n_bits; index++)          /* Read in data         */
    {
    r_data |= read_bit()<<index;
    }
rb3w(0);
return(r_data);
}
```

DS1620CMD.H Figure 5

```
/* 1620CMD.H
This file defines the protocols and bit masks needed for a specific device: the
DS1620.
*/


/* protocols */
#define        Read_Temp          0xAA
#define        Start_Convert      0xEE
#define        Stop_Convert       0x22
#define        Write_TH           0x01
#define        Write_TL           0x02
#define        Read_TH            0xA1
#define        Read_TL            0xA2
#define        Write_Config       0x0C
#define        Read_Config        0xAC
#define        Read_Counter       0xA0
#define        Load_Counter       0x41


/* masks for configuration/status register   */
#define        mode_mask          0x03    /* masks off all but CPU and 1SHOT
                                             bits                           */
#define        nvb_mask           0x10    /* masks off all but NVB bit   */
#define        done_mask          0x80    /* masks off all but DONE bit */
#define        flags_mask         0x60    /* masks off all but THF and TLF flag
                                             bits                           */
#define        flag_offset        5       /* number of right shifts to
                                             right-justify flags            */


#define        res             (int)(1.0/LSB)
                                          /* resolution factor, used in
                                             application to calculate
                                             actual temperatures from raw
                                             data                          */


float          LSB = 0.5;                 /* The value of a LSB on the
                                             DS1620 in degrees C           */
```

DEVFUNC.C C CODE LISTING Figure 6

```
/* Device Specific Functions
These are device specific functions for all 3-wire Dallas Temperature Sensors,
supporting all documented functions of the parts, including high resolution tem-
perature measurements.


Since these routines may be used with many different devices, no attempt is made
in these routines to interpret the data written or read; the application program
which uses these routines must assure that the data written or read is in the
appropriate format for use by these routines and by the application.
*/


/*
Place device-appropriate include file here for specific device commands.
*/
#include <stdio.h>
#include "1620cmd.h"                              /* This example for DS1620    */


/*prototypes*/


extern void write_part();
extern int read_part();


/* StartTempConvert
This routine issues the Start Convert command. No further data is required and no
data is read back.
*/
void StartTempConvert(void)
{
    write_part(Start_Convert,0,0);
}


/* StopTempConvert
This routine isses the Stop Convert command. No further data is required and no
data is read back.
*/
void StopTempConvert(void)
{
    write_part(Stop_Convert,0,0);
}
```

```
/* ReadTemp
This routine reads back the value of the converted temperature reading and returns
that value. num_bits is the number of bits to read back (up to size of int (16)).
Remember that this only returns the raw data; the calling program must know how to
interpret the data (for example, this routine will return an int with 9 bits when
called as ReadTemp(9); the user must then know that the LSB is 0.5°C, and make the
appropriate data conversion.  Likewise, interpreting the sign of the returned tem-
perature is up to the calling program.)
*/
int ReadTemp (int num_bits)
{


     int        temperature;
     temperature = read_part(Read_Temp,num_bits);
     return(temperature);
}


/* WriteTempHigh
This routine writes to the TH thermostat register. Param is value to place in TH,
and should be 9 bits wide. As with ReadTemp, the calling program must assure that
the data sent here is in the appropriate device-specific data form to be used
properly (is LSB 0.5°C or 1°C?, etc).
*/
void WriteTempHigh (int param, int num_bits)
{
     write_part(Write_TH,param,num_bits);
}


/* WriteTempLo
This routine writes to the TL thermostat register. Param is value to place in TL,
and should be num_bits bits wide. As with ReadTemp, the calling program must
assure that the data sent here is in the appropriate device-specific data form to
be used properly (is LSB 0.5°C or 1°C?, etc).
*/
void WriteTempLo (int param, int num_bits)
{
     write_part(Write_TL, param, num_bits);
}


/* ReadTempHigh
This routine reads back from the part the value stored in the TH thermostat regis-
ter and returns that value. num_bits is the number of bits to read back (up to
size of int (16)).  Remember that this only returns the raw data; the calling pro-
gram must know how to interpret the data (for example, this routine will return an
int with 9 bits when called as ReadTempHigh(9); the user must then know that the
LSB is 0.5°C, and make the appropriate data conversion. Likewise, interpreting the
sign of the returned temperature is up to the calling program.)
*/
int ReadTempHigh (int num_bits)
{
```

```
    int tHigh;

    tHigh = read_part(Read_TH,num_bits);

    return(tHigh);
}


/* ReadTempLo
This routine reads back from the part the value stored in the TL thermostat regis-
ter and returns that value. num_bits is the number of bits to read back (up to
size of int (16)). See the caveat for ReadTempHigh regarding returned data.
*/
int ReadTempLo (int num_bits)
{
    int tLow;

    tLow = read_part(Read_TL,num_bits);

    return(tLow);
}


/* LoadCounter
This routine sends the Load Counter command. No further data is required and no
data is read back.
*/
void LoadCounter (void)
{
    write_part(Load_Counter,0,0);
}


/* ReadCounter
This routine reads back from the part the value in the temp sensor count register
and returns that value. num_bits is the number of bits to read back (up to size of
int (16)).
*/
int ReadCounter (int num_bits)
{
    int count;

    count = read_part(Read_Counter,num_bits);

    return(count);
}
```

```
/* WriteConfig
This routine writes to the configuration register. Param is the byte to be written
into the configuration register, and it is always 8 bits wide.
*/
void WriteConfig (int param)
{
     write_part(Write_Config,param,8);
}


/* ReadConfig
This routine reads back the value of the configuration byte and returns that
value. It always returns 8 bits.
*/
int ReadConfig(void)
{
     int regvalue;

     regvalue = read_part(Read_Config,8);
     return(regvalue);
}


/* CheckIfBusy
This routine will check to see if the device is busy and cannot process further
writes to EEPROM. It checks the NVB flag in the configuration register. If the
device is busy, the function returns a 1; if it is not busy, the function will
return a 0.
*/
int CheckIfBusy(void)
{
     if ((ReadConfig() & nvb_mask) == nvb_mask)
               return(1);
     else
               return(0);
}


/* CheckIfDone
This routine will check to see if the device has completed a temperature conver-
sion.  It checks the DONE flag in the configuration register. If the device is
busy, the function returns a 1; if it is not busy, the function will return a 0.
NOTE: If the mode is set so that the temp sensor is doing continuous conversions,
DONE will NEVER be set, so make sure you know what mode you're in if you use this
check routine.
*/
int CheckIfDone(void)
{
     if ((ReadConfig() & done_mask) == done_mask)
               return(0);
     else
               return(1);
}
```

```
/* SetMode
This routine sets the operating mode of the digital temperature sensor. Mode is
determined by two bits in the configuration register: CPU and 1SHOT. The following
modes are possible:
```

Mode	CPU	1SHOT	
1	0	0	Standalone, continuous conversions (typical for thermostat)
2	0	1	Standalone, one shot (used only if conversions can be started by CONV pin)
3	1	0	CPU, continuous conversion (typical for thermostat with readback)
4	1	1	CPU, one shot (typical for low-power temp readings)

```
This routine will set the configuration register bits appropriately for the MODE
value passed to the routine in mode. It returns no data.
*/
void SetMode(int mode)
{
    WriteConfig (mode-1);
}


/* ReadFlags
This routine will read the state of the two temperature flags, THF and TLF. It
returns an integer, t_flag, which is the number of the two bits (0 = 00, 1=01,
2=10, 3=11).
*/
int  ReadFlags(void)
{
    int t_flag;

    t_flag =( ReadConfig() & flags_mask) >> flag_offset;
    return(t_flag);
}
```

SIMPLE DS1620 EXERCISER C CODE LISTING Figure 7

```c
/* DS1620 Exerciser
Example program to show how to perform various functions of a DS1620 Digital Ther-
mometer.  This example is console-driven; that is, it expects input from the user
via a console interface, to set the mode and TH, TL register values.
*/


#include <stdio.h>
#include "devfunc.c"
#include "ds3wire.c"


/*prototypes*/
void    StartTempConvert(void);
void    StopTempConvert(void);
int     ReadTemp(int);
void    WriteTempHigh(int,int);
void    WriteTempLo(int,int);
int     ReadTempHigh(int);
int     ReadTempLo(int);
void    LoadCounter(void);
int     ReadCounter(int);
void    WriteConfig(int);
int     ReadConfig(void);
int     CheckIfBusy(void);
int     CheckIfDone(void);
void    SetMode(int);
int     ReadFlags(void);
void    InitTempSensor(void);
void    SetThermostat(void);
float   CalcHiResTemp(int,int,int);


/* InitTempSensor
This routine will initialize the temperature sensor to a mode specified by the
user.
*/
void InitTempSensor(void)
{
    int        mode, test;

    printf("1. Standalone mode, Continuous Conversions\n");
    printf("2. Standalone mode, One Shot\n");
    printf("3. CPU mode, Continuous Conversions\n");
    printf("4. CPU mode, One Shot\n");
    printf("Enter mode desired:");
    scanf("%d",&mode);
    SetMode(mode);                      /*Write mode bits to Config register */
```

```
        do                                      /* Wait until EEPROM write
                                                complete            */
        {
            test = CheckIfBusy();
            printf("Device Busy");
        } while (test);


}


/* SetThermostat
This routine sets the thermostat registers to the high and low values specified by
the user.
*/
void SetThermostat(void)
{
        float       t_hi, t_lo, high_t, low_t;  /* Since TH and TL are 9-bit
                                                numbers with a 0.5°C
                                                LSB, they must be of type
                                                float. This program
                                                must then assure the data
                                                passed to the other
                                                routines is formatted
                                                properly.           */
        int         hi_t, lo_t, busy;

        printf("Enter a High Temperature Limit:");
        scanf("%f",&t_hi);                      /* Get TH limit       */
        printf("Enter a Low Temperature Limit:");
        scanf("%f",&t_lo);                      /* Get TL limit       */

        hi_t = res*t_hi;                        /* turn into a 9-bit integer -
                                                note that this routine
                                                does not check to see if the
                                                user enters some number that
                                                can't be used, like 22.635. An
                                                actual application would
                                                want to do such error trap
                                                ping.               */
        WriteTempHigh(hi_t,9);                  /* Write to TH register*/

        do                                      /* Wait until EEPROM write
                                                complete            */
        {
            busy = CheckIfBusy();
            printf("Device Busy");
        } while (busy);
```

```
        lo_t= res*t_lo;                          /* same comments as for TH
                                                                          */

        WriteTempLo(lo_t,9);                     /* Write to TL register
                                                                          */


        do                                       /* Wait until EEPROM write
                                                 complete             */
        {
           busy = CheckIfBusy();
           printf("Device Busy");
        } while (busy);

}


/* CalcHiResTemp
This routine handles the calculation of temperature in high resolution mode.
Requires that the part has provided the values of temperature, count_per_degree,
and count_remain. This routine assumes 9 bit data is read back, and that the user
has defined the value of LSB and res in the appropriate DSXXXXCMD.H file.
*/
float  CalcHiResTemp(int temp9, int count_per_degree, int count_remain)
{
        int        temp_read;
        float      frac_deg;

        temp_read = temp9/res;                   /* temp9 is raw data - convert
                                                 for LSB value        */
        if(temp_read>>8>0)                       /* check to see if MSB set -
                                                 works for 9-bit data. */
           temp_read -=256;                      /* convert to negative temper
                                                 ature                */

        if(count_per_degree !=0)
           frac_deg = (count_per_degree-count_remain)/((float)count_per_degree);
        else
           frac_deg = LSB/2.0;

        return(((float)temp_read-(LSB/2.0))+frac_deg);
}


void main(void)
{
        signed int temperature;
        float      read_temp;
        int        Countrem,CountC;
        float      hires_temp;
        int        busy,key;
```

```
   InitTempSensor();                            /* Initialize sensor with mode
                                                 bits                    */
   SetThermostat();                             /* Load in Thermostat values,
                                                 if used                 */
   StartTempConvert();                                   /* Start Temperature
                                                 Conversion(s)           */


   switch(ReadConfig()& mode_mask)              /* Checks for conversion done.
                                                 This switch is used to
                                                 determine what to do depending
                                                 upon the mode the
                                                 DS1620 is in. If continuous
                                                 conversions are going on,
                                                 temperature readings should
                                                 only be made periodically
                                                 based on time. If ONE-SHOT
                                                 mode is used, then you can
                                                 poll the status register for
                                                 the DONE bit to set.    */
   {
      case 0:                                   /* Continuous conversions - so
                                                 read based on time      */
            delay(1000);
            break;
      case 1:                                   /* One Shot mode - poll status
                                                 register for DONE bit */
            do
            {
                  busy =CheckIfDone();
                  printf("\nConversion in progess");
            }while(busy);
            break;
      case 2:                                   /* Continuous conversions - so
                                                 read based on time      */

            delay(1000);
            break;
      case 3:                                   /* One Shot mode - poll status
                                                 register for DONE bit */
            do
            {
                  busy =CheckIfDone();
                  printf("\nConversion in progess");
            }while(busy);
            break;
   }/* end switch */
```

```
read_temp = (float)ReadTemp(9)*LSB;          /* Reads temperature and con
                                             verts for LSB value      */

if ((int)read_temp>>7>0)                      /* Check if MSB set
                                                                      */

    read_temp -=256;                          /* if it is, correct for it
                                                                      */


printf("\nThe temperature is:%5.2f",read_temp);
/* prints temperature on console, to 0.5°C resolution   */


StopTempConvert();                            /* In case you were previously
                                             in continuous mode, stop
                                                                      */

SetMode(4);                                   /* MUST BE in ONE-SHOT to do
                                             High Res                 */

StartTempConvert();                           /* Start Temperature
                                             Conversion(s)            */


do                                            /* Check to see when conver
                                             sion completes by polling DONE
                                             bit                      */

{
   busy =CheckIfDone();
   printf("\nConversion in progess");
}while(busy);


read_temp = (float)ReadTemp(9);               /* get raw 9-bit temperature
                                             data                     */

if ((int)read_temp>>8>0)                       /* Check if MSB set    */
   read_temp -=512;                            /* if it is, correct for it -
                                             9 bits now!              */


temperature = (signed int) read_temp;          /* make sure temp is now a
                                             signed integer           */

Countrem = ReadCounter(9);                     /* reads count remaining
                                                                      */

LoadCounter();                                 /* Loads counter reg with
                                             counts per degree        */

CountC = ReadCounter(9);

hires_temp = CalcHiResTemp(temperature,CountC,Countrem);

printf("\n The temperature is :%5.2f",hires_temp);
/* prints temperature on console, to 0.01°C resolution  */


}
```

RELATED PRODUCTS:
DS1620, DS1621, DS1623, DS1625, DS1820, DS1821, DS2434, DS2435

DALLAS
SEMICONDUCTOR

Related Application Notes
Related Application Notes for
Thermal and Battery Management

Several application notes contained in other sections of this book directly apply to the Thermal and Battery Management product family. These notes are listed below, along with an abstract for how each note applies to Thermal and Battery Management products.

"UNDERSTANDING AND USING CYCLIC REDUNDANCY CHECKS WITH DALLAS SEMICONDUCTOR TOUCH MEMORY PRODUCTS" Note 27
Page 186
Related Thermal and Battery Management products: DS1820

Abstract
The DS1820 utilizes an 8–bit Cyclic Redundancy Check (CRC) to help ensure data integrity over the 1–Wire™ interface. Many Automatic Identification products also feature the same CRC, and this note provides a practical method for calculation of the CRC by the host.

"EXTENDING THE CONTACT RANGE OF TOUCH MEMORIES" Note 55
Page 201
Related Thermal and Battery Management products: DS1820

Abstract
Again, the DS1820 uses the same 1–Wire interface as the Automatic Identification products. This application note addresses the issue of cable length from the master to the last sensor on the bus and the practical number of 1–Wire devices that effectively can exist on the bus while maintaining the integrity of the timing signals.

"READING AND WRITING TOUCH MEMORIES VIA SERIAL INTERFACES" Note 74
Page 218
Related Thermal and Battery Management products: DS1820, DS1821, DS2434, DS2435

Abstract
The DS1820/1 Temperature Sensors and the DS2434/5 Battery ID products all use the same 1–Wire timing as the Automatic Identification products. This application note cites hardware and software examples to interface the 1–Wire bus to TTL and RS–232 serial interface. A schematic is presented for the COM port interface circuit, the DS9097, which is used in the DS2434K, DS2435K, and Minimalist Temperature demo kits.

032796 1/2

"MINIMALIST TEMPERATURE CONTROL DEMO" Note 104
Page 276
Related Thermal and Battery Management products: DS1820

Abstract

The Minimalist Temperature Demo simulates a closed–loop temperature control system using the DS1820 Digital Thermometer and DS2407 Addressable Switch. This application note illustrates the concept of using the DS2407/DS1820 combination to sense air flow (using a Piezo sensor) and temperature (using the DS1820) over a single 1–Wire interface for closed–loop temperature control.

"COMPLEX MICROLANS" Note 106
Page 280
Related Thermal and Battery Management products: DS1820

Abstract

This application note describes how to create a complex MicroLAN network. The topology is analogous to a tree, and Addressable Switches provide the branching from the "trunk." The DS1820 can be used as a function at one of the "leaves." Consult an Automatic Identification Application Engineer before pursuing such a network.

AUTOMATIC IDENTIFICATION

DALLAS
SEMICONDUCTOR

Application Note 27
Understanding and Using Cyclic Redundancy Checks with Dallas Semiconductor Touch Memory Products

INTRODUCTION

The Dallas Semiconductor Touch Memory products are a family of devices that all communicate over a single wire following a specific command sequence referred to as the 1–Wire™ Protocol. A key feature of each device is a unique 8–byte ROM code written into each part at the time of manufacture. The components of this 8–byte code can be seen in Figure 1. The least significant byte contains a family code that identifies the type of Touch Memory product. For example, the DS1990A has a family code of 01 Hex and the DS1991 has a family code of 02 Hex. Since multiple devices of the same or different family types can reside on the same 1–Wire bus simultaneously, it is important for the host to be able to determine how to properly access each of the devices that it locates on the 1–Wire bus. The family code provides this information. The next six bytes contain a unique serial number that allows multiple devices within the same family code to be distinguished from each other. This unique serial number can be thought of as an "address" for each device on the 1–Wire bus. The entire collection of devices plus the host form a type of miniature local area network, or Micro-LAN; they all communicate over the single common wire. The most significant byte in the ROM code of each device contains a Cyclic Redundancy Check (CRC) value based on the previous seven bytes of data for that part. When the host system begins communication with a device, the 8–byte ROM is read,

LSB first. If the CRC that is calculated by the host agrees with the CRC contained in byte 7 of ROM data, the communication can be considered valid. If this is not the case, an error has occurred and the ROM code should be read again.

Some of the Touch Memory products have up to 8K bytes of RAM in addition to the eight bytes of ROM that can be accessed by the host system with appropriate commands. Even if Touch Memories do not have CRC hardware onboard, if the host has the capability to calculate a CRC value for the ROM codes, then a procedure to store and retrieve data in the RAM portion of the devices using CRCs can also be developed. Data can be written to the device in the normal manner; then a CRC value that has been calculated by the host is appended and stored with the data. When this data is retrieved from the Touch Memory, the process is reversed. The host compares the CRC value that was computed for the data bytes to the value stored in memory as the CRC for that data. If the values are equal, the data read from the Touch Memory can be considered valid. In order to take advantage of the power of CRCs to validate the serial communication on the 1–Wire bus, an understanding of what a CRC is and how they work is necessary. In addition, a practical method for calculation of the CRC values by the host will be required for either a hardware or software implementation.

TOUCH MEMORY SYSTEM CONFIGURATION USING DOW CRC Figure 1

Dallas Semiconductor 1-Wire Device DS19xx

MSB **64-BIT ONE-WIRE ROM CODE** LSB

| CRC | UNIQUE | SERIAL | NUMBER | FAMILY CODE |

Byte 7 Byte 6 Byte 5 Byte 4 Byte 3 Byte 2 Byte 1 Byte 0

The CRC (Byte 7) has been computed for the data contained in Byte 0 through Byte 6 and the value has been written into Byte 7 for each Dallas Semiconductor 1-Wire device.

GND

1-WIRE Bus

I/O

Host System

CRC CALCULATOR

If CRC value that is computed for the first 56 data bits of the ROM code agrees with CRC value contained in Byte 7 of ROM code, continue reading data. Otherwise, the ROM should be reread.

BI-DIRECTIONAL I/O PORT

GND

BACKGROUND

Serial data can be checked for errors in a variety of ways. One common way is to include an additional bit in each packet being checked that will indicate if an error has occurred. For packets of 8–bit ASCII characters, for example, an extra bit is appended to each ASCII character that indicates if the character contains errors. Suppose the data consisted of a bit string of 11010001. A ninth bit would be appended so that the total number of bits that are 1's is always an odd number. Thus, a 1 would be appended and the data packet would become 111010001. The underlined character indicates the parity bit value required to make the complete 9–bit packet have an odd number of bits. If the received data was 11010000 1, then it would be assumed that the information was correct. If, however, the data received was 111010101, where the 7th bit from the left has been incorrectly received, the total number of 1's is no longer odd and an error condition has been detected and appropriate action would be taken. This type of scheme is called odd parity. Similarly, the total number of 1's could also be chosen to always be equal to an even number, thus the term even parity. This scheme is limited to detecting an odd number of bit errors, however. In the example above, if the data was corrupted and became 111011101 where both the 6th and 7th bits from the left were wrong, the parity check appears correct; yet the error would go undetected whether even or odd parity was used.

DESCRIPTION

Dallas Semiconductor 1–Wire CRC

The error detection scheme most effective at locating errors in a serial data stream with a minimal amount of hardware is the Cyclic Redundancy Check (CRC). The operation and properties of the CRC function used in Dallas Semiconductor products will be presented without going into the mathematical details of proving the statements and descriptions. The mathematical concepts behind the properties of the CRC are described in detail in the references. The CRC can be most easily understood by considering the function as it would actually be built in hardware, usually represented as a shift register arrangement with feedback as shown in Figure 2. Alternatively, the CRC is sometimes referred to as a polynomial expression in a dummy variable X, with binary coefficients for each of the terms. The coefficients correspond directly to the feedback paths shown in the shift register implementation. The number of stages in the shift register for the hardware description, or the highest order coefficient present in the polynomial expression, indicate the magnitude of the CRC value that will be computed. CRC codes that are commonly used in digital data communications include the CRC–16 and the CRC–CCITT, each of which computes a 16–bit CRC value. The Dallas Semiconductor 1–Wire CRC (DOW CRC) magnitude is eight bits, which is used for checking the 64–bit ROM code written into each

1–Wire product. This ROM code consists of an 8–bit family code written into the least significant byte, a unique 48–bit serial number written into the next six bytes, and a CRC value that is computed based on the preceding 56 bits of ROM and then written into the most significant byte. The location of the feedback paths represented by the exclusive–or gates in Figure 2, or the presence of coefficients in the polynomial expression, determine the properties of the CRC and the ability of the algorithm to locate certain types of errors in the data. For the DOW CRC, the types of errors that are detectable are:

1. Any odd number of errors anywhere within the 64–bit number.

2. All double-bit errors anywhere within the 64–bit number.

3. Any cluster of errors that can be contained within an 8–bit "window" (1–8 bits incorrect).

4. Most larger clusters of errors.

The input data is Exclusive–Or'd with the output of the eighth stage of the shift register in Figure 2. The shift register may be considered mathematically as a dividing circuit. The input data is the dividend, and the shift register with feedback acts as a divisor. The resulting quotient is discarded, and the remainder is the CRC value for that particular stream of input data, which resides in the shift register after the last data bit has been shifted in. From the shift register implementation it is obvious that the final result (CRC value) is dependent, in a very complex way, on the past history of the bits presented. Therefore, it would take an extremely rare combination of errors to escape detection by this method.

The example in Figure 3 calculates the CRC value after each data bit is presented. The shift register circuit is always reset to 0's at the start of the calculation. The computation begins with the LSB of the 64–bit ROM, which is the 02 Hex family code in this example. After all 56 data bits (serial number + family code) are input, the value that is contained in the shift register is A2 Hex, which is the DOW CRC value for that input stream. If the CRC value which has been calculated (A2 Hex in this example), is now used as input to the shift register for the next eight bits of data, the final result in the shift register after the entire 64 bits of data have been entered should be all 0's. This property is always true for the DOW CRC algorithm. If any 8–bit value that appears in the shift register is also used as the next eight bits in the input stream, then the result that appears in the shift register after the 8th data bit has been shifted in is always 00 Hex. This can be explained by observing that the contents of the 8th stage of the shift register is always equal to the incoming data bit, making the output of the EXOR gate controlling the feedback and the next state value of the first stage of the shift register always equal to a logic 0. This causes the shift register to simply shift in 0's from left to right as each data bit is presented, until the entire register is filled with 0's after the 8th bit. The structure of the Dallas Semiconductor 1–Wire 64–bit ROM uses this property to simplify the hardware design of a device used to read the ROM. The shift register in the host is cleared and then the 64 ROM bits are read, including the CRC value. If a correct read has occurred, the shift register is again all 0's which is an easy condition to detect. If a non-zero value remains in the shift register, the read operation must be repeated.

DALLAS 1–WIRE 8–BIT CRC Figure 2

$$\text{Polynomial} = X^8 + X^5 + X^4 + 1$$
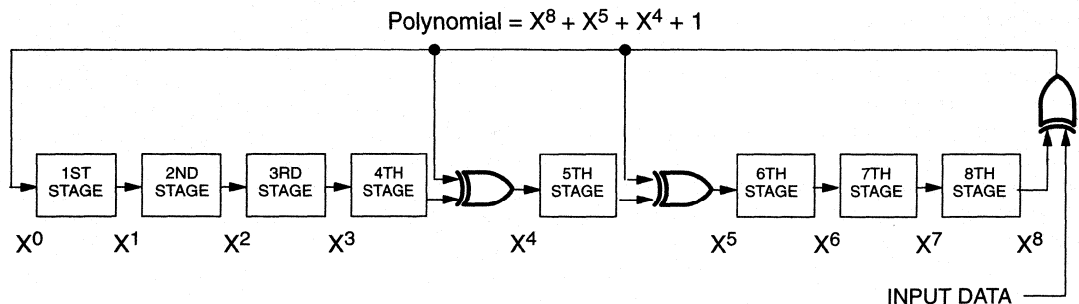
Until now, the discussion has centered around a hardware representation of the CRC process, but clearly a software solution that parallels the hardware methodology is another means of computing the DOW CRC values. An example of how to code the procedure is given in Table 1. Notice that the XRL (exclusive or) of the A register with the constant 18 Hex is due to the presence of the EXOR feedback gates in the DOW CRC after the fourth and fifth stages as shown in Figure 2. An alternative software solution is to simply build a lookup table that is accessed directly for any 8–bit value currently stored in the CRC register and any 8–bit pattern of new data. For the simple case where the current value of the CRC register is 00 Hex, the 256 different bit combinations for the input byte can be evaluated and stored in a matrix, where the index to the matrix is equal to the value of the input byte (i.e., the index will be I = 0–255). It can be shown that if the current value of the CRC register is not 00 Hex, then for any current CRC value and any input byte, the lookup table values would be the same as for the simplified case, but the computation of the index into the table would take the form of:

New CRC = Table [I] for I=0 to 255 ;
where I = (Current CRC) EXOR (Input byte)

For the case where the current CRC register value is 00 Hex, the equation reduces to the simple case. This second approach can reduce computation time since the operation can be done on a byte basis, rather than the bit-oriented commands of the previous example. There is a memory capacity tradeoff, however, since the lookup table must be stored and will consume 256 bytes compared to virtually no storage for the first example except for the program code. An example of this type of code is shown in Table 2. Figure 4 shows the previous example repeated using the lookup table approach. Two properties of the DOW CRC can be helpful in debugging code used to calculate the CRC values. The first property has already been mentioned for the hardware implementation. If the current value of the CRC register is used as the next byte of data, the resulting CRC value will always be 00 Hex (see explanation above). A second property that can be used to confirm proper operation of the code is to enter the 1's complement of the current value of the CRC register. For the DOW CRC algorithm, the resulting CRC value will always be 35 Hex, or 53 Decimal. The reason for this can be explained by observing the operation of the CRC register as the 1's complement data is entered, as shown in Figure 5.

ASSEMBLY LANGUAGE PROCEDURE Table 1

```
DO_CRC:     PUSH    ACC             ;save accumulator
            PUSH    B               ;save the B register
            PUSH    ACC             ;save bits to be shifted
            MOV     B,#8            ;set shift = 8 bits ;

CRC_LOOP:   XRL     A,CRC           ;calculate CRC
            RRC     A               ;move it to the carry
            MOV     A,CRC           ;get the last CRC value
            JNC     ZERO            ;skip if data = 0
            XRL     A,#18H          ;update the CRC value
                                    ;

ZERO:       RRC     A               ;position the new CRC
            MOV     CRC,A           ;store the new CRC
            POP     ACC             ;get the remaining bits
            RR      A               ;position the next bit
            PUSH    ACC             ;save the remaining bits
            DJNZ    B,CRC_LOOP      ;repeat for eight bits
            POP     ACC             ;clean up the stack
            POP     B               ;restore the B register
            POP     ACC             ;restore the accumulator
            RET
```

EXAMPLE CALCULATION FOR DOW CRC Figure 3

Complete 64–Bit 1–Wire ROM Code: A2   00 00 00 01 B8 1C 02

| Family Code: | 0 | 2 | Hex |
| | 0000 | 0010 | Binary |

| Serial Number: | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | B | 8 | 1 | C | Hex |
| | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0001 | 1011 | 1000 | 0001 | 1100 | Binary |

CRC VALUE	INPUT VALUE	
00000000	0	
00000000	1	
10001100	0	2
01000110	0	_____
00100011	0	
10011101	0	
11000010	0	0
01100001	0	_____
10111100	0	
01011110	0	
00101111	1	C
00010111	1	_____
00001011	1	
00000101	0	
10001110	0	1
01000111	0	_____
10101111	0	
11011011	0	
11100001	0	8
11111100	1	_____
11110010	1	
11110101	1	
01111010	0	B
00111101	1	_____
00011110	1	
10000011	0	
11001101	0	1
11101010	0	_____
01110101	0	
10110110	0	
01011011	0	0
10100001	0	_____
11011100	0	
01101110	0	
00110111	0	0
10010111	0	_____
11000111	0	
11101111	0	
11111011	0	0
11110001	0	_____
11110100	0	
01111010	0	
00111101	0	0
10010010	0	_____
01001001	0	
10101000	0	
01010100	0	0
00101010	0	_____
00010101	0	
10000110	0	
01000111	0	0
10101101	0	_____
11011010	0	
01101101	0	
10111010	0	0
01011101	0	_____

10100010 = A2 Hex = CRC Value for [00000001B81C (Serial Number) + 02 (Family Code)]

CRC VALUE	INPUT VALUE	
10100010	0	
01010001	1	
00101000	0	2
00010100	0	___
00001010	0	
00000101	1	
00000010	0	A
00000001	1	___

00000000 = 00 Hex = CRC Value for A2 [(CRC) + 00000001B81C (Serial Number) + 02 (Family Code)]

DOW CRC LOOKUP FUNCTION Table 2

Var
    CRC : Byte;

Procedure Do_CRC(X: Byte);
{
    This procedure calculates the cumulative Dallas Semiconductor 1–Wire CRC of all bytes passed to it. The result accumulates in the global variable CRC.
}
Const
    Table : Array[0..255] of Byte = (

```
  0,   94,  188,  226,   97,   63,  221,  131,  194,  156,  126,   32,  163,  253,   31,   65,
157,  195,   33,  127,  252,  162,   64,   30,   95,    1,  227,  189,   62,   96,  130,  220,
 35,  125,  159,  193,   66,   28,  254,  160,  225,  191,   93,    3,  128,  222,   60,   98,
190,  224,    2,   92,  223,  129,   99,   61,  124,   34,  192,  158,   29,   67,  161,  255,
 70,   24,  250,  164,   39,  121,  155,  197,  132,  218,   56,  102,  229,  187,   89,    7,
219,  133,  103,   57,  186,  228,    6,   88,   25,   71,  165,  251,  120,   38,  196,  154,
101,   59,  217,  135,    4,   90,  184,  230,  167,  249,   27,   69,  198,  152,  122,   36,
248,  166,   68,   26,  153,  199,   37,  123,   58,  100,  134,  216,   91,    5,  231,  185,
140,  210,   48,  110,  237,  179,   81,   15,   78,   16,  242,  172,   47,  113,  147,  205,
 17,   79,  173,  243,  112,   46,  204,  146,  211,  141,  111,   49,  178,  236,   14,   80,
175,  241,   19,   77,  206,  144,  114,   44,  109,   51,  209,  143,   12, ' 82,  176,  238,
 50,  108,  142,  208,   83,   13,  239,  177,  240,  174,   76,   18,  145,  207,   45,  115,
202,  148,  118,   40,  171,  245,   23,   73,    8,   86,  180,  234,  105,   55,  213,  139,
 87,    9,  235,  181,   54,  104,  138,  212,  149,  203,   41,  119,  244,  170,   72,   22,
233,  183,   85,   11,  136,  214,   52,  106,   43,  117,  151,  201,   74,   20,  246,  168,
116,   42,  200,  150,   21,   75,  169,  247,  182,  232,   10,   84,  215,  137,  107,   53);
```

Begin
    CRC := Table[CRC xor X];
End;

TABLE LOOKUP METHOD FOR COMPUTING DOW CRC Figure 4

Current CRC Value (= Current Table Index)	Input Data	New Index (= Current CRC xor Input Data)	Table (New Index) (= New CRC Value)
0000 0000 = 00 Hex	0000 0010 = 02 Hex	(00 H xor 02 H) = 02 Hex = 2 Dec	Table[2]= 1011 1100 = BC Hex = 188 Dec
1011 1100 = BC Hex	0001 1100 = 1C Hex	(BC H xor 1C H) = A0 Hex = 160 Dec	Table[160]= 1010 1111 = AF Hex = 175 Dec
1010 1111 = AF Hex	1011 1000 = B8 Hex	(AF H xor B8 H) = 17 Hex = 23 Dec	Table[23]= 0001 1110 = 1E Hex = 30 Dec
0001 1110 = 1E Hex	0000 0001 = 01 Hex	(1E H xor 01 H) = 1 F Hex = 31 Dec	Table[31]= 1101 110 = DC Hex = 220 Dec
1101 1100 = DC Hex	0000 0000 = 00 Hex	(DC H xor 00 H) = DC Hex = 220 Dec	Table[220]= 1111 0100 = F4 Hex = 244 Dec
11110100 = F4 Hex	0000 0000 = 00 Hex	(F4 H xor 00 H) = F4 Hex = 244 Dec	Table [244]= 0001 0101 = 15 Hex = 21 Dec
0001 0101 = 15 Hex	0000 0000 = 00 Hex	(15 H xor 00 H) = 15 Hex = 21 Dec	Table[21]= 1010 0010 = A2 Hex = 162 Dec
1010 0010 = A2 Hex	10100010 = A2 Hex	(A2 H xor A2 H) = Hex = 0 Dec	Table[0]=0000 0000 = 00 Hex = 0 Dec

CRC REGISTER COMBINED WITH 1'S COMPLEMENT OF CRC REGISTER Figure 5

CRC Register Value **Input**

X_0	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_7^*
1	X_0	X_1	X_2	X_3^*	X_4^*	X_5	X_6	X_6^*
1	1	X_0	X_1	X_2^*	X_3	X_4^*	X_5	X_5^*
1	1	1	X_0	X_1^*	X_2^*	X_3	X_4^*	X_4^*
0	1	1	1	X_0	X_1^*	X_2	X_3	X_3^*
1	0	1	1	0	$X0^*$	$X1^*$	$X2$	$X2^*$
1	1	0	1	0	1	$X0^*$	$X1^*$	$X1^*$
0	1	1	0	1	0	1	$X0^*$	$X0^*$
0	0	1	1	0	1	0	1	Final CRC Value = 35 Hex, 53 Decimal

Note: X_i^* = Complement of Xi

CRC–16 COMPUTATION FOR RAM RECORDS IN TOUCH MEMORIES

As mentioned in the introduction, some Touch Memory devices have RAM in addition to the unique 8–byte ROM code found in all Touch Memories. Because the amount of data stored in RAM can be large compared to the 8–byte ROM code, Dallas Semiconductor recommends using a 16–bit CRC value to ensure the integrity of the data, rather than the 8–bit DOW CRC used for the ROM. The particular CRC suggested is commonly referred to as CRC–16. The shift register and polynomial representations are given in Figure 6. The figure shows that for a 16–bit CRC, the shift register will contain 16

stages and the polynomial expression will have a term of the sixteenth order. As stated previously, the Touch Memory devices do not calculate the CRC values. The host must generate the value and then append the 16-bit CRC value to the end of the actual data. Due to the uncertainty of the Touch Memory's "communication channel," i.e., the two metal contact surfaces, data transfers can experience errors that generally fall into three categories. First, brief intermittent connections can cause small numbers of bit errors to occur in the data, which the normal CRC–16 function is designed to detect. The second type of error occurs when contact is lost altogether, for example when the Touch Memory is

removed from the reader too quickly. This causes the last portion of the data to be read as logic 1's, since no connection to a Touch Memory will be interpreted as all 1's by the host. The normal CRC–16 function can also detect this condition under most circumstances. The third type of error is generated by a short circuit across the reader, which can be caused by a Touch Memory that is not inserted correctly, or tilted significantly once in the reader. A short at the reader causes the data to be read as all 0's by the host. When using CRCs, this can cause problems, since the method to determine the validity of the data is to read the data plus the stored CRC value, and see if the resulting CRC computed at the host is 0000 Hex (for a 16–bit CRC.) If the reader was shorted, the data plus the CRC value stored with the data will be read as all 0's, and a false read will have occurred, but the CRC computed by the host will incorrectly indicate a valid read. In order to avoid this situation, Dallas Semiconductor recommends storing the complement of the computed CRC–16 value (CRC–16*) with the data that is written into the RAM. Using an uncomplemented CRC–16 value, the retrieval of data from the Touch Memory is similar to the DOW CRC case. That is, if the CRC register in the host is initialized to 0000 Hex and then all of the data plus the CRC–16

value stored with the data is read from the Touch Memory, the resulting calculation by the host should have a 0000 Hex, as a final result. If instead, the complement of the CRC–16 value is stored with the data in the Touch Memory, then the CRC register at the host is initialized to 0000 Hex and the actual data plus the stored CRC–16* value is read. The resultant CRC value should be B001 Hex for a valid read. This greatly improves the operation of the system, since it can no longer be fooled by a short at the reader. The reason that the CRC–16 function has these properties can be shown in an analogous manner to the DOW CRC case (see Figures 3 and 5). The operation of the 16–bit CRC is identical in theory to the 8 bit version described earlier, but the properties of the CRC change since a 16–bit value is now available for error detection. For the CRC–16 function, the types of errors that are detectable are:

1. Any odd number of errors anywhere within the data record.
2. All double–bit errors anywhere within the data record.
3. Any cluster of errors that can be contained within a 16–bit "window" (1–16–bits incorrect).
4. Most larger clusters of errors.

CRC–16 HARDWARE DESCRIPTION AND POLYNOMIAL Figure 6

Polynomial $= X^{16} + X^{15} + X^2 + 1$
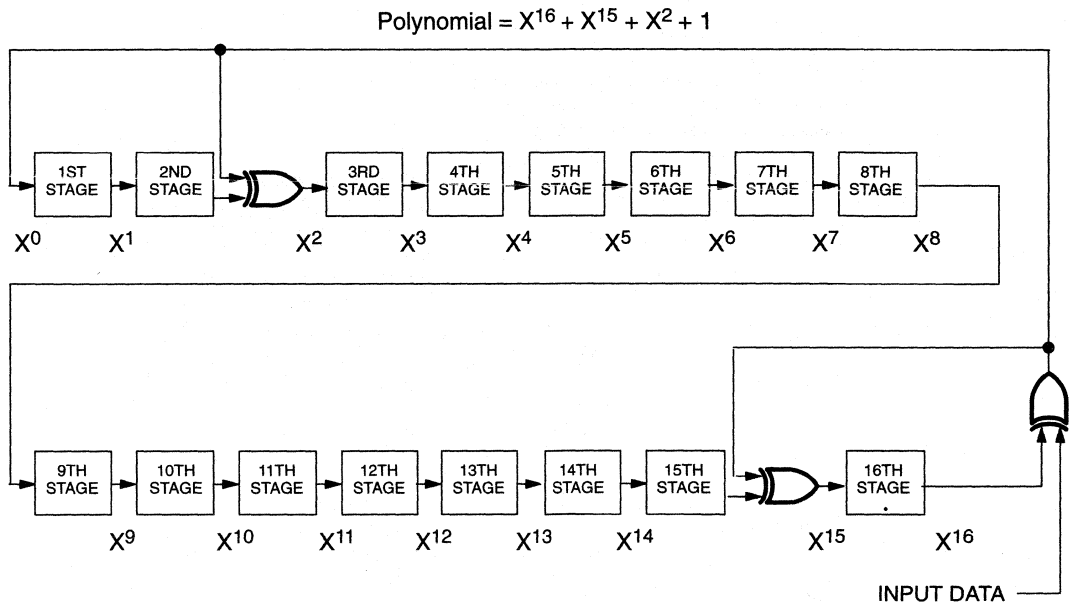
The hardware implementation of the CRC–16 function is straightforward from the description given in Figure 6. Table 3 shows a software solution that is analogous to the hardware operations which compute the CRC–16 values using single–bit operations. As before, a less computation–intensive software solution can be developed through the use of a lookup table. The basic concepts presented for the 8 bit DOW CRC lookup table also work for the CRC–16 case. A slight modification in procedure from the 8–bit case is required, however, because if the entire 16–bit result for the CRC–16 function were mapped into one table as before, the table would have 2^{16} or 65536 entries. A different approach is shown in Table 4, where the 16–bit CRC values are computed and stored in two 256–entry tables, one containing the high order byte and the other the low order byte of the resultant CRC. For any current 16–bit CRC value, expressed as Current_CRC16_Hi for the current high order byte and Current _CRC16_Lo for the current low order byte, and any new input byte, the equation to determine the index into the high order byte table for locating the new high order byte CRC value (New_CRC16_Hi) is given as:

New_CRC16_Hi = CRC16_Tabhi[I] for I=0 to 255; where I = (Current_CRC16_Lo) EXOR (Input byte)

The equation to determine the index into the low order byte table for locating the new low order byte CRC value (New_CRC16_Lo) is given as:

New_CRC16_Lo = (CRC16_Tablo[I]) EXOR (Current_CRC16_Hi) for I=0 to 255; where I = (Current_CRC16_Lo) EXOR (Input byte)

An example of how this method works is shown in Figure 7.

ASSEMBLY LANGUAGE FOR CRC–16 COMPUTATION Table 3

```
crc_lo          data      20h                          ; lo byte of crc calculation (bit addressable)
crc_hi          data      21h                          ; hi part of crc calculation



                                              •

                                              •

                                              •


;--------------------------------------------------------
;               CRC16 subroutine.
;               - accumulator is assumed to have byte to be crc'ed
;               - two direct variables are used crc_hi and crc_lo
;               - crc_hi and crc_lo contain the CRC16 result
;--------------------------------------------------------
crc16:                                                  ; calculate crc with accumulator
                push     b                              ; save value of b
                mov      b,          #08h               ; number of bits to crc.
crc_get_bit:
                rrc      a                              ; get low order bit into carry
                push     acc                            ; save a for later use
                jc       crc_in_1                       ;got a 1 input to crc
                mov      c,          crc_lo.0           ;xor with a 0 input bit is bit
                sjmp     crc_cont                       ;continue
crc_in_1:
                mov      c,          crc_lo.0           ;xor with a 1 input bit
                cpl      c                              ;is not bit.
crc_cont:
                jnc      crc_shift                      ; if carry set, just shift
                cpl      crc_hi.6                       ;complement bit 15 of crc
                cpl      crc_lo.1                       ;complement bit 2 of crc
crc_shift
```

```
        mov     a,              crc_hi          ; carry is in appropriate setting
        rrc     a                               ; rotate it
        mov     crc_hi,         a               ; and save it
        mov     a,              crc_lo          ; again, carry is okay
        rrc     a                               ; rotate it
        mov     crc_lo,         a               ; and save it

        pop     acc                             ; get acc back
        djnz    b,              crc_get_bit     ; go get the next bit

        pop     b                               ; restore b
        ret

        end
```

ASSEMBLY LANGUAGE FOR CRC–16 USING A LOOKUP TABLE Table 4

```
crc_lo                    data            40h          ; any direct address is okay
crc_hi                    data            41h
tmp                       data            42h
```

●

●

●

```
;-------------------------------------------------------
;                    CRC16 subroutine.
;                    - accumulator is assumed to have byte to be crc'ed
;                    - three direct variables are used, tmp, crc_hi and crc_lo
;                    - crc_hi and crc_lo contain the CRC16 result
;                    - this CRC16 algorithm uses a table lookup
;-------------------------------------------------------
crc16:
              xrl       a,              crc_lo           ; create index into tables
              mov       tmp,            a                ; save index
              push      dph                              ; save dptr
              push      dpl                              ;
              mov       dptr,           #crc16_tablo     ; low part of table address
              movc      a,              @a+dptr          ; get low byte
              xrl       a,              crc_hi           ;
              mov       crc_lo,         a                ; save of low result

              mov       dptr,           #crc16_tabhi     ; high part of table address
              mov       a,              tmp              ; index
              movc      a,              @a+dptr          ;
              mov       crc_hi,         a                ; save high result

              pop       dpl                              ; restore pointer
              pop       dph                              ;
              ret                                        ; all done with calculation
crc16_tablo:
              db        000h, 0c1h, 081h, 040h, 001h, 0c0h, 080h, 041h
              db        001h, 0c0h, 080h, 041h, 000h, 0c1h, 081h, 040h
              db        001h, 0c0h, 080h, 041h, 000h, 0c1h, 081h, 040h
              db        000h, 0c1h, 081h, 040h, 001h, 0c0h, 080h, 041h
              db        001h, 0c0h, 080h, 041h, 000h, 0c1h, 081h, 040h
              db        000h, 0c1h, 081h, 040h, 001h, 0c0h, 080h, 041h
              db        000h, 0c1h, 081h, 040h, 001h, 0c0h, 080h, 041h
              db        001h, 0c0h, 080h, 041h, 000h, 0c1h, 081h, 040h
              db        001h, 0c0h, 080h, 041h, 000h, 0c1h, 081h, 040h
              db        000h, 0c1h, 081h, 040h, 001h, 0c0h, 080h, 041h
              db        000h, 0c1h, 081h, 040h, 001h, 0c0h, 080h, 041h
              db        001h, 0c0h, 080h, 041h, 000h, 0c1h, 081h, 040h
              db        000h, 0c1h, 081h, 040h, 001h, 0c0h, 080h, 041h
              db        001h, 0c0h, 080h, 041h, 000h, 0c1h, 081h, 040h
              db        001h, 0c0h, 080h, 041h, 000h, 0c1h, 081h, 040h
              db        000h, 0c1h, 081h, 040h, 001h, 0c0h, 080h, 041h
              db        001h, 0c0h, 080h, 041h, 000h, 0c1h, 081h, 040h
              db        000h, 0c1h, 081h, 040h, 001h, 0c0h, 080h, 041h
              db        000h, 0c1h, 081h, 040h, 001h, 0c0h, 080h, 041h
              db        001h, 0c0h, 080h, 041h, 000h, 0c1h, 081h, 040h
              db        000h, 0c1h, 081h, 040h, 001h, 0c0h, 080h, 041h
```

```
        db      001h, 0c0h, 080h, 041h, 000h, 0c1h, 081h, 040h
        db      001h, 0c0h, 080h, 041h, 000h, 0c1h, 081h, 040h
        db      000h, 0c1h, 081h, 040h, 001h, 0c0h, 080h, 041h
        db      000h, 0c1h, 081h, 040h, 001h, 0c0h, 080h, 041h
        db      001h, 0c0h, 080h, 041h, 000h, 0c1h, 081h, 040h
        db      001h, 0c0h, 080h, 041h, 000h, 0c1h, 081h, 040h
        db      000h, 0c1h, 081h, 040h, 001h, 0c0h, 080h, 041h
        db      001h, 0c0h, 080h, 041h, 000h, 0c1h, 081h, 040h
        db      000h, 0c1h, 081h, 040h, 001h, 0c0h, 080h, 041h
        db      000h, 0c1h, 081h, 040h, 001h, 0c0h, 080h, 041h
        db      001h, 0c0h, 080h, 041h, 000h, 0c1h, 081h, 040h

crc16_tabhi:
        db      000h, 0c0h, 0c1h, 001h, 0c3h, 003h, 002h, 0c2h
        db      0c6h, 006h, 007h, 0c7h, 005h, 0c5h, 0c4h, 004h
        db      0cch, 00ch, 00dh, 0cdh, 00fh, 0cfh, 0ceh, 00eh
        db      00ah, 0cah, 0cbh, 00bh, 0c9h, 009h, 008h, 0c8h
        db      0d8h, 018h, 019h, 0d9h, 01bh, 0dbh, 0dah, 01ah
        db      01eh, 0deh, 0dfh, 01fh, 0ddh, 01dh, 01ch, 0dch
        db      014h, 0d4h, 0d5h, 015h, 0d7h, 017h, 016h, 0d6h
        db      0d2h, 012h, 013h, 0d3h, 011h, 0d1h, 0d0h, 010h
        db      0f0h, 030h, 031h, 0f1h, 033h, 0f3h, 0f2h, 032h
        db      036h, 0f6h, 0f7h, 037h, 0f5h, 035h, 034h, 0f4h
        db      03ch, 0fch, 0fdh, 03dh, 0ffh, 03fh, 03eh, 0feh
        db      0fah, 03ah, 03bh, 0fbh, 039h, 0f9h, 0f8h, 038h
        db      028h, 0e8h, 0e9h, 029h, 0ebh, 02bh, 02ah, 0eah
        db      0eeh, 02eh, 02fh, 0efh, 02dh, 0edh, 0ech, 02ch
        db      0e4h, 024h, 025h, 0e5h, 027h, 0e7h, 0e6h, 026h
        db      022h, 0e2h, 0e3h, 023h, 0e1h, 021h, 020h, 0e0h
        db      0a0h, 060h, 061h, 0a1h, 063h, 0a3h, 0a2h, 062h
        db      066h, 0a6h, 0a7h, 067h, 0a5h, 065h, 064h, 0a4h
        db      06ch, 0ach, 0adh, 06dh, 0afh, 06fh, 06eh, 0aeh
        db      0aah, 06ah, 06bh, 0abh, 069h, 0a9h, 0a8h, 068h
        db      078h, 0b8h, 0b9h, 079h, 0bbh, 07bh, 07ah, 0bah
        db      0beh, 07eh, 07fh, 0bfh, 07dh, 0bdh, 0bch, 07ch
        db      0b4h, 074h, 075h, 0b5h, 077h, 0b7h, 0b6h, 076h
        db      072h, 0b2h, 0b3h, 073h, 0b1h, 071h, 070h, 0b0h
        db      050h, 090h, 091h, 051h, 093h, 053h, 052h, 092h
        db      096h, 056h, 057h, 097h, 055h, 095h, 094h, 054h
        db      09ch, 05ch, 05dh, 09dh, 05fh, 09fh, 09eh, 05eh
        db      05ah, 09ah, 09bh, 05bh, 099h, 059h, 058h, 098h
        db      088h, 048h, 049h, 089h, 04bh, 08bh, 08ah, 04ah
        db      04eh, 08eh, 08fh, 04fh, 08dh, 04dh, 04ch, 08ch
        db      044h, 084h, 085h, 045h, 087h, 047h, 046h, 086h
        db      082h, 042h, 043h, 083h, 041h, 081h, 080h, 040h
```

COMPARISON OF CALCULATION AND TABLE LOOKUP METHOD FOR CRC–16 Figure 7

Example:
CRC register starting value: 90 F1 Hex
Input Byte: 75 Hex

Calculation Method	
Current CRC Value	Input
1001 0000 1111 0001	
0100 1000 0111 1000	1
0010 0100 0011 1100	0
1011 0010 0001 1111	1
1111 1001 0000 1110	0
1101 1100 1000 0110	1
1100 1110 0100 0010	1
1100 0111 0010 0000	1
0110 0011 1001 0000	0
New CRC Value = 63 90 Hex	

Table Lookup Method

Current_CRC16_Lo = F1 Hex
Current_CRC16_Hi = 90 Hex
Input byte = 75 Hex

Tabhi Index= (Current_CRC16_Lo) EXOR (Input byte)
$\quad$ = F1 EXOR 75= 84 Hex = 132 Dec
New_CRC16_Hi = Tabhi[132] = 63 Hex (from Table 4.)

Tablo Index = (Current_CRC16_Lo) EXOR (Input byte) = 132 Dec
Tablo[132] = 00 Hex (from Table 4.)
New_CRC16_Lo = Tablo[132] EXOR (Current_CRC16_Hi)
$\quad$ = 00 EXOR 90 = 90 Hex

New CRC Value = 63 90 Hex

An interesting intermediate method is presented in Table 5. The code will generate a CRC–16 value for each byte input to it by operating on the entire current CRC value and the incoming byte using the equations shown in Figure 8. The derivations for the equations are also shown, using alpha characters to represent the current 16–bit CRC value and numeric characters to represent the bits of the incoming byte. The result after eight shifts yields the equations shown. These equations can then be used to precompute large portions of the new CRC value. Notice, for example, that the quantity ABCDEFGH01234567 (defined as the EXOR of all of those bits) is the parity of the incoming data byte and the low order byte of the current CRC. This method reduces computation time and memory space as compared to both the bit–by–bit and lookup table methods described above. Finally, two properties of the CRC–16 function that can be used as test cases are mentioned as an aid to debugging the code for any of the previous methods.

The first property is identical to the DOW CRC case. If the current 16–bit contents of the CRC register are also used as the next 16–bits of input, the resulting CRC value is always 0000 Hex. A second property of the CRC–16 function is also similar to the DOW CRC case, if the 1's complement of the current 16–bit contents of the CRC register are also used as the next 16–bits of input, the resulting CRC value is always B0 01 Hex. The proof for these two CRC–16 properties follows in an analogous way to the proof for the DOW CRC case.

REFERENCES:
Stallings, William, Ph.D., Data and Computer Communications. 2nd ed., New York: Macmillan Publishing. 107-112.

Buller, Jon, "High Speed Software CRC Generation", EDN, Volume 36, #25, pg. 210.

ASSEMBLY LANGUAGE PROCEDURE FOR HIGH–SPEED CRC–16 COMPUTATION Table 5

```
lo      equ     40h                             ; low byte of CRC
hi      equ     41h                             ; high byte of CRC

                                •

                                •

                                •

crc16:
        push    acc                             ; save the accumulator.

        xrl     a,      lo
        mov     lo,     hi                      ; move the high byte of the CRC.
        mov     hi,     a                       ; save data xor low(crc) for later
        mov     c,      p
        jnc     crc0
        xrl     lo,     #01h                    ; add the parity to CRC bit 0
crc0:
        rrc     a                               ; get the low bit in c
        jnc     crc1
        xrl     lo,     #40h                    ; need to fix bit 6 of the result
crc1:
        mov     c,      acc.7
        xrl     a,      hi                      ; compute the results for other bits.
        rrc     a                               ; shift them into place
        mov     hi,     a                       ; and save them
        jnc     crc2
        xrl     lo,     #80h                    ; now clean up bit 7
crc2:
        pop     acc                             ; restore everything and return
        ret
```

HIGH–SPEED CRC–16 COMPUTATION METHOD Figure 8

CURRENT CRC VALUE = XWUT SRQP HGFE DCBA
INPUT BYTE = 7654 3210

NOTATION: ABC = A EXOR B EXOR C

DEFINITION: DEF EXOR D = (D EXOR D) EXOR EF = 0 EXOR EF = EF

REGISTER STAGE (SEE FIGURE 6 FOR OPERATION)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	INPUT
X	W	U	T	S	R	Q	P	H	G	F	E	D	C	B	A	
A0	X	WA0	U	T	S	R	Q	P	H	G	F	E	D	C	AB0	0
AB01	A0	AB01X	WA0	U	T	S	R	Q	P	H	G	F	E	D	ABC01	1
ABC012	AB01	BC12	AB01X	WA0	U	T	S	R	Q	P	H	G	F	E	ABCD012	2
ABCD0123	ABC012	CD23	BC12	AB01X	WA0	U	T	S	R	Q	P	H	G	F	ABCDE0123	3
ABCDE01234	ABCD0123	DE34	CD23	BC12	AB01X	WA0	U	T	S	R	Q	P	H	G	ABCDEF01234	4
ABCDEF012345	ABCDE01234	EF45	DE34	CD23	BC12	AB01X	WA0	U	T	S	R	Q	P	H	ABCDEFG012345	5
ABCDEFG0123456	ABCDEF012345	FG56	EF45	DE34	CD23	BC12	AB01X	WA0	U	T	S	R	Q	P	ABCDEFGH0123456	6
ABCDEFGH01234567	ABCDEFG0123456	GH67	FG56	EF45	DE34	CD23	BC12	AB01X	WA0	U	T	S	R	Q	ABCDEFGHP01234567	7

THIS YIELDS THE FOLLOWING DEFINITIONS:

NEW CRC REGISTER VALUES AFTER EIGHT SHIFTS

Xnew = ABCDEFGH01234567
Wnew = ABCDEFG0123456 = ABCDEFGH01234567 H7
Unew = G6H7
Tnew = F5G6
Snew = E4F5
Rnew = D3E4
Qnew = C2D3
Pnew = B1C2
Hnew = A0B1X
Gnew = A0W
Fnew = U
Enew = T
Dnew = S
Cnew = R
Bnew = Q
Anew = P ABCDEFGH01234567

DALLAS
SEMICONDUCTOR

INTRODUCTION

Dallas Semiconductor Touch Memories are designed to operate with a contact range exceeding 100 meters. The devices require a single bidirectional bus master for communication over the 1–Wire™ bus. The ideal connection between a Touch Memory and its bus master is a short cable with low resistance and low capacitance. As the distance between the Touch Memory and the bus master becomes longer, more care must be taken to ensure that the waveforms of the communication signals meet specification. The effects of cable length and lumped capacitance on two different bus master types are described as well as some guidelines for successful operation of Touch Memory at extended cable lengths. The two bus master types are shown in Figure 1. The open drain type with a resistor pullup uses Port 0.3 of the Dallas Semiconductor DS5000 (8051–equivalent) microcontroller as an output and Port 0.4 as an input. The COM Port type uses a DS9097 COM Port Adaptor with a computer that has an Intel 8250–equivalent UART chip to drive the line. This COM Port Adaptor is the type provided in the DS9092K Touch Memory Starter Kit. The cable used is 22 gauge, twisted–pair telephone wire with the characteristics shown in Table 1. A single twisted–pair of conductors out of the 30–pair bundle within the cable were used for bus master evaluation (see Figure 2). The lumped capacitance measurements were made using a capacitance decade box applied in parallel with the Touch Memory (see Figure 3). Since many different wire types may be used for interconnect, the lumped capacitance data provides some ability to estimate operating distances for alternate wire if the characteristics of that alternate wire are known. The lumped capacitance data also allows estimation of total capacitive load capability for non–wire connections and contact surfaces for a particular bus master type. The test software for the DS1991, DS1992, DS1993 and DS1994 executed a Search ROM command and read ten bytes of RAM data from

each device present. The DS1990 was tested using the Read ROM command.

GENERAL GUIDELINES FOR OPERATION OVER EXTENDED DISTANCES
1–Wire Timing

The bus master writes a Touch Memory with a minimum low time of 60 µs for a Write 0 and a maximum low time of 15 µs for a Write 1 (see Figure 4a). The bus master reads a Touch Memory by pulling the data line low then releasing and monitoring the condition of the data line. For the Touch Memory to signal a Read 0 time slot, it continues to hold the data line low for a minimum of 15 µs from the initial falling edge. For the Touch Memory to signal a Read 1 time slot, it allows the data line to return high via the pullup resistor. The pullup resistor must return the data line to a logic high level within 15 µs of the initial falling edge (see Figure 4b). The recovery time between time slots must be a minimum of 1 µs. These timing values result in the highest possible data rate. (For more details on 1–Wire timing, see the Dallas Semiconductor Automatic Identification Data Book or Book of DS199x Touch Memory Standards). For short cable lengths and small capacitive loads, this timing can be achieved with a pullup resistor in the 5KΩ to 10KΩ range. Using a resistor in this range maximizes design margin by providing a very small DC load to the Touch Memory and therefore a strong logic low level to the input side of the bidirectional bus master. As cable length becomes longer, the capacitive load presented to the bus master increases and the RC time constant associated with the resistor pulling the line high after the active pulldown transistor releases begins to exceed the time slot conditions for correct operation. Using the lowest capacitance cable available will provide the greatest operating distances for a given pullup value. The twisted–pair cable that was used for testing is approximately 52 pF/m.

072996 1/11

TOUCH MEMORY BUS MASTER TYPES Figure 1
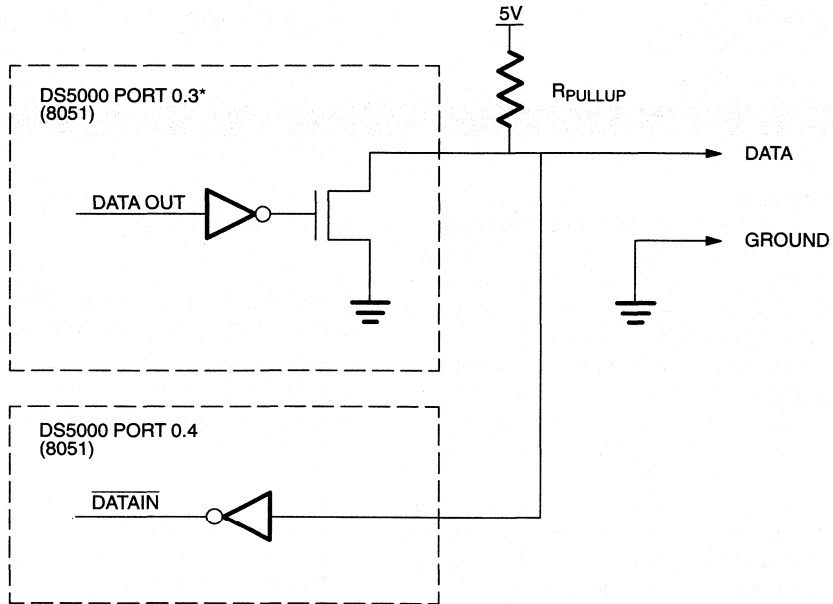
a) Open Drain Output with Resistor Pullup
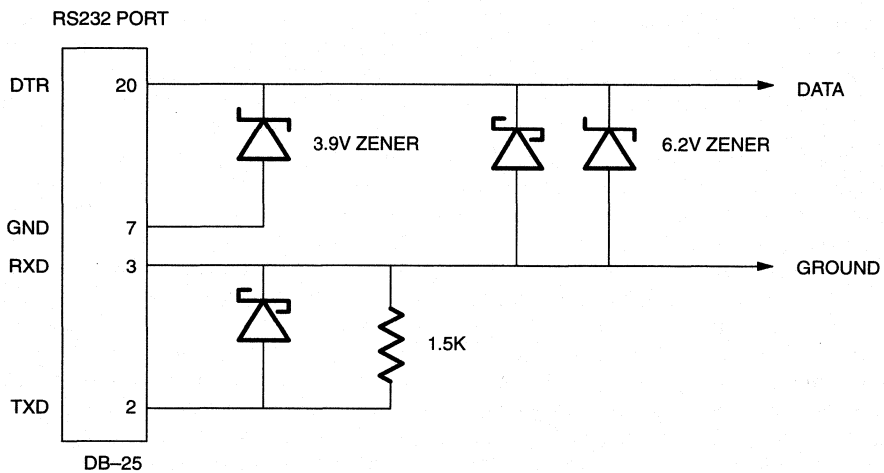
b) DS9097 COM port Adaptor

* Substitution of other port pins that are not open drain but have active pull–ups that are pulsed on briefly may cause unreliable operation or nonfunctionality.

CABLE CHARACTERISTICS – 22 GAUGE PLASTIC INSULATED CABLE AT 25°C Table 1

FREQ Hz	R ohm/km	L mH/km	G μMHO/km	C μF/km	Z$_O$ ohms		GAMMA	
							dB	deg/km
1	108	0.6128	0.000	0.052	12927	−J*12924	0.037	0.24
100	108	0.6128	0.006	0.052	1295	−J*1290	0.363	2.40
1K	108	0.6125	0.047	0.052	416	−J*402	1.130	7.7
10K	109	0.6092	0.405	0.052	154	−J*109	3.071	28.6
100K	134	0.5740	3.498	0.052	107	−J*19.3	5.426	199
1M	362	0.4940	30.192	0.052	98	−J*5.70	16.076	1820

FANOUT TEST SETUP (OPEN DRAIN W/RESISTOR BUS MASTER) Figure 2
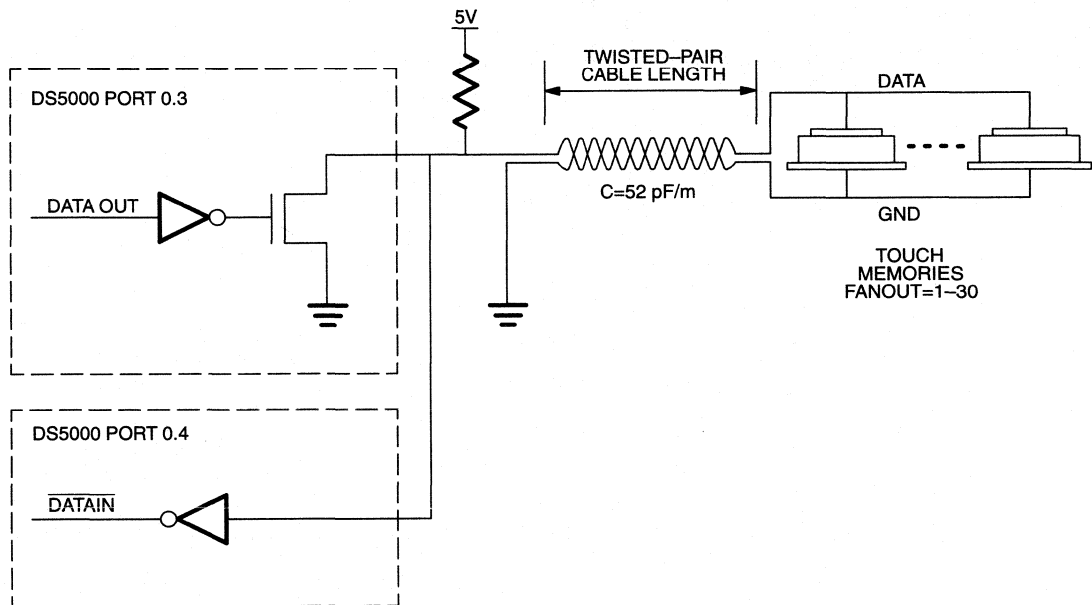
LUMPED CAPACITANCE TEST SETUP (OPEN DRAIN W/RESISTOR BUS MASTER) Figure 3
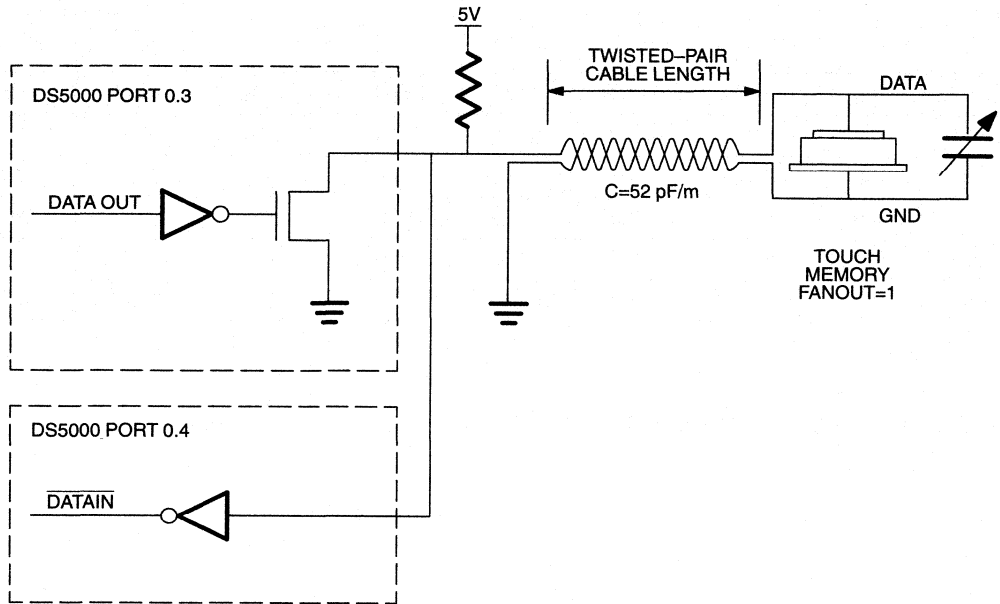
MAXIMUM DATA RATE 1 – WIRE TIMING Figure 4

WRITE 0 TIME SLOT   WRITE 1 TIME SLOT

t_{LOW0}

t_{LOW0} MIN.=60 μs

t_{LOW1}

t_{LOW1} MAX.=15 μs

RECOVERY TIME
MINIMUM=1 μs

—— ACTIVE PULLDOWN
FROM BUS MASTER

- - - - RESISTOR PULLUP
FROM BUS MASTER
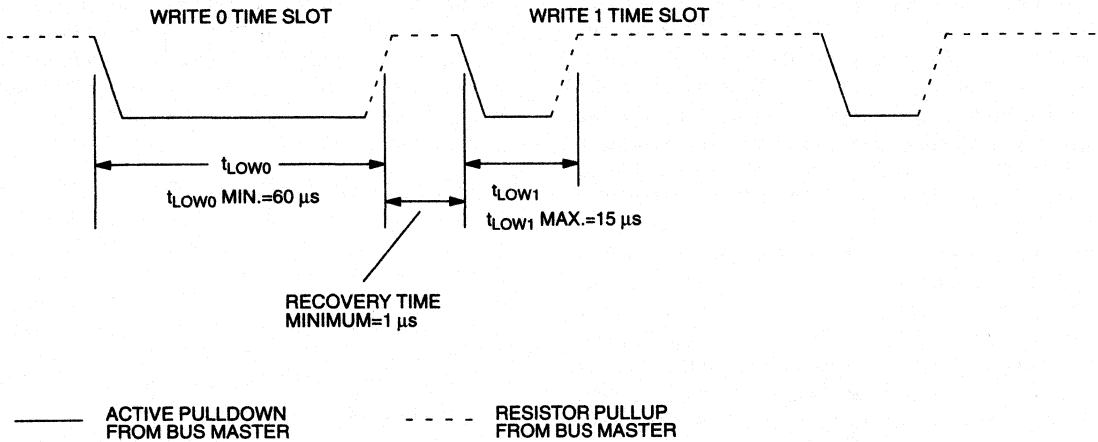
a) 1–Wire Timing – Write Time Slots

READ 0 TIME SLOT   READ 1 TIME SLOT

t_{RDV}

READ 0 MIN.=15 μs

t_{RDV}

READ 1 MAX.=15 μs

RECOVERY TIME
MINIMUM=1 μs

—— ACTIVE PULLDOWN
FROM BUS MASTER

- - - - RESISTOR PULLUP
FROM BUS MASTER

– – – ACTIVE PULLDOWN
FROM TOUCH MEMORY
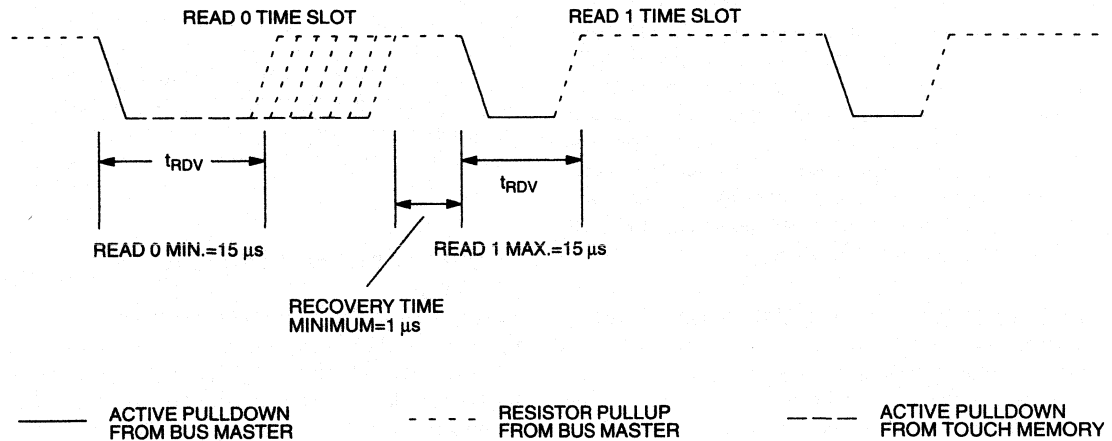
b) 1–Wire Timing – Read Time Slots

Cable Length Effects on Read 1 Time Slots

The Read 1 time slot begins with an active device in the bus master pulling the line low and a resistor returning the line to a high level. A logic high level must be reached within 15 µs of the falling edge in order to meet the 1–Wire timing specification for a logical 1. If the rise time becomes too slow due to loading, the desired logical 1 will be interpreted as being a logical 0 by the input stage of the bus master (see Figure 5). The pullup resistor value in the bus master can be reduced to improve the recovery time for large capacitive loads. The lower limit of the resistor should be approximately 1150 ohms. This value is derived from the Touch Memory current sink capability of 4 mA at 0.4 volts. For a 5–volt system, the resistor will have 4.6 volts across it with 4 mA of current flowing through it resulting in a minimum value of 4.6/4 mA or 1150 ohms (see Figure 6). As this resistor value is further decreased the guaranteed noise margins for logic low levels are compromised. In general, the largest pullup resistor value that still provides sufficient operating margin at the desired distance should be used.

Additional improvement can be realized by releasing the active pulldown device in the bus master as quickly as possible after the data line is driven low. The minimum amount of time that the data line must be held low by the bus master is 1 µs in order to meet the 1–Wire timing specifications. In practice the low time might be several microseconds due to the speed at which the port pin of the microcontroller or microprocessor can operate. The low time must also be sufficiently long to allow a Touch Memory at the end of a long cable to interpret the low–going pulse correctly even though it may have been degraded significantly. For the data that is presented, the bus master pulldown was released 6.5 µs after it was driven low for the open drain type.

Cable Length Effects on Write 0 Time Slots

The 60 µs Write 0 time slot begins with an active device in the bus master pulling the data line low and a resistor returning the line to a high level. At least two consecutive Write 0 pulses (as in the case of the Search ROM command byte F0h) separated by the minimum 1 µs recovery time can be degraded sufficiently over a long cable length that eventually the command byte is not interpreted correctly by the Touch Memory and failure occurs. Figure 7 shows the filtering effect that the long cable can have on the recovery time. Increasing the amount of recovery time between time slots will extend the length of cable over which communication can

occur. By increasing the recovery time from the 1 µs minimum to 15 µs, the worst case condition will always be determined by the Read 1 time slot and failure will occur there first. Increasing the recovery time to 15 µs will reduce the data rate slightly, from 16.3Kbits/s (60 µs + 1 µs) to 13.3Kbits/s (60 µs + 15 µs).

OPEN DRAIN WITH RESISTOR PULLUP

The open drain port pin 0.3 of the DS5000 microcontroller was used with pullup resistor values ranging from 1KΩ to 10KΩ. The input stage of the bus master used port 0.4 of the DS5000, which is a TTL–compatible input that sampled the data line voltage level approximately 14 µs after the line had been driven low. Each Touch Memory device type was tested and the results show that a single device of any type can be read at cable lengths exceeding 300 m with a pullup resistor value of 1KΩ. Some reduction in distance occurs as Touch Memory fanout increases, but the 1KΩ pullup can still read 30 devices at 240 m.

The lumped capacitive load that a particular resistor value can drive for a given cable length is also shown. The data indicates that the loading at long distances is directly proportional to the cable capacitance, as expected. For the DS1990 with a 1KΩ resistor, for example, the maximum 0 distance load is 16800 pF and the device also fails at just over 300 m of cable with a load of 1600 pF. The equivalent load at 300 m is (300 m) x (52 pF/m) or 15600 pF for the cable and a lumped load of 1600 pF for a total of 17200 pF. Selecting the lowest capacitance cable available will increase the communication distance almost linearly for a given resistor value.

COM PORT BUS MASTER WITH DS9097 ADAPTOR

A second bus master type is examined which is created by attaching the DS9097 COM Port Adaptor to the RS–232 port of a 16Mhz 286–class PC that has an Intel 8250–equivalent UART chip. The DS9097 COM Port Adaptor is available as a standard product from Dallas Semiconductor. The resulting bus master has the approximate characteristics of a 5 mA current source. This provides operation that is roughly equivalent to the open drain bus master with a 1.5KΩ pullup. The data confirms that the performance of the COM port bus master falls between that of the open drain bus master with a 1.0KΩ pullup resistor and the open drain bus master with a 2.2KΩ pullup resistor.

The performance of the COM port bus master was very consistent for all types of Touch Memory devices. One difference between the microcontroller–based open drain bus master and the PC–based COM port bus master was the recovery time. The timing constraints imposed by the UART coupled with the computational performance of the PC resulted in a time slot of approximately 118 μs (8.5Kbit/s data rate) which included a 58 μs recovery time. This is significantly longer than the 75 μs time slot which included a 15 μs recovery time that the DS5000 microcontroller created. The fanout and capacitive loading results for the two bus masters were similar, however, because the limiting case for both bus master types was a Read 1 time slot appearing as a Read 0 time slot due to slow rise times.

ALTERNATE BUS MASTERS

The simple open drain bus master or COM port bus master was quite adequate for driving several hundred meters of cable with relatively large capacitive loads and/or device fanout, but other bus masters could be developed for a specific application. The input stage of the resistor–pullup bus master is a port pin on the DS5000 which requires normal TTL input levels for correct operation. A more sophisticated non–TTL input stage may provide some additional improvement. A current source pullup or pulsed active pullup may reduce the time required to return the data line high for a Read 1 Time Slot. In general, however, active pull–ups should be scrutinized for excessive ringing caused by sharp signal transitions. Any new bus master design should be fully characterized before it is deployed into the field.

READ 1 TIME SLOT DEGRADATION FROM CABLE LOADING Figure 5
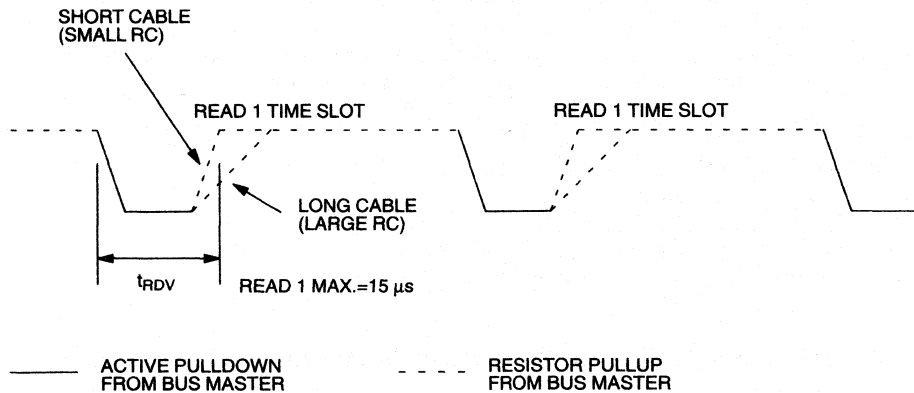
Failure Mode: Pullup at Bus Master is unable to recover data line quickly enough, Read 1 Time Slot rise time increases such that is is misinterpreted by input stage of Bus Master as a Read 0 Time Slot.

Solution: Decrease pullup resistor down to a minimum of 1 kΩ to provide fastest recovery but still maintain adequate logic levels for the Bus Master input. Also minimize time that active pulldown holds data line low to provide greatest amount of time for recovery within the 15 μs Read 1 window.

DETERMINATION OF PULLUP RESISTOR VALUE Figure 6
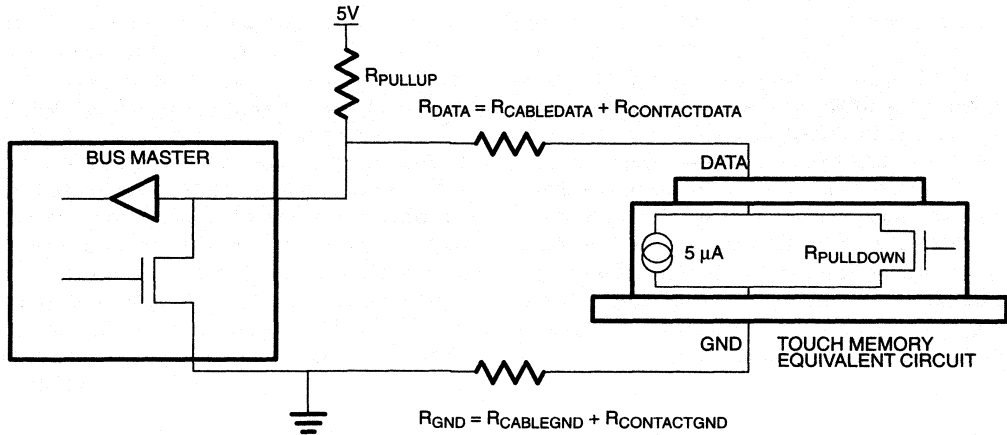
$R_{PULLDOWN}$ Max. = 0.4V/4 mA = 100 ohms (from 199x data sheet)
R_{PULLUP} Min. = (5–0.4)/4 mA = 1150 ohms
Bus Master input stage is TTL equivalent: $V_{IL} = 0.8/V_{IH} = 2.0$

$R_{CABLE} + R_{CONTACT}$ are interconnect resistances that will increase the minimum pullup resistance that will still allow correct operation. To calculate the value for R_{PULLUP} use the following equations:

$0.4V = I_{MAX}(R_{DATA} + 100 + R_{GND})$
$R_{PULLUP} = 4.6V/I_{MAX} = 4.6V/[0.4V/(R_{DATA} + 100 + R_{GND})]$
Assume $R_{DATA} = R_{GND} = R_{INTERCONNECT}$
$R_{PULLUP} = 11.5(2*R_{INTERCONNECT} + 100) = 23(R_{INTERCONNECT}) + 1150$
With negligible interconnect resistance, $R_{PULLUP} = 1150$ as above.
With $R_{PULLUP} = 4.7K$, $R_{INTERCONNECT}$ can be a maximum of 154 ohms.

CONSECUTIVE WRITE 0 TIME SLOT DEGRADATION FROM CABLE LOADING Figure 7
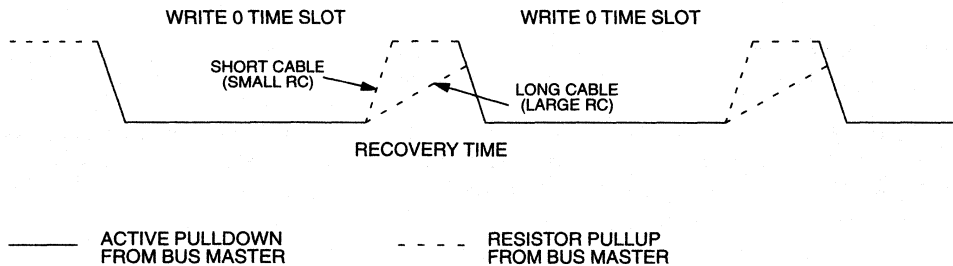
Failure Mode: Touch Memory unable to recognize command byte with consecutive write 0 time slots (Search ROM=F0 hex, for example).

Solution: Increase recovery time from the 1 µs minimum to 15 µs to allow data line to recover completely. This will reduce data rate slightly, but allow operation over much longer distances.

Bus Master Type = Open Drain with Resistor
Cable is 22 gauge twisted–pair

FANOUT vs. CABLE LENGTH vs. RESISTOR PULLUP
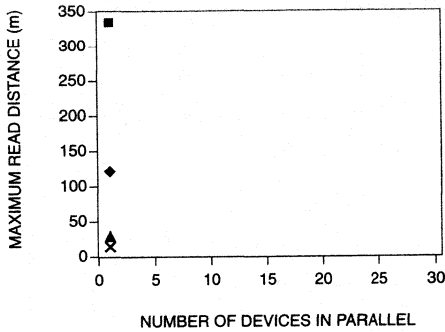
DS1990–R3

✖ R$_{PULLUP}$ = 10 KΩ
▲ R$_{PULLUP}$ = 4.7 KΩ
◆ R$_{PULLUP}$ = 2.2 KΩ
■ R$_{PULLUP}$ = 1.0 KΩ

NOTE: MAXIMUM FANOUT FOR DS1990 IS ONE.

DS1991L–F5
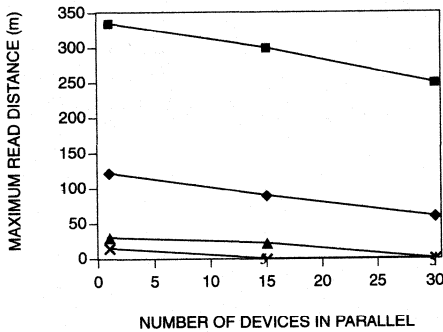
✖ R$_{PULLUP}$ = 10 KΩ
▲ R$_{PULLUP}$ = 4.7 KΩ
◆ R$_{PULLUP}$ = 2.2 KΩ
■ R$_{PULLUP}$ = 1.0 KΩ
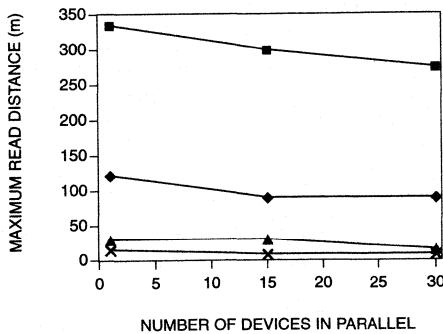
DS1992/3/4L–F5

✖ R$_{PULLUP}$ = 10 KΩ
▲ R$_{PULLUP}$ = 4.7 KΩ
◆ R$_{PULLUP}$ = 2.2 KΩ
■ R$_{PULLUP}$ = 1.0 KΩ

Bus Master Type = Open Drain with Resistor
Cable is 22 gauge twisted–pair
Reading One Device

CABLE LENGTH vs. CAPACITIVE LOAD vs. RESISTOR PULLUP

DS1990–R3

DS1991L–F5

DS1992/3/4L–F5

072996 10/11
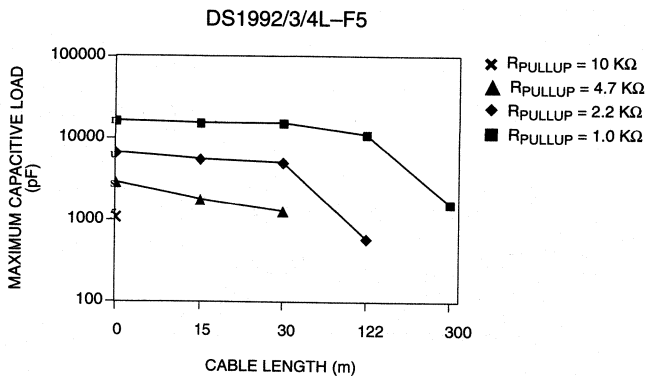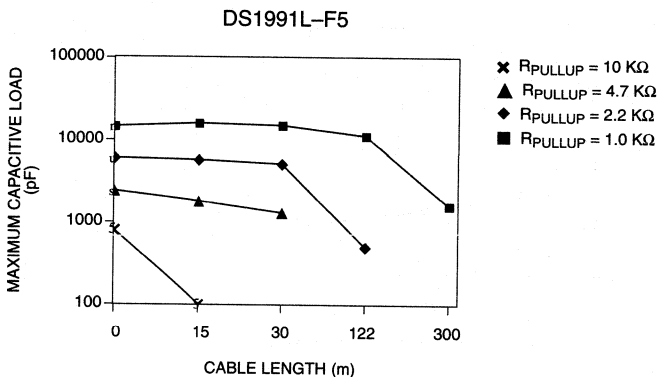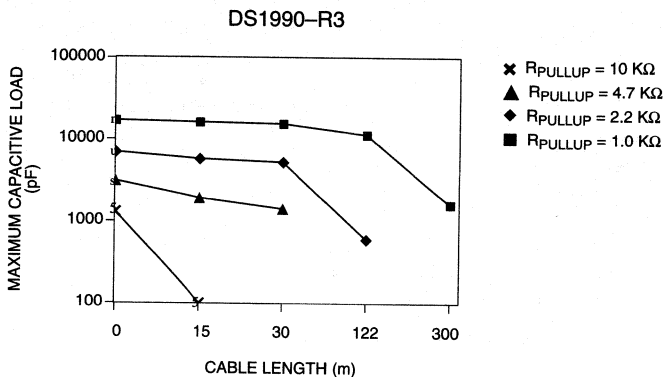
Bus Master Type = COM Port w/ DS9097 Adapter
Cable is 22 gauge twisted–pair

FANOUT vs. CABLE LENGTH vs. DEVICE TYPE
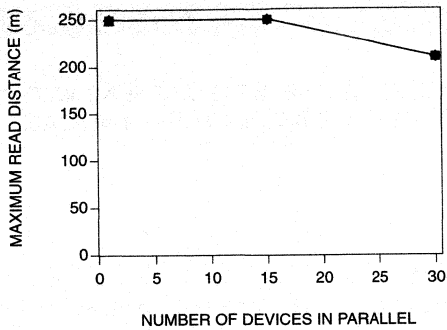
▲ DEVICE TYPE = DS1990–R3 (max. fanout = 1)
◆ DEVICE TYPE = DS1991L–F5
■ DEVICE TYPE = DS1992/3/4L–F5

(GRAPHS ARE COINCIDENT)

CABLE LENGTH vs. CAPACITIVE LOAD vs. DEVICE TYPE
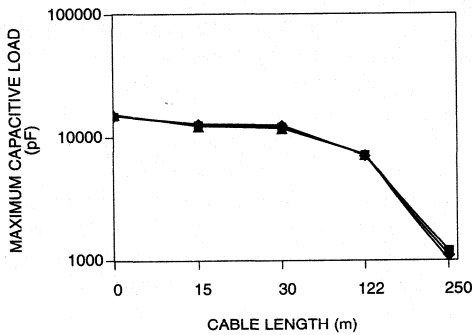
▲ DEVICE TYPE = DS1990–R3
◆ DEVICE TYPE = DS1991L–F5
■ DEVICE TYPE = DS1992/3/4L–F5

(GRAPHS ARE COINCIDENT)

NOTE: READING ONE DEVICE.

DALLAS
SEMICONDUCTOR

TECHNIQUES FOR ACCURATE TIME MEASUREMENT WITH THE DALLAS SEMICONDUCTOR DS1994 TIME–IN–A–CAN

I. HISTORICAL BACKGROUND

The first requirement for accuracy in time measurement arose from the need to determine longitude at sea from celestial measurements. Because of the earth's rotation, a time measurement error of four seconds creates a longitude error of one arc–minute, or about one mile at latitudes near 30 degrees. To determine a ship's position to within a mile, it was essential to have a good sextant and a chronometer capable of giving the correct time to within four seconds, even after many days at sea.

The first navigational chronometers to provide the needed accuracy were precision mechanical devices. These instruments did not provide an accurate time reading directly. Instead, a chronometer was carefully characterized on land to determine exactly how fast or slow it was and how the error depended on temperature and other environmental factors. Each chronometer was supplied with its own unique set of characterization data which could be used at sea to calculate the correct time from the uncorrected time supplied by the instrument.

A modern, cheap, quartz digital wristwatch is now superior to the best early navigational chronometers. With careful characterization and corrective calculation, it is possible to obtain a high degree of accuracy from a modern quartz timepiece. For practical reasons, the correction calculations that were so important for early chronometry are seldom performed with modern timepieces. As a result, a wristwatch, PC clock, or other quartz timepiece may gain or lose a second or more per day in its uncorrected reading. A properly executed cor-

rective calculation can reduce this error substantially. Microprocessors and microcontrollers can now perform this calculation with relative ease, allowing a microprocessor controlled timekeeping device to achieve a dramatic increase in accuracy.

II. SOURCES OF TIMEKEEPING ERROR

The sources of error in a quartz crystal controlled timepiece are listed below:

A. Temperature Dependence

The oscillation frequency of a quartz oscillator depends on temperature. A typical 32,768 Hz wristwatch crystal has a frequency which depends quadratically on temperature, with the maximum frequency occurring near "room" temperature (25°C).

B. Calibration Error

The oscillation frequency of a quartz oscillator depends to a degree on the values of the circuit components that comprise the oscillation circuit. Calibration error is the error in adjusting (trimming) the circuit components at the factory which causes the maximum frequency described in II.A above to be greater or less than the intended value.

C. Relaxation with Time

The manufacturing process induces mechanical stresses in the quartz crystal which affect its oscillation frequency. These stresses spontaneously relax during the lifetime of the crystal, resulting in a slow, long–term change in frequency.

D. Shock History

A severe mechanical shock can suddenly relax existing stresses in the quartz crystal or induce new stresses. This results in a step function change in the crystal frequency, followed by subsequent relaxation of the new stress pattern over time.

Changes in oscillation frequency caused by relaxation with time and shock history are not predictable and therefore not subject to automated correction. However, a recalibration performed after the change in frequency has occurred can eliminate these effects until they occur again. The two most important effects on practical timekeeping accuracy are therefore the temperature dependence of the oscillation frequency and the factory calibration error.

Temperature effects may vary widely in significance depending on the environment. A wristwatch worn continuously is exposed to a temperature environment which is very nearly constant. In this case, the observed error is due almost entirely to calibration error. Even if a watch is left on the dresser at night, it still experiences a daily average temperature environment which is very nearly constant from one day to the next. On the other hand, a timekeeper exposed to daily and seasonal extremes of outdoor temperature would be expected to exhibit substantially larger temperature effects.

In principle, the calibration error can be made arbitrarily small. However, in a manufacturing environment, it is impractical to calibrate very precisely for several reasons. Watches have mechanical adjustments for trimming the oscillator which have backlash and creep that limit the precision with which they can be set. A more important factor is that very exact calibration requires a long period of time to perform because the timekeeper requires time to accumulate a measurable error. As a result, watches and crystal controlled timekeepers used indoors are limited almost exclusively by the imprecision of their calibration.

III. CORRECTION OF TIMEKEEPING ERROR

As noted above, the principal limitation on the accuracy of an indoor, crystal–controlled timekeeper is the precision of its factory calibration. Like the early navigational chronometer, the device is capable of measuring time very accurately but does not display the accurate time directly. To achieve the desired accuracy, it is necessary to determine the correction coefficients and then to perform a corrective calculation to determine the corrected time. This correction takes the following form:

$$\text{Corrected Time} = \text{Uncorrected Time} + (\text{Uncorrected Time} - A) / B$$

In the above equation, the values A and B are the correction coefficients that are obtained by characterization. To set the timekeeper and determine the values of A and B, the following steps are required:

1. At an initial time T1, synchronize the timekeeper with a highly accurate time standard. The National Institute of Standards and Technology (NIST) provides both a radio service (WWV at 5 MHz, 10 MHz, and 15 MHz) and a 2400 bps modem line (303–494–4774) for this purpose. Set the coefficient A equal to the time T1.

2. Wait a long period of time to allow the error in the uncorrected time to build up. Good results can be obtained with a delay of several days to a week, but the longer the delay time, the better. During the long delay time, the timekeeper should be exposed to the average temperature environment in which it will later be used.

3. At a later time T2 provided by the highly accurate time standard, read the uncorrected time T from the timekeeper and solve the following equation for the coefficient B.

$$B = (T - T1) / (T2 - T)$$

Now that the coefficients A and B have been determined, the corrected time can be computed from any uncorrected time measurement by applying the correction equation using the values of A and B obtained from the characterization. Note that the value of B may be positive or negative, depending on whether the timekeeper runs slow or fast respectively. It should also be noted that the value of B can be regarded as a large whole number (signed integer).

The correction described above, while simple in principle, is complicated by the fact that the practical units of time (years, months, days, hours, minutes, seconds) are related in a complicated way. To determine T − T1, for example, it is necessary to find the total number of seconds between the date and time represented by T1 and the date and time represented by T. To facilitate the corrective calculation, it is highly desirable that the time-

keeper measure time in a single standard unit (e.g., seconds). One such standard is that of the UNIX operating system, in which time is kept as the total number of seconds that have elapsed since the beginning of January 1, 1970. Using this standard, the corrective calculations are easily performed with long integer arithmetic.

IV. TIME–IN–A–CAN

The DS1994 Time–In–a–Can manufactured by Dallas Semiconductor is a small, sealed, internally powered timekeeping unit with 512 bytes of nonvolatile RAM memory and numerous specialized timekeeping features. Data transfer to and from the DS1994 is accomplished over a single serial data wire using the Dallas Semiconductor 1–Wire™ protocols for data communication. The DS1994 has several characteristics which make it especially suitable for the type of corrective calculation described above:

1. The DS1994 accumulates date/time information in a five–byte register as the number of 256ths of a second since some arbitrary reference date. (For compatibility with the UNIX operating system, use of the reference date January 1, 1970 is recommended.) This date/time format is ideal for performing the corrective calculation described above and also makes it easy to determine the day of the week. After the correction has been applied, the corrected date and time can be converted easily to the more standard form (MM/DD/YYYY, HH:MM:SS)

2. The correction coefficients A and B can be stored in the nonvolatile RAM memory of the DS1994. This modularity provides a manufacturing advantage because the DS1994 can be characterized and loaded with its A and B coefficients and then used later as a component for assembling the final timekeeping product. It is not necessary to tie up the entire product for the relatively long times needed to perform an accurate characterization of the DS1994. This modularity also facilitates repair or replacement.

3. The oscillation frequency of the DS1994 is fixed at the factory. The fixed–frequency circuit offers long–term stability that is superior to that of mechanically trimmed components because it is free of mechanical relaxation effects in the trim adjustment.

4. Because of the 1–Wire interface, the DS1994 may easily be located at some distance from the rest of the product. This is significant when the product itself generates heat that may affect the accuracy of the timekeeper. (For applications requiring extreme accuracy, in which the timekeeper itself is kept in a constant temperature oven, the packaging and 1–Wire interface of the DS1994 simplify product design.)

V. PROGRAMS TO CALCULATE CORRECTED TIME

Dallas Semiconductor has developed a demonstration program for the DS5000 (Intel 8051 compatible microcontroller) which calculates the corrected time from a characterized DS1994 and displays the results on an LCD display. Using this program and a properly characterized DS1994 with A and B coefficients stored in its nonvolatile memory, accuracies of one second in six months have been observed indoors over a two month period. PC programs to characterize DS1994s and compute and store the correction coefficients have also been developed. The PC programs communicate with the DS1994s using the DS9097 Touch COM Port Adapter and DS9092 Touch Probe. The adapter and probe attach to any RS232C serial port of the PC, and the PC reads and writes data to the DS1994 simply by touching it with the probe. The programs in this demonstration set are described below.

A. SetTime

This is a PC program which performs the calibration step described in section III.1. T1 is obtained from the DOS clock of the PC. Since PCs typically have poor timekeeping accuracy, it is necessary first to synchronize the PC clock with the NIST standard so that it can be used as a secondary time standard. (This synchronization can be performed with a 2400 bps telephone modem and the MS DOS utility program TIMESET, available from Life Sciences Software, Box 1560, 8925 271st NW Suite 112, Stanwood, Washington 98292. A similar program, WTIME, is available as a free utility for Microsoft Windows from PC Magazine, October 27, 1992.) To set the DS1994 time and the first correction coefficient A, type

SETTIME <com port number> <time zone offset>

at the DOS prompt and then touch the DS1994 with the probe. <com port number> is the number of the COM port to which the DS9097 adapter is attached. <time zone offset> is the number of hours of offset required, and is entered as −1 if the PC is set to daylight savings time. The program sets the time register and the A coefficient in the DS1994, responds with "Date and Time Successfully Set", and terminates.

B. CalTime

This is a PC program which performs the calibration step described in section III.3. Before using the program CalTime, the DS1994s should be allowed to remain for several days or weeks in the typical environment in which they will be used. T2 is obtained from the DOS clock of the PC. Again, it is necessary first to synchronize the PC clock with the NIST standard so that it can be used as a secondary standard. To compute and set the second correction coefficient B, type

CALTIME <com port number> <time zone offset>

at the DOS prompt and then touch the DS1994 with the probe. The program sets the B coefficient in the DS1994, responds with "Date and Time Successfully Calibrated", and terminates.

C. LCDTime

This is a program for the DS5000 microcontroller which reads the time and correction coefficients from a DS1994, computes the correction, and displays the corrected time, date, day of the week, and daylight/standard time indicator on an attached LCD display. (To avoid the problem of daylight savings time, the program reads standard time from the DS1994 and automatically converts it to daylight time when daylight time is in effect.) The program is written in KSC Pascal, and the Hitachi LCD controller is attached to the DS5000 in the manner recommended by KSC, with the base address changed from 6000 hex to 0000 hex. The program can be compiled with the SYSTEM51 (Ver 3.10) software development system from KSC Software Systems, Ludvig Holsteins Alle 137, DK–2750 Ballerup, Denmark, Phone (Int. 45) 44 97 69 11, Fax (Int. 45) 44 97 96 12.

D. ReadTime

This is a PC program which reads a DS1994 using the serial port adapter and displays corrected and uncor-

rected time values, calibration constants, and measurement errors. To read the DS1994 and display these results, type

READTIME <com port number> <time zone offset>

at the DOS prompt and then touch the DS1994 with the probe. The program displays the uncorrected date and time, the corrected date and time, the difference in seconds between the corrected and uncorrected times, the values of the correction coefficients A and B, the current DOS time, and the difference in seconds between the corrected time and the current DOS time.

Note that in all of the above programs, the values of A and B are stored in the unused Real Time Alarm and Interval Time Alarm registers in the "clock page" of the DS1994. This was done so that the 512 bytes of nonvolatile RAM memory could be kept free for other possible uses. The values of A and B can be independently reset at any time in the future with the SetTime and CalTime programs respectively.

VI. TESTS OF THE CALIBRATION PROCESS

To investigate the effectiveness of the calibration process, a set of fifty DS1994s was divided into five groups of ten parts each. The first calibration point was set with SetTime for all parts at 11:50 a.m. Central Standard Time on July 14, 1992. Each of the five groups, designated A, B, C, D, and E, were recalibrated after different periods of time. The second calibration point was set with CalTime for the various groups as follows:

Group A at 11:50 a.m. CST on July 16, 1992. Group B at 9:30 a.m. CST on July 22, 1992. Group C at 3:40 p.m. CST on July 29, 1992. Group D at 9:10 a.m. CST on August 3, 1992. Group E at 7:35 a.m. CST on August 13, 1992.

When the parts were not being read or calibrated, they remained in a desk drawer in an air–conditioned office environment. On September 15, 1992, at 11:50 a.m. CST, all parts were read with ReadTime and the errors in the corrected times were tabulated in seconds as shown in the table below:

PART #	GROUP A	GROUP B	GROUP C	GROUP D	GROUP E
1	0.62	0.63	0.56	−0.29	0.05*
2	2.38	1.31	0.48	−0.09*	0.13*
3	3.03	0.74	0.20*	−0.30	0.21
4	2.41	0.50	0.13*	−0.07*	−0.02*
5	2.79	0.81	0.79	0.01*	−0.08*
6	1.02	0.43	0.21*	0.02*	0.08*
7	2.20	0.95	0.16*	0.04*	0.23
8	1.20	0.67	0.48	0.09*	0.04*
9	1.12	0.29*	0.07*	0.22*	0.19
10	1.79	1.3	0.76	0.07*	−0.32
Mean	1.856	0.770	0.384	−0.030	0.051
Std.Dev.	0.827	0.355	0.265	0.164	0.165
Days Cal.	2.00	7.90	15.16	19.89	29.82
Days Idle	61.00	55.10	47.84	43.11	33.18

Days Cal. in the table above indicates the number of days between the first and second calibration points, and Days Idle represents the number of days after the second calibration point before the final reading was taken on September 15.

Note that the corrected times marked with asterisks represent errors of less than two seconds per year. The parts calibrated for shorter times show larger errors and larger standard deviations, indicating the effects of end–point calibration inaccuracy and short–term temperature fluctuations. (By comparison, the uncorrected times from the fifty DS1994s showed a mean error of 526 seconds/year, with a standard deviation of 210 seconds/year.)

A final measurement was made on July 14, 1993 at 11:50 a.m., one year after the initial setting. The uncorrected times showed a mean error of 550.6 seconds with a standard deviation of 207.8 seconds, consistent with the two–month measurement cited above. In parts per million (ppm), this is 17.5 ppm ± 6.6 ppm. The mean errors in the corrected times for the five groups were as follows:

Group A:	26.83 sec ± 6.81 sec	(0.85 ppm ± 0.22 ppm)
Group B:	22.91 sec ± 6.94 sec	(0.73 ppm ± 0.22 ppm)
Group C:	18.30 sec ± 3.83 sec	(0.58 ppm ± 0.12 ppm)
Group D:	15.48 sec ± 2.81 sec	(0.49 ppm ± 0.09 ppm)
Group E:	15.94 sec ± 3.70 sec	(0.51 ppm ± 0.12 ppm)

While not as accurate as the measurements made over the two month period, these accuracies are still impressive. The deviations in the corrected readings are all in the same direction, indicating a possible seasonal change in the temperature of the environment in which the parts were stored. As expected, the group having the shortest calibration period (Group A) had the greatest error. The average error in the corrected readings of groups D and E is 35 times smaller than the average error in the uncorrected readings.

VII. SUMMARY

Significant improvement of timekeeping accuracy can be achieved by using a real time clock with supplemental nonvolatile SRAM for the storage of calibration constants. Whenever the timekeeper is reset, improved calibration constants can be stored in the nonvolatile SRAM, so that successive resets adaptively enhance accuracy. When the timekeeper is read, these constants are supplied along with the uncorrected timebase to a microprocessor program that calculates and displays a more accurate reading. If the timekeeper maintains time by counting the seconds from a designated reference time then a simple program code can quickly compute a more accurate, compensated time. In this way the practical effect of errors from temperature, initial calibration, quartz aging, and shock history can be minimized. Experimental data indicate that an accuracy of ±2 seconds per year is achievable over a two–month period, and approximately 16 seconds over a full year. To obtain copies of the programs described above, further information on the demonstration circuit, and information on how to obtain components, contact John Adams at Dallas Semiconductor, (214) 450–3803.

DALLAS
SEMICONDUCTOR

I. INTRODUCTION

Touch Memory is a chip housed in a stainless steel enclosure. The electrical interface is reduced to the absolute minimum, i.e., a single data line plus a ground reference. The energy needed for operation is either "stolen" from the data line ("parasitic power") or is taken from an embedded lithium cell. The logical functions range from a simple serial number to password–protected memory, to 64K bits and beyond of nonvolatile RAM or EPROM, to a Touch Thermometer, to a real time clock plus 4K bits of nonvolatile RAM. Common to all Touch Memories is a globally unique registration number, the serial 1–Wire™ protocol, presence detect, and communication in discrete time slots. Table 1 gives an overview of the available devices.

For read operations all devices are satisfied with a 5kΩ pull–up resistor to supply energy and to terminate the 1–Wire bus. Touch Memory devices based on non–volatile RAM (DS1991 to DS1996) can also be written using this same interface. Due to their different technology, EPROM based Touch Memories (DS1982 to DS1986) also require pulses of up to 12V for programming. Since they cannot be erased, EPROM Touch Memories are referred to as Add–Only Memories. Another device, the DS1920 Touch Thermometer, gets its energy for temperature conversion through a low impedance active pull–up to 5V. Different requirements for writing or special functions are the reason for several types of interfaces.

TOUCH MEMORY DEVICES Table 1

Device Type	Family Code	Serial Number	Memory Bits Type	Protected NV RAM bits	Real Time Clock	Interval Timer	Cycle Counter
DS1990A	01H	yes	—	—	—	—	—
DS1991	02H	yes	512, NVRAM	3 * 384	—	—	—
DS1992	08H	yes	1K, NVRAM	—	—	—	—
DS1993	06H	yes	4K, NVRAM	—	—	—	—
DS1994	04H	yes	4K, NVRAM	—	yes	yes	yes
DS1995	0AH	yes	16K, NVRAM	—	—	—	—
DS1996	0CH	yes	64K, NVRAM	—	—	—	—
DS1982	09H	yes	1K, EPROM	—	—	—	—
DS1985	0BH	yes	16K, EPROM	—	—	—	—
DS1986	0FH	yes	64K, EPROM	—	—	—	—
DS1920	10H	yes	16, EEPROM	TOUCH THERMOMETER			

II. 1–WIRE INTERFACE

A. General Information

Touch Memories are self–timed silicon devices. The timing logic provides a means of measuring and generating digital pulses of various widths. Data transfers are bit–sequential and half–duplex. Data can be interpreted as commands (according to the prearranged format identified by the family code) that are compared to information already stored in the Touch Memory to make a decision, or can simply be stored in the Touch Memory for later retrieval. Touch Memories are considered slaves, while the host reader/writer is considered a master.

B. DC Requirements

Touch Memories operate in an open drain environment on voltage levels ranging from 2.8V (minimum pull–up voltage) to 6V (maximum pull–up voltage). All voltages greater than 2.2V are interpreted as logic 1 or HIGH, voltages less than 0.8V are considered as logic 0 or LOW. The pull–up voltage must be a minimum of 2.8V to recharge an internal storage capacitor that is used to supply power during periods when the data line is low. The size of this capacitor is about 800 pF. This capacity is seen for a short time when a Touch Memory is contacted by a probe. After the capacitor is charged, only a very small fraction of this capacity is recognizable, according to the charge required to refill to full charge. The total time constant to charge the capacitor is defined by the capacitor itself, the internal resistances of about 1 $k\Omega$, the resistance of the cable and contacts, the cable capacitance, and the resistor pulling up the data line.

C. AC Requirements

Timing relationships in Touch Memories are defined with respect to time slots. Because the falling slope is the least sensitive to capacitive loading in an open drain environment, Touch Memories use this edge to synchronize their internal timing circuitry. By definition the active part of a 1–Wire time slot (t_{SLOT}) is 60 μs. After the active part of the time slot, the data line needs to be inactive for a minimum of 1 μs at a voltage of 2.8V or higher to recharge the internal capacitor.

Under nominal conditions, a Touch Memory will sample the line 30 μs after the falling edge of the start condition. The internal time base of Touch Memory may deviate from its nominal value. The allowed tolerance band ranges from 15 μs to 60 μs. This means that the actual slave sampling may occur anywhere between 15 and 60 μs after the start condition, which is a ratio of 1 to 4. During this time frame the voltage on the data line must stay below V_{ILMAX} or above V_{IHMIN}.

C.1. Write Time Slots

In the 1–Wire system, the logical values of 1 and 0 are represented by certain voltage levels in special waveforms. The waveforms needed to write commands or data to Touch Memories are called write–1 and write–0 time slots. The duration of a low pulse to write a 1 (t_{LOW1}, Figure 1) must be shorter than 15 μs. To write a 0, the duration of the low pulse (t_{LOW0}, Figure 2) must be at least 60 μs to cope with worst–case conditions.

The duration of the active part of a time slot can be extended beyond 60 μs. The maximum extension is limited by the fact that a low pulse of a duration of at least eight active time slots (480 μs) is defined as a Reset Pulse. Allowing the same worst–case tolerance ratio, a low pulse of 120 μs might be sufficient for a reset. This limits the extension of the active part of a time slot to a maximum of 120 μs to prevent misinterpretation with reset.

At the end of the active part of each time slot, Touch Memory needs a recovery time t_{REC} of a minimum of 1 μs to prepare for the next bit. This recovery time is the inactive part of a time slot, since it must be added to the duration of the active part to obtain the time it takes to transfer one bit. The wide tolerance of the time slots and the non–critical recovery time allow even slow microprocessors to meet the timing requirements for 1–Wire communication easily.

WRITE–ONE TIME SLOT Figure 1

___ RESISTOR

▬▬ MASTER

▬▬ TOUCH MEMORY

$60 \ \mu s \leq t_{SLOT} < 120 \ \mu s$

$1 \ \mu s \leq t_{LOW1} < 15 \ \mu s$

$1 \ \mu s \leq t_{REC} < \infty$
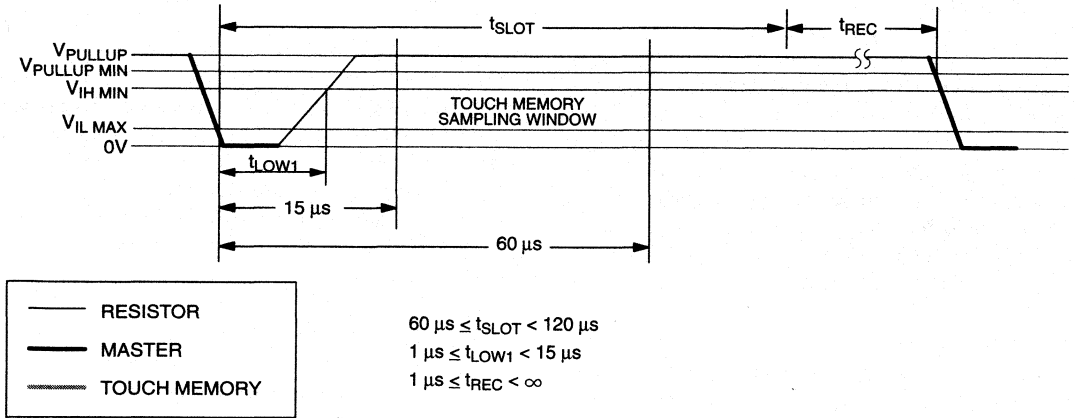
WRITE–ZERO TIME SLOT Figure 2

___ RESISTOR

▬▬ MASTER

▬▬ TOUCH MEMORY

$60 \ \mu s \leq t_{LOW0} < t_{SLOT} < 120 \ \mu s$

$1 \ \mu s \leq t_{REC} < \infty$
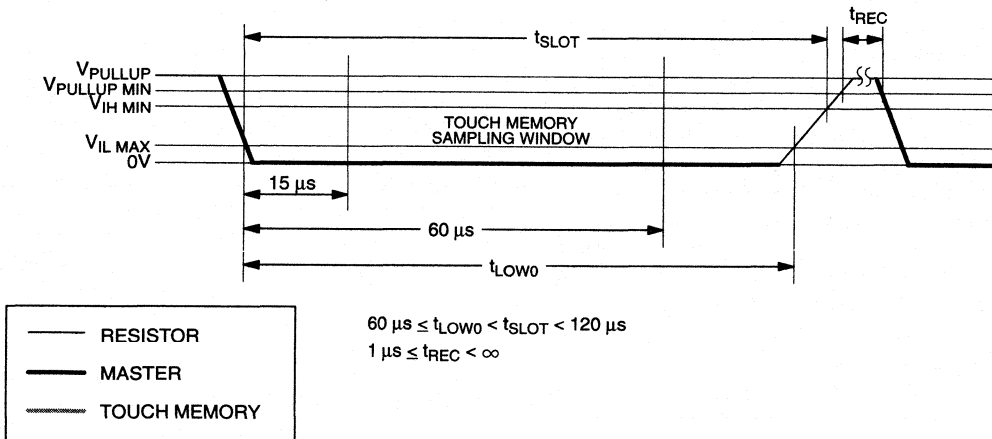
C.2. Read Time Slots

Commands and data are sent to Touch Memories by combining Write–Zero and Write–One time slots. To read data, the master has to generate Read–Data time slots to define the start condition of each bit. The Read–Data time slot looks essentially the same as the Write–One time slot from the master's point of view. Starting at the high–to–low transition, the Touch Memory sends one bit of its addressed contents. If the data bit is a 1, the Touch Memory leaves the pulse unchanged. If the data bit is a 0, the Touch Memory will pull the data line low for t_{RDV} or 15 µs (Figure 3). In this time frame data is valid for reading by the master. The duration t_{LOWR} of the low pulse sent by the master should be a minimum of 1 µs with a maximum value as short as possible to maximize the master sampling window. In order to compensate for the cable capacitance of the 1–Wire line the master should sample as close to 15 µs after the synchronization edge as possible. Following t_{RDV} there is an additional time interval, $t_{RELEASE}$, after which the Touch Memory releases the 1–Wire line so that its voltage can return to V_{PULLUP}. The duration of $t_{RELEASE}$ may vary from 0 to 45 µs; its nominal value is 15 µs.

C.3. Reset and Presence Detect

The Reset Pulse provides a clear starting condition that supersedes any time slot synchronization. It is defined as a single low pulse of minimum duration of eight time slots or 480 µs followed by a Reset–high time t_{RSTH} of another 480 µs (Figure 4). After a Reset Pulse has been sent, the Touch Memory will wait for the time t_{PDH} and then generate a Presence Pulse of duration t_{PDL}. No other communication on the 1–Wire bus is allowed during t_{RSTH}. The Presence Pulse can be used to trigger a hardware interrupt or to automatically power up equipment like Touch Pens. If a Touch Memory is discon-nected from the probe, it will pull its data line low via an internal current source of 5 µA. This simulates a Reset Pulse of unlimited duration. As soon as the Touch Memory detects a high level on the data line, it will generate a Presence Pulse.

The nominal values are 30 µs for t_{PDH} and 120 µs for t_{PDL}. With the same worst–case tolerance band, the measured t_{PDH} value indicates the internal time base of the fastest device. The sum of the measured t_{PDH} and t_{PDL} values is five times the internal time base of the slowest device. If there is only one device on the line, both values will deviate in the same direction. This correlation can be used to build an adaptive system. Special care must be taken to recalibrate timing after every reset since the individual timing characteristics of the devices vary with temperature and load.

The accuracy of the time measurements required for adaptive timing is limited by the characteristics of the master's input logic, the time constant of the 1–Wire line (pullup resistor x cable capacitance) and the applied sampling rate. If the observed rise time or fall time exceeds 1 µs or the highest possible sampling rate is less than 1 MHz, adaptive timing should not be attempted.

C.4. Example Pulse Train

For illustrative purposes, a reference pulse train has been defined (Figure 5) to explain how the waveforms are generated rather than showing a complete session. It starts with a reset sequence including a Presence Pulse. Further time slots show all waveforms of reading and writing 1s and 0s. Any communication session can be constructed from the waveforms of this pulse train.

READ–DATA TIME SLOT Figure 3

	RESISTOR
	MASTER
	TOUCH MEMORY

$60 \ \mu s \leq t_{SLOT} < 120 \ \mu s$

$1 \ \mu s \leq t_{LOWR} < 15 \ \mu s$

$0 \leq t_{RELEASE} < 45 \ \mu s$

$1 \ \mu s \leq t_{REC} < \infty$

$t_{RDV} = 15 \ \mu s$
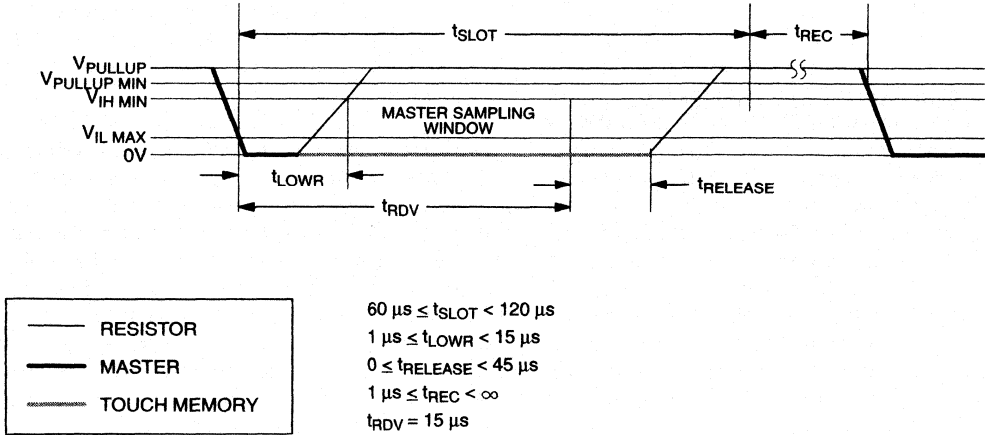
RESET AND PRESENCE PULSE Figure 4

	RESISTOR
	MASTER
	TOUCH MEMORY

$480 \ \mu s \leq t_{RSTL} < \infty$ *

$480 \ \mu s \leq t_{RSTH} < \infty$ (includes recovery time)

$15 \ \mu s \leq t_{PDH} < 60 \ \mu s$

$60 \ \mu s \leq t_{PDL} < 240 \ \mu s$
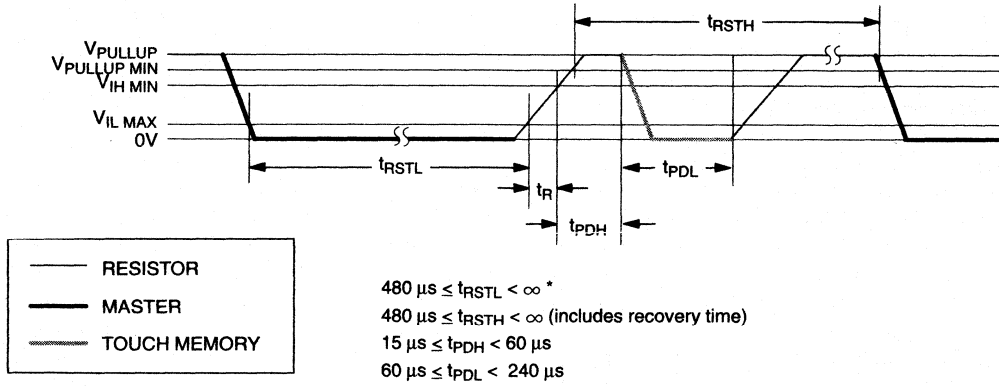
* In order not to mask interrupt signalling by other devices on the 1–Wire bus, $t_{RSTL} + t_R$ should always be less than 960 µs.

REFERENCE PULSE TRAIN Figure 5
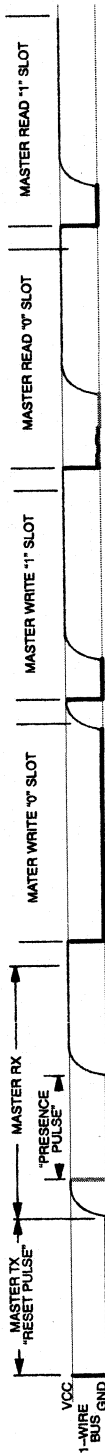
III. FUNDAMENTALS

A. TTL Interface

This category includes all logic families and microprocessors that use positive logic with a maximum 0.8V for a logical 0 or LOW and a minimum of 2.2V for a logical 1 or HIGH. These voltages combined with a current source capability of at least 1 mA and a sink capability of more than 4 mA interface to a broad class of digital electronics.

Since the 1–Wire bus is an open drain system, an open drain/collector driver is required to connect the output port to the bus (Figure 6). This driver can be a general purpose NPN transistor with a resistor connected between base and output port or an n–channel MOS-FET or any open drain/collector driver available in the logic family as long as the pullup voltage is equal to the driver voltage. Even a tri–state driver with its logic input tied to ground can be used, connecting the output gate to the tri–state control input. Depending on the characteristics of the driver (inverting/non–inverting), it may be required to complement the logic value of the output gate to compensate for the driver's signal inversion.

Reading from the 1–Wire bus can usually be accomplished by directly connecting the 1–Wire bus to the input port of the master. If the pullup–voltage of the 1–wire bus is too low or if the capacity of the cable produces slopes too slow for the logic family, it may be required to employ a comparator as interface and to adjust the reference voltage to optimize noise margins and timing characteristics. If the comparator inverts the signal, this inversion needs to be compensated by the software.

Generally it is recommended to test this type of interface carefully, starting with reset pulses generated by software and watching the slopes with an oscilloscope. If the timing specifications of Figure 4 are met, and the presence pulse is seen, one may proceed and test the software to generate the Write–Zero and Write–One time slots. After this works properly, the next step is reading the ROM. This is done by performing a reset cycle first, followed by 2 Write–One time slots, 2 Write–Zero time slots, 2 Write–One time slots and 2 Write–Zero time slots (this is equivalent to sending 33H, least significant bit first). After this, 64 Read Data time slots need to be generated.

Since all timing depends on the clock frequency of the microcontroller, it is required to count the number of clock cycles for the execution of each command within the loop. The level of an output pin of a microprocessor will usually not change at the end of a command, but a few clock cycles earlier. The actual reading from any input pin also occurs some clock cycles before the end of the command. If the description of the microprocessor does not give sufficient details on this, a test series with different clock frequencies may be required. As long as the microprocessor is executing time–sensitive code, i.e., reading from the 1–Wire bus or writing to it, jumps or calls may occur only while the 1–Wire bus is idle. Interrupts from other sources than the 1–Wire bus must be disabled.

B. RS232 Interface

This section covers all interfaces that use a special controller to generate all timing and reference signals required for serial communication. The typical controller for this type of interface is the UART 8250. It relieves the microprocessor of the burden of time–critical software execution. The microprocessor simply puts the character code to be transmitted into the transmit register of the UART and the UART will do the work. A character is received by the microprocessor just by reading the UART's receive register. If the serial transmission is finished or if there is data for the microprocessor, this condition is signalled by the UART through flags that can be polled or by interrupts.

To function properly, the UART requires configuration with respect to baud rate, number of data bits per character, parity and number of start and stop bits. These terms are common for serial communication, but fit the needs of 1–Wire networks with their time slots and separate synchronization if a bit rather than a character is framed by the start condition. For 1–Wire communication the UART is set up for a high baud rate and each character delivered by the UART represents a bit on the 1–Wire bus. The microprocessor must separate the bits of a byte, least significant bit first, and write them as appropriate characters to the UART. To read data, the microprocessor has to assemble the bits received through characters back into bytes. These functions are not time–sensitive and can easily be programmed in a high level language.

TTL INTERFACING Figure 6
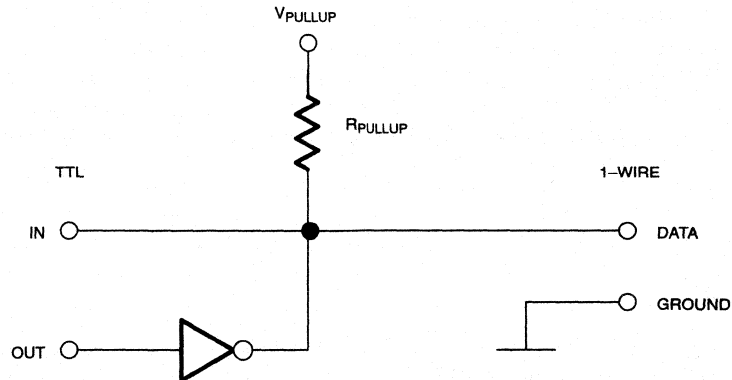
RS232 Conventions

Unlike TTL logic, RS232 has been established to transport data over long lines. Therefore different current drive characteristics and higher voltages are required to represent the logic levels 0 and 1.

The values to be expected are:

+3V to +15V for 0,

which is identical to the polarity of the start bit and

−3V to −15V for 1,

which is identical to the polarity of the stop bit and the idle state. The voltage range from −3V to +3V is undefined.

All voltages are measured with respect to ground. The receive channel and the transmit channel are independent wires, called RXD and TXD. Since RS232 ports are often used for communication via phone lines, several control signals are also included in the standard. Not all of these control signals need to be implemented with a communication device. For 1–Wire applications only the control signals DTR (Data Terminal Ready) and RTS (Request to send) are needed. Other signals often found with RS232 are DSR (Data Set Ready), which is the response to DTR, and CTS (Clear To Send), which is the response to RTS. How these signals are provided on a connector is detailed in Table 2. Full documentation on RS232 is beyond the scope of this application note. For a complete description please refer to other literature.

Table 2

SIGNAL	9–PIN CONNECTOR	25–PIN CONNECTOR	DESCRIPTION	FUNCTION
RXD	2	3	Receive Data	input
TXD	3	2	Transmit Data	output
DTR	4	20	Data Terminal Ready	output
RTS	7	4	Request to Send	output
GND	5	7	Ground	(reference)
DSR	6	6	Data Set Ready	input
CTS	8	5	Clear to Send	input

Hardware Simplified Model

The standard hardware of a RS232 interface is shown in Figure 7a. The UART is hooked to the system bus like an 8–bit memory device. The three address inputs A0 to A2 make the UART appear as a block of 8 read/write memory locations. On the other side there are the serial communication signals and the control signals mentioned above. Since the UART is a 5V device, special drivers and receivers for handling higher voltages are required. Circuit diagrams of bipolar integrated drivers (1488) and receivers (1489) are shown on Figures 7b and 7c. CMOS drivers and receivers are also available as standard products.

By design, the output current of an RS232 driver is limited to ±10 mA typically. Since the voltage definitions for 1 and 0 on the RS232 channel are reversed with respect to the conventions of positive logic, RS232 drivers and receivers are actually inverting devices. The inversion in the transmit channel is compensated through an inversion in the receive channel. Thus a 1 written to the transmit register will appear as a 1 at the serial output, as −15V at TXD, as a 1 at the serial input of the receiver and finally be read as a 1 in the receive register of the receiving UART.

For energy efficiency with battery operated equipment, the RS232 drivers and receivers are often replaced by simple 5V inverting drivers. This is definitely not compatible with the RS232 standard, but may be sufficient to control a modem or to transfer data through a short cable. Interfaces like this are called 5V RS232 within this application note. They run on the same software as the standard RS232, but are electrically almost the same as the TTL interface.

Programmer's Model

To write software for the 8250 UART one must know the basic address where the registers of the UART are hardwired to. This address is generally an equipment specific variable, and therefore will be referenced by the name SPA (Serial Port Address) rather than by a physical address. Of the 8 theoretically accessible addresses within the UART only 7 are really implemented, using the relative addresses 0 to 6. The names of these registers are as follows:

address:
SPA   +0   Receive (read)/Transmit (write)
            Data Register

+1	Interrupt Enable Register
+2	Interrupt Identification Register (read only)
+3	Line Control Register
+4	Modem Control Register
+5	Line Status Register
+6	Modem Status Register

The meaning of the UART's control bits and their position within the control registers is detailed in Figure 8. For 1–Wire communication, the Interrupt Identification Register and Modem Status Register are not used.

To define the speed or baud rate of a serial communication, there is a 16 bit register, called Divisor Latch. This register is accessed as two bytes using the same addresses as the Data Register (least significant byte) and the Interrupt Enable Register (most significant byte). To access the Divisor Latch, the Divisor Latch Access Bit DLAB must be set to 1. DLAB is the most significant bit of the Line Control Register. As long as DLAB is set, the Data Register and the Interrupt Enable Register are not accessible. For the commonly used crystal frequency of 1.8432 MHz, the divisor latch must contain a number between 2304 (900hex, 50 bps) and 1 (115.2k bps).

1–Wire Communication through the UART

As mentioned above, to write one bit to the 1–Wire bus, the UART is programmed to transmit one character. Since the receive and transmit channels of the UART are operating independently, but using the same communication setup, reading from and writing to the 1–Wire bus can occur at the same time. The start condition generated at the UART's serial output is fed to the 1–Wire bus and is simultaneously returned to the serial input, triggering the process of reading one character. The waveform is completely defined by the baud rate, the polarity of the start and stop bits and the bit pattern of the character. The serial output of the UART is high (~ 5V, idle) between characters, low (~0V) for the start bit and equal to the value of the data bit being transmitted. An idle (~5V) to low (~0V) transition at the UART's serial input triggers the process of receiving a character. The first bit is understood as start bit; the remaining bits are shifted into the receive register in the same polarity as they arrive at the serial input. Bits received after the receive register is full, are ignored.

HARDWARE OVERALL CONCEPT Figure 7a
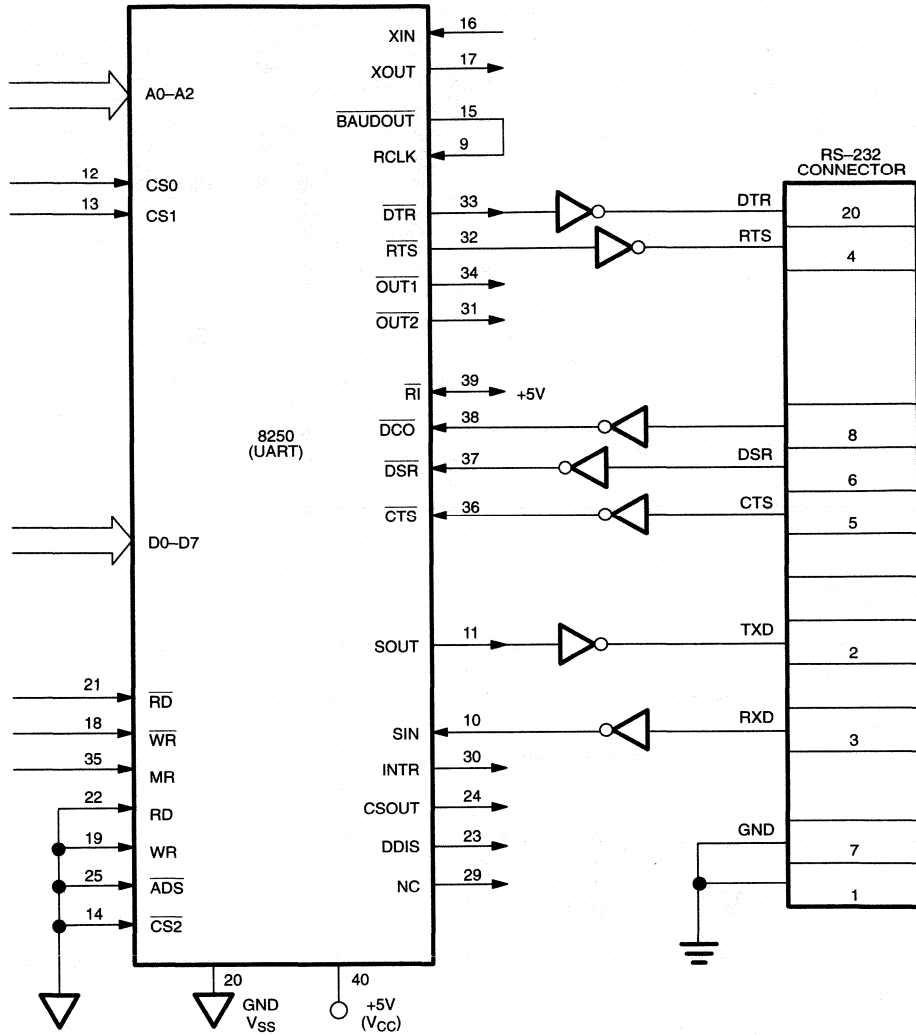
TYPICAL DRIVER Figure 7b
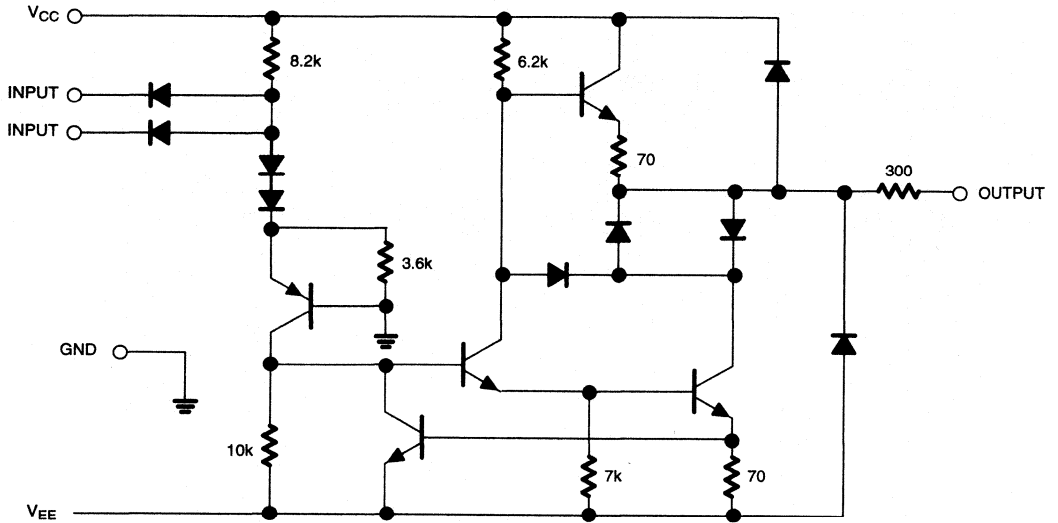
TYPICAL RECEIVER Figure 7c
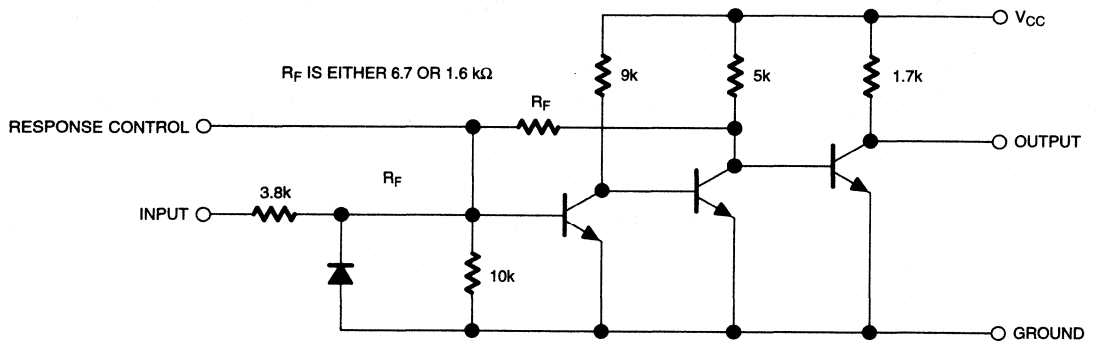
Figure 8

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	
SPA + 1	bit 15	baud rate divisor most significant byte						bit 8	DLAB=1
SPA + 0	bit 7	baud rate divisor least significant byte						bit 0	DLAB=1
SPA + 0	receive/transmit data register								DLAB=0
SPA + 1	0	0	0	0	Interrupt enable; 0=disabled				DLAB=0
SPA + 2	0	0	0	0	0	Interrupt ident. register, not used			DLAB=0
SPA + 3	DLAB, Divisor Latch Address Bit	Set Break, 0=dis-abled	Stick Parity, 0=dis-abled	Even Parity don't care	Parity Enable, 0=dis-abled	Nr. of stop bits, 0=1 stop bit	WLS1 Word Length	WLS0 Word Length	DLAB=0
SPA + 4	0	0	0	Loop	Out2	Out1	RTS, Request to Send	DTR, Data Terminal Ready	DLAB=0
SPA + 5	0	TSRE, transmit shift reg-ister empty	THRE, transmit hold reg-ister empty	Break Interrupt	Framing Error	Parity Error	Overrun Error	Data Ready	DLAB=0
SPA + 6	Modem Status Register, not used								DLAB=0

Reset and Presence Detect

The reset and presence detect cycle is performed by setting the baud rate to 10473 bps (Divisor Latch = 11decimal), the character length to 8 bits (WLS0=1, WLS1=1) and transmitting the character code F0. Including the start bit, this produces a pulse of 5 x 95.5 µs low (start bit plus four 0's) followed by 5 x 95.5 µs high (four 1's plus stop bit) at the serial output of the UART. If a Touch Memory is present, then it will assert its presence pulse during the time interval where the most significant bits of the character code are transmitted. If after the transmission the receive data register reads F0, then there is no Touch Memory. If one or more bits of the transmitted F are changed to 0s, than a presence pulse was received.

Read/Write One Bit

To generate data time slots, the UART must be set to a baud rate of 115.2k bps (Divisor Latch = 1). With a character length of 6 bits (WLS0=1, WLS1=0), any transmitted character will consist of a pulse train of 8 x 8.68 µs, beginning with a low start bit, followed by true data bits and a high stop bit. This matches the waveform for fast 1–Wire communication. If the character code is 00,

than a Write–Zero time slot is generated. A character code of FF will produce a Write–One time slot.

As long as writing to the 1–Wire bus is desired, all characters received by the UART must be read and can be discarded. To read data, Write–One time slots must be generated. Bits received from the Touch Memory are returned in their true form in the least significant bit of the character code found in the receive data register. If the Touch Memory sends a one, all bits of the character code will read one. If the Touch Memory sends a zero, one or more of the least significant bits of the character code will be zero, depending on the internal time base of the Touch Memory.

Not all UARTs behave exactly the same as the 8250. Some of them do not support the character length of 6 bits. To circumvent potential problems from this restriction, the software examples in this document always use the character length of 8 bits. This extends the transmission of one bit by 17.36 µs and reduces the effective baud rate from 14.4k bps to 11.5k bps, but remains well within the specification of 1–Wire timing.

IV. CIRCUITS FOR 5V INTERFACES (TTL AND RS232)

A. TTL Read All

This is the simplest interface for Touch Memory applications. It is suitable for reading all Touch Memories and writing NVRAM based devices. The circuit diagram (Figure 9) conforms to the principle of Figure 6. The diodes D1 and D2 protect transistor Q1 and the input of the microprocessor, respectively, against damage from electrostatic discharge (ESD). R1 is the 1–Wire pullup resistor. If the microprocessor runs on 5V, the same supply can directly be connected to R1. If only a higher supply voltage than 5V is available, any monolithic or discrete positive 5V regulator can be used to provide the pullup voltage for the 1–Wire bus.

The characteristics of the components are not critical. The transistor 2N7000 has been chosen since it is a very common product and has a low threshold voltage.

If desired, a small signal bipolar transistor or any available open–drain or open–collector inverting driver can be used instead of Q1. If Q1 is an npn–transistor, a resistor bypassed by a small capacitor between the TX–input and the base terminal is required to limit the input current.

The logic level high at TX will produce a low on the 1–Wire bus. To generate a Write–One or Read Data time slot, a short high pulse (1 μs < t < 15 μs) must be applied to the TX input. A Write–Zero Time Slot is formed by a 60 μs high pulse at TX. Data from Touch Memories is received in its true form. If idle, TX must be held at a logic low level. The reference pulse train and other relevant waveforms for this circuit are shown on Figure 9a. The timing for this type of interface is directly generated by the microprocessor. A software example for this type of interface is found later in this document.

TTL READ ALL CIRCUIT Figure 9
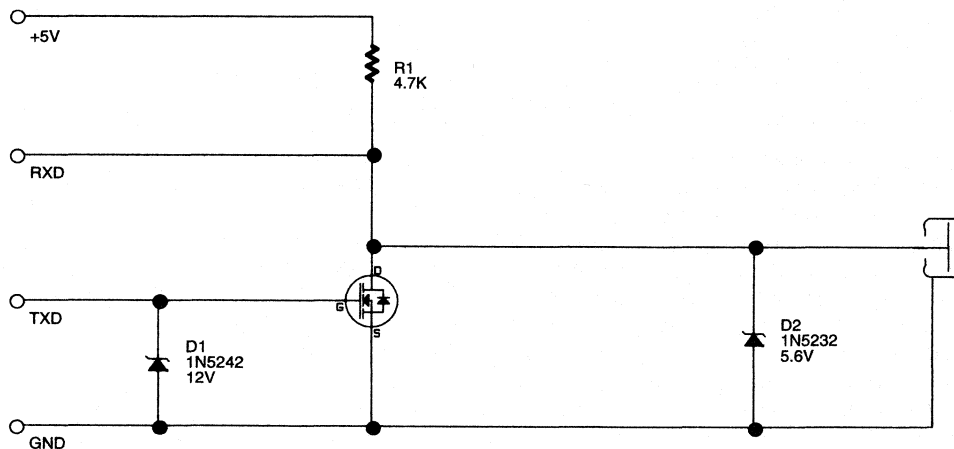
TTL READ ALL WAVEFORMS Figure 9a
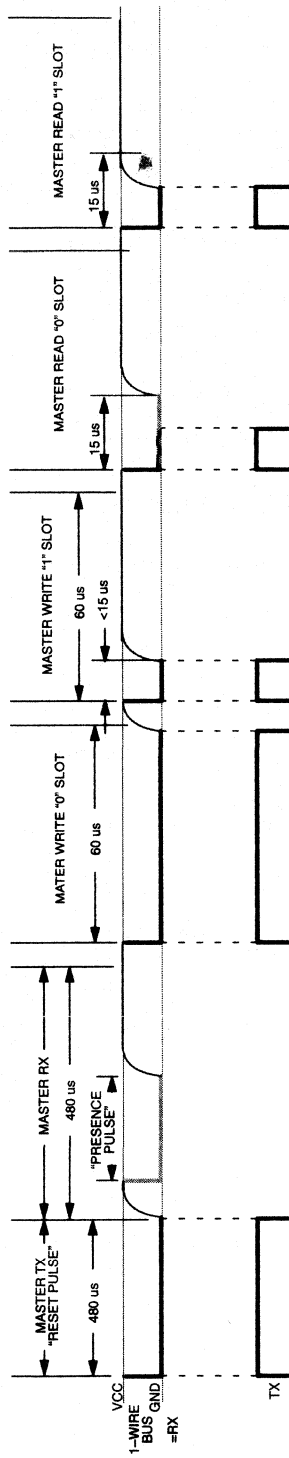
B. 5V RS232 Read All

This interface is required for equipment that uses a UART to provide all timing but does not conform to the voltage levels of the RS232 standard. The logic levels, however, are the correct polarity so that it can be operated with the same software as the standard RS232 interface.

The circuit shown on Figure 10 is suitable for reading all Touch Memories and writing NVRAM based devices. New compared to Figure 9 are D3, Q2 and R2. D3 has the same function as D2, i.e., ESD protection. Q2 and R2 form an inverter to restore the correct polarity for the receive channel of the UART. Since there is no pin defined as a power supply with RS232, the DTR signal is used to provide the pullup voltage for the 1–Wire bus

and power for the inverter in the receive channel. Everything else is identical to Figure 9.

The reference pulse train and other important waveforms for this circuit are shown on Figure 10a. Due to the double inversion in both the transmit and the receive channel (inverters between UART and RS232 connector in Figure 7, Q1 and Q2 in Figure 10) the logic levels of the 1–Wire bus are the same as those at the UART. A 1 written to the UART's transmit register will appear as a 1 on the 1–Wire bus, a zero on the 1–Wire bus will be received as a 0. No software–inversion is needed. Further details on programming the UART for 1–Wire communication are found section III.b. For software examples please refer to a later section of this document.

5V RS232 READ ALL CIRCUIT Figure 10
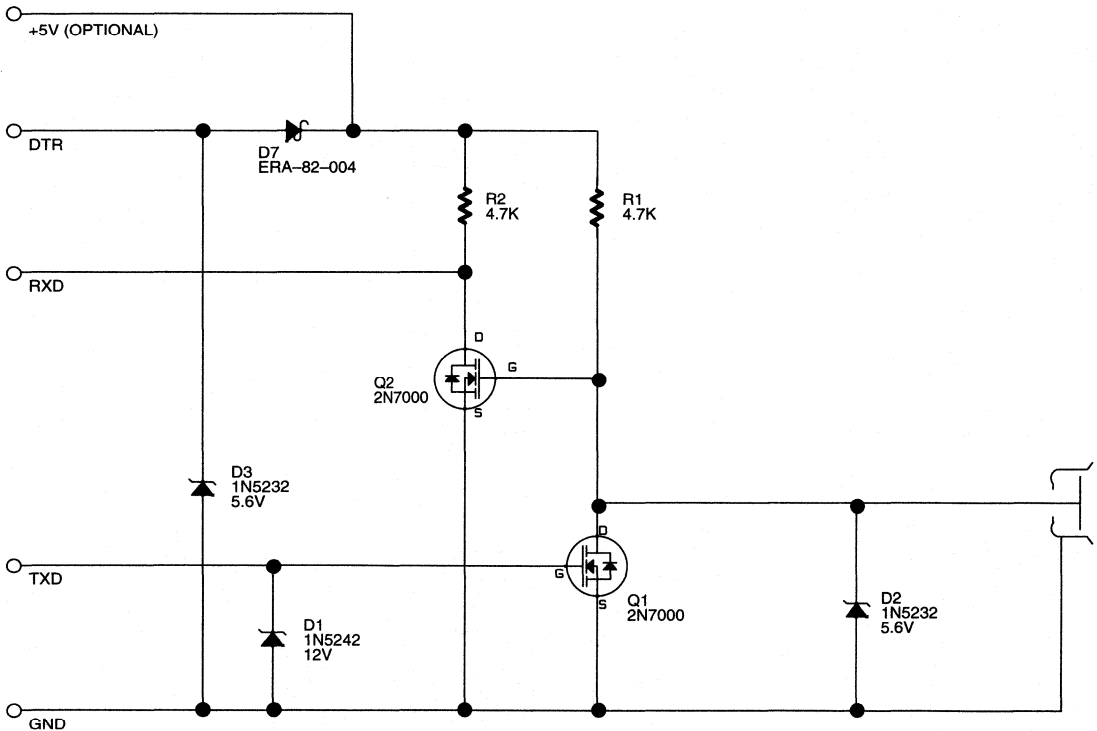
5V RS232 READ ALL WAVEFORMS Figure 10a
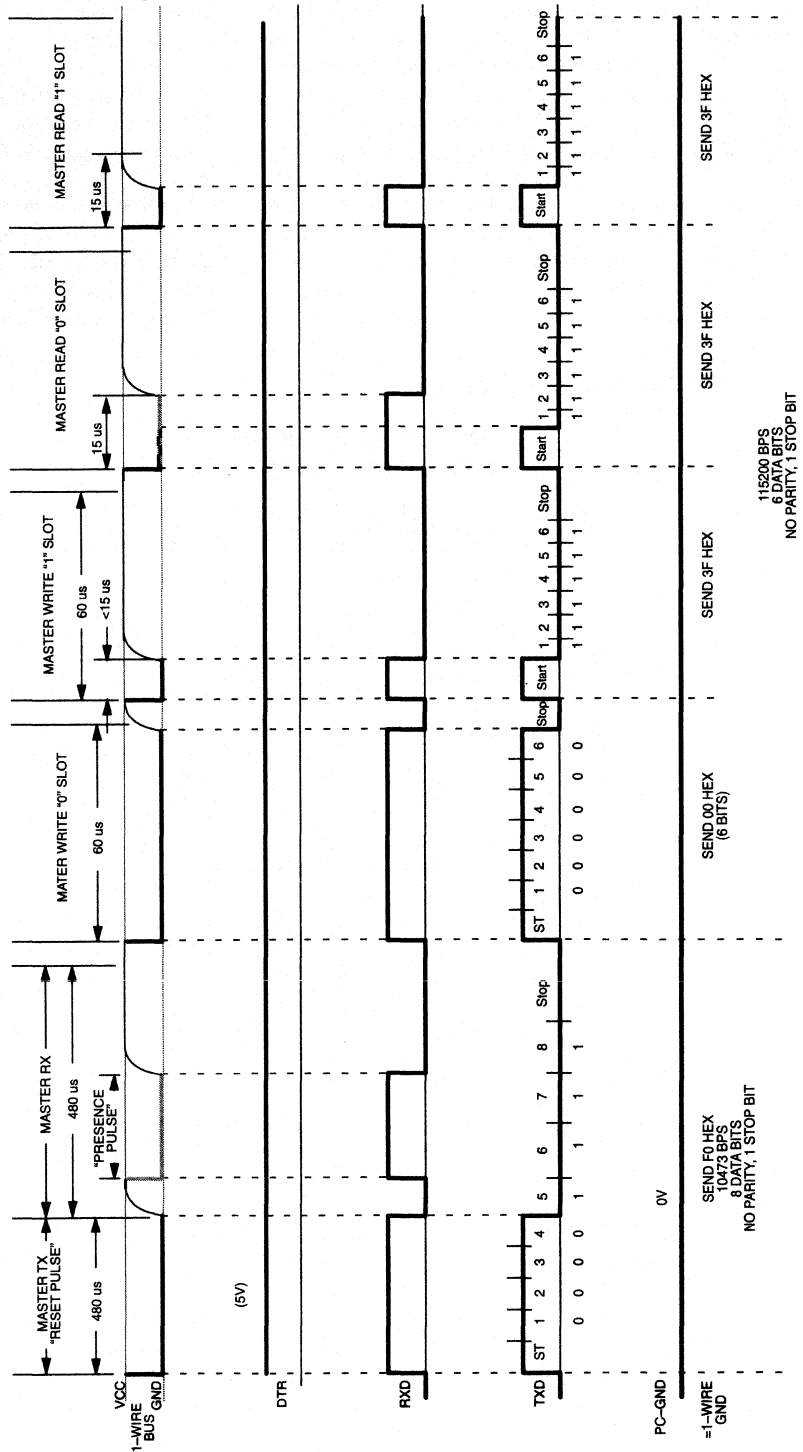
C. TTL R/W All (Voltage Converter)

The circuits described so far can read all Touch Memories and write NVRAM–based devices. For other technologies, however, voltage requirements other than 5V are necessary. One important group of Touch Memories, called Add–Only Memories, is based on EPROM technology and therefore needs a programming pulse of 12V to copy data from the scratchpad to the EPROM cells. Another device, the Touch Thermometer, will operate on 5V but requires a low impedance pull–up to 5V while measuring the temperature. To fulfill the requirements of these and future devices, the circuits described above need to be upgraded. These two new functions require two more signals and the 12V programming supply can be provided by a DC–to–DC voltage converter.

The complete circuit of such a universal interface in a TTL–version is shown in Figure 11. It is a compatible superset of Figure 9. The components Q1, R1, D1, D2 and their functions are the same as before. Additional requirements include the control input $\overline{\text{PGMEN}}$ with the diode D4 for ESD protection, the diode D5, the cascaded inverters R3, Q3, R4 and Q4 & R5, C1, the pull–up switch Q5 and the controlled voltage converter IC1 with its external components L1, D6 and C2.

TTL R/W ALL CIRCUIT Figure 11
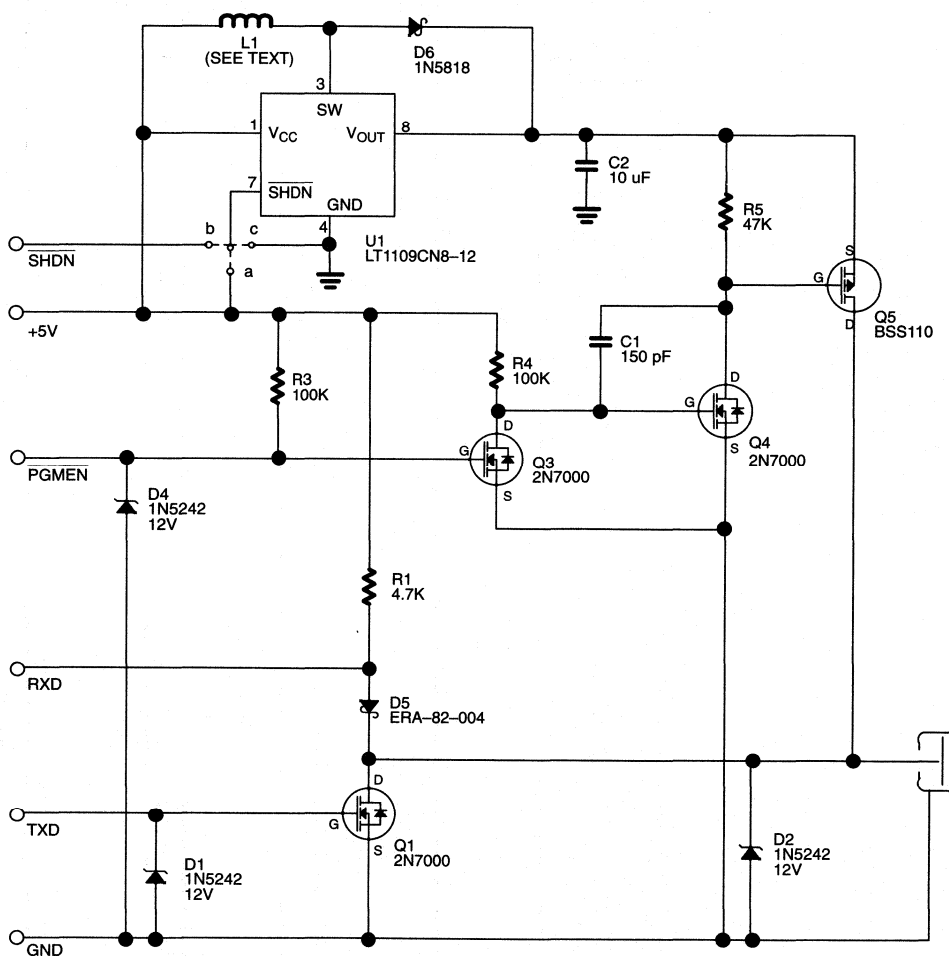
$\overline{\text{PGMEN}}$ is the active low input to activate the pull–up switch. If not connected, $\overline{\text{PGMEN}}$ will be held high through R3 to avoid unwanted activation of the pull–up switch. The voltage converter can be controlled in three ways: a) hard–wired for continuous operation, b) activated by an external signal, or c) permanently shut down. Case a) is intended for applications which never require a strong 5V pull–up. If there is no control signal available from the master and strong 5V pull–up as well as EPROM programming is required, then a mechanical switch can be used to switch between case a) and c). Case b) offers the most flexibility. For EPROM programming $\overline{\text{SHDN}}$ needs to be high, for strong 5V pull–up it should be low. If the voltage converter is shut down, L1 and D6 together with a conducting Q5 provide the required low–impedance path to 5V.

If the signal $\overline{\text{PGMEN}}$ becomes active (i.e., is low) then the voltage at the gate of Q4 rises from 0V to approximately 5V, causing Q4 to conduct. This is equivalent to feeding a low level from $\overline{\text{PGMEN}}$ to the gate of Q5. The capacitor C1 between gate and drain of Q4 slows down the rise and fall of Q4's gate–source voltage and therefore determines the ramp rate of the programming pulse. As soon as the gate voltage of Q5 changes from the quiescent state of 5 or 12V (depending on mode of operation) to 0V, the P–channel transistor becomes conducting and pulls the 1–Wire bus either to 12V (if the voltage converter is on) or to 5V, bypassing R1.

From the strictly logical point of view, the double inversion (Q3, Q4) is unnecessary. The reasons for this circuit are to convert from a TTL–level system to a 12V system (5 volts on the gate of Q5 is not sufficient to turn the transistor off if the voltage converter is running), to avoid high voltage feedback from the voltage converter through R5 to the TTL–level control input and to extend the rise and fall time of the 12V programming pulse to the required minimum of 5 µs. High voltage feedback from the 1–Wire bus to the receive input of the microprocessor is avoided by the diode D5, which becomes conducting only when the voltage on the 1–Wire bus is lower than 5V.

The monolithic voltage converter IC1 requires L1, C2 and D6 for operation. It is activated by a low level at its TTL–compatible input $\overline{\text{SHDN}}$. The right choice of L1, D6 and C2 is essential for reliable operation. D6 is a Schottky diode, recommended part number 1N5818, C2 is a low ESR tantalum capacitor of 10 µF. L1 must be a low ESR device between 20 to 100 µH, capable of

withstanding current peaks of approximately 0.5 A without magnetic saturation. To avoid EMI problems, L1 should be a pot–core or toroid type; a rod core type is not recommended. For further details on the voltage converter and its external components please refer to the appropriate data sheet and application notes. The LT1109 is just one example of available parts. Other manufacturer's components or modules can be used as well.

The duration of the programming pulse (pulse width of $\overline{\text{PGMEN}}$) or the strong pull–up is determined by software. Program examples are given later in this document.

D. 5V RS232 R/W All (Voltage Converter)

The universal upgrade of the interface of Figure 10 is shown in Figure 12. The components Q1, R1, D1, D2, Q2, R2, D3 and their functions are the same as before. Additional requirements include the control signal RTS with the diode D4 for ESD protection, the diodes D5, D7, the cascaded inverters R3, Q3, R4 and Q4 & R5, C2, the pull–up switch Q5 and the controlled voltage converter IC1 with its external components L1, D6 and C2.

RTS is used to activate the pull–up switch Q5. DTR may act as power supply, if it is able to source 5V at 25 mA. The strong pull–up is made possible by L1 and D6 if the voltage converter is shut down. If idle (i.e., the bit controlling RTS in the Modem Control Register of the UART is set to 1), RTS will be at 5V. To activate the pull–up switch, the RTS bit in the UART (see Figure 8) must be cleared to 0. This will cause a 0V level at the RTS pin of the RS232 connector. The DTR bit in the Modem Control Register of the UART must be set to 1. If power is supplied from the outside, the status of DTR becomes a don't care. In this case, D7 prevents driving the DTR line with the external power supply.

For reliable operation of this circuit, DTR must reach a high level of 5V minimum. If this is not possible, an external 5V supply must be connected as shown in Figure 12.

The function of the other components of this circuit has already been explained in the section IV.C. RTS is equivalent to $\overline{\text{PGMEN}}$ in Figure 11. D5 now prevents feedback from the 1–Wire bus to the internal power supply during the programming pulse. The duration of the programming pulse (pulse width of RTS) cannot be controlled by the UART alone. Software to provide the correct timing is found later in this document.

5V RS232 R/W ALL CIRCUIT Figure 12
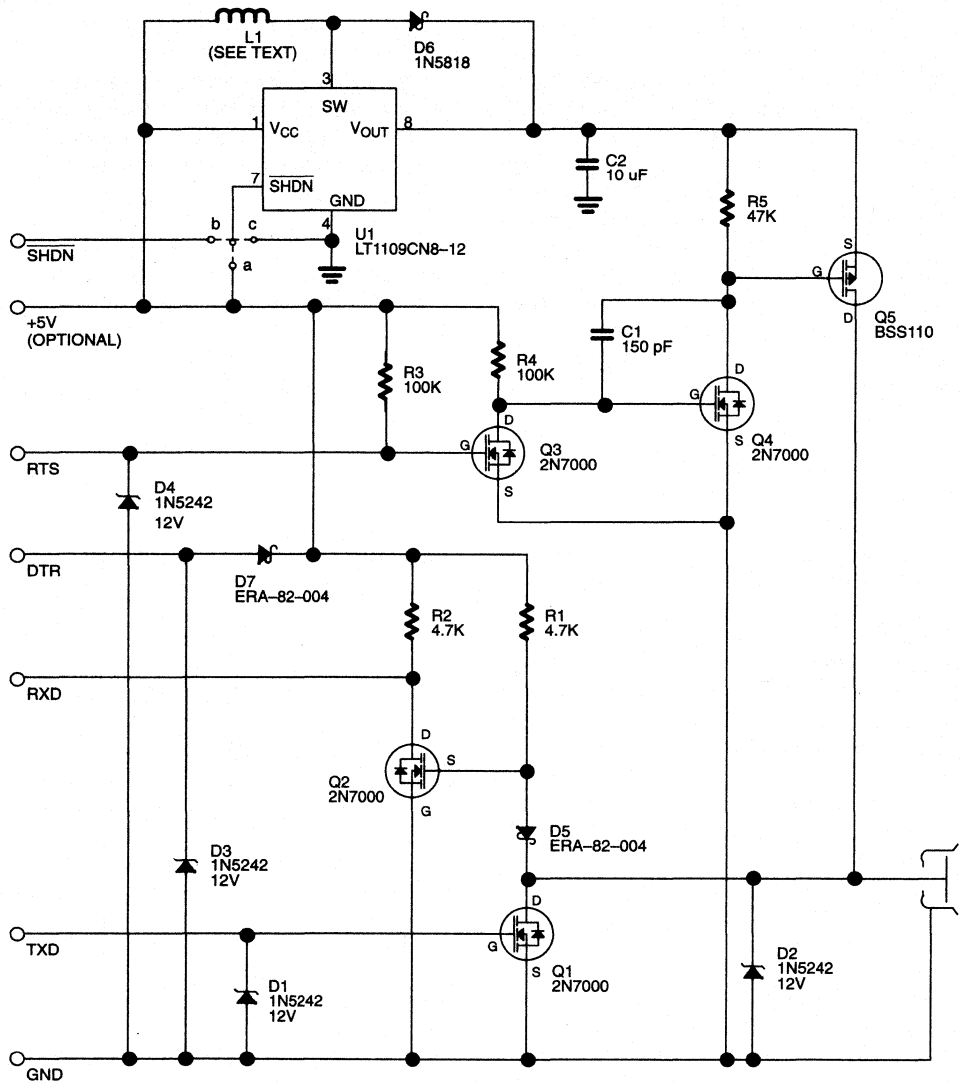
V. CIRCUITS FOR 12V RS232 INTERFACES (COM PORT)

A. Read All

If equipment has a true RS232 port using current limited drivers with voltage capabilities of at least ±8V, then a simple passive circuit is sufficient to interface to the 1–Wire bus. Figure 13 shows all details; the waveforms are found on Figure 13a. This interface operates on the same software as the circuit shown in Figure 10. The waveforms at RXD and TXD with respect to the computer's ground are basically the same. The major difference is that instead of 0V, a true negative voltage will be found, representing the idle state or a logic 1. Neglecting absolute voltage levels, the waveform observed at RXD is essentially an inversion of the waveform that would be observed on a 1–Wire data line for a 0V to 5V system.

The ground potential of the computer is different from the 1–Wire ground. This allows the Touch Memory to experience typical voltage levels (0V to 6V) on the 1–Wire data line relative to the 1–Wire ground, while the serial port generates both positive and negative voltages relative to the serial port ground. D1 clamps the data line to a constant potential of nominally 3.9V. The time slots for 1–Wire communication are generated by changing the potential of the 1–Wire ground with respect to the 1–Wire data line. The 1–Wire pullup resistor is in the path from the 1–Wire ground to TXD, which provides the voltage for the 1–Wire bus. D2 limits the voltage swing on the 1–Wire bus to a maximum value of 6.2 Volts. Since DTR is kept at 3.9V, D2 also limits the most negative voltage occurring at RXD to –2.3V. D2 is conducting only when the voltage at TXD is negative with respect to the computer's ground. D3 limits the voltage between 1–Wire ground and 1–Wire data when the voltage at TXD is positive. D4 couples TXD with RXD when TXD is positive and bypasses R1 to provide a low resistance path to initiate a time slot on the 1–Wire bus. D4 is nonconducting when the voltage at TXD is negative with respect to the computer's ground. If a Touch Memory pulls the 1–Wire data line low (e. g. at a presence pulse or when sending out a zero data bit), it shorts DTR to RXD, resulting in a positive voltage at RXD. This positive voltage arrives as a zero in the UART's receive data register.

Probing the 1–Wire bus at the data contact and the 1–Wire ground with an oscilloscope, would show nothing unusual, except that the voltage swing is at the upper end of the tolerable range. Probing RXD and TXD with another oscilloscope hooked up at the computer's ground, would also reveal quite normal waveforms. The only thing one might be concerned about is the poor negative voltage of about –2.3V found at RXD. Probing all three signals (1–Wire bus, TXD, RXD) with one oscilloscope hooked up at the computer's ground, would show a nearly constant voltage of 3.9V at the 1–Wire bus.

The positive voltage of approximately +4.1V at RXD is well within the RS232–specification, the negative voltage is slightly out of specification. Since the total swing on the 1–Wire bus is limited to approximately 6V by the characteristics of Touch Memories, a more negative voltage could be produced by replacing D1 by a zener diode of 3.2V, for example. Unfortunately, this would increase the permanent current load for DTR.

Fortunately, real RS232 receivers are more sensitive to a weak positive voltage than they are to a poor negative voltage. Although it is not completely within the specification of RS232, this interface with the components specified on Figure 13 has proven to be reliable with most desktop personal computers. Small computers, especially battery powered models, might not have the required current capability to run this interface. In case of difficulties, one should use the 5V RS232 interface instead.

This interface is shipped with the Touch Memory Starter Kit DS9092K and is also sold separately as COM Port Adapter DS9097. It is applicable for reading all Touch Memories and writing to SRAM–based devices. Due to the high pullup voltage and the low pullup resistor, this interface together with the right software also allows operation of the Touch Thermometer directly through the COM Port. Software to operate this interface is explained later in this document.

12V RS232 READ ALL CIRCUIT Figure 13
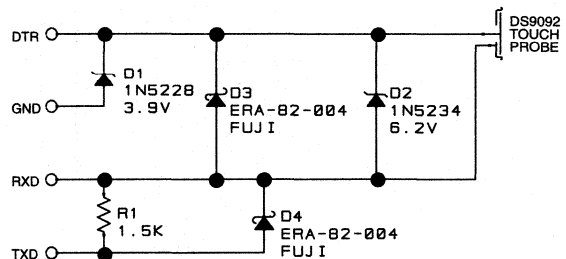
12V RS232 READ ALL WAVEFORMS Figure 13a
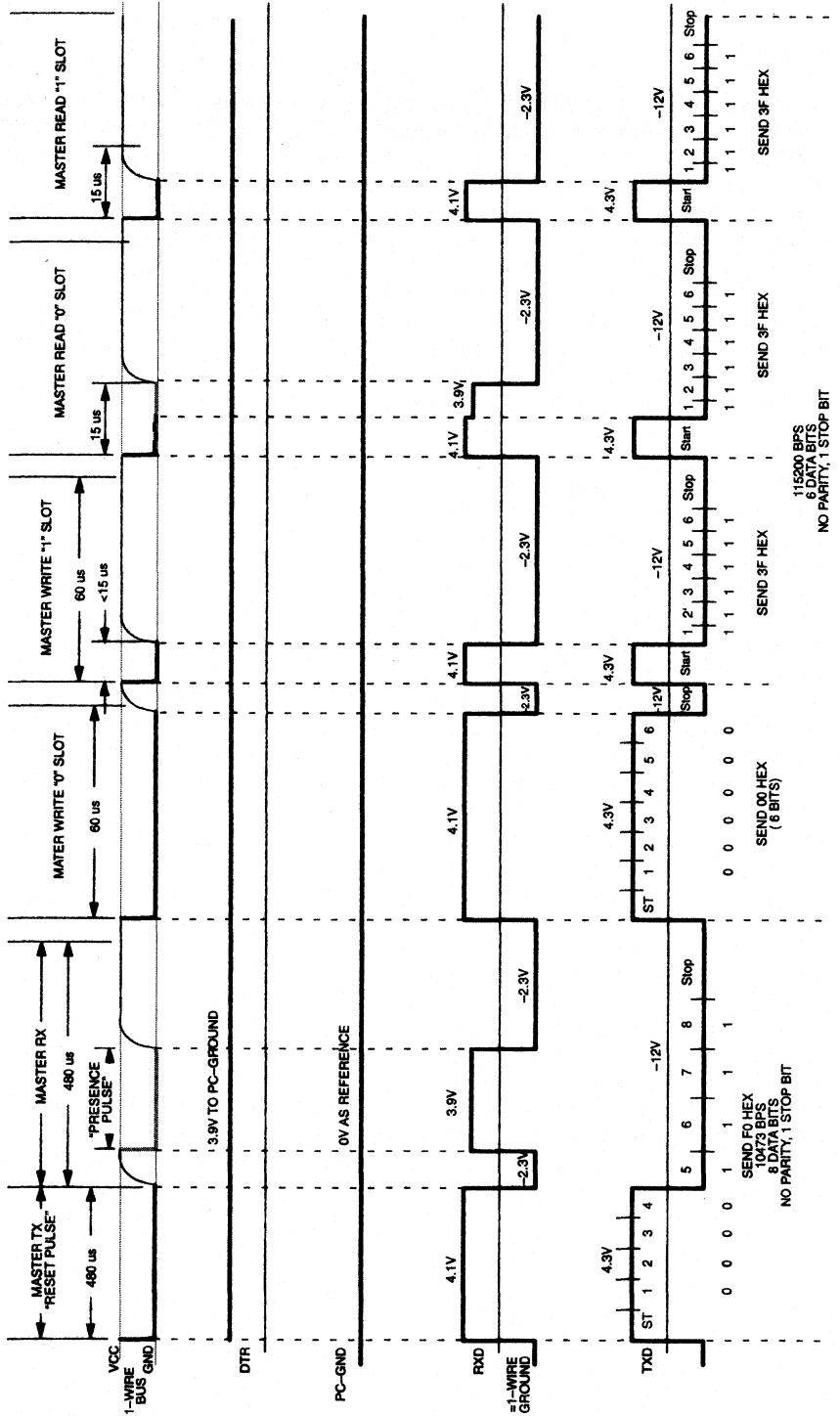
B. R/W All

The simple adapter of Figure 13 can be upgraded for programming Add–Only Memories. Figure 14 shows the details. R1 and the diodes D1 to D4 are the same in both circuits. There is a new signal in use, called RTS. When doing normal 1–Wire communication, RTS is constantly at a high positive voltage of nominally +12V. As long as the voltage at RTS remains positive, Q1 and Q2 are conducting. This allows D1 and D2 to provide the same functions as in Figure 13. Since D1 and D2 are conducting, there is not enough voltage across D5 and D6 to draw any current. The gate–source voltage of Q3 is determined by R2, R4, the voltage between DTR and RTS and the position of the contacts inside the connector for the external DC supply. D7 prevents current flow from source to drain through the substrate diode of Q3 when VDS is negative. If no external power supply is connected, R2 and R4 form a voltage divider providing a negative VGS for Q3. With an external power supply connected, R2 will hold the gate of Q3 at the same level as the source (VGS = 0V). None of these conditions will allow any current flow through Q3. With a positive voltage at RTS, the p–channel transistor Q4 remains non–conducting, regardless if an external power supply is connected or not. Thus the upgraded circuit behaves the same as the simple COM–Port adapter.

To generate a programming pulse for EPROM based devices, RTS needs to be switched to a negative voltage. This is done under software control by simply clearing the associated control bit in the UART and resetting the bit as soon as the programming pulse needs to be ended. Depending on whether an external supply is connected or not, the behavior of this circuit is slightly different.

If there is no external supply, Q4 has no function and the following sequence occurs (Figure 14a): A negative voltage at RTS will switch off Q1 and Q2, activating D5 and D6 instead of D1 and D2. This increases the voltage at DTR from 3.9V to 6.8V and defines a limit of 12V for the voltage on the 1–Wire bus. Simultaneously, the negative voltage at RTS will make VGS of Q3 equal to the voltage between DTR and RTS, which is a positive value. This will switch Q3 into a conducting state, feeding the negative voltage from RTS through D7 to the

1–Wire ground. As soon as the voltage on the 1–Wire bus reaches 12V, D6 will become active as voltage limiter. Thus the voltage on RTS is limited to approximately –5.4V. During the programming pulse, the voltage at RXD will be –5.2V instead of –2.3V. This has no impact on the logic of the UART, since a positive voltage is required to trigger the reception of a character.

The COM–Port powered mode of this circuit works properly if the RS232–drivers are able to provide enough current for programming. If more than one bit of the addressed memory byte need to be altered, then more current is needed during the programming pulse. Depending on the RS232 drivers, the available energy may not be sufficient for programming all bits of a byte. There are two possible solutions to this problem. One possibility uses an adaptive programming algorithm, where multiple passes are made for each byte. For example the software may program the first four bits of every byte on the first pass, and the remaining bits on the second pass. The other possibility is to provide an auxiliary energy source, i.e., by connecting an external 12V DC supply.

If the external supply is connected and operating, the contacts inside the connector will open. This prevents any current flow through Q3. As the voltage at RTS goes negative, Q1, Q2, D5, D6 will do the same as without the external supply. Instead of having no function, Q4 now will connect the negative end of the external voltage source with the 1–Wire ground, providing the desired low–impedance programming voltage. Since RTS has no other load than the gates of four transistors, there will be the full voltage swing. The voltages at RXD and TXD are again determined by D5 and D6 and exhibit the same waveforms as before.

This upgraded version of a COM–Port adapter is shipped with the EPROM–supporting revision of the Touch Memory Starter Kit DS9092K and is also available as the DS9097E. It is applicable for reading and writing all Touch Memories. Due to the high pullup voltage and the low pullup resistor, this interface together with the right software allows operation of the Touch Thermometer directly through the COM Port.

12V RS232 R/W ALL CIRCUIT Figure 14

PART NUMBERS ARE SPECIFIED
FOR SURFACE-MOUNT TECHNIQUE.
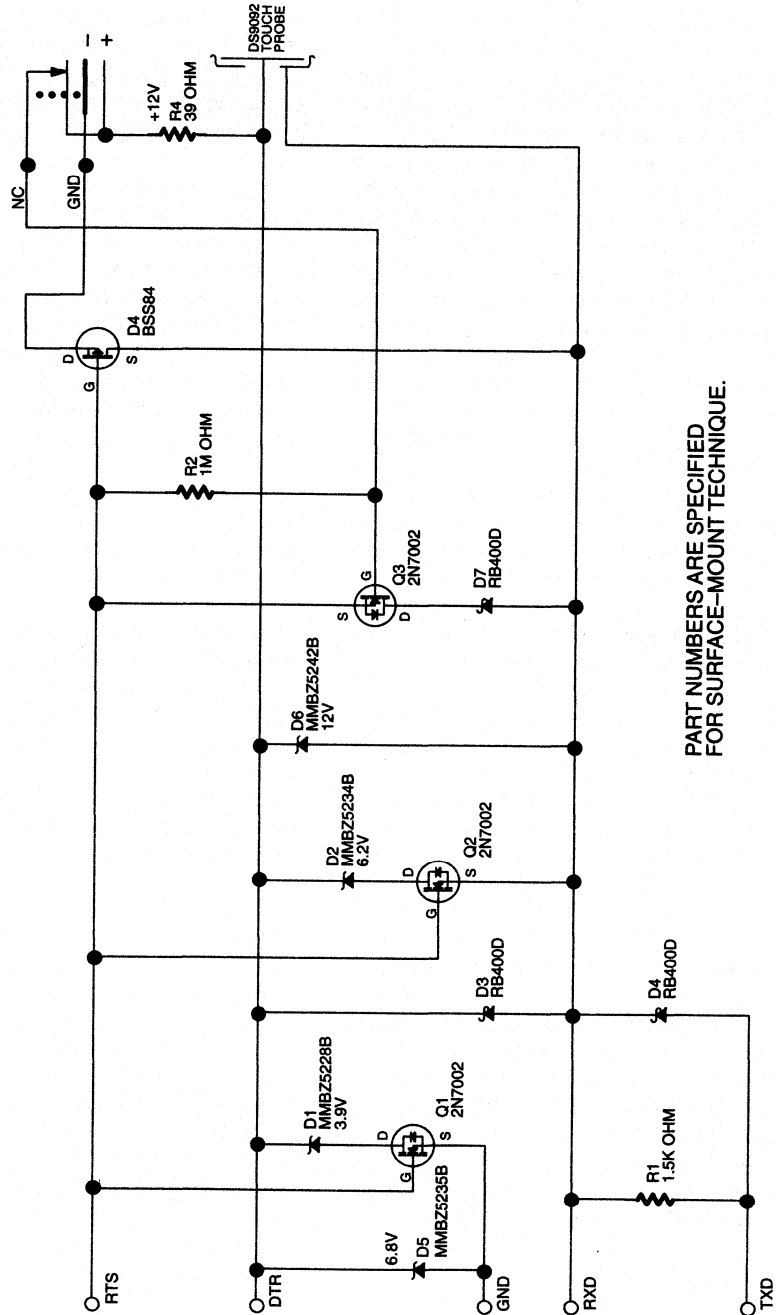
12V RS232 R/W ALL PROGRAMMING WAVEFORMS Figure 14a

```
RTS    ___12V____              _____          ___12V____
                 |            |                    |        |
                 |_____|                    |_____|
                      -5.4V

DTR    __3.9V (D1)__  _____6.8V (D5)_____  __3.9V (D1)__
                    |                                  |
                    |_____|

RXD    __-2.3 (D2)__              _____  __-2.3 (D2)__
                    |            |                     |
                    |_____|                     |
                         -5.2V (D6)

TXD    _____-12V_____

                         _____12V (D6)_____
1-WIRE  __6.2V (D2)__   |                              |   __6.2 (D2)__
BUS                  |  |                              |  |
                     |__|                              |__|
```
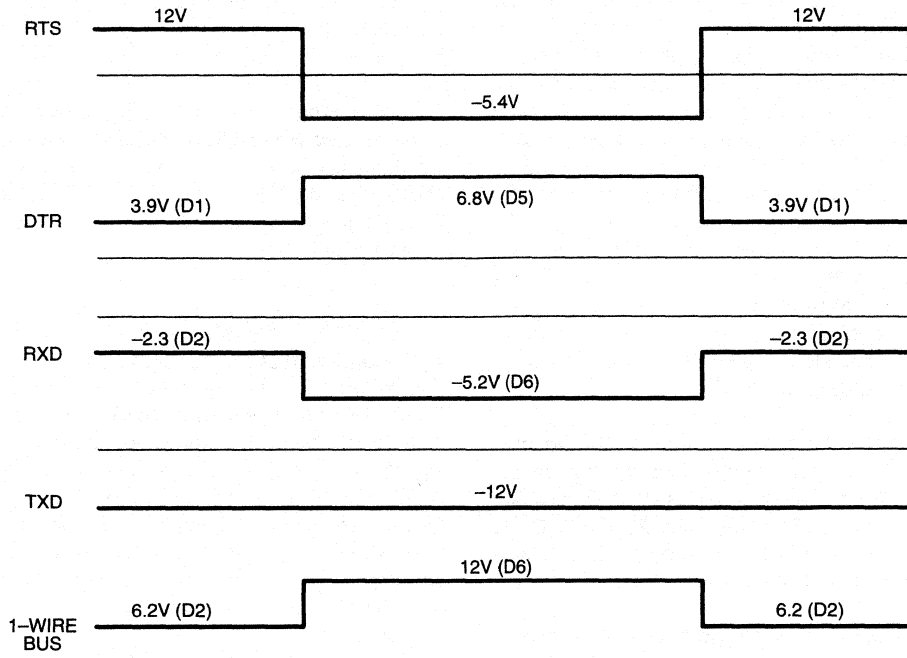
VI. INTRINSICALLY SAFE

A. Definition

Touch Memories satisfy a very high safety standard which makes them well suited for applications in hazardous environments. Touch Memories meet the UL#913 (4th Edit.) requirements as Intrinsically Safe Apparatus, Approved under the Entity Concept for use in Class I, Division 1, Group A, B, C, and D Locations. Intrinsically safe means that the probability of causing an explosion or accident in hazardous locations is not increased when using approved equipment, even if this equipment should be faulty. Since Touch Memories have been certified as intrinsically safe under the entity concept, they may reside in a hazardous environment but cannot be read or written in that environment unless the unit performing the reading or writing has also been certified as intrinsically safe under the same entity concept.

The Touch Memory might be affixed to a tanker truck and contain maintenance information. The truck could enter and exit a hazardous location (fuel depot, for example) without concern over an increased potential for an explosion due to the Touch Memory device. If no intrinsically safe reader/writer is available, any updates to the Touch Memory would have to occur outside of the hazardous area. Should it be necessary to read or update the Touch Memory within the hazardous area (record the fuel dispensed, for example), a certified intrinsically safe reader/writer must be used.

There are two options available to provide an intrinsically safe reader/writer. The first involves taking any piece of equipment capable of reading or writing Touch Memories and submitting it to an approved NRTL (Nationally Recognized Testing LAB) to be tested and certified under the same entity concept as the Touch Memories. The second option takes advantage of reader/writer equipment that has already been tested and certified as intrinsically safe (laptop computer, handheld reader, etc.) and uses those test results along with a specially designed Touch Memory probe adapter to create an entire system that is intrinsically safe. One such unit utilizing this second option is the PSION Organizer II, Model LZ64. This unit is used as an example to show how an intrinsically safe system can be realized.

B. Example of an Intrinsically Safe System

To use the LZ64 as an intrinsically safe Touch Memory Reader/Writer, an adapter is required. This adapter limits voltages and currents to safe values in the event a fault occurs. In the case of a fault, due to its internal construction, a maximum voltage of 17.22V at the 16–pin connector of the LZ64 may occur allowing a current of up to 1.55A. The adapter discussed here limits these values at the Touch Probe to a maximum voltage of 15V and a maximum current of 10mA (Values required by the Touch Memories in order to be certified as intrinsically safe). These values together with the maximum inductance of 18 µH and maximum capacitance of 0.2nF of a Touch Memory fulfill the requirements for a complete system that is intrinsically safe according to UL specifications.

Figure 15 shows the complete circuit of the adapter. Essential components are the four current shunts Q1/R1 to Q4/R4, three Zener diodes D2 to D4 and one self–resetting fuse SS1. The Schottky diode D1 is optional. It protects Touch Memories by suppressing negative undershoots on the 1–Wire bus.

If by fault the LZ64 presents up to 17.22V at its connector and there is an open circuit at the Touch Probe, then the 12V Zener diodes D2 to D4 will limit the voltage at the Touch Probe to a value well below 15V. The intrinsically–safe regulations demand that this limitation will work correctly even if two of the protecting devices should fail. Therefore three Zener diodes are provided instead of one. If a Zener diode fails, it will either represent a short or it will be non–conducting. In either case, the voltage at the Touch Probe is limited to a safe value.

If by fault the LZ64 presents up to 17.22V at its connector and there is a short at the Touch Probe, then a current will flow through the resistors R1 to R4. The voltage drop across these resistors acts to turn on their respective transistors and causes base currents to flow. These base currents multiplied by the current gain of the transistors will direct most of the current to ground and limit the current at the Touch Probe to less than 10mA. Q1 will sink most of the current. Two of these Q/R stages are required to limit the current available at the Touch Probe under worst case conditions to less than 10mA. The other two current shunts are redundant since again the circuit must operate correctly with up to two faults. The self–resetting fuse with a trip point below 100mA

will open in less than one second and thus prevent thermal damage to the transistors. It will also serve as an indicator to the operator that a fault or malfunction has occurred.

Although this adapter is designed for use with the LZ64, it can be used with similar intrinsically safe equipment with the same or lower faulted open–circuit voltage or short–circuit current to form a Touch Reader/Writer. This adapter does not impede writing to EPROM based Touch Memories.

VII. COMMENTED SOFTWARE

A. Software Architectural Model

The software that manages data transfer to and from Touch Memories is related to the ISO reference model of Open System Interconnection (OSI). This model specifies a layered protocol having up to seven layers, denoted as Physical, Link, Network, Transport, Session, Presentation, and Application. The Application layer represents the final application designed by the customer. A Session layer may or may not be needed, depending on the environment in which the Touch Memories are used.

According to the ISO model, the electrical and timing requirements of Touch Memory and the characteristics of the 1–Wire bus comprise the Physical layer. Details have already been given in section II of this document.

The Link layer defines the basic communication functions of Touch Memories, which are the hardware dependent functions of Reset, Presence Detect and bit transfer. Circuits for interfacing Touch Memories and general information on the software to operate these interfaces have already been presented in sections III, IV, and V. In this section, the software itself, specifically the functions TouchReset and TouchByte, are discussed in detail.

The Network layer provides the identification of Touch Memories and the associated network capabilities based on the unique lasered identification number. Software for this layer is built up using the low–level functions of the Link Layer. Since this software is independent of any particular interface, it is not within the scope of this document.

INTRINSICALLY SAFE ADAPTER (EXAMPLE) Figure 15
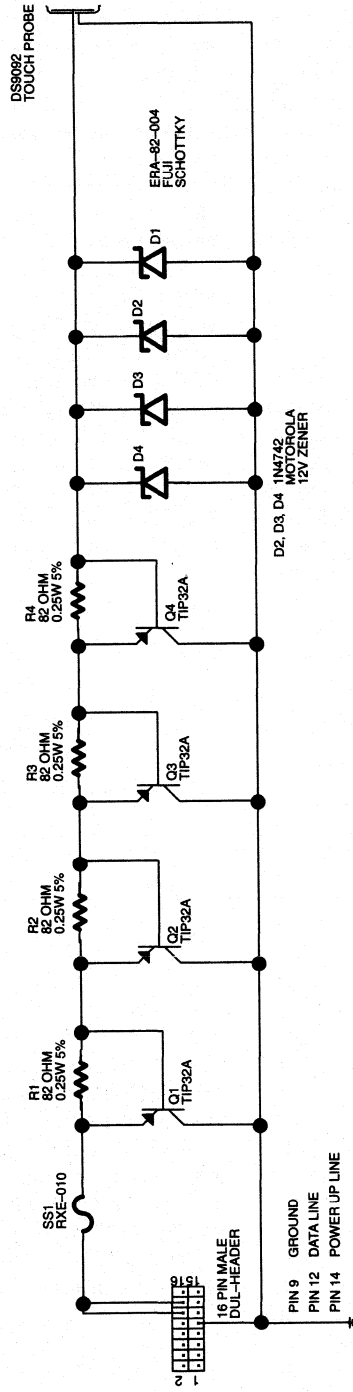
The Transport layer is responsible for the data transfer between the non–ROM segments of Touch Memories and the master, and the data transfer from the scratchpad to the final storage areas and special registers of the Touch Memory. Due to their EPROM technology, Add–Only Memories require special attention for writing data. The Touch Thermometer may require special hardware together with appropriate software to do a temperature measurement. To comply these devices, the hardware specific function PulWidth has been provided on the Transport layer. Details are given in this chapter. All other software of the transport layer is independent of the type of interface, and therefore is not discussed here.

The layers Link, Network, and Transport are the foundations of the Presentation layer. This layer provides a DOS–like file system supporting functions like Format, Directory, Type, Copy, Delete, Optimize, and integrity check. Since the Presentation layer itself is based on software of the lower layers, its software is independent of any particular interface. Full details of the Presentation layer are given in the Touch Memory EXecutive

DS0620. For software examples beyond the hardware dependent functions TouchReset, TouchByte and PulWidth, please refer to the "Book of DS19xx Touch Memory Standards" and the Touch Memory Starter Kit DS9092K.

A matrix that indicates which software of this section matches with which hardware is given in Table 3. For the 5V TTL type interface, assembly language code for the 8051 has been provided. For the group of interfaces based on the UART 8250, code examples in Pascal and C are included. This particular software has been adapted to and verified with IBM–compatible PCs employing a 8253 timer at 2.3863633 MHz and running under DOS. The timing is practically independent of the CPU clock rate. Under WINDOWS there is a lot more software being executed around an application program. This overhead introduces a significant influence from the CPU clock rate to the desired timing with the function PulWidth. The functions TouchReset and TouchByte are timed by the UART only and therefore are independent of the operating system.

SOFTWARE/HARDWARE MATRIX Table 3

LANGUAGE	8051 ASM	PASCAL AND C
TIMING	CPU CRYSTAL	8250 UART (1.8 MHz) and 8253 TIMER (2.4 MHz)
Electric Type	5V TTL	5V RS232, 12V RS232

SRAM R/W EPROM Read	TouchReset, TouchByte 1.8 or 11 MHz	TouchReset and TouchByte Pascal or C–Language
EPROM Write	0.5 ms pulsewidth: PULWIDTH(1) at 1.8 MHz PULWIDTH(6) at 11 MHz	0.5 ms pulsewidth: PULWIDTH(1193) under DOS

B. TTL–Interface R/W All

As a representative for all microprocessor timed 1–Wire interfaces the industry–standard 8051 microcontroller has been chosen. The following pages show two versions of assembly language code to provide the functions TouchReset and TouchByte. The first example is written for an 11.0592 MHz crystal, the second one for 1.8432 MHz. The higher frequency is very common since it supports all standard baud rates with the highest accuracy. The lower frequency is the lowest that can comply with the 1–Wire timing. The port to be used as 1–Wire bus is defined in the parameter DATA_BIT. Parameter passing from the subroutines TouchReset

and TouchByte is very simple: If a Touch Memory is present on the 1–Wire bus, TouchReset will return a set carry flag; otherwise carry is cleared. To send one byte to the 1–Wire bus, the byte to be sent is loaded into the accumulator before calling TouchByte. If one intends to read, the accumulator is loaded with FFH. This generates correct Read Data Time Slots and returns data from the 1–Wire bus to the calling program through the accumulator. These conventions are valid for both versions of TouchReset and TouchByte.

The procedure to generate a programming pulse is the same for both clock frequencies. It generates a 0.5 ms

LOW pulse at the port named PROGRAM. If the clock frequency is 1.8 MHz, then the accumulator needs to be loaded with 1 before calling this procedure. For a clock frequency of 11 MHz the value of 6 loaded into the accumulator will generate a pulse of the same duration. Software considering the Touch Thermometer will be published as soon as the device is available. Generally, it is not a difficult task to adapt the procedure PulWidth to a pulsewidth of 2 seconds.

8051 ASSEMBLY LANGUAGE, 11.0592 MHz

```
DATA_BIT   BIT        P0.0

; The following 8051 code uses a bi-directional port pin (specified by
; DATA_BIT) for 1-wire I/O. This code was written for an 11.0592 MHz
; crystal.
;
;           Procedure TouchReset
;
;           This procedure transmits the Reset signal to the Touch
;           Memory and watches for a presence pulse. On return,
;           the Carry bit is set if a presence pulse was detected,
;           otherwise the Carry is cleared. The code is timed for
;           an 11.0592 MHz crystal.
;
TOUCHRESET:
           PUSH      B                    ;           Save the B register.
           PUSH      ACC                  ;           Save the accumulator.
           MOV       A,         #4        ;           Load outer loop variable.
           CLR       DATA_BIT             ;           Start the reset pulse.
           MOV       B,         #221      ;    2.     Set time interval.
           DJNZ      B,         $         ;  442.     Wait with Data low.
           SETB      DATA_BIT             ;    1.     Release Data line.
           MOV       B,         #6        ;    2.     Set time interval.
           CLR       C                    ;    1.     Clear presence flag.
WAITLOW:
           JB        DATA_BIT,  WH        ;           Exit loop if line high.
           DJNZ      B,         WAITLOW   ;           Hang around for 3360
           DJNZ      ACC,       WAITLOW   ;             us if line is low.
           SJMP      SHORT                ;           Line could not go high.
WH:
           MOV       B,         #111      ;           Delay for presence detect.
HL:
           ORL       C,         /DATA_BIT ;  222.     Catch presence pulse.
           DJNZ      B,         HL        ;  222.     Wait with Data high.
SHORT:
           POP       ACC                  ;           Restore accumulator.
           POP       B                    ;           Restore B register.
           RET                            ;           Return.
;
;           Procedure TouchByte
;
;           The procedure TouchByte sends the byte in the accumulator
;           to the Touch Memory and simulatneously returns one
;           byte from the Touch Memory in the accumulator. Note that
```

```
;          the NOPs in the following code are intended to give the
;          optimum performance when using a 11.0592 MHz crystal.
;          Their purpose is to make the pulses as long as
;          possible consistent with the Touch Memory timing
;          constraints. When using other crystal frequencies,
;          the delays in this code should be adjusted to conform
;          to the timing requirements of the Touch Memory.
;
TOUCHBYTE:
          PUSH      B                          ;          Save the B register.
          MOV       B,        #8               ;          Setup for 8 bits.
BIT_LOOP:
          RRC       A                          ;   1.     Get bit in carry.
          CALL      TOUCHBIT                   ;   2.     Send bit.
          DJNZ      B,        BIT_LOOP         ;   2.     Get next bit.
          RRC       A                          ;          Get final bit in ACC.
          POP       B                          ;          Restore B register.
          RET                                  ;          Return to caller.
TOUCHBIT:
          CLR       DATA_BIT                   ;   1.     Start the time slot.
          NOP                                  ;   1.     Delay to make sure
          NOP                                  ;   1.       that the Touch Memory
          NOP                                  ;   1.         sees a low for at
          NOP                                  ;   1.           least 1 microsecond.
          MOV       DATA_BIT, C                ;   2.     Send out the data bit.
          NOP                                  ;   1.     Delay to give the
          NOP                                  ;   1.       data returned from
          NOP                                  ;   1.         the Touch Memory
          NOP                                  ;   1.           time to settle
          NOP                                  ;   1.             before reading
          NOP                                  ;   1.               the bit.
          MOV       C,        DATA_BIT         ;   1.     Sample input data bit.
          PUSH      B                          ;   2.     Save B register.
          MOV       B,        #12H             ;   2.     Delay until the end
          DJNZ      B,        $                ;  36.       of the time slot.
          POP       B                          ;          Restore B register.
          SETB      DATA_BIT                   ;          Terminate time slot.
          RET                                  ;          Return to caller.
;
          END                                  ; End of module._
```

8051 ASSEMBLY LANGUAGE, 1.8432 MHz

```
;          Touch Memory I/O Procedures for use with a 1.8432 MHz crystal.
```

TOUCHRESET:

```
          CLR       DATA_BIT                   ;  - Pull the data line low.
          MOV       B,        #35              ;  2 Hold the data line
          DJNZ      B,        $                ; 70    low for 481.77
          NOP                                  ;  1       microseconds.
          SETB      DATA_BIT                   ;  1 Release the data line.
```

```
            MOV       B,          #130        ;   2 Short circuit timeout.
            CLR       C                       ;   1 Presence pulse detector.
WAITLOW:
            JB        DATA_BIT, WAITHIGH     ; 260 Go look for Presence pulse.
            DJNZ      B,          WAITLOW    ; 260 Abort on short circuit.
            SJMP      ABORT                   ;     Short circuit (3.8877 ms).
WAITHIGH:
            MOV       B,          #18         ;   2 Prepare for high period.
HL:
            ORL       C,          /DATA_BIT  ;  36 Trap the Presence pulse.
            DJNZ      B,          HL          ;  36 Wait out 481.77 microsec.
ABORT:
            RET                                   Return.
TOUCHBYTE:
            MOV       B,          #8          ;     Prepare to move 8 bits.
BIT_LOOP:
            RRC       A                       ;     Move LSB to Carry.
            JC        SENDONE                 ;     If Carry then send 1.
            CLR       DATA_BIT                ;     Otherwise send 0.
            SJMP      DELAYSET                ;   2 Wait out rest of time slot.
SENDONE:
            CLR       DATA_BIT                ;     Start read/write 1 slot.
            SETB      DATA_BIT                ;   1 Set data line.
            MOV       C,          DATA_BIT   ;   1 Read data line.
DELAYSET:
            NOP                               ;   1 Delay 7 more cycles
            NOP                               ;   1    to produce enough
            NOP                               ;   1       delay to complete
            NOP                               ;   1          the time slot.
            NOP                               ;   1
            NOP                               ;   1
            NOP                               ;   1
            SETB      DATA_BIT                ;   1 Done (65.1 microseconds).
            DJNZ      B,          BIT_LOOP   ;     Repeat to send 8 bits.
            RRC       A                       ;     Align final result.
            RET                               ;     Return.
```

8051 ASSEMBLY LANGUAGE PULSEWIDTH (1.8432 AND 11.0592 MHz)

```
PROGRAM    BIT       Pn.i
;
;               This procedure generates a 0.5 ms low pulse on port
;               Pn.i of an 8051 microprocessor, where 0 <= n <= 3
;               and 0 <= i <= 7.  The frequency of the crystal, in
;               multiples of the minimum frequency 1.8432 MHz, must
;               be passed in the accumulator.
;
PULWIDTH:
            MOV       B,          #38     ; Number of loops at 1.8432 MHz.
            MUL       AB                  ; AB = # of loops given frequency.
            INC       B                   ; Adjust count value for
```

```
                                        ;    use with DJNZ instruction.
          PUSH     PSW                  ; Preserve state of interrupts.
          CLR      EA                   ; Inhibit all interrupts.
          CLR      PROGRAM              ; Bring the port pin low.
LOOP:
          DJNZ     ACC,      LOOP       ; Count while
          DJNZ     B,        LOOP       ;    pin is low.
          SETB     PROGRAM              ; Bring the port pin high.
          POP      PSW                  ; Restore state of interrupts.
          RET                           ; Return.
```

C. RS232 Interface R/W All

UARTs like the 8250 can be connected to any microprocessor to implement a RS232 type interface. The software to operate the UART mainly consists of reading and writing the UART's internal registers (Figure 8). This can easily be done in any high level language. Depending on the computer, the physical address of the UART will be different, but the crystal will usually be a 1.8432 MHz type. Not regarding the UART's physical address, the software examples for TouchReset and TouchByte given on the following pages are very general. The languages C and Pascal are very common and a variety of compilers is available.

Unfortunately, the UART does not control the timing of the signals DTR and RTS. It only allows activation or deactivation of these signals by setting or clearing bits inside its control registers. The timing itself is left to the microprocessor and its peripheral timing circuits. From the software developer's point of view this is a step backwards to assembly language, where every command and its execution time at a specified clock frequency need to be counted. For this reason it is not possible to provide machine–independent software to generate the programming pulse.

The most common computer using a 8250 type UART to implement a RS232 interface is the IBM–compatible PC. These machines employ a programmable interval timer 8253 running at 2.3863633 MHz for general timing purposes. This timer is involved in controlling the timing of the software examples of PulWidth. The pulsewidth is specified by a formal parameter passed to PulWidth. For 0.5 ms the value of this parameter is 1193 decimal. Under DOS, the software examples given below will perform accurately and almost independent of the CPU clock. Due to a very different environment and use of resources under WINDOWS, the pulses will be longer and also dependent on the CPU clock. This can be compensated for experimentally by reducing the value of the parameter passed to PulWidth. Software considering the Touch Thermometer will be published as soon as the device is available. For the 5V–type RS232 interface a pulsewidth of 2 seconds will be required for the strong pullup to 5V. The 12V RS232 interface has enough power available to run one Touch Thermometer without extra power switching.

C LANGUAGE FOR UART 8250 SYSTEMS

```
/*
    In the following C language code, 1-wire I/O is accomplished using the
    serial port of an IBM PC or compatible. The serial port must be capable
    of a 115,200 bps data rate. Setup must be called before any of the
    touch functions to verify the existence of the specified com port and
    initialize it.

    ----------------------------------------------------------------
    The setup function makes sure that the com port number passed to it
    is from 1 to 4 and has a valid address associated with it.
*/
uchar Setup(uchar CmPt)
{
```

```
        uint far *ptr = (uint far *) 0x00400000;
        uint SPA;

        /* check to see if it is a valid com port number and address */
        SPA = *(ptr+CmPt-1); /* get the address */
        if (CmPt < 1 || CmPt > 4 || !SPA )
            return FL;

        /* serial port initialization */
        outportb(SPA+3,0x83);   /* set DLAB */
        outportb(SPA  ,0x01);   /* bit rate is 115200 */
        outportb(SPA+1,0x00);
        outportb(SPA+3,0x03);   /* 8 dta, 1 stp, no par */
        outportb(SPA+1,0x00);   /* no interrupts */
        outportb(SPA+4,0x03);   /* RTS and DTR on */

        return TR;
}


/*--------------------------------------------------------------------
 * Do a reset on the 1 wire port and return  0 no presence detect
 *                                           1 presence pulse no alarm
 *                                           2 alarm followed by presence
 *                                           3 short circuit to ground
 *                                           4 no com port found
 *
 * The global variable 'com_port' must be set to the com port that the
 * DS9097 COM Port Adapter is attached to before calling this routine.
 *
 */
uchar TouchReset( void )
{
        uint SPA,F,X,Y,tmp,trst=0;
        uint far *ptr = (uint far *) 0x00400000;
        ulong far *sysclk = (ulong far *) 0x0040006c;
        ulong M;

        /* get the serial port address */
        SPA = *(ptr+com_port-1);

        /* return if there is no address */
        if (!SPA) return 4;

        /* serial port initialization */
        outportb(SPA+3,0x83);   /* set DLAB */
        outportb(SPA  ,0x01);   /* bit rate is 115200 */
        outportb(SPA+1,0x00);
        outportb(SPA+3,0x03);   /* 8 dta, 1 stp, no par */
        outportb(SPA+1,0x00);   /* no interrupts */
        outportb(SPA+4,0x03);   /* RTS and DTR on */
```

```
/* Initialize the time limit */
M = *sysclk +1;

/* loop to clear the buffers */
do { tmp = inportb(SPA+5) & 0x60;  } while (tmp != 0x60);

/* flush input */
while (inportb(SPA+5) & 0x1) X = inportb(SPA);

outportb(SPA+3,0x83);   /* set DLAB */
outportb(SPA+1,0x00);   /* baud rate is 10473 */
outportb(SPA  ,0x0B);
outportb(SPA+3,0x03);   /* 8 dta, 1 stp, no par */
outportb(SPA  ,0xF0);   /* send the reset pulse */

/* wait until character back or timeout */
do
{
    Y = inportb(SPA+5);
    F = Y & 0x1;
} while ( !F && (*sysclk <= M) );

if (F) X = inportb(SPA);
else    return 3;

if (X != 0xF0)             /* if more bits back than sent then there */
{                          /* is a device  if framing error or break */
    trst = TR;
    if ( (Y & 0x18) != 0 )
    {
        trst = 2;

        /* loop to clear the buffers */
        do { tmp = inportb(SPA+5) & 0x60;  } while (tmp != 0x60);

        /* wait until character back or timeout */
        do
        {
            Y = inportb(SPA+5);
            F = Y & 0x1;
        } while ( !F && (*sysclk <= M) );

        if (F) X = inportb(SPA);
        else return 3;
    }
}

outportb(SPA+3,0x83);   /* set DLAB */
outportb(SPA  ,0x01);   /* bit rate is 115200 */
outportb(SPA+3,0x03);   /* 8 dta, 1 stp, no par */
```

```
        return trst;
}


/*------------------------------------------------------------------
 * This is the 1-Wire routine 'TouchByte', sometimes called 'DataByte'.
 * It transmits 8 bits onto the 1-Wire data line and receives 8 bits
 * concurrently. The global variable 'com_port' must be set to the
 * com port that the serial brick is attached to before calling this
 * routine. This com port must also be set to 115200 baud, 8 dta, 1 stp,
 * and no parity. This routine returns the uchar 8 bit value received.
 * If it times out waiting for a character then 0xFF is returned.
 */
uchar TouchByte(uchar outch)
{
        uchar inch=0,sendbit,Mask=1;
        uint SPA;
        uint far *ptr = (uint far *) 0x00400000;
        ulong far *sysclk = (ulong far *) 0x0040006c;
        ulong M;

        /* get the serial port address */
        SPA = *(ptr+com_port-1);

        /* Initialize the time limit */
        M = *sysclk +2;

        /* wait to TBE and TSRE */
        do {} while ( (inportb(SPA+5) & 0x60) != 0x60 );

        /* flush input */
        while ( (inportb(SPA+5) & 0x1) )
                inportb(SPA);

        /* get first bit ready to go out */
        sendbit = (outch & 0x1) ? 0xFF : 0x00;

        /* loop to send and receive 8 bits */
        do
        {
                outportb(SPA,sendbit);   /* send out the bit */

                /* get next bit ready to go out */
                Mask <<= 1;
                sendbit = (outch & Mask) ? 0xFF : 0x00;

                /* shift input char over ready for next bit */
                inch >>= 1;

                /* loop to look for the incoming bit */
                for (;;)
                {
                        /* return if out of time */
```

```
                 if ( *sysclk > M )
                    return 0xFF;

                 if ( inportb(SPA+5) & 0x01 )
                 {
                     inch |= ((inportb(SPA) & 0x01) ? 0x80 : 0x00);
                     break;
                 }
             }

        } while (Mask);

     return inch;    /* return the input char */
}
```

C LANGUAGE PULSEWIDTH FOR SYSTEMS USING 8253 AND 8250

```
// standard include header file
#include <dos.h>

// function prototype
void PulWidth(unsigned int);

// global variable
int SPA;

//-----------------------------------------------------------------------
//   This procedure creates a fixed pulse width for programming that is
//   approximately independent of system clock speed.  X is in units of
//   0.419 microseconds for values greater than about 1000.
//
void PulWidth(unsigned int X)
{
    unsigned int N,M;

    disable();                                  // turn off interrupts
    outportb(SPA+4,(inportb(SPA+4) & 0xFD));    // apply program pulse to rts
    outportb(0x43,0);                           // freeze value in timer
    M = inportb(0x40);                          // read value in timer
    M |= (inportb(0x40) << 8);
    do
    {
        outport(0x43,0);                        // freese value in timer
        N = inportb(0x40);                      // read value in timer
        N |= (inportb(0x40) << 8);
    }
    while (X > (M - N));

    outportb(SPA+4,(inportb(SPA+4) | 0x02));    // remove program Voltage
    enable();                                   // turn interrupts on
}
```

PASCAL LANGUAGE FOR UART 8250 SYSTEMS

```
{
    In the following pascal code 1-wire I/O is accomplished using the
    serial port of an IBM PC or compatible. The serial port must be capable
    of a 115,200 bps data rate.
}
Const
    SPA : Word = 0;             { Currently active serial port address }
```

Function TouchReset(N: Byte): Boolean;

```
{
    This function transmits the one-wire protocol reset sequence to
    the device connected to COM port number N. This sequence consists
    of a low pulse lasting a mimimum of 480 us followed by a high dead
    time lasting a mimimum of 480 us. The function returns True if a
    presence detect pulse occurs during the dead time, otherwise
    it returns False.
}
Const
    Init : Array[1..4] of Boolean = (True, True, True, True);

Var
    S : Array[1..4] of Word Absolute $40:0;
    T : LongInt Absolute $40:$6C;
    M : LongInt;
    F : Boolean;
    X, Y : Byte;

Begin
    SPA := 0; TouchReset := False;                    { Assume failure }
    If (N > 0) and (N < 5) and (S[N] > 0) then Begin  { Legal port # }
        SPA := S[N];                   { Save active serial port address }
        If Init[N] then Begin   { Serial port requires initialization }
            Port[SPA +3] := $83;                       { Set the DLAB }
            Port[SPA]     :=   1;                       { Bit rate is   }
            Port[SPA +1] :=   0;                    {      115200 bps }
            Port[SPA +3] :=   3;             { 8 dta, 1 stp, no par }
            Port[SPA +1] :=   0;                      { No interrupts }
            Port[SPA +4] :=   3;                    { RTS and DTR on }
            Init[N] := False;             { Initialization completed }
        End;
        M := T +1;                      { Initialize the time limit }
        Repeat until Port[SPA +5] and $60 = $60;  { Await TBE & TSRE }
        While Odd(Port[SPA +5]) do X := Port[SPA];     { Flush input }
        Port[SPA +3] := $83;                            { Set DLAB }
        port[SPA+1] := 0;                       { Baud rate is 10473 }
        Port[SPA] := 11;
        Port[SPA +3] := 3;                  { 8 data, 1 stop, no parity }
        Port[SPA] := $F0;                      { Send the reset pulse }
        Repeat              { Wait until character back or timeout }
            Y := Port[SPA +5];
            F := Odd(Y);
```

```
         until F or (T > M);
         If F then X := Port[SPA] else X := $F0;
         If (X <> $F0) Then Begin          { If more bits back than sent }
            TouchReset := True;            {      then there is a device }
            If ((Y and $18) <> 0) Then Begin  { Framing error or break }
               Repeat until Port[SPA +5] and $60 = $60;  { TBE & TSRE }
               Repeat F := Odd(Port[SPA +5]) until F or (T > M);
               If F then X := Port[SPA];
            End;
         End;
         Port[SPA +3] := $83;                        { Set the DLAB }
         Port[SPA]    :=   1;              { Bit rate is 115200 bps }
         Port[SPA +3] :=   3;              { 8 dta, 1 stp, no par }
      End;
End;
```

Function TouchByte(X: Byte): Byte;
```
{
    This function transmits the byte X to the device attached to the
    currently active serial port SPA, and returns a byte from the
    device as its value.
}
Var
    T    : LongInt Absolute $40:$6C;
    M    : LongInt;
    I, J : Byte;
Begin
    If SPA = 0 then TouchByte := X else Begin
        M := T +1;                         { Initialize the time limit }
        Repeat until Port[SPA +5] and $60 = $60;  { Await TBE & TSRE }
        While Odd(Port[SPA +5]) do I := Port[SPA];    { Flush input }
        I := 0; J := 0;       { Initialize output & input bit counters }
        Repeat
            If Odd(Port[SPA +5]) then Begin
                Inc(J); If Odd(Port[SPA]) then X := X or $80;
            End else If (I<=J) and (Port[SPA+5] and $20 = $20) then Begin
                If Odd(X) then Port[SPA] := $FF else Port[SPA] := 0;
                X := X shr 1; Inc(I);
            End;
        Until (J = 8) or (T > M);
        While (J < 8) do Begin
            X := X shr 1 or $80;
            Inc(J)
        End;
        TouchByte := X;
    End;
End;
```

PASCAL LANGUAGE PULSEWIDTH FOR SYSTEMS USING 8253 AND 8250

```
Procedure PulWidth(X : Word);
{
    This procedure creates a fixed pulse width for programming
    that is approximately independent of system clock speed.  When
    used in an IBM PC or compatible computer operating under MS-
    DOS, X is in units of 0.419 microseconds for values of X
    greater than about 1000.  The procedure can be used with any
    processor having an 8250 UART I/O mapped to base port address
    SPA and an 8253 timer mapped to base port address $40,
    operating with an input clock of 2.386363 MHz and with its
    count limit set to the maximum value.
}
Var
    M, N : Word;
Begin
    Inline($FA);                              {Turn off interrupts}
    Port[SPA +4] := Port[SPA +4] and $FD;     {Apply Program Pulse on RTS}
    Port[$43] := 0;                           {Freeze value in timer}
    M := Port[$40] shl 8 or Port[$40];        {Read value in timer}
    Repeat                                    {Loop to consume real time}
        Port[$43] := 0;                       {Freeze value in timer again}
        N := Port[$40] shl 8 or Port[$40];    {Read new value in timer}
    Until M - N >= X;                         {See if "X" usec have elapsed}
    Port[SPA +4] := Port[SPA +4] or 2;        {Remove Program Voltage}
    Inline($FB);                              {Turn interrupts on}
End;
```

VIII. SUMMARY

This application note explains the hardware of different types of 1-Wire interfaces and software examples adapted to this hardware. Depending on the types of Touch Memories required for a project and the type of computer to be used, the most economic interface is easily found. The hardware examples shown are basically two different types: 5V general interface and 12V RS232 interface. Within the 5V group a common printed circuit board could be used for all four circuits. The variations can be achieved by different population of components (Table 4). The same principle is used for the 12V RS232 interface. The population determines if it is a Read all or a Read/Write all type of interface (Table 5).

There are other possible circuit implementations to create a 1-Wire interface. The circuits described in this application note cover many different configurations. For a custom application, one of the described options can be adapted to meet individual needs. The circuits can be used for reading and writing SRAM based Touch Memories, for individually programming EPROM based Touch Memories and for temperature measurement with the DS1920 Touch Thermometer. For programming large quantities of EPROM based Touch Memories with the same data (gang programming) commercial programmers are available from several independent companies. A list of vendors is available from Dallas Semiconductor on request.

PARTS LIST FOR 5V SERIAL TO 1–WIRE PORT ADAPTERS (FOUR OPTIONS) Table 4

POSITION	TTL READ ALL	5V RS232 RD ALL	TTL RW ALL	5V RS232 RW ALL
C1	empty	empty	10µ tantalum	10µ tantalum
C2	empty	empty	150p, ceramic	150p, ceramic
D1	1N5242 (12V)	1N5242 (12V)	1N5242 (12V)	1N5242 (12V)
D2	1N5232 (5.6V)	1N5232 (5.6V)	1N5242 (12V)	1N5242 (12V)
D3	empty	1N5232 (5.6V)	empty	1N5242 (12V) optional
D4	empty	empty	1N5242 (12V)	1N5242 (12V)
D5	empty	empty	ERA–82–004	ERA–82–004
D6	empty	empty	1N5818	1N5818
D7	short	short	empty	ERA–82–004 optional
IC1	empty	empty	LT1109CN8–12	LT1109CN8–12
Q1	2N7000	2N7000	2N7000	2N7000
Q2	short GD	2N7000	short GD	2N7000
Q3	empty	empty	2N7000	2N7000
Q4	empty	empty	2N7000	2N7000
Q5	empty	empty	BSS110	BSS110
R1	4.7 kΩ	4.7 kΩ	4.7 kΩ	4.7 kΩ
R2	empty	4.7 kΩ	empty	4.7 kΩ
R3	empty	empty	100 kΩ	100 kΩ
R4	empty	empty	100 kΩ	100 kΩ
R5	empty	empty	47 kΩ	47 kΩ
L1	empty	empty	see text	see text

PARTS LIST FOR 12V COM PORT TO 1–WIRE ADAPTERS (TWO OPTIONS) Table 5

POSITION	READ ALL	RW ALL
D1	empty	1N5228 (3.9V)
D2	empty	1N5234 (6.2V)
D3	ERA–82–004	ERA–82–004
D4	ERA–82–004	ERA–82–004
D5	1N5228 (3.9V)	1N5235 (6.8V)
D6	1N5234 (6.2V)	1N5242 (12V)
D7	empty	ERA–82–004
Q1	empty	2N7000
Q2	empty	2N7000
Q3	empty	2N7000
Q4	empty	BSS110
R1	1.5 kΩ	1.5 kΩ
R2	empty	1000 kΩ
R3	empty	empty
R4	empty	39 Ω

DALLAS
SEMICONDUCTOR

I. INTRODUCTION

The fare payment system used by the Bay Area Rapid Transit (BART) system in San Francisco is an example of an application in which monetary equivalent data is read and written electronically. In this system, the user can obtain a transit ticket and deposit any desired amount of money into it from an automatic vending machine. The information is stored in the ticket magnetically in the form of encoded data written on a magnetic stripe. Each time the user travels from one place to another, the system deducts the fare from the amount represented by the magnetically encoded data, thus reducing the value of the ticket. When the value of the ticket is nearly exhausted, it can be restored to a high value by inserting it again into an automatic vending machine and depositing additional funds.

The BART system eliminates the need for handling money and making change at the point of entry to the transit system, thereby reducing labor costs and increasing efficiency. A similar advantage can be realized in many other circumstances where an electronically readable and alterable "token" can eliminate the costs and delays associated with money handling at the point of use. Such a token might therefore be used as a meal ticket on a college campus, as a ride ticket at an amusement park, or wherever tickets or tokens are now used to speed monetary payments and/or eliminate unnecessary labor.

The system described above suffers from three significant disadvantages:

A. Paper tickets with magnetic stripes deposited on them are subject to wrinkling or tearing which can cause loss of the monetary equivalent data. Also, the magnetic stripes are subject to erasure by environmental magnetic fields, even if the paper carrier and magnetic material are physically intact.

B. Since magnetic recording is a read/write technology, it is possible for a technologically sophisticated person to read the contents of the magnetic stripe when the ticket has a large monetary value, use the ticket until the value is nearly gone, then rewrite the original data into the ticket to restore its original value. It is not necessary for the person to understand the encoding of the monetary data in order to do this. Therefore, the use of a read/write technology makes the tickets vulnerable to counterfeiting.

C. The magnetic recording technology requires uniform motion of the magnetic material across the read/write heads in order to read and write data reliably. This makes it necessary to use a relatively complex mechanical ticket–handling mechanism to read, debit, and rewrite the monetary equivalent data.

II. ADD–ONLY TOUCH MEMORY AS AN ALTERNATIVE TECHNOLOGY FOR MONETARY TOKENS

Add–Only Touch Memory provides a viable alternative technology for storage of monetary equivalent data which delivers the advantages described above but does not suffer from the disadvantages. Touch Memory is sealed in a durable stainless steel microcan which protects it against environmental damage. Reading and writing data is accomplished with a momentary contact to a simple electrical probe, requiring no sophisticated mechanical handling mechanisms. The add–only attribute of Add–Only Touch Memory provides protection against counterfeiting, since the data in these memories can never be restored to its original value once it has been modified.

Add–Only Touch Memory contains many bits of information, with each bit having either a one or a zero value. Initially, all the bits in the memory are ones. The

read/write probe can read these bits, and it can also selectively change one or more of the bits to zero. Once a bit has been changed to a zero, it cannot be changed back to a one. Writing a bit is therefore much like punching a hole in a meal ticket card. The electrically alterable bits are organized into memory pages having 256 bits each. In addition to these electrically alterable bits, each Touch Memory also contains a unique 64 bit registration number which cannot be altered. No two Touch Memories ever have the same registration number. Finally, each page has a status register that can be read to determine which pages have been used up, and error detection circuitry (CRC) which allows the reader to determine if it has read the data correctly.

With this feature set, it is possible to design a system in which monetary equivalents can be added to or removed from the part many times before it must be replaced, and which is highly resistant to counterfeiting. The basic principle of this system is described below.

III. ELECTRONIC CREDITING AND DEBITING OF ADD–ONLY TOUCH MEMORY

One possible technique which allows storage of credits and debits in Add–Only Touch Memory is as follows. Monetary units are added by changing one bits to zero bits starting from the least significant bit of each page and progressing toward the most significant bit. Monetary units are debited by changing one bits to zero bits starting from the most significant bit of each page and progressing toward the least significant bit. As the memory is repeatedly debited and credited, the rows of zero bits grow toward the middle of the page. When they meet, the page is marked as exhausted with the status byte and the process continues on the next page. (It is possible to ignore pages and treat the entire memory as a single page, but that would require the reading of the entire memory, increasing the time needed to complete a transaction. The electronic read/write process is more efficient when only a portion of the stored data needs to be read). With this technique, assuming the credit units and debit units have equal value, a 1024 bit memory could credit and debit 512 monetary units before it was used up. If credit units are taken to represent some multiple of the debit unit, then more debits are allowed. (For example, if each credit unit is the equivalent of three debit units, then a 1024 bit memory would allow 768 debits).

The problem with the simple system described above is that anyone with the necessary knowledge and equipment to read and write data in Add–Only Touch Memory can easily increase the value by adding additional credit units. This is possible because there is a direct, straightforward correspondence between a bit location and its value. If the bits were scrambled (permuted) in an apparently random manner, it would no longer be possible to determine how to add credit units to the memory. For example, if 15 bits on a page are still set to one, only one of these bits will add a credit unit to the memory. Similarly, only one of the bits will add a debit unit to the memory. If any one of the other 13 bits were written to zero, it would appear out of sequence and would signify that the memory had been tampered with, thereby invalidating it. Therefore, if a person guesses which bit to write next, he has one chance in 15 of adding a credit unit, one chance in 15 of adding a debit unit (decreasing the value), and 13 chances in 15 of invalidating the memory and flagging it as having been subject to tampering. Although there is a chance of guessing correctly which bit to change, the laws of probability are stacked against this event. (This is the kind of statistical analysis that makes lotteries predictable).

The unique registration number in each Touch Memory can be used to permute the bits in each part differently, so that one cannot determine by studying the data in one part how to add credit units to a different part. Many different techniques are possible to determine a unique bit permutation from the unique registration number supplied with each part. A few of these techniques are described below.

IV. CALCULATING BIT PERMUTATIONS FROM UNIQUE REGISTRATION NUMBERS

The number of different permutations of the 256 bits in each page is very large, approximately ten to the power of 507. Only a minute fraction of these permutations can be enumerated with the unique registration number, since the registration number represents a range of 281 trillion unique numerical values, or about ten to the power of 14. The permutations that can be derived from the unique registration number are thus buried in the much larger population of possible permutations. 281 trillion is in fact an extremely large set of unique registration numbers that is sufficient for all practical purposes. The enormously larger number of different permuta-

tions greatly multiplies the task of deducing the permutation from the registration number. To select a permutation based on the registration number from this enormous population, the following method could be used.

A. Replace the CRC in the publicly readable registration number with the page number of the page to be scrambled and then encrypt it with a standard block cypher encryption algorithm (such as DES), using a secret encryption key. This produces a 64 bit encrypted number which is unique to each page of each part and is known only to the reader.

B. Divide the 64 bit encrypted number by 256 to obtain a quotient and a remainder that lies in the range 0–255. The value of the remainder gives the position of bit 1 in the scrambled data, and leaves 255 other bit positions unfilled.

C. Divide the quotient from step B by 255 to obtain another quotient and a remainder that lies in the range 0–254. The value of this remainder gives the position of bit 2 in the remaining 255 bits that were unfilled after step B.

D. Repeat step C for each successive bit, decreasing the divisor by 1 each time, until all 64 bits have been placed in their scrambled positions. Each time the quotient reaches zero during this process, replace it with the original encrypted number from step A.

The steps B – D above are numerically intensive and may not be suitable for microcontroller–based equipment. A simpler but less secure technique is to start with an initial, secret, randomly chosen permutation and then further permute it based on the 64 bit encrypted number by interchanging certain bits or not depending on whether a bit in the encrypted number is a one or a zero. This method provides a simpler set of permutations but may still provide adequate security in many applications. The complexity of the technique used to derive permutations from the unique registration number can be selected based on the degree of security needed in the application and the amount of computing power available in the equipment.

V. DESCRIPTION OF OPERATION

Using the methods described above, the automatic debiting equipment operates as follows to decrease the value of the memory by one monetary unit:

A. The equipment detects an Add–Only Touch Memory by means of the presence pulse that it generates, reads the unique registration number, and checks its validity with the CRC.

B. The equipment reads the status registers to find the first page that has not been used up. It then reads that page, making use of the built–in CRC calculation circuitry to confirm the validity of the read.

C. Using a secret encryption key that can be changed periodically, the equipment applies a standard encryption algorithm (such as DES) to the unique registration number (with the CRC replaced with the active page number) to generate a unique secret number, and then uses this number to reorder the bits read from the active page using any of the techniques described in section IV above.

D. After the above reordering, the zero bits starting from the least significant bit represent credits, and the zero bits starting from the most significant bit represent debits. The data, beginning with the least significant bit, should therefore appear as an unbroken sequence of zero bits (credits), followed by an unbroken sequence of one bits (not yet used), followed by an unbroken sequence of zero or more zero bits (debits). The equipment checks the integrity of these three sequences. If there is a break in any of these sequences or if the number of debits exceeds the number of credits, then there is evidence of tampering and the equipment may take appropriate action (such as recording the registration number, or even sounding an alarm or summoning an official).

E. If the number of credits is greater than the number of debits, the equipment adds one more zero bit to the unscrambled sequence, checks to make sure that the page has not been used up, and then uses the bit permutation in reverse to determine where the debit bit falls in the original scrambled bit sequence. Any time a page is filled, the equipment writes the status bytes to mark the page as used up and proceeds to the next page.

F. The equipment performs a write operation to write the bit identified in step E above from a one to a zero, then reads back the page to make sure that the write operation was completed correctly. When a successful write of the debit bit is detected, the equipment activates a peripheral device (passenger gate, etc.) to signal a completed, successful operation.

The operation of the crediting equipment is similar to that described above. The crediting equipment receives cash from the user and sets one or more credit bits to zero to indicate the amount of added value. When a page is half full of credit bits, the equipment proceeds to the next page to add additional credits. The bits are written in the scrambled order so that it is impossible to distinguish the credit bits from the debit bits and the bits that have not yet been used.

Both the debiting and crediting equipment can make use of a secure microprocessor (such as the DS5002 secure micro) so that even if the equipment is stolen, it cannot be made to reveal the secret encryption key which is used in step C above. This makes it possible to limit the knowledge of this information to a very small number of individuals. It is important to note that a blank Add–Only Touch Memory has no monetary value until it his been credited with monetary equivalents using its unique bit scrambling algorithm. Therefore, there is no advantage to a counterfeiter to obtain a supply of blank Touch Memories, and it is unnecessary to take special precautions to safeguard these supplies.

Assuming that a high–performance processor is used so that the time required to perform the calculations described above can be neglected, the minimum time required for a debiting transaction is the time required to read the unique registration number, read the status

bytes, read the appropriate page, and write out the bit that represents the debit. This time, equal to 31.7 milliseconds, is scarcely perceptible and would be regarded as essentially instantaneous by the user.

VI. SUMMARY

Add–Only Touch Memory has the following special characteristics which make it uniquely suitable for applications requiring secure crediting, debiting, and portable storage of monetary equivalent data:

A. A unique, unalterable registration number which allows the data on each different part to be encrypted differently. This makes it impossible to determine how to counterfeit a part by studying how data is written into a different part.

B. Random–access memory which is one–way alterable, that is, having bits that can be changed from a one to a zero but not from a zero back to a one. This makes it impossible to write into a part the data pattern it held earlier when it was more valuable. (This type of memory is commonly referred to as one–time–programmable EPROM, but this terminology is misleading in the current application because it suggests that the part can be written only once).

C. A small, durable Touch Memory package with a simple electrical connection, allowing data to be read or written with a momentary contact.

DALLAS
SEMICONDUCTOR

<div align="right">

Application Note 99
UniqueWare™ Project Setup Manual
Revision 2.00B

</div>

SPECIAL FEATURES

- Registered Silicon chips providing globally unique addresses

- Meets the identification requirements of Ethernet, Token Ring, cellular phones, etc.

- EPROM technology for custom programming and quick delivery

- No NRE charge

- Two device options: 512–bit (two pages) and 1024–bit (four pages)

- Dallas Semiconductor handles serialization and does the bookkeeping of tracking the last number used

- Pages are programmed and locked under strict manufacturing control

- Two data structures: one unnamed file or up to three named files (TMEX™ format)

- Unused memory pages can be programmed in the application

- The unique registration number authenticates the chip and tracks the transfer of ownership

- Customer specific Project ID number together with Custom ROM for highest level of product security

- Programmed devices are made available to the owner of the Project ID or their authorized agents only

- Built–in multidrop controller ensures compatibility with other MicroLAN™ products

- Reduces control, address, data signals and power to a single pin

- Directly connects to a single port pin of a microprocessor and communicates at up to 16.3k bits per second

- Low–cost TO–92 or 8–pin SOIC and 6–pin C–Lead surface mount packages

- Tape & Reel option available

- Reads over a wide voltage range of 2.8V to 6.0V from −40°C to +85°C

IEEE # = 0F 74 A8 36 EE D9
SERIAL # = 12 5E 70 00 00 54 CF 91

DALLAS
DS2501.2
UNW
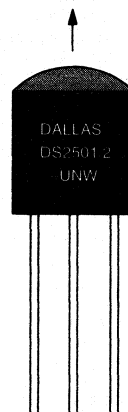
APPLICATIONS

- After–market protection of products by restricted availability

- Node–ID for network cards (IEEE–assigned number)

- Wireless Phone ID plus battery information

- Electronic product identification label with serialization

I. INTRODUCTION

UniqueWare™ is a convenient and affordable way to electronically label items with a unique serialization and to optionally add some text information around the serialization field.

There are several options that allow you to customize what is programmed into UniqueWare™ devices. This manual will help you when filling in the screens and explain the effect of your entries to take full advantage of all the options. The data you enter will be used directly as a master to program the UniqueWare™ devices. Therefore it is important that you understand how data is used and what you can expect to find programmed in your UniqueWare™ devices.

II. DEVICE DATA STRUCTURE

A. Structure Of ROM Section

As with other Dallas Semiconductor Automatic Identification Products, UniqueWare™ devices have a ROM Registration Number (Table 1) that serves several purposes:

1. it identifies the logical behavior of the device by its family code

2. it stores the device's unique serial number

3. it validates that a device is genuine UniqueWare™. Only UniqueWare™ Devices have the special identifier code 5E7. Other devices have a different code stored in this location.

4. it provides an 8-bit Cyclic Redundancy Check to verify the integrity of the complete ROM section

5. it may act as node address to singulate and communicate with one device if several Automatic Identification devices are connected in parallel as a 1-Wire™ network.

UNIQUEWARE™ DEVICES ROM SECTION Table 1

512-bit UniqueWare™ Device DS2501-UNW

64-BIT ROM REGISTRATION NUMBER				
8-BIT CRC	**UNIQUEWARE™ IDENTIFIER**			**DEVICE SERIAL NUMBER**	**FAMILY CODE**
	5	E	7	MSB (hex format) LSB	91
1 byte	12 bits			36 bits	1 byte

high address   low address

1024-bit UniqueWare™ Device DS2502-UNW

64-BIT ROM REGISTRATION NUMBER				
8-BIT CRC	**UNIQUEWARE™ IDENTIFIER**			**DEVICE SERIAL NUMBER**	**FAMILY CODE**
	5	E	7	MSB (hex format) LSB	89
1 byte	12 bits			36 bits	1 byte

high address   low address

In addition to the ROM, UniqueWare™ devices have 64 bytes (512-bit device) or 128 bytes (1024-bit device) of one-time programmable data memory organized as pages of 32 bytes each. These pages are factory programmed with a unique Project ID Number, a customer-specified serialization and additional constant data patterns. The UniqueWare™ identifier in the ROM section together with a unique Project ID number guarantees the authenticity of original UniqueWare™ parts that are sold only to the owner of the Project ID or other authorized parties. Once programmed, the protected pages of UniqueWare™ devices cannot be altered.

B. Structure Of Data Memory

This section explains how data within the data memory of UniqueWare™ Devices is organized. There are basically two alternatives (Table 2):

1. combine all pages into one unnamed file (Default Data Structure) containing UniqueWare™ data; advantage: up to 57 (121) bytes of user data* available

2. use one page as a directory and store the UniqueWare™ data as a named file (TMEX™ format); advantages of TMEX™ format:

   a. up to 24 (80) bytes of user data* available

   b. give the UniqueWare™ data a file name

   c. use TMEX™ high level functions to read the UniqueWare™ file

   d. store up to two additional files together with the UniqueWare™ file in the same device (1024-bit device only).

*User data includes the customer specified serialization field.

Regardless of which data structure is chosen, the format of the UniqueWare™ file is very similar. With Default Data Structure it always starts at physical address 0000H, with TMEX™ format it starts at address 0020H. Table 3 shows the general structure of the UniqueWare™ file in case of Default Data Structure. With TMEX™ format the UniqueWare™ file may occupy up to 3 pages (1024–bit device). Each page begins with a length byte and ends with a control byte and a CRC16, leaving up to 28 bytes per page for data.

As with Default Data Structure the Project ID is stored in the beginning of the UniqueWare™ file. Pages not occupied by the UniqueWare™ file or device directory are available for programming by the user.

In the case of TMEX™ format, there will be a device directory starting at address 0000H. Details are shown in Table 4. According to TMEX™ rules, the most significant bit of the 1–byte file extension will be set to 1 to indicate that the UniqueWare™ file is write protected.

DATA ORGANIZATION OPTIONS Table 2
512–bit UniqueWare™ Device

	DEFAULT DATA STRUCTURE	TMEX™ FORMAT
Page 0	UniqueWare™ File	Device Directory
Page 1	UniqueWare™ File (cont.) or blank	UniqueWare™ File

1024–bit UniqueWare™ Device

	DEFAULT DATA STRUCTURE	TMEX™ FORMAT			
	Only Case	Case A	Case B	Case C	Case D
Page 0	UniqueWare™ File	Device Directory			
Page 1	UniqueWare™ File (cont.) or blank	UniqueWare™ File			
Page 2	UniqueWare™ File (cont.) or blank	UniqueWare™ File (cont.) or blank	UniqueWare™ File (cont.)	second file	second file
Page 3	UniqueWare™ File (cont.) or blank	UniqueWare™ File (cont.) or blank	second file or blank	second file (cont.) or blank	third file

UNIQUEWARE™ APPLICATION DATA Table 3
Default Data Structure

LENGTH	PROJECT ID LSB MSB	TEXT 1 const. data	SERIALIZATION NUMBER LSB (hex. or BCD) MSB MSB (character format) LSB	TEXT 2 const. data	CRC 16 LSB MSB
1 byte	4 bytes assigned by DS		up to 121 bytes (no file name) extending over adjacent pages		2 bytes

low address   high address

UNIQUEWARE™ DEVICE DIRECTORY (TMEX™ FORMAT) Table 4

LENGTH	DEVICE CONTROL FIELD							FILE ENTRY 1			
	AA	00	00	00	00	00	01	Name	Ext.	Address	Length
1 byte	7 bytes							5 bytes		2 bytes	

low address   medium address

	FILE ENTRY 2				FILE ENTRY 3				CONTROL	CRC16	
continued	N.	E.	A.	L.	N.	E.	A.	L.	00	LSB	MSB
	7 bytes				7 bytes				1 byte	2 bytes	

medium address   high address

File Entries 2 and 3 are only possible with the 1024-bit device.

III. COMPUTER–AIDED DATA SETUP

A. Overview

Several pieces of information are needed to customize UniqueWare™ devices and to get them programmed for your application. To simplify the process and to avoid misunderstanding and errors introduced by retyping data from paper forms, Dallas Semiconductor has developed a program that prompts for all relevant information, does error checking and writes the result to a machine readable data carrier. Using this program to specify your special UniqueWare™ device is called a Project Setup Session.

A session to create a set of UniqueWare™ data consists of seven steps, each having its own screen:

Screen 1   Address Section, (your office address, for book–keeping purposes)

Screen 2   Device specification: 512–bit or 1024–bit device

Screen 3   UniqueWare File, Global Specification (data structure, file name)

Screen 4   UniqueWare File, Serialization Field (length, counting style, code)

Screen 5   UniqueWare File, Text Section (optional text around the Serialization Field)

Screen 6a   Second file specification (if space available)

Screen 6b   Third file specification (if space available)

Screen 7   Saving Data, (write to a Touch Memory or disk file)

You may jump forward or backward between screens by using the "Page Up" and "Page Down" buttons of your keyboard. You may reach filled–in fields of a screen by using the cursor keys "up" or "down". To replace data, use the backspace key or cursor "left" to empty the field and then type the new data. A session can be ended at any time. If the screen "Saving Data" is reached, the entered data can be saved for later review, approval and ordering.

B. Getting Started

To specify the data of your UniqueWare™ devices, take the UniqueWare™ Setup Disk, insert it in the appropriate disk drive of your computer and access it. You may do this from WINDOWS using the File Manager or from DOS using the <drive:> and DIR command. In either case you will see the file listing of this disk. The program to execute is called UNIB.BAT.

UNIB.BAT allows up to two parameters to be specified when starting. The first parameter is a 1–digit number specifying the COM–port that will be used to program or read a Touch Memory with your data. If you don't have a COM–Port Adapter with probe for Touch Memories, this parameter becomes a don't care. You may specify any value from 1 to 4 that your computer accepts. However, you will not be able to program or read a Touch Memory. You will have to store your data on a disk file instead.

The second parameter is alphanumeric. It becomes important if you would like to revise data that you have already specified in an earlier session. If you are using a Touch Memory as data storage, a capital R will tell the

program to read data from a Touch Memory through the COM–port specified in the first parameter. If you are using a disk file to store your data, just specify the full name including the path of that file to load its data for review.

Examples:

UNIB 2

starts the project setup software and assigns COM– Port 2 to write to a Touch Memory

UNIB 2 R

starts the project setup software and assigns COM– Port 2 to first read data from a Touch Memory and to later write back to a Touch Memory

UNIB 2 DATA.DAT

starts the project setup software, assigns COM– Port 2 to write to a Touch Memory and loads data from a file with the name DATA.DAT residing in the same directory as UNIB.BAT for review.

C. Ending the Program

The program is finished after data has been written to a Touch Memory or disk file. It will then prompt for your permission to quit. However, there is an easy way to quit any time. The ESCAPE button of your keyboard can ALWAYS be used to end the program. You may be asked if you really want to quit. Enter a Y to quit, otherwise the program will ignore the ESCAPE and resume work at the current location.

IV. A COMPLETE SESSION

After a start without the second parameter, the empty Address Section screen appears. This screen is used to enter the customer address including phone and FAX numbers. This information is required to label the container with the programmed parts and to distinguish it from orders of other customers. It also provides a convenient means to contact you to double–check requirements in the case of extremely unusual data specifications.

One can move from one line to the next using the cursor keys "up" and "down" only after data has been entered. However, fields that are not always applicable, such as Mail Stop, State, FAX, may be left empty. The area code (may also include the country code) is separated from the phone number.

UNIQUEWARE™ PROJECT SETUP, SCREEN 1

```
                        UniqueWare Project Setup Ver. 2.00


        Address Section

        Company Name      :
        Contact           :
        Street Address    :
        Mail Stop (optional):
        City              :
        State (optional)  :
        ZIP               :
        Country           :
        Phone           Area Code  :
                        Number     :
        FAX (optional)  Area Code  :          (numeric input including ()/– only)
                        Number     :
```

After this screen is filled in, one reaches the device specification screen, either after the FAX number has been entered with <RETURN> or with "Page Down". This screen is required to select one of the two possible UniqueWare™ devices, DS2501 (512 bits) or DS2502 (1024 bits). Depending on which device is selected the global specification screen looks slightly different. In addition, if the 1024–bit device is chosen and the Uni- queWare™ file itself is very short, the project–setup program may have one or two more screens (screens 6 and 6a).

After the device type is chosen, either by accepting the 512–bit default device or by overstriking the device type 1 by a 2 and pressing <RETURN> or "Page Down" one reaches screen 3 or 3a, Global Specification.

UNIQUEWARE™ PROJECT SETUP, SCREEN 2

```
                    UniqueWare Project Setup Ver. 2.00


UniqueWare Device Specification

Select Device Type
         enter 1 for 512-bit device
         enter 2 for 1024-bit device
DEVICE TYPE    :1

Hit <Enter> after entering Device Type to continue.
```

The screen "UniqueWare File, Global Specifications" is required to select between two possible data structures, called TMEX™ Format and Default Data Structure. For details on data structures see Table 2, Data Organization Options. Depending on the selected device type either screen 3 (512–bit) or screen 3a (1024–bit) will appear. The major difference between these screens is that screen 3a allows preselection of three different sizes for the UniqueWare™ file if TMEX™ Format is selected.

With TMEX™ Format (file size option 1 on screen 3, options 1 through 3 on screen 3a), one may either accept the default name IDNR.0 for the UniqueWare™

file (UniqueWare file selection # 1) or specify a different name and extension (UniqueWare file selection # 2). The device directory should be hardware write protected (default).

With Default Data Structure (file size option 2 on screen 3, option 4 on screen 3a) a file name cannot be specified.

After screen 3 or 3a has been reviewed or updated one reaches the next screen with "Page Down" or, if the cursor was positioned in the last applicable field, with <RETURN>.

UNIQUEWARE™ PROJECT SETUP, SCREEN 3 (512–bit device)

```
                    UniqueWare Project Setup Ver. 2.00


UniqueWare File, Global Specifications

Total File Size (including Serialization Field)
         enter 1 for up to 24 bytes (with file name)
         enter 2 for up to 57 bytes (no file name)
FILE SIZE      :1

Name of UniqueWare File
         enter 1 for default name IDNR.0
         enter 2 to specify desired file name
UNIQUEWARE FILE SELECTION      :1                        only applicable with
                                                         File Size Option 1
         File Name             :<IDNR>
         File Extension (0-126)  :<0>

WRITE PROTECT DIRECTORY (Y/N)    :Y
```

UNIQUEWARE™ PROJECT SETUP, SCREEN 3A (1024–bit device)

```
                        UniqueWare Project Setup Ver. 2.00


        UniqueWare File, Global Specifications

        Total File Size (including Serialization Field)
                enter 1 for up to 24 bytes
                enter 2 for 25 to 52 bytes
                enter 3 for 53 to 80 bytes
                enter 4 for up to 121 bytes (no file name)
        FILE SIZE      :1

        Name of UniqueWare File
                enter 1 for default name IDNR.0
                enter 2 to specify desired file name
        UNIQUEWARE FILE SELECTION     :1                    not applicable with
                                                            File Size Option 4

                File Name             :<IDNR>
                File Extension (0-126) :<0>

        WRITE PROTECT DIRECTORY (Y/N)     :Y
```

The screen "UniqueWare File, Serialization Field" is the central portion of the whole project setup system. Here the specification of the serialization is done. You may specify how much space (number of bytes) is to be reserved for serialization, if incrementing is to occur on a hexadecimal (as done internally in computers) or a deci-mal (familiar to humans) basis, if the serialization is to be formatted for printing (direct ASCII character representation) or for efficient internal computation (binary number representation) and the starting value for the serialization. The screen example below shows the default settings for the fields.

UNIQUEWARE™ PROJECT SETUP, SCREEN 4

```
                        UniqueWare Project Setup Ver. 2.00


        UniqueWare File, Serialization Field

        Number of bytes for serialization (2-24)     :4
                                                            limits the maximum number of devices that
        Counting Style                                      can be programmed with the same project ID
                enter 1 for hexadecimal
                enter 2 for decimal
        COUNTING STYLE  :1                              preferred for machine-readable serialization
                                                        e.g., node addresses
        Number Representation
                enter 1 for binary (includes 4-bit BCD or hexadecimal)
                enter 2 for character (8-bit ASCII code, with decimal counting only)

        NUMBER REPRESENTATION :1            best for easily printed, human readable serialization
                                            such as product identification labels

        Serialization starts with number:
        $ ———————  if a $ appears, the starting number has to be entered in hexadecimal form
```

The value you specify for "Number of bytes for serialization" directly determines the highest number and with it the maximum possible number range for serialization. If you have decided on the counting style (either hexade-cimal or decimal) and the format of number representation (unsigned integer true binary/binary coded decimal BCD or character ASCII) then you can calculate the number of bytes required for the application.

As an example, if you need up to but no more than a million different serial numbers, the maximum number to be represented is 999 999.

- Counting in **decimal style** and using **character representation, each** of these digits takes one byte of storage. The Number of bytes for serialization would be 6 in this case.

- If you opt for **BCD number representation** (which also implies decimal counting) then two digits fit into one byte. You would only need 3 bytes for serialization for the same number range.

- If you use **hexadecimal counting** style (which implies binary number representation) then you can distinguish 256 different numbers with one byte. You would therefore also need 3 bytes for serialization using hexadecimal numbers.

Table 5 shows examples for the maximum serialization number depending on the number of bytes reserved and the code to be used. The maximum supported size for the serialization field is 24 bytes. When using character representation for serialization the serialization field must be at least 4 bytes.

EXAMPLES FOR THE MAXIMUM SERIAL NUMBER Table 5

Number of Bytes	2	3	4	5	6
hexadecimal	$65.535*10^3$	$16.7772*10^6$	$4.29496*10^9$	$1.09951*10^{12}$	$281.474*10^{12}$
decimal, BCD	9 999	999 999	99 999 999	9 999 999 999	999 999 999 999
decimal, character	(99) N/A	(999) N/A	9 999	99 999	999 999

As the last item on this screen, the starting number for serialization may be specified. If the counting style is hexadecimal, the starting number also needs to be specified in hexadecimal (0,1,2, ... 9, A, B, C, D, E, F). This situation is indicated by a $ sign at the beginning of the entry field. If the $ is not shown, data is understood as a decimal number. If not otherwise specified, the first value of the serialization number is 0. With character representation, unused bytes ("leading zeros") are programmed as spaces (code 20H). In the case of subsequent UniqueWare[TM] orders, note that it is not possible to choose a starting value that would create overlap in serialization with a previous order.

Usually the application determines the counting style and the type of number representation. Hexadecimal counting is the standard choice to realize node addresses for networks with UniqueWare[TM]. The decimal counting with character number representation allows one to easily generate printable as well as machine readable electronic labels similar to the standard manufacturers product identification labels.

After the serialization screen has been completed one reaches the next screen with "Page Down" or, if the cursor was positioned in the last applicable field, with <RETURN>. The screen "UniqueWare File, Text Section" allows you to specify up to two text sections surrounding the serialization field. The screen example below shows the default setting for the fields. Depending on your requirements you may select one option of "text section". Any option other than 0 will prompt for the way you want to enter the text.

The most common way (default) is entering the text directly from the keyboard as if you were running a text editor. To select this way, press RETURN. The software will then accept any characters that your keyboard can generate. The character code corresponds to the ASCII code page selected for your computer. (Verification of your text entry on the byte level is explained in section VI. of this document.) While entering text you may use the RETURN key to split the text into several lines. Numbers entered as text are stored with their most significant digit at the lower physical address. (Numbers such as the project ID or a binary serialization are commonly stored in the reverse direction, i. e. least significant byte at the lower address).

UNIQUEWARE™ PROJECT SETUP, SCREEN 5

```
                        UniqueWare Project Setup Ver. 2.00


        UniqueWare File, Text Section

        The UniqueWare file may contain up to two text sections.
                enter 0 to omit additional text.
                enter 1 to define one text preceding the serialization field.
                enter 2 to define one text following the serialization field.
                enter 3 to define both text sections.
        TEXT SECTION  :0

        Text consists of characters only? (Y/N) :N/A
                                                 ╲  if you enter N you may enter any data in hexadecimal format
        Text 1  :<none>
                     ╲_____ precedes serialization field

        Text 2  :<none>
                     ╲_____ follows serialization field
                     ╲_____ required since RETURN is treated as valid character, displayed as  ◀
        Hit <PageDown> when finished entering text.
```

If you need to store information in a code other than ASCII, enter N followed by RETURN. Now you may enter any information representing text, data or control in hexadecimal form. Enter a space–character ("blank") as a separator between individual bytes. With every byte entered the physical address location within the UniqueWare™ device will increment.

If you specify two text sections, be aware that both texts are to be entered in the same way (either character or hexadecimal, not mixed). The program will count the characters or bytes you enter and prevent further input if the limit has been reached. Independent of the way you enter text, you may jump between either text section by using the cursor keys "up" and "down". To make corrections, use the backspace key to reach the character or byte to be replaced and re–enter the trailing data. The insert/overwrite function from the DOS command–line editor is not supported. When ready press the PAGE DOWN key to finish the text entry.

With the 512–bit UniqueWare™ device data entry is now finished and Screen 7 will appear. The same is true for the 1024–bit device if Default Data Structure was selected on screen 3a. With TMEX™ Format, however, if File Size Option 1 or 2 was selected on Screen 3a, the program will next display Screen 6. The program will not release unused pages if the actual length of the Unique-Ware™ file is shorter than preselected. All possible combinations of files are shown in Table 2. If, for example, two pages are still available the program assigns them both to the second file; a third file will be accepted only after the data of the second file has been entered and fits on one page. If the UniqueWare™ file occupies two pages the size of the second file is limited to one page. After all memory is used up or if one does not specify additional files, Screen 7 will appear.

On Screen 6, Additional File Specification, one basically specifies the name of the additional file and enters data. This screen assumes that one intends to store another file. Answering the question "Store Second File ?" with N will transfer to Screen 7. The specification of the file name is done in the same way as on screen 3 or 3a. However, there is no default name for the file. Possible file names consist of up to 4 printable characters. After the file name is entered one reaches the field for the file extension by using the cursor down key or by <RETURN>. The default file extension 0 can be over-typed by any decimal number in the range of 1 to 126 or accepted by again using the cursor or <RETURN>. The program will then check if this file name plus extension is already used for another file in the device. If the file name plus extension is duplicated it will display an error message and prompt for a different file extension. After a valid file name is entered one reaches the section of text entry. This section follows the same scheme as with Screen 5 option 1 to 3. Here also <RETURN> is a valid character displayed by a special symbol. To finish text entry one has to use the cursor down key to reach the write protect field. If the file is marked as write protected the file extension stored in the device will be modified to

a value of 128 higher than the original value. The file will also be hardware write–protected, even if you did not enter any text data.

Pressing <RETURN> or "page down" finishes the screen and one either reaches Screen 6a or Screen 7.

Screen 6a works in the same way as screen 6, but it applies to a third file. After this screen is finished one reaches Screen 7, Saving Data.

UNIQUEWARE™ PROJECT SETUP, SCREEN 6

```
                    UniqueWare Project Setup Ver. 2.00


        Additional Files Specification (File 2)

        There is space for one more data file within the device.
        Do you want to store another file together with the UniqueWare?
        Store Second File? (Y/N)  :Y


                File Name              : ——————  a file name has to be specified
                File Extension (0-126)   :0

        Text consists of characters only? (Y/N) :Y
                                              \  if you enter N you may enter
        Text  :                                  data in hexadecimal format

                                          Use the "cursor down" key when finished entering text.
        Write Protect File (Y/N)  :Y      <RETURN> will be treated as valid character, displayed as  ◀
```

UNIQUEWARE™ PROJECT SETUP, SCREEN 6A

```
                    UniqueWare Project Setup Ver. 2.00


        Additional Files Specification (File 3)

        There is space for one more data file within the device.
        Do you want to store another file together with the UniqueWare?
        Store Third File? (Y/N)   :Y


                File Name              :
                File Extension (0-126)   :0

        Text consists of characters only? (Y/N) :Y

        Text  :                          (same rules as with screen 6)


        Write Protect File (Y/N)  :Y
```

Screen 7, "Saving Data to Touch Memory and/or Disk File", is the last of the UniqueWare™ Project Setup Screens. It handles the dialog on how to store the generated data. The picture below shows how this screen looks initially.

You have two choices: a) storing the result of the session in a Touch Memory and b) saving the result to a disk file. If your computer is equipped for reading/writing Touch Memories, you have specified a valid COM port number at the time of starting this program and you have a DS1993 or DS1994 available then you can program the Touch Memory with all your UniqueWare™ specifications. To do this, just press ENTER and follow the instructions on screen.

After a Touch Memory has been programmed, or if you have selected N before pressing ENTER, you reach the section which specifies the name and path of the disk file to save the data of the session. The default file name is either IDNR.0 or is identical to the name specified on screen "UniqueWare File, Global Specification". The default path is copied from the drive/directory where he Project Setup Software is started from. You may replace the default path and file name with any other specification within the DOS conventions. If you want to finish this session without saving at all, accept the default file information and specify N at the subsequently reached field or press ESCAPE.

If you would like to review the session before saving, simply use the Page Up and Page Down keys to jump from one screen to another. The arrow up/down keys allow you to reach every applicable field. You may erase and retype any data.

UNIQUEWARE™ PROJECT SETUP, SCREEN 7

```
                        UniqueWare Project Setup Ver. 2.00


        Saving Data to Touch Memory and/or Disk File

        Data Entry for UniqueWare Project Setup is finished.

        Copy Data to Touch Memory DS1993?    (Y/N)   :Y


                                                enter N if you are NOT equipped to
                                                read/write a Touch Memory



        Disk File Copy information:
             File Name      :IDNR.0
             Path    :B:              ——— DOS file name and path conventions apply

        Copy Data to Disk? (Y/N)      :Y ——— it is recommended to save the result of a session
```

V. REVIEWING DATA

After the data generated in a session has been saved either to a Touch Memory or a disk file you can load it into the Project Setup program, check it, modify it and save it again some time later. The examples in the section "Getting Started" in the beginning of this manual show how the program knows if it has to read back data and where to get it from. If you load data from a Touch Memory (second command line parameter is R), the program will prompt you to probe the Touch Memory before the Address Section screen appears.

After data has been read from a device or file it is checked for consistency. A modification, for example, from loading the data file to a word processor and saving it even without intentional changes will be identified by non–matching checksums or other inconsistencies. The data will then be useless and has to be re–entered from scratch. A word processor such as NOTEPAD, for example, replaces a 00 byte with a 20H byte, which represents a space rather than a binary number. This modification is detected by the UniqueWare™ Project Setup Program and the alterations will not be accepted.

VI. READING OUTPUT DATA

The format of the first two pages (512–bit device) or four pages (1024–bit device) of the data carrier (disk or Touch Memory) that contains the UniqueWare™ application data are described in detail in the beginning of this document. The only difference between the data submitted and actual programmed devices is that the Project ID field contains zeros rather than your personal Project ID number. Pages 2 and 3 are not accessible with the 512–bit device. They always read "FF" hex in a programmed DS1993 or disk file.

To get the application data into its intended location and to be able to track products during manufacturing, additional information is required. Beginning at page 4 and extending up to page 15, this information is stored as Default Data Structure with several embedded sections. Each of these sections is preceded by an Output ID String, directly followed by the associated data and terminated by the ASCII code for RETURN acting as a delimiter. Table 6 gives details.

The additional information is invisible for TMEX™. However, it can be displayed with the Touch Memory Utility TOUCH.EXE. When stored in a disk file, each 32–byte page of a Touch Memory corresponds to a 32–byte binary record. To examine a file one can use the function "Display File Contents" that the DOSSHELL provides. One should select the "Display HEX" mode to examine on hexadecimal level. Switching to the "Display ASCII" mode clearly shows the Output ID strings, one per line, with its associated data section.

OUTPUT ID STRINGS Table 6

OUTPUT ID STRING: MEANING	SPECIFIED WHERE/ THROUGH	VALID CODES/ MEANING
DV: device & data structure	device specification; global specification/file size	1D/512-bit, Default Data Structure 1E/512-bit, TMEX™ Format 2D/1024-bit, Default Data Structure 2E/1024-bit, TMEX™ Format
P0: page 0 control info	(derived by combining several data inputs from different screens)	first character: P/program and protect first character: N/no programming first character: Y/program, no protection second char.: C/display text as character second char.: H/display text in hex. format second char.: (space)/C, H not applicable
P1: page 1 control info	(same as with P0:)	(same as with P0:)
P2: page 2 control info	(same as with P0:)	(same as with P0:)
P3: page 3 control info	(same as with P0:)	(same as with P0:)
SESTA: Ser. Start Address	(various sources)	16-bit binary number/byte address of the beginning of the serialization field
LEN: Length of serialization	serialization field/number of bytes	8-bit binary number/number or bytes for serialization
CNT: Counting Style	serialization field/counting style	HEX/true binary counting (not BCD) DEC/decimal counting (BCD or character)
COD: code for counting	serialization field/number representation	BIN/binary coding (hex. or BCD) ASC/ASCII character coding (decimal only)
START#: Staring number	serialization field/ser. start number	binary number of LEN bytes/serialization start number
CN: company name	address section/company name	ASCII string up to 40 characters
CT: contact person	address section/contact	ASCII string up to 40 characters

OUTPUT ID STRINGS Table 6 (cont'd)

OUTPUT ID STRING: MEANING	SPECIFIED WHERE/ THROUGH	VALID CODES/ MEANING
AD: mailing address	address section/street address	ASCII string up to 40 characters
MS: mail stop	address section/mail stop	ASCII string up to 10 characters
CI: city name	address section/city	ASCII string up to 30 characters
ST: state name	address section/state	ASCII string up to 10 characters
ZI: ZIP code	address section/ZIP	ASCII string up to 10 characters
CO: country name	address section/country	ASCII string up to 20 characters
PH: phone number	address section/phone	ASCII string, two section of up to 10 characters each, separated by a space
FX: FAX number	address section/FAX	ASCII string, two section of up to 10 characters each, separated by a space
DC: date code	(computer clock)	ASCII string MMDDYY hh:mm:ss R.rr/date of data creation and program revision

VII. HAND–CRAFTING UNIQUEWARE™ PROTOTYPES

If you are equipped for programming Add–Only Memories such as DS1982, DS1985, DS1986 (DS9097E COM port adapter, EPROMPC.EXE) then you have everything you need to make UniqueWare™ prototype devices. These prototypes differ in two important details from real UniqueWare™ parts: They have neither the UniqueWare™ ROM code nor a valid Project ID. Such prototypes therefore cannot provide the desired authentication. However, they allow you to look ahead at what you will get and you can continue with the development of your application software. To make such electrically functional prototypes you also need some unprogrammed DS2502 devices, cable to connect the DS2502 as well as a Touch Memory to the DS9097E and a DS1993 for intermediate data storage. As preparation, set up your UniqueWare™ data structure and write the result to the DS1993.

Next run the program EPROMPC.EXE and copy the data from the Touch Memory to the DS2502. You may then write–protect the DS2502 with another function of EPROMPC. Now run the UniqueWare™ project setup software again, restore the data from the Touch

Memory, manually increment the Serialization number and store the result back in the Touch Memory. Now you can again run EPROMPC and program the next device, and so on.

VIII. SKIPPING NUMBER RANGES

Together with your first shipment of ready–to–use UniqueWare™ devices you will receive a DS1993 Touch Memory containing all your project data plus Project ID and the first serialization number for your next order. To order the next quantity of UniqueWare™ devices you have to submit this device together with your order form. This Touch Memory eases keeping track of your UniqueWare™ information and greatly simplifies order processing.

As long as you need more devices with continuously incrementing serialization you have to do nothing with the device but submit it with your order. However, you might have a reason to skip a certain block of numbers and continue at a much higher number. In this case you have to replace the default number by a new, higher number without changing anything else inside the data structure before you submit the device with your order.

Updating the serialization number requires a COM–port adapter DS9097(E) and a Touch Probe DS9092GT (or equivalent electrical connection) installed at your computer. After your computer is ready, start the UniqueWare™ Project Setup program to load data from the DS1993 (UNIB <port #> R). You will get the prompt:

```
Probe Touch Memory to read data.
```

If the device you probe has a project ID assigned you will get the following message:

```
PID =   xxxxxxxx
PID field is already set.
```

(xxxxxxxx will show your Project ID)

```
To update start number only enter (U)
To begin new project enter (N):
```

After pressing "U" the following message appears:

```
Length Byte =   nn
Current Start Number:   yyyyyy
Enter new Start Number: <new number>
```

Now enter your new Start Number for serialization, followed by <RETURN>. This will transfer you to the screen "Saving Data", where you have to copy the updated set of UniqueWare™ data back to exactly the same DS1993 as you loaded it from. Following these steps ensures that the Project ID assigned to your set of data is not erased.

Should you press "N" instead of "U" the program will transfer to Screen 1 and allow all possible other modifi-cations with the loaded data, but the Project ID will be erased when writing back to a data carrier. If you have pressed "U" and you decide to update the number later, you can quit the program by hitting the <ESCAPE> key.

IX. ORDERING INFORMATION

The UniqueWare Project Setup Software as described in this document is available on a 3 1/2" DOS floppy disk. To order a copy of this disk, please call (214) 450–8167, send a Fax to (214) 450–3715 or send an EMAIL to AutoID.Support@dalsemi.com.

For details on UniqueWare Touch Memories please refer to the DS1981U/DS1982U data sheet. Details on UniqueWare Add–Only Memories are found in the DS2501–UNW/DS2502–UNW data sheet. These data sheets are available through the World Wide Web and the Dallas Semiconductor Fax–Back Service, phone (214) 450–0441. They are also included in the Automatic Identification Data Book.

All DS2501–UNW and DS2502–UNW UniqueWare Add–Only Memories can be ordered on Tape & Reel. When ordering Tape & Reel, please be aware that there is a certain yield loss in the process of putting the devices on the tape. For the TO–92 and SOIC package this loss is $1.0 \pm 0.2\%$, and $2.0 \pm 1.0\%$ for the TSOC package. To get full reels, it is a good idea to increase the order size by the expected yield loss and have excess devices (if any) shipped separately. Each reel holds 2000 devices in the TO–92 package or 2500 devices in the SOIC or TSOC package.

DALLAS
SEMICONDUCTOR

Application Note 104
Minimalist Temperature Control Demo

FEATURES

- Direct–to–digital, direct–to–PC instrumentation with graphical user interface demo
- Ready to use temperature control demonstration including piezo–sensor for sensing of mechanical activity and LED to simulate opto–isolators
- Features minimalist sensor chips and 1–Wire line–powered MicroLAN™ networking technique
- Self–configuring TMEX–based software automatically identifies devices and assigns communication port
- Includes DS1820 1–Wire Thermometer and DS2407 Addressable Switch as a Sensor/Actuator

DESCRIPTION

The Minimalist Temperature Control Demo is a closed loop 1–Wire MicroLAN control system. It consists of a ready to use printed circuit board (Figure 1), a cable with two RJ–11 connectors and a disk with software. In addition to this, the demo requires the DS9097 COM Port Adapter

To run the demo, connect the circuit board to the DS9097 adapter using the RJ–11 cable and plug the adapter into the RS232 serial communication port of your computer. Load the demo disk into the 3 1/2" disk drive and start the program TEMPCONT.EXE.

TEMPERATURE CONTROL CIRCUIT BOARD DESCRIPTION Figure 1
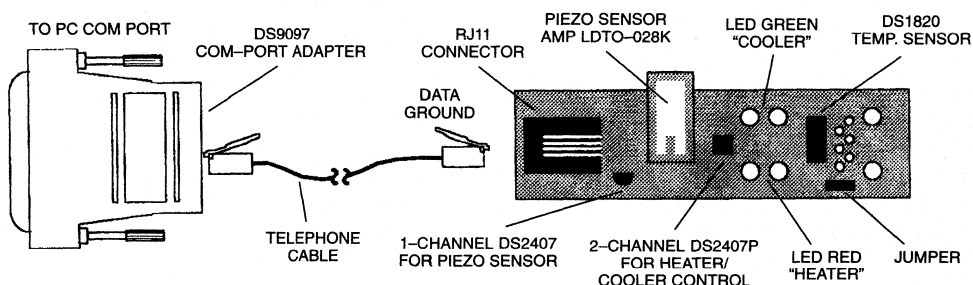
SOFTWARE OPERATION

The demo program comes up with the Temperature Control Graphical User Interface Tab (Figure 2). The actual temperature is displayed in the center of the window. The value is read from the DS1820 Thermometer and interpolated to the 0.1°C or °F resolution. A short beep marks every new reading. On startup, the heater and cooler trip points are close to the actual temperature and the heater is running. One may change the trip points up or down by clicking on the arrows. For this demo, the heater trip point must be at least 1.0°C or 1.8°F lower than the cooler trip point. The demonstration assumes that one warms up the DS1820 by touching it with a finger. This will raise its temperature and eventually switch off the heater, and, after some time, switch on the cooler.

Movement of the Piezo–Sensor will result in the program temporarily switching to the Piezo–Activity Tab (Figure 3), where the Dallas Skyline appears every time the piezo–sensor registers activity. To prevent the program from automatically switching back to the Temperature Control Tab, press P. To return back to temperature control, press T.

More information about the demo program is obtained by clicking on the question mark or by displaying the README.TXT file found on the demo disk.

The software will not work if the jumper is removed from the circuit board.

GRAPHICAL USER INTERFACE TEMPERATURE CONTROL TAB Figure 2

ACTUAL TEMPERATURE DISPLAY

CLICK HERE TO
SWITCH TO THE
PIEZO TAB

CLICK HERE TO
SWITCH ON/OFF
THE SOUND

CLICK HERE TO
DISPLAY HELP
INFORMATION

25.9° C

CLICK ON ARROW TO
CHANGE THE COOLER
TRIP POINT IN STEPS
OF 1°C OR 1.8°F

ACTUAL HEATER
TRIP VALUE

26° C

28° C

CLICK HERE TO TOGGLE BETWEEN
CELSIUS AND FAHRENHEIT

CLICK ON ARROW TO
CHANGE THE HEATER
TRIP POINT IN STEPS
OF 1°C OR 1.8°F

ACTUAL COOLER TRIP VALUE

GRAPHICAL USER INTERFACE PIEZO–ACTIVITY TAB Figure 3

CLICK HERE TO
RETURN TO THE
TEMPERATURE TAB

Piezo activity!

CLICK HERE TO
SWITCH ON/OFF
THE SOUND

CLICK HERE TO
DISPLAY HELP
INFORMATION

DEMO BOARD CIRCUIT DIAGRAM Figure 4
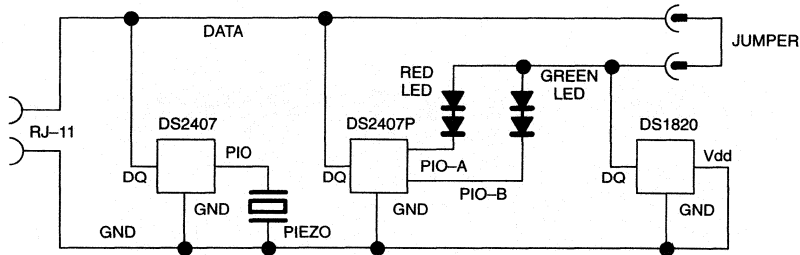
MINIMALIST TEMPERATURE CONTROL DEMO CONTENTS Table 1

KIT PARTS LIST	
DESCRIPTION	QUANTITY
Demo Board	1
RJ–11 Cable	1
DS9097 COM–Port Adapter	1
3 1/2" Demo Disk	1

SPECIAL DEVICES ON THE DEMO BOARD		
PART NUMBER	DISCRIPTION	QUANTITY
DS1820	Direct–To–Digital Temperature Sensor	1
DS2407	one–channel Addressable Switch	1
DS2407P	two–channel Addressable Switch	1
LTDO–028K	AMP Piezo Sensor for info call AMP at (610) 666–3545	1

1–WIRE LINE–POWERED MICROLAN Figure 5
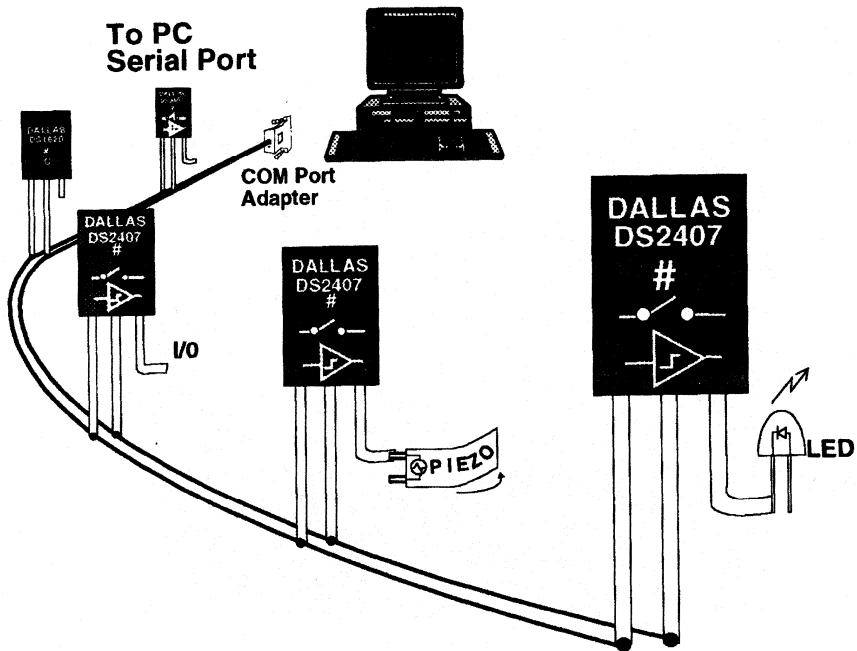
DS2407 ADDRESSABLE SWITCH CONTROL PANEL REPRESENTATION OF ALL RESOURCES AS VIEWED FROM PC Figure 6

DALLAS
SEMICONDUCTOR

Application Note 106
Complex MicroLANs
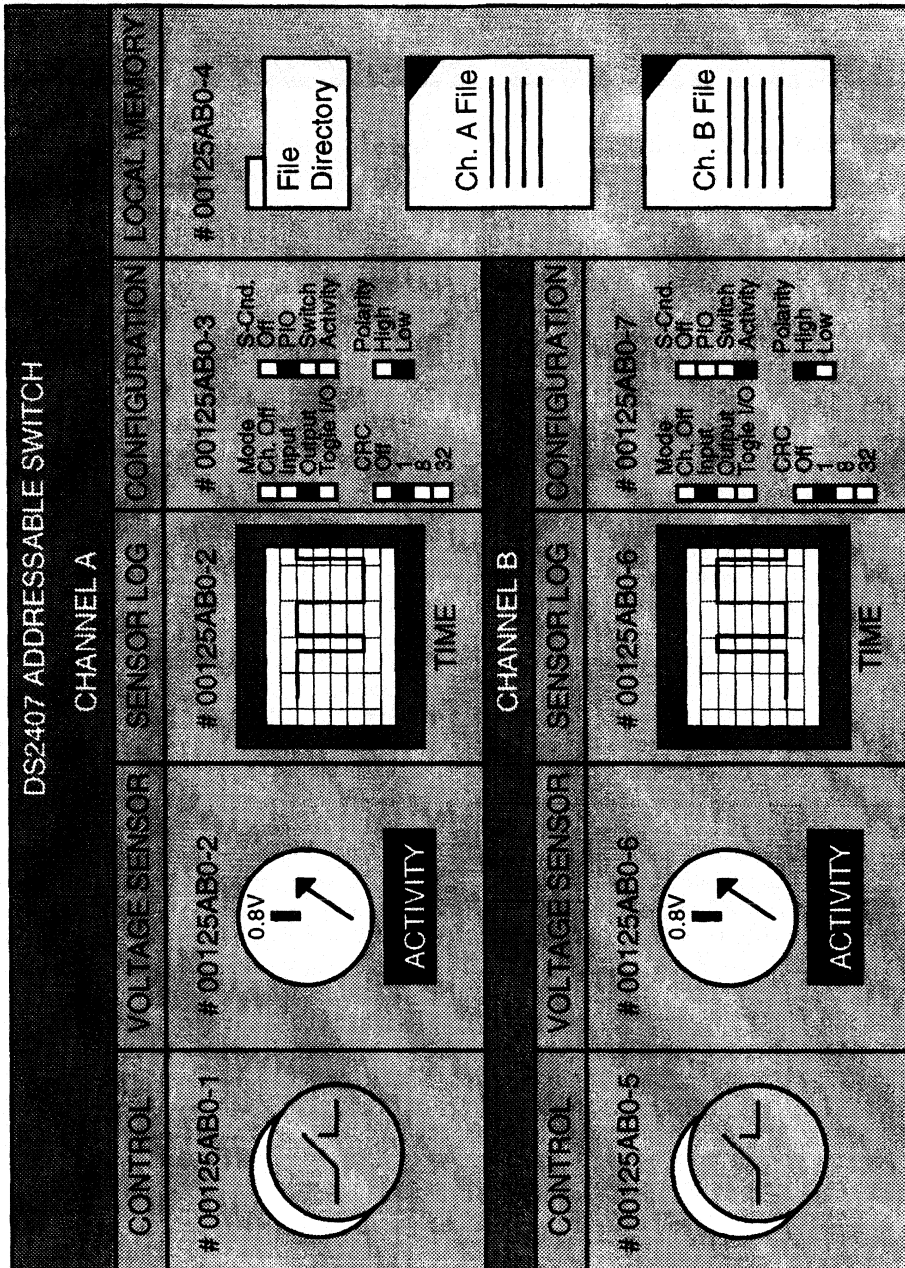
I. GENERAL CONSIDERATIONS

A. Topology

The MicroLAN is a feeder network using a single data line plus ground reference for digital communication. It provides digital information at very low cost to computers and their networks. A MicroLAN (Figure 1) can be likened to a tree consisting of a trunk and many branches. Its trunk connects via a 1–Wire™ driver to an RS232 COM port of a Personal Computer which can act as a MicroLAN server to a larger network. This computer is also referred to as Bus Master. It runs TMEX (Touch Memory EXecutive) 1–Wire communication software under MS–DOS or WINDOWS.

At the most distant end of the trunk, a Touch Memory called Trunk Marker is attached. Along the trunk there are addressable switches that branch to extensions called limbs. The limbs also have their markers at the ends of the cables. Branches extending from a limb are called twigs, branches extending from a twig are called stems. In any case, each branch point has its own marker at the most distant end of the path. The function of markers is to verify that communication can reliably pass all the way to the path extremities. Markers are distinguished from other devices by their data contents. In the general case, mobile Touch Memory Buttons (the sub–network), sensors and I/O switches are connected like leaves to limbs, twigs and stems, but not to the trunk. If a network has no branches, however, the "leaves" are directly connected to the trunk.

To communicate with a device on a twig, first the addressable switch connecting the limb to the trunk and then the switch connecting the twig to the limb must be activated. Activating branches with software controlled switches allows the building of huge feeder networks, yet avoids having all devices loading the cable at the same time. In addition, the addressable switch can provide information about physical location. This is not possible if all components are residing on the trunk.

Data communicated via the MicroLAN is packetized. Either the MicroLAN device adds a CRC (cyclic redundancy check) to the data or the data itself is packetized before it is written to a device. If for any reason a packet gets corrupted during data transfer, the Bus Master can detect the error by checking the CRC and repeat the data transfer. (See the "Book of DS19xx Touch Memory Standards" for details).

Communication on MicroLAN branches is not event driven; the Master typically polls the nodes and checks if an activity is required. A Personal Computer equipped with several COM ports can support several independent MicroLANs (trees), thereby increasing the performance or total size of the feeder network.

B. Building Branches

To realize huge feeder networks without simultaneously increasing the electrical load, MicroLAN uses a tree–like topology consisting of several levels of branches that finally connect to the trunk. The core component to activate and deactivate branches (i. e., making connection under software control) is the addressable switch. This device is basically a transistor that can be remote–controlled (switched on/off or sensed) via its MicroLAN interface. How the addressable switch connects a limb to the trunk (or creates a new branch level somewhere in a MicroLAN) is detailed in Figure 2. The Data Line of the MicroLAN is permanently routed to all devices in the system. The Return Line, however, is conducting only in those branches that are needed to access the leaves the master intends to communicate with.

Note that the MicroLAN Return Line is not identical to system ground. To avoid ground loops, a circuit employing optoisolators is required to communicate with circuits that must be grounded. Figure 3 gives an example of such an optoisolated interface.

UNLIMITED NETWORKING Figure 1
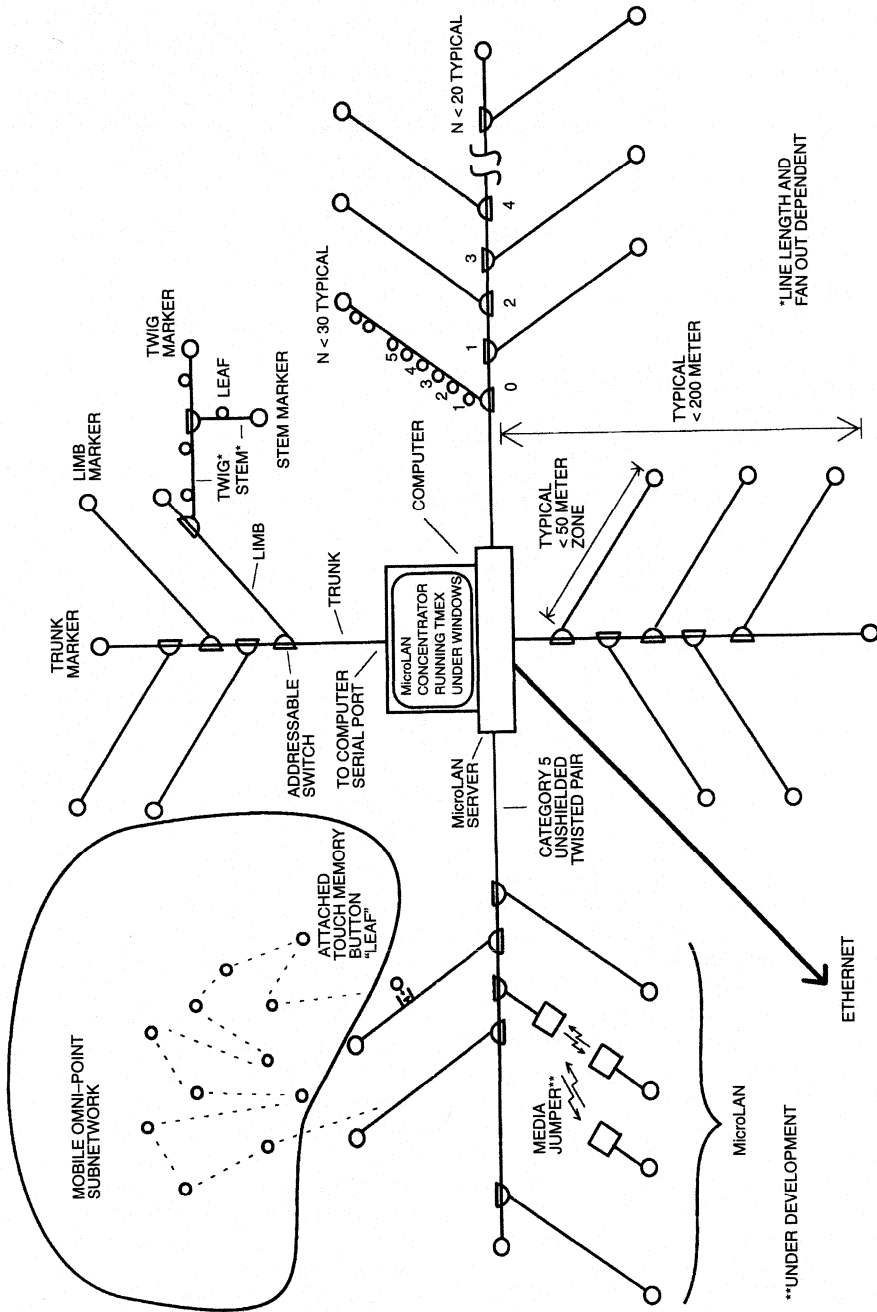
BUILDING BRANCHES Figure 2
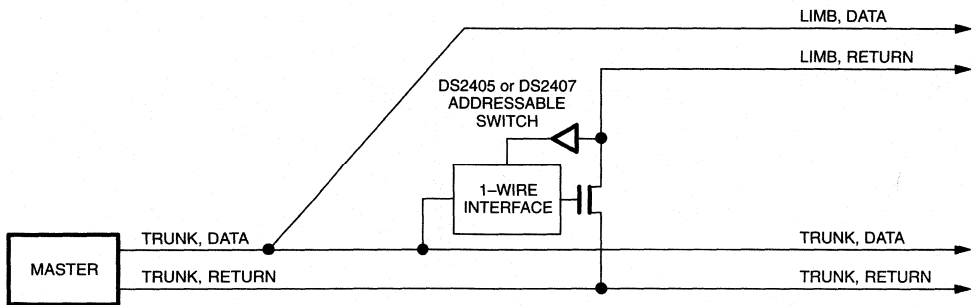
OPTO–ISOLATED INTERFACE Figure 3
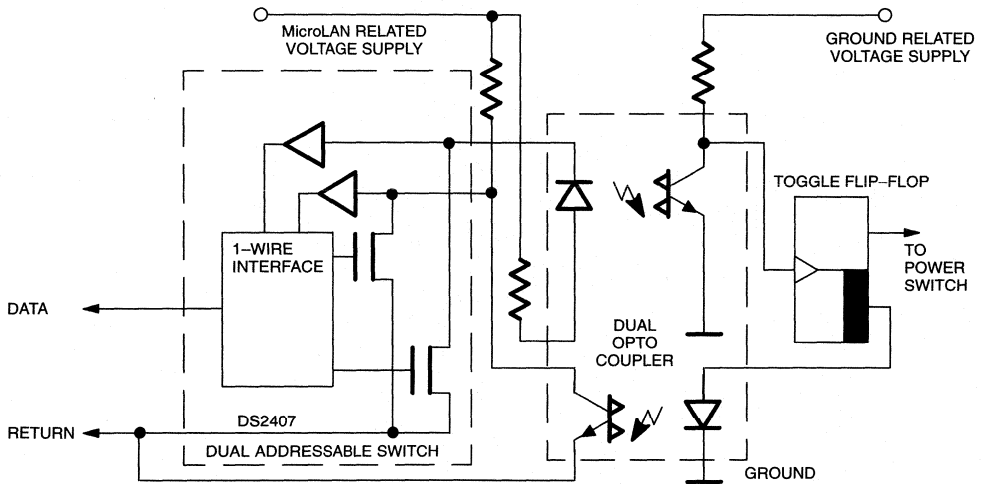
C. Software Considerations

After powering–up a MicroLAN, the bus master will see only those 1–Wire devices that reside on the trunk. In the general case these are addressable switches and the trunk marker. Before any communication beyond the trunk can take place the bus master needs to learn the topology of the network. The first step therefore is to perform an analysis of only the Addressable Switches on the MicroLAN. Beginning with the Trunk, the bus master systematically opens the switches to every limb one by one and records the registration numbers of the switches that it finds, then opens those switches one by one, etc., until it has built a complete model of the network topology in memory. In the second step the master needs to identify all other devices on the MicroLAN that are not addressable switches. Beginning with the Trunk and using the internal tree developed in the first step, the bus master systematically opens the switches to every limb one by one and records the registration numbers of the devices that it finds which are not Addressable Switches until it has added all additional devices to the memory model of the network topology. As the master knows the topology and population of the MicroLAN it now is able to build the path to any device in the network and to communicate with it. After the master determines which device the user wants to access it opens the switches necessary to reach the device, and addresses it with a Match ROM command. At this point, all devices have been deselected except the target device, which is prepared to accept a memory function command. This

scheme allows data to be read from or written to any Touch Memory in the network, regardless of its position.

Software like this has already been developed by Dallas Semiconductor and will be available as TMEX Micro-LAN Manager, part number DS0630N. This software is designed for the Microsoft Windows operating environment as a Dynamic Link Library which the software developer can link with his Windows programs. The MicroLAN Manager supports MicroLANs with rigid as well as permanently changing topology. An example of changing topology is a network with docking stations where sub–networks may connect at any time. Such a sub–network could be a cart with containers that are wired as a MicroLAN themselves.

II. IMPACT OF COMPONENTS

A. Effects From the Cable

With long distances, any cable shows transmission line effects unless the signal slew rate is slow compared to the signal propagation time on the cable. Undesirable signal excursions (reflections) which can disrupt communications show up if the cable is not properly terminated and the transition time of the transmitted signals is too short for the electrical length of the cable. In the case of the MicroLAN it is not possible to terminate the cable, so controlling the signal slew rate is important.

Each cable type has its own characteristic capacitance, inductance and resistance. These parameters are determined by the cable geometry, the type of conductor (solid, stranded, gauge size, etc.) and the type dielectric between the conductors. These define the characteristic impedance of the cable, the signal damping and the signal propagation velocity which for a practical cable is approximately 2/3 the speed of light. Typically, the specific capacitance which can range from 30 pF/m to 100 pF/m, needs to be taken into account since it contributes significantly to the total capacitive load of a network. Also the differential inductance of the cable becomes increasingly important as cable length increases.

The ohmic losses inside the cable reduce the noise margin of the digital signals. When building large networks, the branches and addressable switches connecting them add to the ohmic losses of the trunk. Usually the cable capacitance together with the parasitic power supply of the 1–Wire devices are the major factors that limit the size of the network. This requires a precharge time of a few milliseconds before communication can start on the MicroLAN, especially if a passive (resistive) pull–up is used for providing energy and communication voltage. A precharge time of one millisecond should also be allowed if one activates a new branch of the network. The parasitic power supply of the new arriving devices will cause a droop in the 1–Wire idle voltage until the energy reservoirs of the new devices are filled sufficiently. After that, the recovery time after each time slot will be long enough to replenish energy used during the previous time slot.

B. Open Drain Considerations

The MicroLAN is an open drain (wired–AND) environment with typically a passive resistor pull–up to the operating voltage of nominally 5V. Communication is done in time slots of 60 µs minimum per bit plus a short recovery time between time slots. Due to the low impedance active pull–down to generate a logic 0, the fall time of the communication signals is very short. The rise time is determined by the product of pull–up resistor and the total capacitive load of the active branches of the network, counting the cable itself and the input capacitance of the devices. The maximum high voltage on the Micro-LAN is determined by the value of the pull–up resistor and the total idle current of all devices on the activated segments of the network. The more devices that are listening, the higher the voltage drop across the pull–up resistor will be, limiting the maximum voltage the network can reach. The higher the voltage drop, the longer it will take for the network to reach the logic 1 level of 2.2V and the minimum operating voltage of 2.8V. This has some consequences for the 1–Wire communication.

The critical part of 1–Wire communication is the read data time slot, especially if a 1 is being transmitted. This becomes most critical in the ROM search process, which is required to identify the devices on the bus. In the general case, there may be many switches scattered on the trunk as well as leaves on the selected limb and twig and stem. Each of these devices sees the trigger provided by the falling edge of the time slot issued by the master at a slightly different time point (variations up to 2 µs due to transmission line delay). Theoretically, any device may have a cycle time (t_{RDV}) as short as 15 µs. Since the position of the devices on the limbs and trunk is arbitrary and the cable can never be terminated with its characteristic impedance echoes in the form of spikes and undershoots may occur unless the master's

pull down transistor has its LOW–going slew rate limited.

The rise time of the communication signals can be improved by reducing the value of the pull–up resistor, by using lower capacitance cable, or reducing the load of devices in the active segments of the network, whichever is applicable. The pull–up resistor, however, should not be below 1.5kΩ. Using a very low value will increase the logic 0 voltage the master sees when reading from the network, reducing the noise immunity of the system. In difficult situations an active pull–up driver may be used to provide optimized wave forms and highest noise margins. Such a driver could also include circuitry for echo cancellation.

C. Addressable Switches

The addressable switch is the core component for creating branches of complex MicroLANs. It is a three–terminal device with a open drain output of a transistor controlled by the bus master through the 1–Wire network. When switched on, the impedance of the transistor is typically 15 Ω (DS2405, < 10 Ω with DS2407); switched off the impedance is minimum 10 MΩ. Due to its low but non–zero impedance, every switch in the path to a device in the MicroLAN creates a small voltage drop of approximately 30 mV (15 Ω x 2 mA, for example, assuming a 2.5kΩ pull–up resistor and a pull–up voltage of 5V). The more switches involved in building a path, the higher the logic 0 voltage the bus master will see when reading from the network. The addressable switch has almost no impact on the logic 1 voltage; it adds to the load of the network as any other component on the activated branches.

If a limb of a MicroLAN is deactivated by switching its addressable switch into the high impedance state, all devices on that limb and beyond loose their parasitic power. The addressable switches affected will loose their status. If deselected for approximately 1 second or greater and then powered up again, they all will have their switching transistor non–conducting. Due to the parasitic power supply, the power–up cycle will cause a short drop in the 1–Wire voltage until the energy reservoirs of the devices in the newly activated branch are filled sufficiently.

If addressable switches on branches are used to permanently (de-)activate equipment through the Micro-LAN, the optoisolated interface (see previous chapter, "Building Branches") should be used to keep the equipment running during times where the path leading to that particular switch is not powered.

D. 1–Wire Interface, Parasitic Power Supply

To understand the voltage and timing issues of the MicroLAN, one should refer to its electrical equivalent circuit (Figure 4). The MicroLAN consists of three segments, a) bus master, b) the wiring and coupling and c) the 1–Wire devices themselves.

At the master's side there is a DC voltage source representing the pull–up voltage, the single network pull–up resistor and the pull–down switch, a digitally operated transistor controlled by the bus master.

The wiring between the master and the 1–Wire devices is modeled by the inductive and ohmic resistance of the data and return lines and the lumped capacitance of the cable. The cable capacitance is calculated by multiplying cable length between the master and the most distant device by the specified cable capacitance. The inductance of importance is the differential inductance which is that measured across the cable input with the two wires of the bus shorted together at the far end. Differential inductance is substantially lower than the inductance of a single wire because the current flows in opposite directions in the pair and in an ideal case would cancel completely. The resistance of the data line represents the cable length multiplied by the specified resistance of a single wire. The resistance of the return line is the same as the resistance of the data line plus a few Ohms for each addressable switch between the bus master and the most distant branch.

A 1–Wire device is represented by its input capacitance (CINPUT), a constant discharge current (IDISC, of typically 5 μA), the parasitic power supply circuitry (Di, Ri, CLOAD) and its operating current (Iop) of 10 μA during communication. Any device residing on the network will consume operating current, even if not addressed. This current is required to keep its 1–Wire interface synchronized with the communication protocol. When conducting, its impedance is nominally 100 Ω which results in a current sink capability of 4 mA at a voltage drop of 0.4V. If multiple 1–Wire devices are residing on the bus, CINPUT, IDISC, Iop and CLOAD need to be multiplied by the number of devices; Ri needs to be divided by the number of devices. The MOSFET inside the equivalent circuit of the 1–Wire device allows the device to respond to the command by putting a logic 0 level on the network.

Except for the presence detect cycle, Search ROM command, Skip ROM and Read ROM command, only one such transistor may be conducting for short periods of a time slot when reading from an addressed device.

MicroLAN ELECTRICAL EQUIVALENT CIRCUIT Figure 4
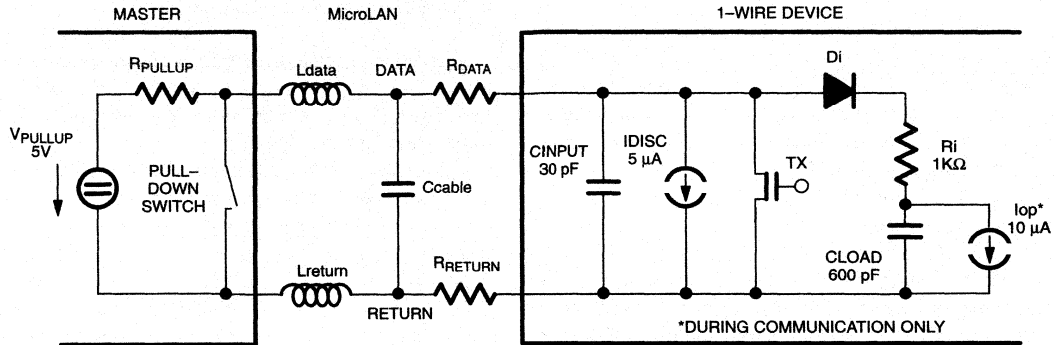
The fanout of a MicroLAN depends on the value of the pull–up resistor and the pull–up voltage. The limit is reached as the voltage drop across the pull–up resistor reduces the MicroLAN voltage to 2.8V. This is the minimum voltage required to recharge the parasitic power supply of the 1–Wire devices. From this DC requirement, Table 1 was obtained.

1–WIRE DC FANOUT Table 1

PULL–UP RESISTOR	PULL–UP VOLTAGE		
	4.0V	5.0V	6.0V
1.5kΩ	53	98	142
1.8kΩ	44	81	119
2.2kΩ	36	67	97
2.7kΩ	30	54	79
3.3kΩ	24	44	65
3.9kΩ	21	38	55
4.7kΩ	17	31	45

As shown in Figure 4, MicroLAN devices contribute to the capacitive load on the network. While the input capacitance of the device is always present, the typical 600 pF energy storage capacitance of the internal power supply only temporarily shows up when voltage on the line is first applied and the devices start to charge their internal energy reservoirs. Its effects are also seen when the devices replenish their power supply. This happens above the logic switching level at voltage levels of 2.8V minimum and therefore does not add to the system capacitance during normal communication. On heavily loaded long lines, the effect of recharging the parasitic power reservoir can be seen as a dip on the ris-

ing edge. When activating a new branch with many 1–Wire devices, however, the communication is delayed until this precharge is satisfied.

E. UART Operation

The UART is a bi–directional, self–timed serial communication device found in most computers. After setting its communication parameters (data rate, character length, parity, number of stop bits) the UART will transmit each character that it gets from the CPU (central processing unit) according to the specified parameters. Whenever activity is found on the receive channel, that waveform is assumed to follow the same communica-

tion parameters that apply to the transmit channel. Both, the receive and the transmit channel are tied together inside the DS9097 COM port adapter to form a 1–Wire interface.

To do 1–Wire communication, the UART is set up for a data rate of 115.2 Kbps. For the UART, each time slot is equivalent to a complete character. Depending on the bit pattern of the character written to the transmit register of the UART, the result will be a Write–One or a Write–Zero time slot. The UART is in effect creating and digitizing the 1–Wire timing by choosing a particular data pattern. The Read Data time slot is essentially the same as the Write One time slot. In contrast to the Write–One time slot, however, the CPU will not discard the data it finds in the receive register after a Read Data time slot was generated. Since all data is transmitted least significant bit first, any response from the MicroLAN is found in the least significant bits of the received character.

When reading, the logic of the UART's receive channel triggers on the transition from the idle status (voltage found during times of no communication or between characters) to the active status. With every character there is a preamble, the start bit, transmitted before the data of the character itself. This helps the UART to get ready for receiving valid data. At 115.2 Kbps the duration T of the start bit as well as the data bits is 8.68 μs (the reciprocal of 115200). The highest safety margin against non–ideal communication characteristics of serial data is reached if one samples the voltage of the communication signal at the midpoint of the window that represents a bit. The best time points for sampling therefore are at 1.5T, 2.5T, 3.5T, etc., after the receive channel is triggered. For 115.2 Kbps these sampling time points are 13.02 μs, 21.70 μs, 30.38 μs, etc., after the beginning of a character or time slot, respectively.

These numbers need to be correlated to the conditions of the read data time slot, as described in data sheets. When transmitting, a 1–Wire device is guaranteed to provide the data value of a bit for a minimum of 15 μs after the beginning of a read data time slot. This means that the first sampling of the UART's receive channel at

13.02 μs occurs at a good time to catch the response of a 1–Wire device, provided that in the case of reading a logic 1 the network was able to return to the logic 1 voltage within 4.34 μs (the time from the end of the start bit until the center of the first data bit).

This condition is difficult to meet, especially if there is a long cable between bus master and 1–Wire device. Fortunately, a 1–Wire device holds the value of a bit to be transmitted for typically 30 μs rather than 15 μs, which is the absolute worst case. As a consequence, if the operating conditions of all 1–Wire devices of a MicroLAN are kept within certain limits, using the second bit the UART reads during a time slot produces more reliable results. At 21.70 μs one is still on the safe side for reading a logic 0 and there are 13.02 μs for the data line to reach a logic 1 level in case the response from the network is a logic 1. One can rely on the second bit as long as all 1–Wire devices of a MicroLAN are within the temperature range of 0°C to 50°C and the 1–Wire pull–up voltage is 5V minimum. The DS9097 COM Port adapter, for example, provides a pull–up voltage of 6V; its pull–up impedance is approximately 1.5kΩ.

The PULLUP resistor of the MicroLAN together with the cable capacitance and the input capacitances of the 1–Wire devices on the active branches of the network represent the network time constant τ. As long as the load capacitors of the parasitic power supply are not yet being recharged, this network time constant determines the speed at which the MicroLAN data line returns to a logic 1 voltage. This waveform can be calculated as

$$V(t) = V_{PULLUP} * (1 - \exp(-t/\tau))$$

With the requirement that at t = 13.02 μs the 1–Wire voltage needs to have reached the 2.2V threshold of a logic 1, the value of τ can be calculated for any valid PULLUP voltage as

$$\tau = 13.02 \ \mu s / \ln(V_{PUP}/(V_{PUP}-2.2V))$$

The results are as shown in the table below:

MAX. NETWORK TIME CONSTANT (13.02 μs SAMPLING)	PULL–UP VOLTAGE		
	4.0V	5.0V	6.0V
τ	16.3 μs	22.4 μs	28.5 μs

Assuming the MicroLAN is loaded with the maximum number of devices as shown in Table 1, the cable capacitance alone must not exceed the value in Table 2 to yield the network time constant shown above.

MAXIMUM CABLE CAPACITANCE AT MAXIMUM NUMBER OF DEVICES (13.02 µs SAMPLING) Table 2

PULL–UP RESISTOR	PULL–UP VOLTAGE		
	4.0V	5.0V	6.0V
1.5kΩ	9.27 nF	12.00 nF	14.73 nF
1.8kΩ	7.72 nF	10.00 nF	12.28 nF
2.2kΩ	6.32 nF	8.18 nF	10.05 nF
2.7kΩ	5.15 nF	6.67 nF	8.19 nF
3.3kΩ	4.21 nF	5.46 nF	6.70 nF
3.9kΩ	3.56 nF	4.62 nF	5.67 nF
4.7kΩ	2.96 nF	3.83 nF	4.70 nF

Assuming a typical specific cable capacitance of 50 pF/m, Table 2 translates into the cable lengths shown in Table 3. The length values can easily be scaled to other cable capacitances by multiplication with the actual specific cable capacitance divided by 50 pF/m.

MAXIMUM CABLE LENGTH AT 50 pF/m (13.02 µs SAMPLING) Table 3

PULL–UP RESISTOR	PULL–UP VOLTAGE		
	4.0V	5.0V	6.0V
1.5kΩ	185 m	240 m	295 m
1.8kΩ	154 m	200 m	246 m
2.2kΩ	126 m	164 m	201 m
2.7kΩ	103 m	133 m	164 m
3.3kΩ	84 m	109 m	134 m
3.9kΩ	71 m	92 m	113 m
4.7kΩ	59 m	77 m	94 m

If using less than the maximum possible number of devices, the maximum cable capacitance as shown in Table 2 may be increased by 30 pF for each device less than the maximum number shown in Table 1.

So far, the UART description was not specific in the number of bits per character for the generation of 1–Wire time slots. Using six bits per character plus start bit, a character data pattern of all zeros would result in a logic 0 level for 7 x 8.68 µs, which is 60.76 µs. This matches the descriptions of the Write Zero time slot as specified in data sheets. A reasonable Write One time slot is achieved if the data pattern of the character is all ones; in this case the resulting wave form would have a logic 0 level for 8.68 µs caused by the start bit. Between characters there is an idle time inserted by the UART, called stop bit. If not otherwise specified, this stop bit has the same duration as the start bit, which is 8.68 µs at a data rate of 115.2 Kbps. All together, six data bits, one start bit and one stop bit result in a total length of 8 x 8.68 µs or 69.4 µs for a time slot including an idle time for recovery of 8.68 µs. These 69.4 µs are equivalent to a data rate of 14.4 Kbps, which is the fastest 1–Wire data rate possible through a UART, assuming that no time is

required by the microprocessor for reading and reloading the UART to provide the next character pattern.

In case of a network with a capacitive load close to the maximum possible value, the idle time of 8.68 μs following a write zero time slot is too short for the voltage on the data line to reach the minimum of 2.2V for the

1–Wire devices to recognize the end of a time slot. To increase the idle time, one can increase the character length by 1 or 2 bits. These additional bits need to be set to logic 1 to achieve the desired effect. Otherwise they would add to the time the data line stays at logic 0 level during a write zero time slot. The possible options and their practical results are shown in the table below.

IDLE TIME	8.68 μs	17.36 μs	26.04 μs
CHAR. LENGTH	6 BITS	7 BITS	8 BITS
WRITE ZERO PATTERN	00 0000	100 0000	1100 0000
DATA RATE	14.4 kbps	12.8 kbps	11.5 kbps

As long as the network time constant does not exceed 30% of the maximum allowed value for a given pull–up voltage, the shortest idle time is sufficient. For heavier loaded MicroLANs, the 17.36 μs idle time or more may be required. In any case, before building a huge Micro-LAN it is highly recommended to check the operating conditions as explained in the section III.D., Electrical Verification.

As the table shows, longer idle times reduce the possible data rate of the MicroLAN. A lower data rate increases the response time of the network for detecting changes in topology and/or population. At 14.4 Kbps the ROM search will discover 66 devices per second, not taking into account the limited speed of the bus master itself and the time to select the switches and precharge the limbs.

F. Computer Speed and Operating System

The timing on the MicroLAN is not solely determined by the UART and the network characteristics. The hardware design of the bus master itself and the software running on it also have impact on the performance of the network. Among other things, the task of the master is to read data from the UART and to reload it with the character for the next time slot. When transferring data exclusively (read or write many bytes) the behavior of the master is very predictable; it adds a constant delay between the characters read from or written to the UART. In case of a Search ROM procedure, the master has to do some extra data processing to determine the value of the next character before loading it into the UART's transmit register. This adds a varying delay between the characters, potentially extending the idle time between time slots significantly with respect to

transferring data only. In any case, the delay depends on the clock frequency applied to the microprocessor, the size of the program cache memory, the design of the application software and the operating system.

If the program loop being executed for reading or writing a stream of data fits into the fast program cache memory, the delay will be very short since no access to the slow main memory will be required to load the next block of program code. A sophisticated operating system burdens the bus master with some overhead that needs to be done at regular intervals, thereby slowing down servicing the UART.

Some very recent computer designs try to compensate for this delay by using a UART with FIFOs in the receive and transmit channel. Unfortunately, for 1–Wire communication the master often needs to know the value of the most recent bit before being able to provide the next bit, this approach will not work for MicroLAN applications. Therefore, if one is going to use such a computer as bus master for a MicroLAN, the FIFOs need to be disabled. This can be done under software control.

The delay between characters or time slots introduced by the bus master is difficult to predict. Measurements have been made using a 80386DX–based IBM–compatible computer running at 33 MHz. The computer had to measure the total time it needed to access a DS1993 and read the entire data memory. The UART was setup for a character length of 8–bits. Under DOS the computer–introduced delay was 20 μs; the Windows–result was 30 μs. For an 80486–based computer at 33 MHz clock rate one might expect half of this delay. These numbers cannot be scaled for other clock rates; if

required they need to be measured in a test setup. Although the computer–introduced delay slows down the communication speed of the MicroLAN, it gives the 1–Wire devices another opportunity to recharge their parasitic power supply.

III. MicroLAN OPTIMIZATION

A. COM Port to MicroLAN Adapter

For most MicroLANs the DS9097 COM port adapter is sufficient. For heavily loaded lines, extreme temperatures or maximum speed a more sophisticated adapter is required. Such an adapter consists of an RS232 level translator, a MicroLAN driver and a passive pull–up as outlined in the functional block diagram of Figure 5. The driver circuit includes an active pullup and a slew–rate controlled pull–down transistor that translates the TXD signal from the COM port into 1–Wire time slots.

The value of the 1–Wire pull–up resistor may range from 1.5 kΩ to 4.7 kΩ, as explained in chapter II. The purpose of the diode in series with the pull–up resistor is to prevent energy flow into the circuit if the interface is not powered up. In the unpowered state the interface should have a high impedance to allow another bus master to be connected. Due to its low 0.2V forward voltage drop a Schottky diode is preferred over a standard silicon diode.

In Figure 5 the supply energy for the interface is taken from the RS232 control lines DTR and RTS as well as from the transmit data line TXD. This approach works reliably if the COM port drivers meet the RS232C standards. The level translator directly connects to the MicroLAN and converts its 0V/5V levels to +12V/–12V

RS232 levels. Its negative voltage supply is derived from TXD during idle times and read data time slots.

Since on longer cables there could be reflections causing positive edges, the active pull–up must be made immune to such noise during the time reflections can occur. For this reason the Noise Filter and Pull–up Control is found as part of the MicroLAN driver block in Figure 5. The control circuit is coupled to the MicroLAN via an RC high pass filter.

How the noise filter and pull–up–control circuit works is shown in Figure 6. It consists of two parallel paths feeding into a NAND gate that directly controls the pull–up transistor. The upper path of the control circuit is basically a slope detector that generates a HIGH pulse of duration $\tau 1$ at the node Noise Gate every time the MicroLAN is pulled LOW by the master. The duration of $\tau 1$ is nominally 7.5 µs. The lower path of Figure 6 is very similar to the upper path. The principle difference is the additional analog inverter in front of the slope detector. Inverter and slope detector together generate a HIGH pulse of duration $\tau 2$ at the node "Pull–up Control" every time the MicroLAN goes back to a HIGH level when it is released from a LOW. The gain of the analog amplifier determines at which voltage level the Pull–up Control signal becomes active. The duration of $\tau 2$ is nominally 7.5 µs. The NAND gate finally lets the Pull–up control signal pass to the pull–up transistor if the noise gate is inactive. This prevents the active pull–up from being activated by reflections that can occur during the first few microseconds after the line is pulled LOW. Depending on the cable length it may be necessary to limit the active pull–up current by adjusting the resistor between the pull–up transistor and the MicroLAN (Rise time adjust).

OPTIMIZED COM PORT TO MicroLAN ADAPTER Figure 5
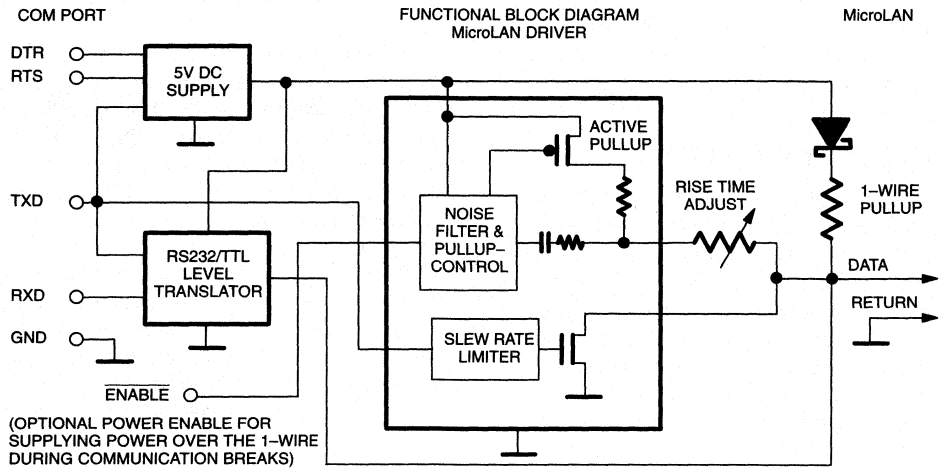
NOISE FILTER AND PULLUP CONTROL CIRCUIT Figure 6
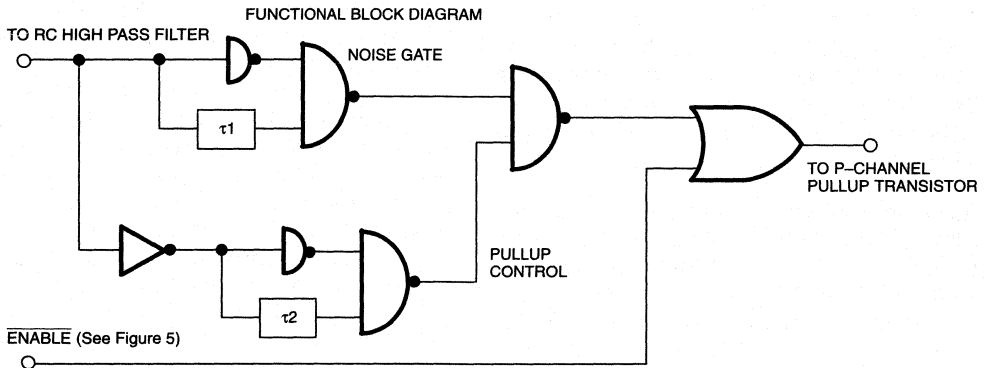
B. MicroLAN Topology

For best performance of a MicroLAN installation one has to find the conditions under which the loading of the MicroLAN is minimized for a given number of nodes. Assuming a topology where the number of branches from each branch level is constant, the optimum number of branch levels can be calculated.

To explain how this optimization is done one has to define a few parameters:

M = Number of target devices (not counting address able switches) to be reached via the feeder net-work.

N = Number of branch levels (limb is first level, twig second level, stem is third level, etc.)

S = Number of branches including leaves (not count ing markers) branching off/connected to each level.

L = The total load, defined as the number of listening MicroLAN devices when a target device is addressed.

It can be shown theoretically that the optimum value of branches on each branch level is approximately 3.6 and that this value does not depend on M. This result applies to a configuration where each branch has one marker in addition to the addressable switches. The target devices are located on the most distant branches. Since the non–integer value of 3.6 cannot be realized in practice, for the following example the value of four has been chosen. The Table 4 shows how the capacity M and the load increases with each branch level. It also proves that even with more than 4096 target devices the DC fanout is not the limiting factor for the size of the Micro-LAN.

Figure 7 shows a MicroLAN example of four branch levels according to this table. The number of listening devices is found by counting the addressable switches and markers on the active branches and the leaves (= target devices) on the most distant branch (total of 20). Since there are four leaves per stem, four stems per twig, four twigs per limb and four limbs at the trunk, the number of target devices (not counting addressable switches and markers) is 4 x 4 x 4 x 4 = 256 without having more than 20 devices listening at a time.

MINIMUM LOAD TOPOLOGY Table 4

N (NUMBER OF BRANCH LEVELS)	L (LOAD OF LISTENING DEVICES)	M (TARGET DEVICES)
1 (TRUNK ONLY)	5	4
2 (TRUNK, LIMBS)	10	16
3 (TRUNK, LIMBS, TWIGS)	15	64
4 (TRUNK, LIMBS, TWIGS, STEMS)	20	256
5	25	1024
6	30	4096

MINIMUM LOAD TOPOLOGY Figure 7
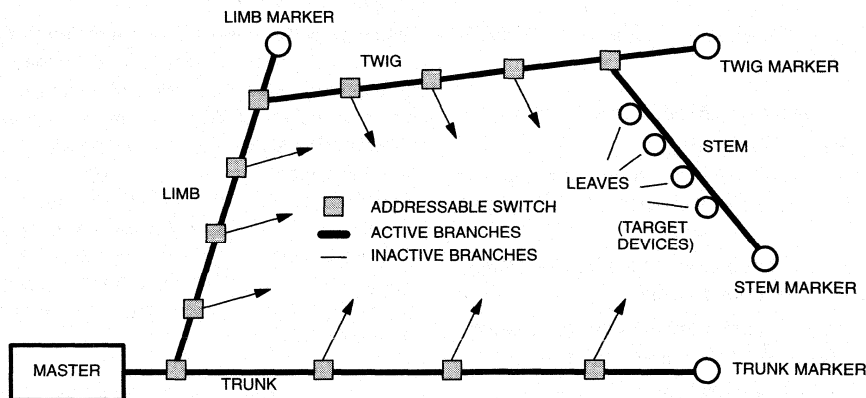
C. Protection and Noise

When interfacing a MicroLAN to a computer one has to provide a means to protect the system from damage by ESD (electrostatic discharge) and EMI (electromagnetic interference) from radio stations, communication equipment and lightning, etc. All MicroLAN compatible 1–Wire devices have built–in ESD protection circuitry to withstand ESD events (±10kV minimum Human Body Model). At the bus master's end of the MicroLAN, reliable protection can be provided by means of diodes shorting negative spikes or feeding positive spikes higher than 5V to the power supply unit where they are smoothed by capacitors and other protection devices.

When selecting protection diodes or surge suppresses to be connected to the data line one should choose devices with minimum junction capacitance and fast switching times.

The cable recommended for MicroLANs is regular unshielded twisted pair category 5 cable. Such cable is available with two or more pairs of wires. The fact that the wires are twisted reduces unwanted coupling from nearby power cables or analog phone lines by canceling the induced voltages. The specific capacitance between the wires of a pair is approximately 50 pF/m; between wires of different pairs a value close to 30 pF/m

can be expected. Unused wires need to be left unconnected at both ends of the cable. Grounding them is worse than leaving them floating: it can increase the capacitive load on the MicroLAN so significantly that communication becomes impossible. It is also not recommended to run two MicroLANs through the same cable, unless each pair is separately shielded, and the cable also shielded. The recommended shielded cable is IEEE 1394 "Firewire".

D. Electrical Verification

After the topology of a MicroLAN application has been defined it is recommended to check the electrical feasibility. This is done using the equivalent electrical circuit of Figure 4. The total capacitance of the cable is lumped into Ccable. R_{DATA} is the impedance of the data line proportional to the cable length. R_{RETURN} represents the impedances of all addressable switches in the branch plus the wire impedance of the return line (same length as the data line). The diode is considered to have a voltage drop of 0.7V and be ideal otherwise. For simplicity, the effect of Ldata and Lreturn is not included in this calculation.

To verify the electrical feasibility of a MicroLAN perform the following steps:

1. calculate L1, the maximum cable length when activating the most distant branch

2. calculate L2, the cable length when activating the most heavily loaded branch (branch with most target devices)

3. count N1, the number of devices loading the MicroLAN in case of the longest cable

4. count N2, the maximum number of devices loading the MicroLAN in case of the heaviest load

5. calculate the total capacitances;
   Ccable1 = L1 x ccable
   Ccable2 = L2 x ccable
   C1 = Ccable1 + N1 x CINPUT
   C2 = Ccable2 + N2 x CINPUT
   Ctotal is the higher value of C1 and C2.
   Ccable is the specific cable capacitance.

6. Using N2, select the pull–up resistor for the available pull–up voltage from Table 1. If possible select a pull–up resistor even lower than the table indicates, but not less than 1.5kΩ.

If the load is too high for the available pull–up voltage, split up the most heavily loaded branch to reduce the DC load and go back to step 1.

7. calculate the network time constant by multiplying Ctotal x R_{PULLUP}.

The passive pull–up may be sufficient if the network time constant does not exceed 16.3 µs at 4V pull–up voltage, 22.4 µs at 5V or 28.5 µs at 6V, respectively. If the time constant is higher, reduce R_{PULLUP} as far as possible. If this is not possible or sufficient, an active pull–up driver is required.

8. calculate the maximum LOW voltage at the master's end of the MicroLAN
   Vml = V_{PULLUP} x (R_{DATA} + R_{RETURN}) / (R_{DATA} + R_{RETURN} + R_{PULLUP}).
   The result needs to be less than 0.8V; otherwise the bus master might have difficulties in recognizing a logic 0 level on the MicroLAN.

9. check the recharge condition, i.e., verify that between time slots there is sufficient idle time for the MicroLAN to allow recharging the parasitic power supply of its 1–Wire devices.

With a parasitic power supply capacitor CLOAD = 600 pF and a current demand of 10 µA the capacitor will lose 1.0V during a write 0 time slot. To be able to tolerate an indefinite series of Write 0 time slots, the idle time between time slots needs to be long enough to restore the energy consumed previously. To calculate the recharge time it is assumed that the operating current stops and recharging CLOAD starts as soon as 2.8V are reached on the data line of the MicroLAN. The discharge current Idisc will continue flowing. As soon as CLOAD is being recharged the network time constant changes from

$$\tau 1 = R_{PULLUP} \times (Ccable + N \times CINPUT) \text{ to}$$
$$\tau 2 = R_{PULLUP} \times (Ccable + N \times (CINPUT + CLOAD)).$$

At a load of 20 1–Wire devices , for example, this easily doubles the value of the time constant, even if cable was already a high capacitive load.

The effective voltage available on the MicroLAN to recharge the parasitic power supply will be

$$\Delta U = V_{PULLUP} - 2.8V - N \times R_{PULLUP} \times IDISC.$$

Since 1.0V was lost during the write 0 time slot, the voltage on the MicroLAN needs to rise by 1V at a time constant of $\tau 2$ and a voltage of ΔU available for recharging. Solving the differential equation the required recharge time is:

$$trecharge = \tau 2 \times \ln (\Delta U / (\Delta U - 1V)).$$

This recharge time begins as the voltage on the data line has reached 2.8V. With a network that reaches the 2.2V level at 13 µs after the bus master has stopped pulling the data line to 0 V, the 2.8V level will be reached after no more than 20 µs maximum. The idle time between time slots therefore needs to be:

$$tidle = 20 \ \mu s + trecharge.$$

The recharge condition is fulfilled if the idle time generated by the UART plus the inter character delay introduced by the bus master is greater or equal to the required tidle as calculated above. For example, a UART that is setup for 8–bits character format provides an idle time of 26 µs. If the bus master is not one of the fastest models, it may add 20 µs to the idle time between time slots. In this case the recharge condition is fulfilled if the recharge time does not exceed 26 µs.

If the recharge condition is not fulfilled with the R_{PULLUP} and V_{PULLUP} chosen, R_{PULLUP} needs to be reduced further (not below 1.5kΩ) or V_{PULLUP} needs to be raised (up to 6.0V), if possible. If this is not sufficient to fulfill the recharge condition, the network topology needs to be changed to reduce the load from the devices on the network or an active pull–up driver as outlined in Figure 5 will be required.

Checking the recharge condition in case of a 4V pull–up voltage, one might find very long recharge times. With a high number of devices it might be impossible to calculate the recharge time since the maximum voltage for recharging is less than the 1.0V required to compensate for the energy loss. Nevertheless, the network might work if the idle time is very long, e. g., in the range of 100 µs or more. The reason for this comes from the fact that IDISC is not as constant as assumed for the calculation.

It may be in the range of 1.5 to 3 µA if the voltage on the data line is very low. Loss of energy at the parasitic power supply will reduce the threshold voltage where the recharging starts to a value close to 2.3V. This together with a lower value of IDISC will allow a slow MicroLAN operation even at 4.0V pull–up voltage. However, for reliable operation at a reasonable speed one should avoid such a low pull–up voltage. Whenever possible, the pull–up voltage should be in the range of 5.0 to 6.0V.

Sample Calculation
(The cable inductance is not included in this calculation.)

rcable = 0.085 Ω/m
ccable = 30 pF/m
V_{PULLUP} = 5V
L1 = 220 m
L2 = 250 m
N1 = 40
N2 = 45
C1 = 6.6 nF + 1.2 nF = 7.8 nF
C2 = 7.5 nF + 1.35 nF = 8.85 nF
Ctotal = 8.85 nF
R_{PULLUP} = 2.7kΩ (taken from fanout Table 1)
Network Time Constant = 2.7kΩ x 8.85 nF = 23.9 µs

This is too long for 5V pull–up voltage. R_{PULLUP} therefore needs to be reduced to 2.2kΩ. This results in a network time constant of 19.5 µs, which is sufficient for the data line to reach a logic 1 value when sampling at 2.5T or 21.7 µs after the beginning of a time slot.

R_{DATA} = 250m x 0.085 Ω/m = 21 Ω
maximum 3 addressable switches (15 Ω each) required to reach the target branch.

R_{RETURN} = R_{DATA} + 3 x 15 Ω = 66 Ω
Vml = 0.19 V.
$\tau 2$ = 97.0 µs
ΔU = 1.59 V
trecharge = 96 µs

This value is not acceptable. R_{PULLUP} is now reduced to 1.5 kΩ. These are the new results:

$\tau 2$ = 53.9 µs
ΔU = 1.86 V
trecharge = 41.6 µs

This value is better, but still very long. The pull–up voltage is now raised to 6.0V. This results in:

$\tau2 = 53.9\ \mu s$
$\Delta U = 2.86\ V$
$trecharge = 23.2\ \mu s$

This value is reasonably short. Since the pull–up resistor and pull–up voltage were changed, the new maximum LOW voltage needs to be checked.
$Vml = 0.33\ V$

This value is within the allowed range. The MicroLAN in this example will work if the UART is setup for 8 bits (yielding 26 μs idle time) and the bus master itself is slow enough to add 17 μs minimum to each time slot.

IV. SUMMARY
A heavily loaded large MicroLAN implementation works most reliably if the following conditions are met:

- controlled slew rate of about 3 ... 4 μs

- minimum cable capacitance and inductance (Category 5 twisted pair or IEEE 1394 "Firewire")

- line capacitance completely discharged in every time slot

- reverse–biased Schottky diode termination at the far end of the cable

- active pull–up driver circuit (Figure 5)

- late sampling at 2.5T or 21.7 μs respectively

- maximum 1–Wire pull–up voltage

- multi–level branching topology (Figure 7)

A more detailed discussion of the most important of these recommendations is found in Application Note 108, "MicroLAN – In The Long Run."

TELECOMMUNICATIONS

DALLAS
SEMICONDUCTOR

DS2141A, DS2143, DS2151, DS2153
Three Channel Drop and Insert
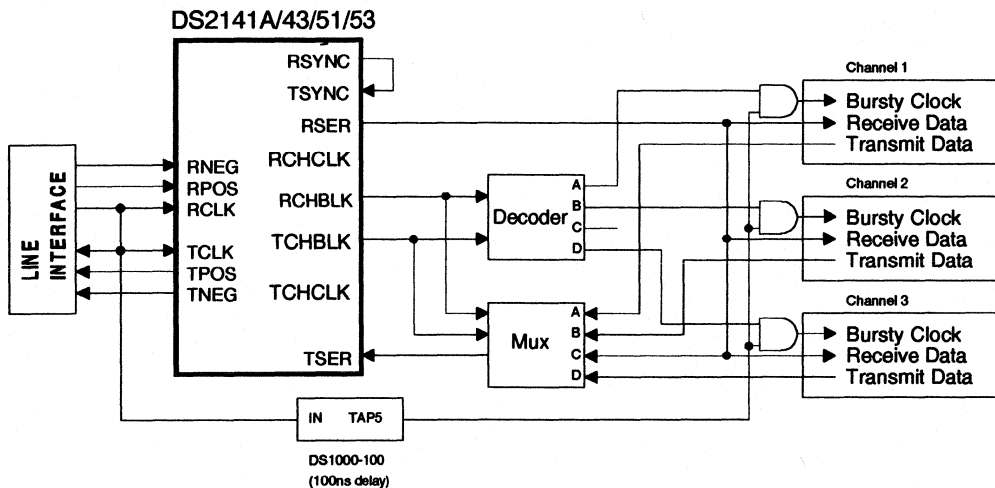
DS2141A/43/51/53

Notes:
1. a "looped-timed" application is shown
2. this application assumes the dropped channels occupy the same timeslots as the inserted channels
3. the idle registers in the DS2141A/43/51/53 can be used to fill unused (if any) channels
4. the used channels do not need to be contiguous nor do they need to be only one channel wide
5. the line interface function is included onboard the DS2151 and DS2153 devices
6. the delay is used to adjust the clock to account for the delay in generation of the signals that create the bursty clock
7. additional channels can be added by allowing the RCHCLK signal to be driven into a counter & decoded; the decoded signal would then be used to provide additional selects for the decoder & mux
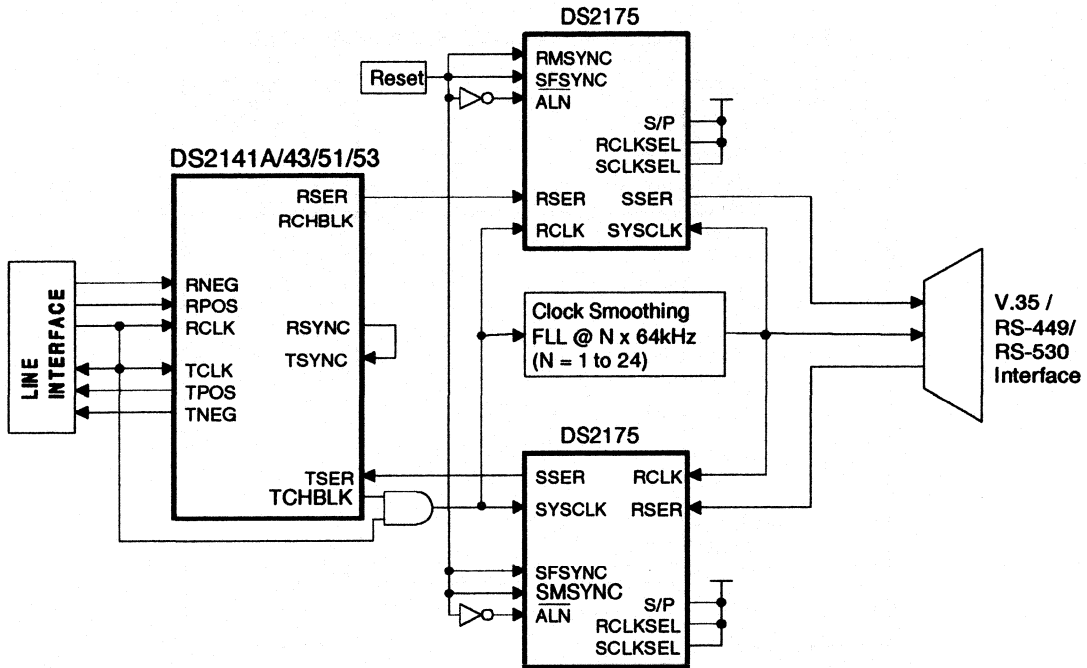
Notes:

1. a "looped-timed" application is shown
2. this application assumes the transmit and receive FT1/E1 streams occupy the same channels
3. the idle registers in the DS2141A/43/51/53 can be used to fill unused channels
4. the used channels do not need to be contiguous
5. reset should be activated after all the clocks are stable
6. the elastic stores in the DS2141A/43/51/53 are disabled
7. RCHBLK can used in conjunction with TCHBLK to allow up to three additional FT1/E1 ports
8. the line interface function is included onboard the DS2151 and DS2153 devices

DALLAS
SEMICONDUCTOR

DS2151, DS2153
Interfacing to the MC68MH360 QUICC32

Interconnections between the DS2151 or DS2153 and the Motorola MC68MH360 (QUICC32) are shown in Figure 1. The MC68MH360 can be configured as an HDLC controller implementing protocols such as LAPD for both D channel and the FDL. In the configuration shown, TDM channel A is used for timeslots 0-23 (T1) or 0-31 (E1) and TDM channel B is used for the FDL. For more information see the application note on Interfacing to a Non-multiplex Bus

Figure 1. **TRANSCEIVER - QUICC32 INTERCONNECTIONS**
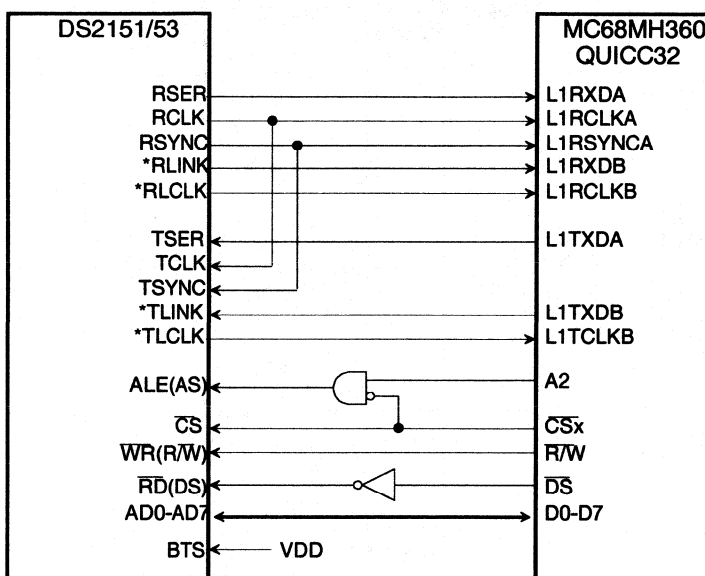
* HDLC on the FDL can be implemented either by the second serial input (TDM CHANNEL B) or via the port by the host processor (CPU32 internal to the QUICC32).

DS2151/53 NOTES

1. Other signals affecting operation of device are not shown.
2. Example circuit has RSYNC in output mode.

MC68360 NOTES

1. Other signals affecting operation of device are not shown.
2. Use SI mode register to:
    A. Set up transmit and receive frame sync delays (0 - 3 clocks) to mask the F-Bit in T1 applications. RFSDA = 1 for DS2151, 0 for DS2153
    B. Set clock edges for transmit on rising edge and receive on falling edge. CEA = CEB = 0
    C. In the above example, TDM channel A has a common transmit/ receive clock and sync. CRTA = 1
3. Use the TIMESLOT ASSIGNER to ignore Timeslot 0 for the DS2153.

DALLAS
SEMICONDUCTOR

DS2151, DS2153
Secondary Over–Voltage Protection

The DS2151 and DS2153 Single-Chip Transceivers are used in applications connecting directly to the outgoing telephone lines and hence the devices can be exposed to hazardous overvoltage conditions. For such applications, protection networks are used to direct high voltage or current away from the more sensitive low voltage CMOS semiconductor devices. The circuitry shown below details how to construct a protection network that provides for both longitudinal (common mode) and metallic (differential) surges as well as for power-line cross. The network described targets compliance with UL1459, FCC Part 68, Bellcore TR-NWT-1089, and ITU K17 - K20.

DS2151 / DS2153 PROTECTION NETWORK Figure 1
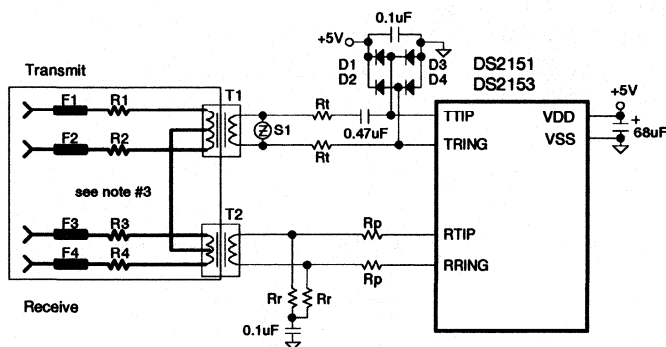
Components

Name	Description	Part #	Source
Rr	see note 1	1/8 watt 1%	
Rp	470 ohms	1/8 watt 10%	
Rt	2.2 ohms	1/8 watt 10%	
D1 to D4	Signal Diode	1N4148 or equal	
S1	30V Transient Supressor	P0300EA70	TECCOR
		TPN3021	SGS-Thomson
T1, T2	Transformers	(see Transformer Application Note)	
F1 to F4	Fuse		
R1 to R4	Power Resistor		

Notes

1. The value of Rr is determined by the line impedance (75, 100, or 120 ohms) and the value of R3 and R4 according to the following formula: Rr = [Line Impedance - (R3 + R4)] / 2
2. In the shaded area, layout is critical. Traces should at least 20 mils wide and be separated by at least 150mils.
3. Only needed in applications that require a DC path for the transmit and receive paths (i.e. T1 CSU applications).
4. See the Transformer Selection Guide for picking a transformer with the proper isolation.
5. F1 to F4 and R1 to R4 are required to protect the transformer during power-line cross according to UL1459; contact the factory for information on how to chose F1 to F4 and R1 to R4 to meet the application.

TYPICAL PROTECTION NETWORK Figure 2
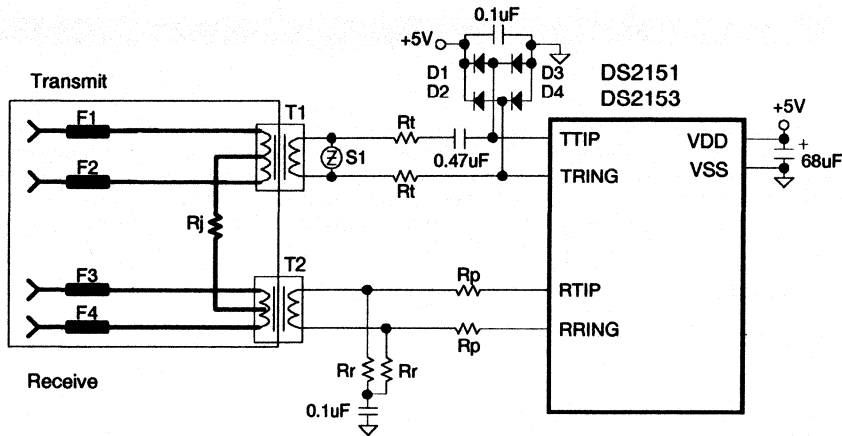
Components

Name	Description	Part #	Source
Rr	See Table 1	1/8 watt 1%	
Rp	470 ohms	1/8 watt 10%	
Rt	See Table 1	1/8 watt 10%	
Rj	See Table 1	1/8 watt 10%	
D1 to D4	Signal Diode	1N4148 or equal	
S1	30V Transient Supressor	P0300EA70	TECCOR
T1	Transformer 1:1.36CT 3KV Low DCR	PE-65832	Pulse Engineering
T2	Transformer 1CT:2CT 3KV Low DCR	PE-65835	Pulse Engineering
F1 to F4	1.25 Amp Slow Blow Fuse	2301.25	Littlefuse

RESISTOR VALUE SELECTION Table 1

Application	Rr	Rt	Rj
T1	50	4.7	0
E1 75 ohm (L2/L1/L0 = 110)	37	18	open
E1 120 ohm (L2/L1/L0 = 100)	60	27	open

Notes
1. Resistors are not needed to protect the fuse since the fuse can survive the applied surges.
2. Transformers with a low DC resistance are needed to allow them to withstand power line cross at the rated fuse limit (times 1.35) without compromising the transformers isolation.

DALLAS
SEMICONDUCTOR

DS2153 Selectable 75 and 120 Ohm E1 Interface

When interfacing to the E1 line with Secondary Over Voltage protection or in a high return loss configuration, resistors are added between the DS2153 outputs and the transformer. Normally the resistors are of different values depending on whether the interface is driving a 75Ω coax or 120 Ω twisted wire pair line. Also a change in the termination resistor is required. A single set of resistors can be utilized in both situations by using the circuit shown in Figure 1. The dotted lines indicate jumper positions for selecting 75 Ω or 120 Ω operation.

E1 INTERFACE Figure 1
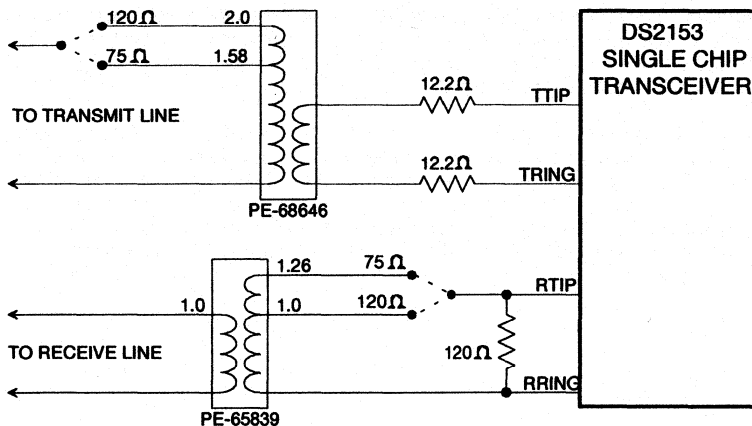
LINE INTERFACE CONTROL REGISTER SETTINGS Table 1

LINE TYPE	L2	L1	L0
75 Ω	0	1	0
120 Ω	0	1	0

NOTES:

1   Transformers shown are available from Pulse Engineering.

2   Resistors shown are within 1% of best value.

DALLAS
SEMICONDUCTOR

DS2141A, DS2143, DS2151, DS2153
Interfacing to a Non–Multiplexed Bus

The DS2141A/43/51/53 contains a some internal circuitry to allow it to more easily interface to an external controller that has a non-multiplexed address/data bus. In multiplexed schemes like the DS2141A/43/51/53 uses, a signal exists (usually called Address Strobe or Address Latch Enable) to denote that a valid address is present on the bus. This signal is necessary because the address and data lines share the same bus. In non-multiplexed schemes, this signal may not be supplied by the controller because the address and data lines are separated. The DS2141A/43/51/53 contains some onboard circuitry that enables it to internally create an address latch if it is used on a non-multiplexed bus.

The creation of this internal address latch occurs as follows. In the Intel timing set, when the ALE pin on the DS2141A/43/51/53 is high, if the WR\ pin goes active (i.e. low), then the address will be latched on the rising edge of WR\. In the Motorola timing set, when the AS pin is high and the R/W\ pin is low, if the DS pin goes active (i.e. high), then the address will be latched on the falling edge of DS.

As an example, in a non-multiplexed system, A0 could be tied to the ALE (or AS) input. Please see Figure 1. Each read or write to the DS2141A/43/51/53 would consist of two operations. To write to the DS2141A/43/51/53, first the controller would write the address and then the data. To read from the DS2141A/43/51/53, the controller would first write the address and then read the data. The addresses would be treated as data by the controller. The addresses would reside in odd numbered bytes while the data would reside in even numbered bytes. Please see Figures 2 through 4 for timing information. The parametrics listed in the DS2141A/43/51/53 data sheet are valid for non-multiplexed bus operation as well.

Non-Multiplexed Bus Configuration   Figure 1

Non-Multiplxed Intel Read Timing   Figure 2
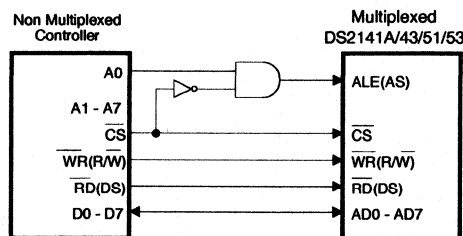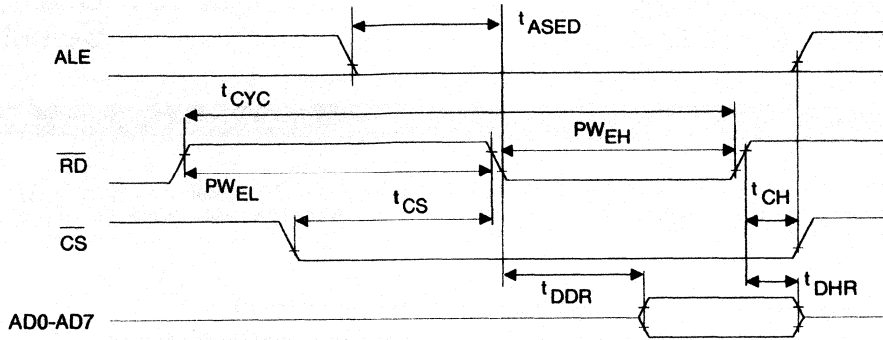
Non-Multiplexed Intel Write Timing   Figure 3
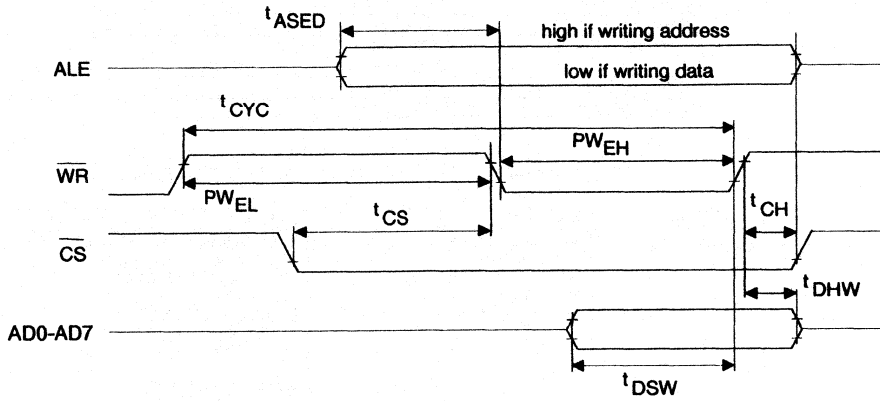
Non-Multiplexed Motorola Timing   Figure 4

DALLAS
SEMICONDUCTOR

DS2151, DS2153 Crystal Selection Guide
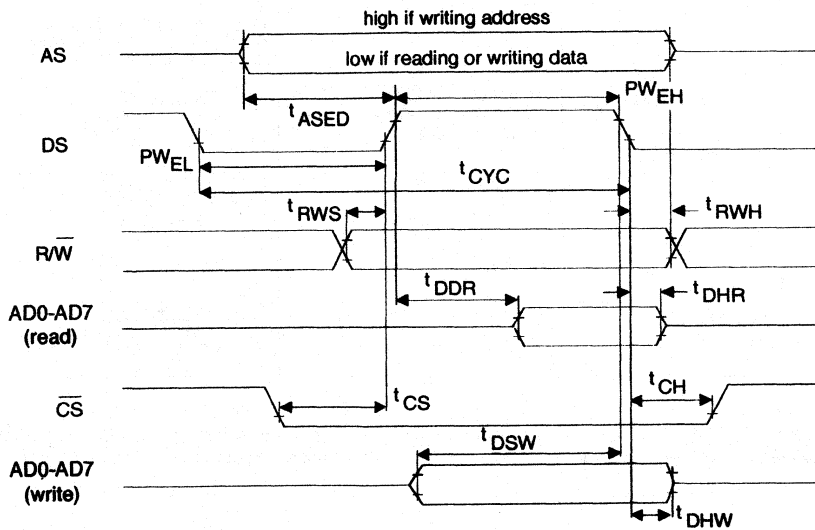
Specifications for selecting the appropriate crystal in a T1 or E1 application are given in Table 1. A list of manufacturers are in Table 2. The part numbers given are for leaded packages. Surface mount devices typically do not meet the pullability specification.

CRYSTAL SPECIFICATIONS Table 1

PARAMETER	SPECIFICATION
Parallel Resonant Frequency	6.176 (T1), 8.192 (E1)
Mode	Fundamental
Load capacitance	18pF to 20pF
Tolerance	± 50ppm
Pullability	CL= 10pF, Δf = + 175 to +250 ppm
	CL= 45pF, Δf = - 175 to -250 ppm
Effective Series Resistance	40 Ω max
Crystal Cut	AT

RECOMMENDED CRYSTAL MANUFACTURERS Table 2

MANUFACTURER	ADDRESS	PART #
M-TRON	100 DOUGLAS AV. P.O. BOX 630 YANKTON, SD 57078-0630 PH: (605) 665-9321 FAX: (605) 665-1709	COMMERCIAL TEMP RANGE (T1, 6.176MHz) 4575-032 (E1, 8.192MHz) 4575-031 INDUSTRIAL TEMP RANGE (T1, 6.176MHz) 4144-002 (E1, 8.192MHz) 4144-001 NOTE: DO NOT USE MP-1 PREFIX
RALTRON	2315 N.W. 107th AV. MIAMI, FL 33172 PH: (305) 593-6033 FAX: (305) 594-3973	COMMERCIAL TEMP RANGE (T1, 6.176MHz) A-6.176-18.5-DS (E1, 8.192MHz) A-8.192-18.5-DS INDUSTRIAL TEMP RANGE (T1, 6.176MHz) A-6.176-18.5-DSE (E1, 8.192MHz) A-8.192-18.5-DSE
SUNNY-EMI CO. SUNTRAC (DIST)	11925 VENTURA BLVD. STUDIO CITY, CA 91604 PH: (818) 509-8985	INDUSTRIAL TEMP RANGE (T1, 6.176MHz) SE 061-32 (E1, 8.192MHz) SE 081-30
JAN	2341 CRYSTAL DR. P.O. BOX 6017 FORT MYERS, FL 33906-6017 PH: (800) 526-9825 PH: (941) 936-2297 FAX: (941) 936-3750	COMMERCIAL TEMP RANGE (T1, 6.176MHz) JC8B18 (E1, 8.192MHz) JC8B18
Saronix	Strategic Marketing, Inc 624 W. University Suite 265 Denton, Tx 76201	Commercial Temp Range (T1, 6.176MHz) SRX5310(L) (E1, 8.192MHz) SRX5469(L)

Dallas Semiconductor has qualified a sample of each device from the above manufactures and have found that the device meets or exceeds our specifications. We do not conduct an on going qualification of these manufactures.

DALLAS
SEMICONDUCTOR

DS2151, DS2153 Transformer Selection Guide

This application note lists a number of vendors and model numbers for transformers that will work with the DS2151 and DS2153 Single-Chip Transceivers.

Vendor	Model Number	Turns Ratio	Temp. Range	Thru-Hole or SMD	Comments
Pulse	PE-65388(3)	1:1.15	0 to 70C	Thru-Hole	
Pulse	PE-65586	1:1.36	0 to 70C	Thru-Hole	
Pulse	PE-65770	1:1.15	-40 to 85C	Thru-Hole	
Pulse	PE-65340	1:1.36	-40 to 85C	Thru-Hole	
Pulse	PE-64936(2)	1:1	0 to 70C	Thru-Hole	
Pulse	PE-65778	1:1	-40C to 85C	Thru-Hole	
Pulse	PE-65565	1:1* & 1:1.15	0 to 70C	Thru-Hole	
Pulse	PE-64952	1:1* & 1:1.36	0 to 70C	Thru-Hole	
Pulse	PE-65567(4)	1:1* & 1:1.15	-40C to 85C	Thru-Hole	
Pulse	PE-65774	1:1* & 1:1.36	-40C to 85C	Thru-Hole	
Pulse	PE-65765	1:1* & 1:1.15	0 to 70C	SMD	
Pulse	PE-65762	1:1* & 1:1.36	0 to 70C	SMD	
Pulse	PE-65825	1:1* & 1:1.15	-40C to 85C	SMD	
Pulse	PE-65822	1:1* & 1:1.36	-40C to 85C	SMD	
Pulse	PE-68645	1:1.36	0 to 70C	Thru-Hole	0.25 ohm DCR
Pulse	PE-68644	1:1	0 to 70C	Thru-Hole	0.25 ohm DCR
Pulse	PE-65882	1:1.36	0 to 70C	Thru-Hole	3000Vrms isolation
Pulse	PE-65883	1:1	0 to 70C	Thru-Hole	3000Vrms isolation
Bel Fuse	A553-0013-07(4)	1:1* & 1:1.15	0 to 70C	Thru-Hole	
Bel Fuse	0553-0013-AC	1:1	0 to 70C	Thru-Hole	
Bel Fuse	0553-0013-BC(2)	1:1	0 to 70C	Thru-Hole	
Bel Fuse	0553-0013-RC(3)	1:1.15	0 to 70C	Thru-Hole	
Bel Fuse	0553-0013-7J	1:1* & 1:1.15	0 to 70C	Thru-Hole	
Bel Fuse	S553-0013-06	1:1* & 1:1.15	0 to 70C	SMD	
Midcom	671-5792	1:1	0 to 70C	Thru-Hole	
Midcom	671-6599	1:1.15	0 to 80C	Thru-Hole	use pins 2 & 3 on primary
Midcom	671-7734	1:1	0 to 70C	Thru-Hole	
Midcom	671-7735	1:1.15	0 to 70C	Thru-Hole	use pins 10 & 6 as primary
Midcom	671-7808	1:1.36	-40 to 85C	Thru-Hole	0.20 ohm DCR, 1.7A line side
Midcom	671-7809	1:1	-40 to 85C	Thru-Hole	0.20 ohm DCR, 1.7A line side
Midcom	671-7815	1:1.36	-40 to 85C	Thru-Hole	0.20 ohm DCR, 1.7A line side, Small footprint
Midcom	671-7816	1:1	-40 to 85C	Thru-Hole	0.20 ohm DCR, 1.7A line side, Small footprint
VITEC	16Z5936(2)	1:1	0 to 70C(5)	Thru-Hole	
VITEC	16Z5952(3)	1:1.15	0 to 70C(5)	Thru-Hole	
VITEC	16Z5937	1:1.36	0 to 70C	Thru-Hole	
VITEC	24Z4665(4)	1:1* & 1:1.15	0 to 70C(5)	Thru-Hole	
VITEC	24Z4652	1:1* & 1:1.36	0 to 70C	Thru-Hole	
VITEC	18Z4883	1:1.15	0 to 70C	Thru-Hole	3000Vrms isolation
VITEC	18Z4882	1:1.36	0 to 70C	Thru-Hole	3000Vrms isolation
VITEC	18Z4885	1:1*	0 to 70C	Thru-Hole	3000Vrms isolation

HALO	TD02-1205A	1:1	0 to 70C	Thru-Hole	
HALO	TD38-1505A	1:1.15	0 to 70C	Thru-Hole	
HALO	TD70-1205A	1:1.36	0 to 70C	Thru-Hole	
HALO	TD63-1505D	1:1* & 1:1.15	0 to 70C	Thru-Hole	
Schott	67130840	1:1	0 to 70C	Thru-Hole	
Schott	67129310	1:1.17	0 to 70C	Thru-Hole	
Schott	67134190	1:1.36	0 to 70C	Thru-Hole	
Premier	PM-IS32	1:1.37	0 to 70C	Thru-Hole	3000Vrms isolation
Premier	PM-IS33	1:1*	0 to 70C	Thru-Hole	3000Vrms isolation
YCL	PT163045	1:1 & 1:1.15	0 to 70C	SMD	

Notes:
1. * = actually a 1:2ct winding which can be made into a 1:1 winding
2. PE-64936 is compatible with 0553-0013-BC & 16Z5936
3. PE-65388 is compatible with 0553-0013-RC & 16Z5952
4. PE-65565 is compatible with A553-0013-07 & 24Z4665
5. VITEC offers extended temperature (-40C to +85C) models; add an "A" suffix

VENDOR INFORMATION

Pulse Engineering
P.O. Box 12236
San Diego, CA 92112
Phone (619) 674-8100
Fax (619) 674-8262

Bel Fuse Inc.
198 Van Vorst Street
Jersey City, NJ 07302
Phone (201) 432-0463
Fax   (201) 432-9542

MidCom
PO Box 1330
121 Airport Drive
Watertown, SD 57201
Phone (800) 643-2661
Fax   (605) 886-4486

VITEC
8530 W. Roosevelt Ave.
Visalia, CA 93291
Phone (209) 651-1535
Fax   (209) 651-1538

HALO
P.O. Box 5826
Redwood City, CA 94063
Phone (415) 969-7313
Fax   (415) 367-7158

Schott
1000 Parkers Lake Road
Wayzata, MN 55391
Phone (612) 475-1173
Fax   (612) 475-1786

Premier Magnetics
27111 Aliso Creek Rd
Suite 175
Aliso Viejo, CA 92656
Phone (714) 362-4211
Fax   (714) 362-4212

YCL
Taiwan, R.O.C.
Phone 886 - 2 - 7962878
Fax   886 - 2 - 7912635

DALLAS
SEMICONDUCTOR

DS2141A, DS2151 D4 Framing Applications

Transmit Framing

In D4 framing applications, the transmit side of the DS2141A and DS2151 can be set up to automatically insert both the Ft and the Fs framing pattern. To have the devices insert the Ft pattern, set the TCR1.6 bit to zero. To have the devices insert the Fs pattern, first set the CCR2.5 bit to one, then set TCR1.2 to zero, and finally program the TFDL register with a value of XX011100 (or 1C hex). The DS2141A/DS2151 will automatically pull the Fs pattern from the TFDL register. The user can corrupt the Fs pattern by changing the value in the TFDL register.

If the user wishes to have the DS2141A/DS2151 not alter the Ft pattern that is present in the data stream, then the TCR1.6 bit should be set to one. If the Fs bits should not be altered, then the TCR1.2 bit should be set to one and the TLINK pin should be tied to the TSER pin.

Synchronizing

In all applications, it is recommended that the cross coupling sync criteria (RCR1.3 = 1) be used. This feature makes the synchronizer in the DS2141A/DS2151 impervious to false framing patterns such as Digital Milliwatt. Under special circumstances, the user may find it necessary to relax the sync criteria and via the RCR1.3 bit this can be accomplished.

Yellow Alarm Options

In most applications, the user will use the normal definition for yellow alarm which is the setting of bit 2 of all channels to zero. However, the DS2141A/DS2151 does offer an alternative yellow alarm in the D4 framing mode which is to set the Fs bit position in frame 12 to one. This yellow alarm is sometimes referred to as the "Japanese Yellow Alarm" and it is controlled in the devices via the RCR2.2 bit.

DALLAS
SEMICONDUCTOR

<div align="right">

DS2141A, DS2151
Controlling the FDL

</div>

Note: Contact the factory for C code firmware that implements the FDL requirements as per ANSI T1.403 and AT&T TR54016.

Due to expected future changes in the requirements for the Facility Data Link (FDL), the DS2141A/DS2151 T1 Controller makes use of software in the host to support the communications that occur on the FDL. The use of host's software to help control the FDL allows designs to be easily modified as the standards progress. The DS2141A/DS2151 implements in hardware the more basic functions of controlling the FDL such as byte aligning to the FDL data stream. In current T1 networks, there are two common protocols that exist on the FDL in the Extended Superframe (ESF) framing mode. One is defined in the American National Standards Institute (ANSI) document T1.403-1989 and the other is defined in the AT&T publication TR54016. Depending on the carrier used, either one of these protocols (or both) may be required. This Application Note details how a host would interface with the DS2141A/DS2151 to extract information from the FDL and to insert data into the FDL. The user has the option on the DS2141A/DS2151 to either use its onboard RFDL and TFDL registers or to use the RLINK and TLINK pins to extract/insert data from/into the FDL data stream. This Application Note covers the usage of the RFDL and TFDL registers. The specifics of the two FDL protocols will be not be fully covered. Please refer to the two documents mentioned for more details. (Note: all references in this Application Note to specific data values are in Hexadecimal which is denoted by the value enclosed within < >.)

Receive FDL Control

The DS2141A/DS2151 contains an eight-bit register called the RFDL which is automatically loaded with data from the FDL. There is also a set of two match registers called RFDLM1 and RFDLM2 which can be used relieve the host from constantly polling the RFDL to see if any important data has arrived. If either of the values in the match registers corresponds to the current value in the RFDL, then the host will be alerted. Since most of the communications on the FDL follows the LAPD protocol, the DS2141A/DS2151 also contains a zero destuffer. The zero destuffer should always be enabled when using the RFDL to decode the FDL data stream. Also, when decoding the FDL, the DS2141A/DS2151 should have the ZBTSI (RCR2.6 = 0) and SLC-96 (CCR2.1 = 0) modes disabled.

Figure 1 displays a flowchart of how to decode data off of the FDL. The Match Registers are programmed to respond to the opening addresses of either a Performance Report Message (PRM) in the ANSI T1.403 protocol or a request message in the AT&T TR54016 protocol. The DS2141A/DS2151 will only match on the first byte following an opening flag of <7E> and it will automatically byte align to the incoming LAPD stream. The external controller is normally waiting for a match to occur. Once a match has occured, the external controller will wait for the RFDL register to fill and then read it. The Abort interrupt (SR2.1) takes on a new function once a match has occurred; it will now also report when the closing flag <7E> is received. Hence the external controller can monitor this flag during the extraction of data from the FDL to determine when all of the data has been captured.

If an abort flag is ever received (eight consecutive ones), then the SR2.1 bit will be set to a one. An Abort flag normally represents the beginning of an Unscheduled (bit-oriented) Message in the ANSI T1.403 protocol. The Unscheduled Message in ANSI T1.403 is a repeating 16-bit pattern of the form "...0CCCCCC011111111..." (the abort flag is transmitted first; the C's represent one of 64 possible code words). The Unscheduled Message will be reported in the RFDL as shown below.

	(msb)						(lsb)
RFDL	1	0	C	C	C	C	C	C

It is possible for an Unscheduled Message to interrupt a PRM or a request message, hence the external controller should monitor the Abort interrupt at all times.

Transmit FDL Control

The DS2141A/DS2151 contains an eight-bit register called the TFDL. Data that is to be transmitted onto the FDL will be loaded into the TFDL register, one byte at a time. Since most communications on the FDL follow the LAPD protocol, the DS2141A/DS2151 contains a zero stuffer to insure that the data between the opening and closing flags does not resemble either an opening or closing flag. The zero stuffer in the DS2141A/DS2151 should only be enabled for the data bytes within the opening and closing flags. If the TFDL register is not updated, it will retransmit the previous value contained in the register. When using the DS2141A/DS2151 to insert data into the FDL, the device should be set up to source the FDL from the TFDL register (TCR1.2 = 0), and both the ZBTSI (TCR2.5 = 0) and SLC-96 (CCR2.5 = 0) modes should be disabled. Figures 2 and 3 display two different flowcharts for inserting data into the FDL. Figure 2 details how to transmit an Unscheduled (bit-oriented) Message in an ANSI T1.403 environment. The specification calls for Unscheduled Messages to be transmitted at least 10 times (Priority Messages are to be sent for at least one second). Figure 3 details how to transmit either a Performance Report Message (PRM) according to T1.403 or a response message in the AT&T TR54016 protocol. According to ANSI T1.403, PRM's are to be transmitted once a second. The One Second Timer in Status Register 2 can be used to determine when a PRM should be sent. In TR54016, response message are only sent once a request has been received. When the FDL is idle and not sending any PRM's, the flag value of <7E> is to be transmitted.

Receive FDL Decoding   Figure 1

```
        +-----------------------------------------------+
        | Write <38> to the RFDLM1 Register (T1.403)    |
        | Write <C3> to the RFDLM2 Register (TR54016)   |
        +-----------------------------------------------+
                              |
                              v
        +-----------------------------------------------+
        | Wait for the FDL Match to Occur via the SR2.2 |
        | bit or for an Abort flag to be received via   |
        | the SR2.1 bit                                 |
        | (ignore the RFDL Full Status bit)             |
        +-----------------------------------------------+
                              |
                              v
        +-----------------------------------------------+
        | Wait for the RFDL Register to Fill via the    |
        | SR2.4 bit or for an Abort flag to be received |
        | via the SR2.1 bit                             |
        | (ignore the FDL Match Status bit )            |
        +-----------------------------------------------+
                              |
                              v
  If the Abort flag is set   +-------------------------------+
  (either signals end of data| Read Data Byte from the RFDL  |
  or that an Abort signal has | Register                      |
  been received)             | (must be done within 2ms      |
                             | after the setting of SR2.4)   |
                             +-------------------------------+
```

Status Bit(s) Used:
SR2.1 - Receive FDL Abort
SR2.2 - Receive FDL Match Occurence
SR2.4 - Receive FDL Buffer Full
Register(s) Used:
RFDL - Receive FDL Register
RFDLM1 - Receive FDL Match Register 1
RFDLM2 - Receive FDL Match Register 2

Transmit T1.403 Unscheduled Message Figure 2

```
┌─────────────────────────────────────────────┐
│       Write <FF> to the TFDL Register        │
│      Disable Zero Stuffer (CCR2.4 = 0)       │
└─────────────────────────────────────────────┘
                      │
                      ▼
┌─────────────────────────────────────────────┐
│        Wait for the TFDL Register to         │
│           Empty via the SR2.3 bit            │
└─────────────────────────────────────────────┘
                      │
                      ▼
┌─────────────────────────────────────────────┐
│     Write <14> to the TFDL Register    (1)   │
│ (must be done within 2ms after the setting of SR2.3) │
└─────────────────────────────────────────────┘
                      │
                      ▼
┌─────────────────────────────────────────────┐
│        Wait for the TFDL Register to         │
│           Empty via the SR2.3 bit            │
└─────────────────────────────────────────────┘
                      │
                      ▼
┌─────────────────────────────────────────────┐
│       Write <FF> to the TFDL Register        │
│ (must be done within 2ms after the setting of SR2.3) │
└─────────────────────────────────────────────┘
                      │
                      ▼
┌─────────────────────────────────────────────┐
│          Write <7E> to the     (2)           │
│              TFDL Register                    │
└─────────────────────────────────────────────┘
```

Loop at least 10 times
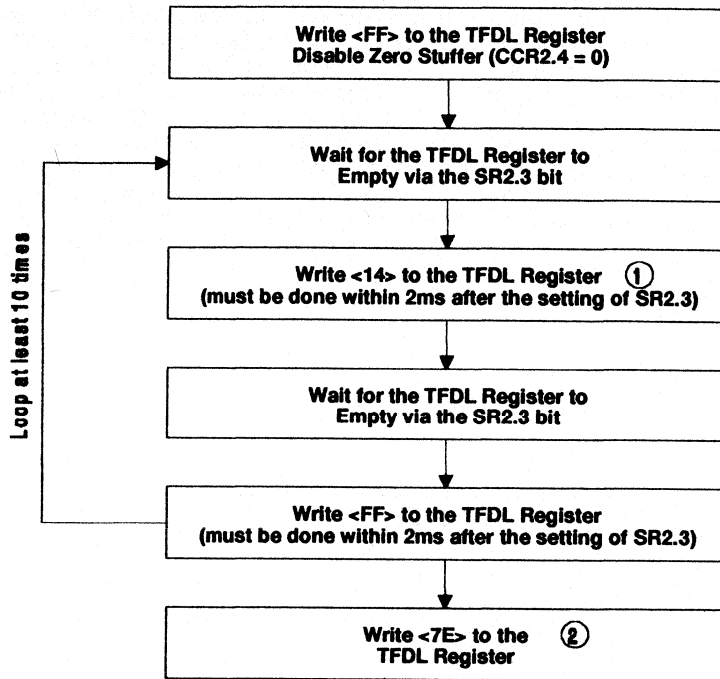
Notes(s):
1. Transmitting the Unscheduled Message for Payload Loopback Activate is shown.
2. This <7E> will be the FDL idle code.

Status Bit(s) Used:
SR2.3 - Transmit FDL Buffer Empty
Register(s) Used:
TFDL - Transmit FDL Register
Control Bit(s) Used:
CR2.4 - Transmit Zero Stuffer Enable

Transmit FDL Coding Figure 3

```
┌─────────────────────────────────────────────┐
│      Write <7E> to the TFDL Register         │
│      Disable Zero Stuffer (CCR2.4 = 0)       │
└─────────────────────────────────────────────┘
                      │
                      ▼
┌─────────────────────────────────────────────┐
│            Wait at least 54ms                │
│      (for >27 flags to be transmitted)       │
└─────────────────────────────────────────────┘
                      │
                      ▼
┌─────────────────────────────────────────────┐
│         Wait for the TFDL Register to        │
│            Empty via the SR2.3 bit           │
└─────────────────────────────────────────────┘
                      │
                      ▼
┌─────────────────────────────────────────────┐
│     Write Data Byte to the TFDL Register;    │
│      Enable Zero Stuffer (CCR2.4 = 1)    ①   │
│ (must be done within 2ms after the setting of SR2.3) │
└─────────────────────────────────────────────┘
                      │
                      ▼
┌─────────────────────────────────────────────┐
│         Wait for the TFDL Register to        │
│            Empty via the SR2.3 bit           │
└─────────────────────────────────────────────┘
                      │
                      ▼
┌─────────────────────────────────────────────┐
│    Write <7E> to the TFDL Register;   ②      │
│      Disable Zero Stuffer (CCR2.4 = 0)       │
│ (must be done within 2ms after the setting of SR2.3) │
└─────────────────────────────────────────────┘
```

Loop Until all the Data has been Sent

Note(s):
1. The CCR2.4 bit only needs to be set when the first data byte is written.
2. This <7E> will be the closing flag.

Status Bit(s) Used:
SR2.3 - Transmit FDL Buffer Empty
Register(s) Used:
TFDL - Transmit FDL Register
Control Bit(s) Used:
CCR2.4 - Transmit Zero Stuffer Enable

DALLAS
SEMICONDUCTOR

DS2151 and DS2153 Initialization

GENERAL INITIALIZATION FOR THE DS2151 AND DS2153

After power-up, when supplies and clocks have stabilized, internal registers must be initialized. It is a good idea to clear, set to 00H, **ALL** R/W registers. Certain registers have bits which control special test modes and features which can provide confusing indications. These registers are;

DS2151: **TEST, TCR2, LICR**

DS2153: **TEST1, TEST2, LICR**

Depending on the interrupt structure implemented, it may be a good idea to clear **IMR1** and **IMR2** first. Prior to clearing these two registers however, without an external interrupt disable, there is no way to protect against spurious interrupt signals prior to initialization.

Once the registers have be initialized and set up with the transceiver's mode of operation. , the Line Interface Reset bit should be set high then low. Also, if the elastic store is enabled, the ESR bit should be set then cleared.

SPECIAL INITIALIZATION FOR DS2153

TCLK (Transmit Clock) must be present for proper port initialization. Network signals cannot be guaranteed to replace a missing TCLK during initialization. The following sequence should be used to initialize the **DS2153** in which TCLK is not present or TCLK is derived from RCLK (a loop timed system).

1. WRITE 04H TO **CCR2** REGISTER (SET **LOTCMC** BIT)
2. WAIT 10ms MINIMUM
3. WRITE 00H TO ALL OTHER R/W REGISTERS
4. WRITE DEVICE CONFIGURATION DATA
5. SET **LIRST** BIT IN **CCR3** REGISTER HIGH
6. WAIT FOR SYSCLK TO STABILIZE (IF ELASTIC STORE(S) ENABLED)
7. SET **ESR** BIT IN **CCR3** REGISTER HIGH (IF ELASTIC STORE ENABLED)
8. CLEAR **LIRST** AND **ESR** BITS.

NOTES: In Loop Timed configurations or when TCLK is not guaranteed to always be present it is recommended that **LOTCMC** in **CCR2** be enabled (= 1).

If the SYSCLK pin is tied high, registers can be initialized (written to), but not read.

DALLAS
SEMICONDUCTOR

DS2141, DS2143, DS21Q41, DS21Q43 Initialization

GENERAL INITIALIZATION FOR DS21Q41, DS21Q43

After power-up, when supplies and clocks have stabilized, internal registers must be initialized. It is a good idea to clear, set to 00H, **ALL** R/W registers. Certain registers have bits which control special test modes and features which can provide confusing indications. There are some registers containing functions that if not cleared, can cause unexpected conditions on device pins. These registers are;

> DS21Q41:   **TEST, TCR2**
>
> DS21Q43:   **TEST1, TEST2**

Depending on the interrupt structure implemented, it may be a good idea to clear **IMR1** and **IMR2** first. Prior to clearing these two registers however, without an external interrupt disable, there is no way to protect against spurious interrupt signals prior to initialization.

Once the registers have be initialized and set up with the framer's mode of operation, if the transmit or receive elastic store is enabled, the **ESR** bit in **CCR3** should be set then cleared.

SPECIAL INITIALIZATION FOR DS21Q43

TCLK (Transmit Clock) must be present for proper port operation. This clock can be sourced externally from the TCLK pin or internally from RCLK via the LOSS of TRANSMIT CLOCK mux. This mux is enableb by setting CCR2.2 = 1. The following sequence should be used to initialize the **DS21Q43** in which TCLK is not present or TCLK is derived from RCLK (a loop timed system).

1. SET **LOTCMC** BIT IN **CCR2** REGISTER
2. WAIT 10ms MINIMUM
3. WRITE 00H TO ALL OTHER R/W REGISTERS
4. WRITE DEVICE CONFIGURATION DATA
5. WAIT FOR TSYSCLK AND/OR RSYSCLK TO STABILIZE (IF ELASTIC STORES ENABLED)
6. SET **ESR** BIT IN **CCR3** REGISTER HIGH (IF ELASTIC STORES ENABLED)
7. CLEAR **ESR** IN **CCR3** BIT.

NOTES: In Loop Timed configurations or when TCLK is not guaranteed to always be present it is recommended that **LOTCMC** in **CCR2** be enabled (= 1).

If the RSYSCLK pin (DS21Q41, DS21Q43) is tied high, registers can be initialized (written to), but not read.

DALLAS
SEMICONDUCTOR

DS2151 and DS2153 Interfacing

The Circuit in Figure 1 shows how to interface the DS2172 BERT to the DS2151 or DS2153. The receive side of the DS2172 can be connected directly to the receive side of the 51 or 53. RCHBLK is connected to the Receive Disable (RDIS) pin. RCHBLK and TCHBLK outputs from the 51 and 53 are used to determine the time slot or band in which the BERT is to transmit or receive data. On the transmit side, normal traffic is multiplexed with BERT data. Figure 1 shows the DS2151 or DS2153 with the elastic stores disabled. See Table 1. for clock connections with elastic stores enabled.

Figure 1. DS2151/53 to DS2172 Interface
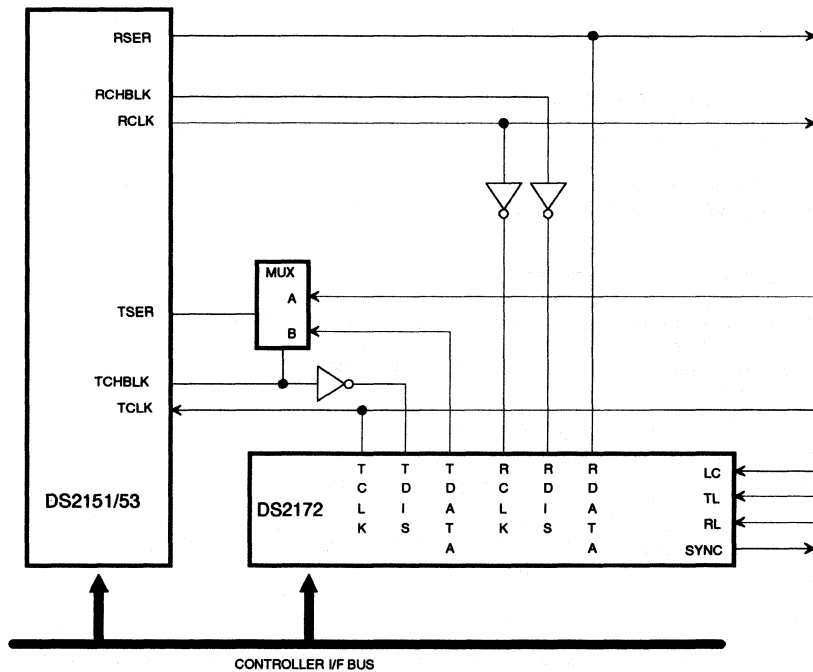
TABLE 1.

ELASTIC STORES (51/53)	TCLK SOURCE (72)	RCLK SOURCE (72)
NONE	TCLK (51/53)	RCLK (51/53)
RECEIVE ONLY	TCLK (51/53)	SYSCLK (51/53)
RECEIVE AND TRANSMIT	SYSCLK (51/53)	SYSCLK (51/53)

DALLAS
SEMICONDUCTOR

DS2151 Special Modes

Special Mode #1:　　**3 State Outputs**

When the SYSCLK pin is held high for 256 RCLK periods, all output pins, including the parallel port are 3-stated.

Special Mode #2:　　**Allow Access Between Framer and LIU on Transmit Side**

Writing 40hex to the TEST resgister will invoke this mode. When enabled, the internal TLINK node is tied to the TSER pin. This access is useful in applications where it is desired to purposely corrupt the data stream before ti is to be transmitted out onto the T1 line via TTIP and TRING pins. There is an exact delay from the TSER pin to the bipolar outputs fo the formatter so the user has the ability to know when the needed bits appear at RCLK and RLINK pins. This delay is **10 TCLK** periods.

Special Mode #3:　　**Tap the Transmit & Recieve Bipolar Data Stream**

Writing 60hex **(REV B ONLY)** to the TEST register will invoke this mode. When enabled, the internal TLINK node is tied to the TSER pin.

Special Mode #4:　　**Asynchronous, Elastic Stores Operation**

Writing 07hex **(REV B ONLY)** to the TEST Register will invoke this mode.

Special Mode #5:　　**LIU Bypass**

Writing 60hex **(REV A)** , 20hex **(REV B)** to the TEST register will invoke this mode. When enabled, the internal TLINK node is tied to the TSER pin.

MODE 2: DS2151 Special Mode to Allow Access to Transmit Bipolar Data Stream

JITTER ATTEN. REMOTE LOOPBACK FRAMER LOOPBACK Receive Framer Transmit Framer

SIMPLIFIED PORTION OF DS2151
() REASSIGNED PIN

| TPOS | TNEG | TPOS | TNEG |
| (TLINK) | (TLCLK) | (RLINK) | (RLCLK) |
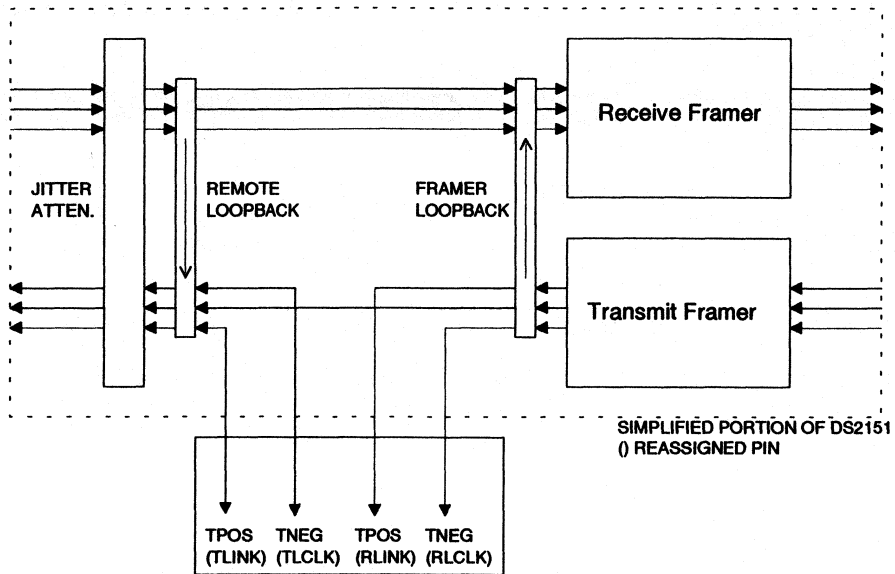
MODE 3: DS2151 Special Mode to Tap the Transmit & Receive Bipolar Data Streams

| RPOS | RNEG |
| (TLINK) | (TLCLK) |

JITTER ATTEN. REMOTE LOOPBACK FRAMER LOOPBACK Receive Framer Transmit Framer

SIMPLIFIED PORTION OF DS2151
() REASSIGNED PIN

| TPOS | TNEG |
| (RLINK) | (RLCLK) |

MODE 4: DS2151 Special Mode to Allow Asynchronous Backplane Clocks

RECEIVE FRAMER

PAYLOAD
LOOPBACK

TRANSMIT FRAMER

Xmit
Elastic
Store

CLOCK
(TCHCLK)

SYNC
(TCHBLK)

TSER

SIMPLIFIED PORTION OF DS2151
() REASSIGNED PIN
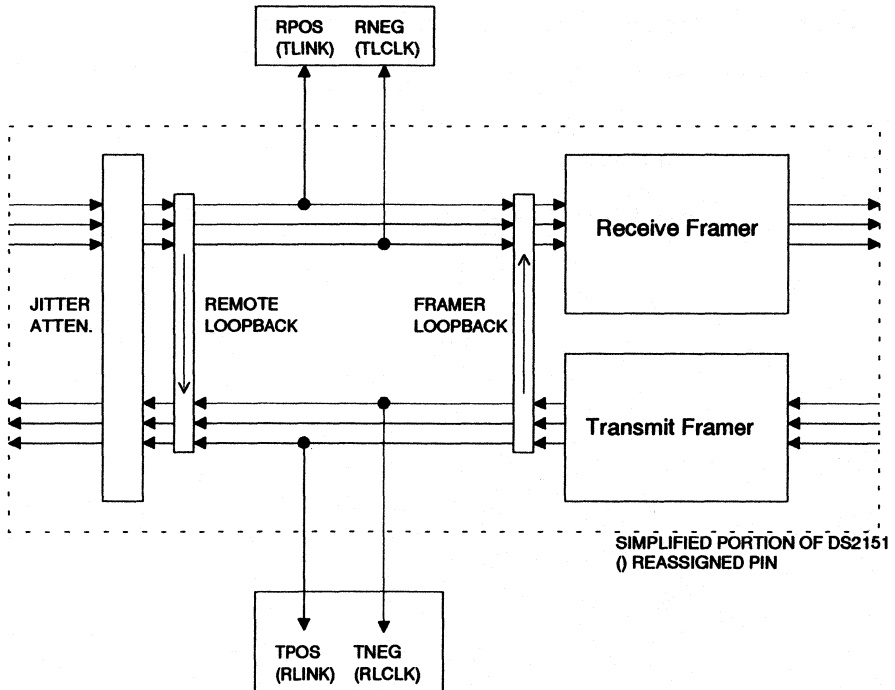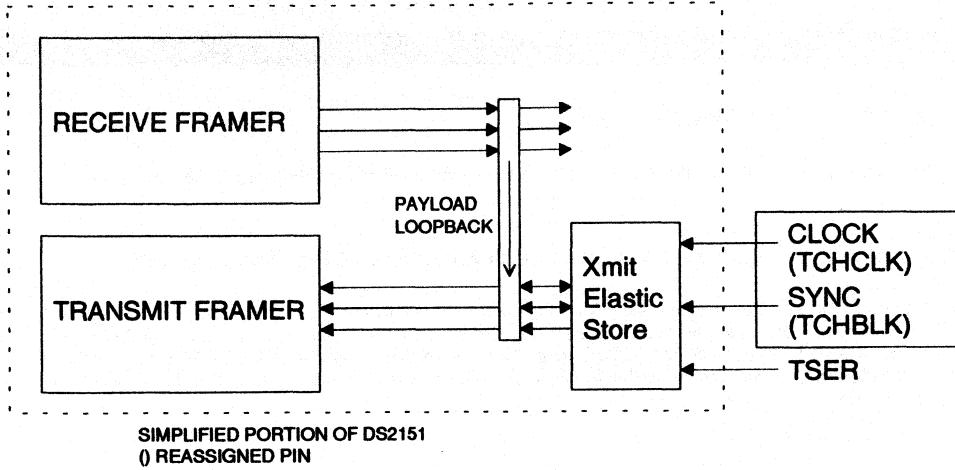
MODE 5: DS2151 Special Mode to Bypass the LIU

RPOS   RCLK   RNEG
(ACLKI)   (TLINK)   (TLCLK)

SIMPLIFIED PORTION OF DS2151
() REASSIGNED PIN

RECEIVE FRAMER

LOCAL
LOOPBACK

JITTER
ATTEN.

FRAMER
LOOPBACK

REMOTE
LOOPBACK

TRANSMIT FRAMER

TPOS   TNEG
(RLINK)   (RLCLK)

DALLAS
SEMICONDUCTOR
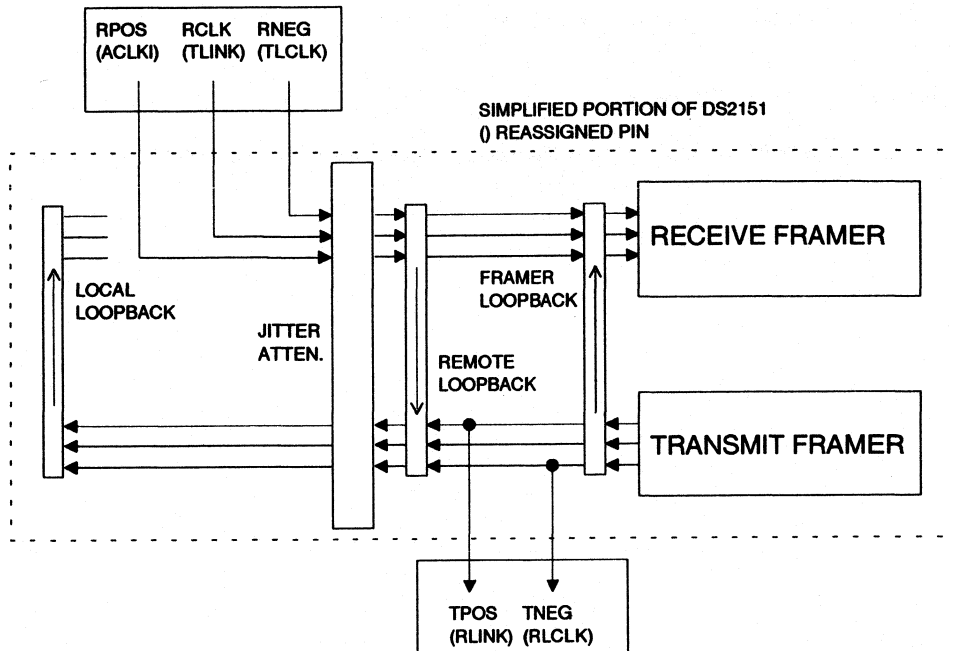
DS2153 Special Modes

Special Mode #1:   **3 State Outputs**

When the SYSCLK pin is held high for 256 RCLK periods, all output pins, including the parallel port are 3-stated.

Special Mode #2:   **Allow Access Between Framer and LIU on Transmit Side**

Setting the 40hex bit in the TEST2 resgister will invoke this mode. When enabled, the internal TLINK node is tied to the TSER pin. This access is useful in applications where it is desired to purposely corrupt the data stream before ti is to be transmitted out onto the E1 line via TTIP and TRING pins. There is an exact delay from the TSER pin to the bipolar outputs fo the formatter so the user has the ability to know when the needed bits appear at RCLK and RLINK pins. This delay is **5 TCLK** periods.

Special Mode #3:   **Tap the Transmit & Recieve Bipolar Data Stream**

Setting the 60hex bit in the TEST2 register will invoke thsi mode.   When enabled, the internal TLINK node is tied to the TSER pin.

Special Mode #4:   **Allow Separate Transmit and Recieve Backplane Clocks**

Setting the CCR3.2 bit in the Common Control Register 3 will invoke this mode.

Special Mode #5:   **LIU Bypass**

Setting the 20hex bit in the TEST2 register will invoke this mode.   When enabled, the internal TLINK node is tied to the TSER pin.

MODE 2: DS2153 Special Mode to Allow Access to Transmit Bipolar Data Stream
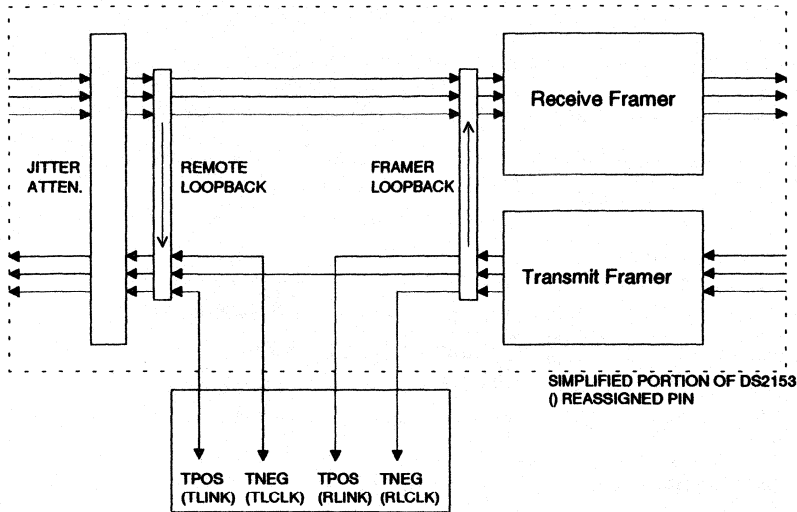
MODE 3: DS2153 Special Mode to Tap the Transmit & Receive Bipolar Data Streams

MODE 4: DS2153 Special Mode to Allow Asynchronous Backplane Clocks

RECEIVE FRAMER

PAYLOAD
LOOPBACK

TRANSMIT FRAMER

Xmit
Elastic
Store

CLOCK
(TCHCLK)

SYNC
(TCHBLK)

TSER

SIMPLIFIED PORTION OF DS2153
() REASSIGNED PIN
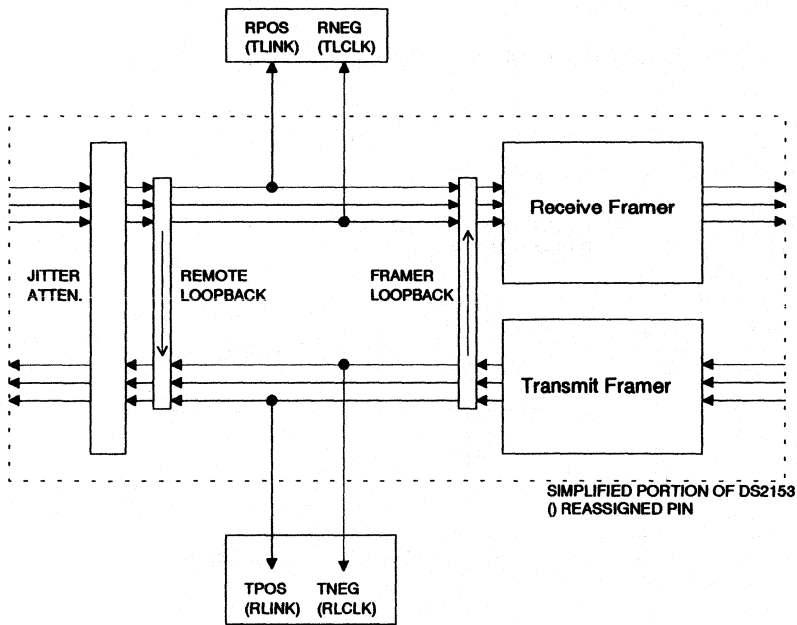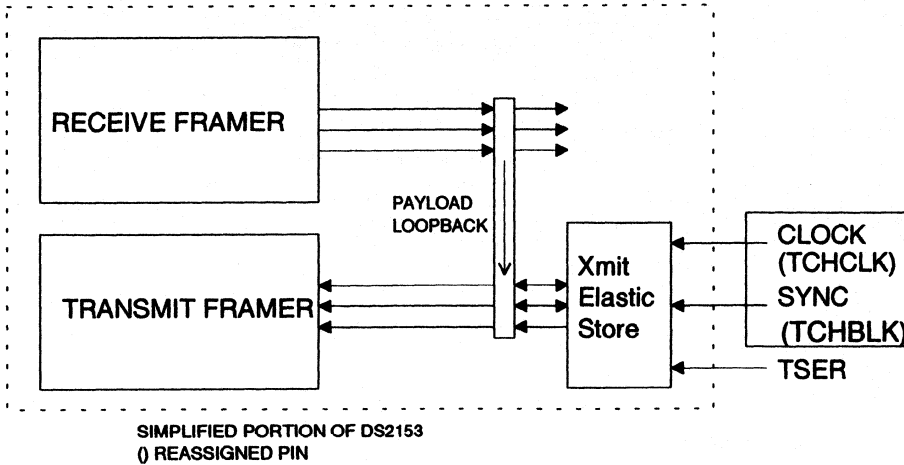
MODE 5: DS2153 Special Mode to Bypass the LIU

RPOS
(ACLKI)

RCLK
(TLINK)

RNEG
(TLCLK)

SIMPLIFIED PORTION OF DS2153
() REASSIGNED PIN

RECEIVE FRAMER

LOCAL
LOOPBACK

FRAMER
LOOPBACK

JITTER
ATTEN.

REMOTE
LOOPBACK

TRANSMIT FRAMER

TPOS
(RLINK)

TNEG
(RLCLK)

MICROCONTROLLERS

DALLAS
SEMICONDUCTOR

Application Note 56
The DS80C320 as a Drop–In
Replacement for the 8032
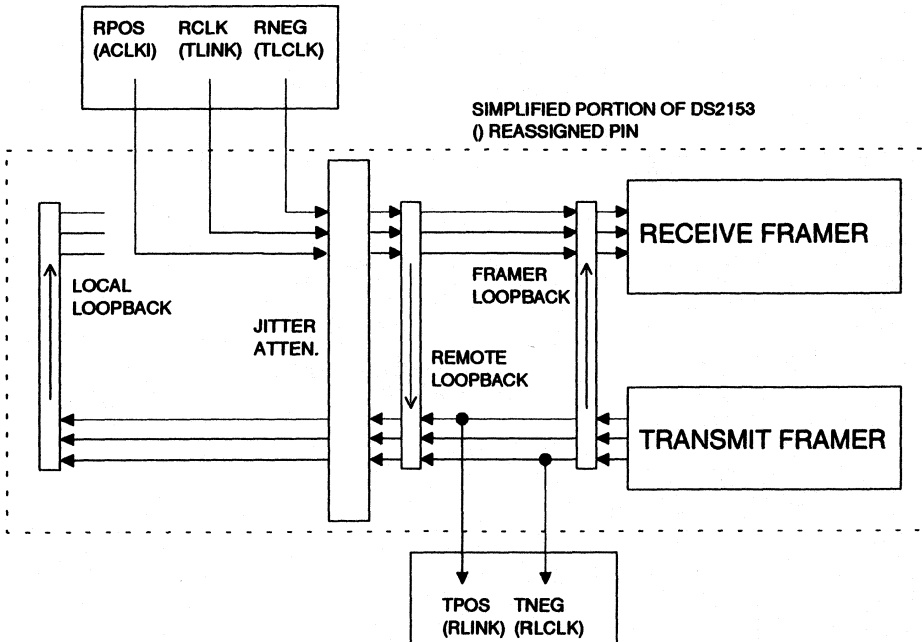
INTRODUCTION

The DS80C320 High–Speed Micro is another member of Dallas Semiconductor's 8051 instruction set compatible family of microprocessors. It was designed with the same pinout and basic resources as a conventional 8032, but has significantly enhanced performance capabilities and a number of additional resources. Since the instruction set and pinout are the same, many situations allow it to be used as a drop–in replacement. When doing so however, there are some issues that must be taken into account. This application note discusses those issues.

PROCESSOR SPEED

While the DS80C320 is 100 percent compatible with the 8051 instruction set, the execution of the instructions has been streamlined for increased performance. A single byte instruction that previously required 12 clocks to complete now executes in 4 clocks. In addition, the DS80C320 can accept clocks up to 25 MHz where in some versions of the 8032 the maximum was 12 MHz. Because of this higher performance, there are issues relating to processor speed that must be considered when evaluating the DS80C320 as a drop–in replacement for the 8032.

Memory Interface[1]

Since the basic instruction execution time has been streamlined in the DS80C320, the time available to transfer data to and from memory has also been reduced. This means that for the same frequency crystal, there is less time available for memory access. A simple example illustrates this point. The data sheet for the 8032[2] stipulates that, when using a 12 MHz crystal, the program memory must have an address access time of 302 ns or less (neglecting any address latch overhead). A DS80C320 also using a 12 MHz crystal requires a memory with an address access time of 230 ns of less. While this is not a tremendous difference, it is

something that must be considered, and may be important in some systems.

Software Loops

The other issue having to do with speed considerations is the use of software timing. It is frequently the case that software writers will use the presumed constant execution speed of a processor as a real time reference. Often a tight loop that requires a known number of clocks to execute will be used for generating delays. Since the DS80C320 executes instructions much more quickly than the standard 8032, these previously designed timing loops will no longer produce the originally intended results. While using software timing loops is generally accepted as undesirable software design, in practice they are used rather frequently in embedded applications. The DS80C320 was designed so that the internal timers default to a condition where they behave exactly as the timers in the 8032. If application code is written to make use of these timers rather than software delays, the code will run as originally intended.

POWER–ON RESET

The DS80C320 incorporates circuitry to generate its own power–on reset function. While the RST pin may still be connected to an external reset generating circuit, this on–board feature is provided as a convenience to new designs. The fact that the processor has its own reset function is a benefit in most cases, however, there are situations where the on–board reset is not exactly what the user wants. One could conceive of situations where the reset may not be at exactly the desired voltage level or last for exactly the desired duration. One example of this is could be the case where battery backed RAM is used for storage. If the RAM contains its own voltage detection circuitry and does not become unprotected at the same voltage as the DS80C320 leaves reset (4.0 volts), then the processor could be accessing protected RAM. While these cases are not

---

1. Details on selecting the correct speed memory devices for the DS80C320 may be found in Dallas Semiconductor's Application Note 57 entitled "DS80C320 Memory Interface Timing."

2. Data for the 8032 from the Intel "8–Bit Embedded Controllers" data book dated 1991.

102594 1/2

common, they remain something to consider for each specific application.

POWER CONSUMPTION

In addition to being a higher performance device, the DS80C320 is also a lower power device than the 8032 when equivalent work is considered. All CMOS parts exhibit the property that they consume more power as their speed goes up. Since the DS80C320 is a higher speed part, it will consume more power for a given crystal frequency. However, if an equivalent amount of work is considered, it consumes slightly less power than a conventional 8032. This difference in power consumption is probably only important for battery powered applications, in which case stop mode power is likely to be more important.

DALLAS
SEMICONDUCTOR

Application Note 57
DS80C320 Memory Interface Timing

INTRODUCTION

Dallas Semiconductor's DS80C320 processor provides extensive new application opportunities due to its increased throughput. The increased speed, however, also requires attention to the timing requirements of the memory that interfaces with the processor. This application note identifies the critical timing paths associated with the memory interface and identifies memory speeds required for various CPU crystal frequencies.

A typical configuration for a DS80C320 processor system is shown in Figure 1. A 32K x 8 EPROM is used to hold program information, and a 8K x 8 Static RAM is used for data storage. The Least Significant Byte (LSB) of the memories' address is time multiplexed with data on processor pins AD7 through AD0. The signal ALE from the processor goes high before a new address is placed on the bus, and goes low before it is removed. In Figure 1, the action of ALE going low is used to latch the address into a 74HCT373 8–bit transparent latch. The 373 then provides its latched address output to the memories while the AD7 through AD0 CPU bus carries data. The MSB of the address is not multiplexed, and is available on port pins P2.7 through P2.0 (A15–A9).

A TYPICAL DS80C320 SYSTEM CONFIGURATION Figure 1
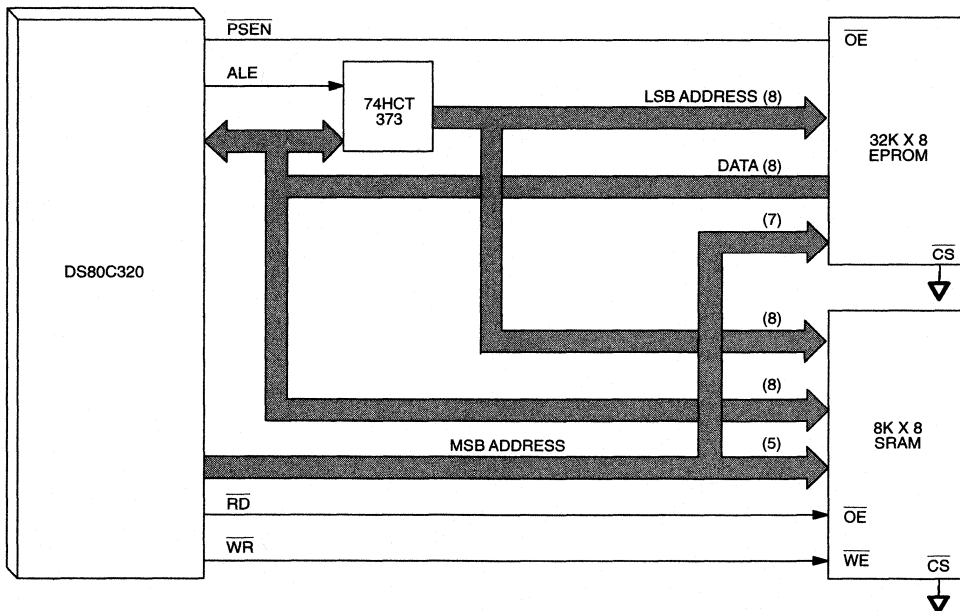
PROGRAM MEMORY

Some of the signals used in interfacing the DS80C320 with EPROM Program memory are shown in Figure 2. As can be seen, timing relationships for two latch technologies (HCT, and F) are shown. The technology selected for this latch is critical to the memory selection. The 74HCT373 has a worst case propagation delay from input to output (D to Q) of 44 ns[1] while the 74F373's is 8 ns[2]. This results in significantly different memory address access timing requirements depending on the family used. Examining timing parameters from the DS80C320 data sheet reveals that an instruction must be ready to read into the processor within 60 ns (parameter $t_{AVIV1}=3t_{CLCL}-27$)[3], assuming a 33 MHz clock[4]. If the 44 ns propagation delay through the HCT latch is subtracted from this, you arrive at a required address access time of 26 ns.

PROGRAM MEMORY INTERFACE TIMING Figure 2
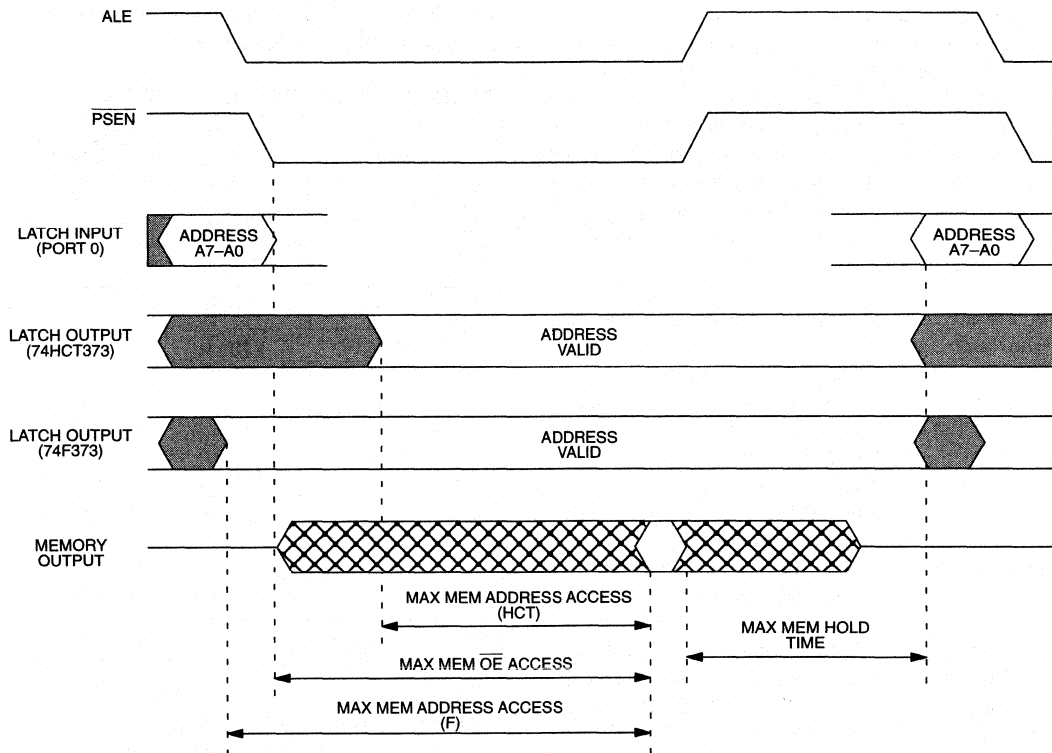
While EPROM devices with access times of 26 ns or less may be available, they are likely to be expensive. A simple and more cost effective approach to solving this timing constraint is to use a faster latch technology; an 74F373 for instance. Using the same analysis as above, if the propagation delay of the F373 latch, 8 ns, is subtracted from the t_{AVIV1} parameter (64 ns) you arrive at an address access requirement of 56 ns. This is much easier to realize than 26 ns.

1. HCT timing information from the 1988 Texas Instruments "High Speed CMOS Logic" data book.
2. F timing information from the Signetics "Fast Data Manual 1986" data book.
3. For details on the equations presented in this document, refer to the DS80C320 data sheet.
4. t_{CLCL} is the period of the crystal frequency, and is equal to 30.3 ns for a 33 MHz crystal.

There is another timing constraint that suggests the use of an "F" type part in faster applications. On a 74HCT373 latch, the minimum required hold time of the input after the latch enable (ALE) goes low may be as much as 13 ns. The latch's input, the address out of the processor, is held until it is driven by the memory's output. This output is enabled by $\overline{PSEN}$. Again referring to the data sheet, it can be seen that $\overline{PSEN}$ may occur as quickly as 0.5 ns after ALE falls (parameter t_{LLPL}). If the memory's output begins to drive the bus immediately after $\overline{PSEN}$ enables it, then there may only be 0.5 ns hold time. That obviously violates the latch's requirement. While it is reasonable to assume that there will be some delay before the memory's output will drive the bus, this is not a specified (or tested) parameter. Therefore a conservative estimate of 5 ns will be assumed. From above numbers and the equation for the t_{LLPL} parameter ($0.25t_{CLCL}-7$), it can be calculated that clock frequencies of 19.23 MHz and below will allow sufficient hold time to satisfy the 74HCT373 latch's requirement. Note that the hold time for a 74F373 latch is 3 ns which is met under all conditions.

From the above analysis, you could choose 19.32 MHz as the latch technology switchover point, however, there is a reason why this was not done. Referring back to the address access equation ($3t_{CLCL}-24$), it can be calculated that if a 74HCT373 latch is used for frequencies higher than 16.31 MHz, then a 90 ns or faster memory device is required. Therefore as a trade–off between latch speed and EPROM speed, the 74F373 latch (less expensive than a fast EPROM) is recommended for crystal frequencies greater than or equal to 16.31 MHz, and the 74HCT373 for lower frequencies.

As shown in Figure 1, $\overline{PSEN}$ enables the EPROM's output, so the timing for this signal must also be considered when choosing a device. The DS80C320 data sheet specifies the time from $\overline{PSEN}$ low to a valid instruction in must be no more than 70 ns (parameter t_{PLIV}). This is therefore the maximum allowed access time from the memory's $\overline{OE}$ pin. In summary, the two timing requirements for the selected EPROM are that the address access time must be less than 92 ns and the $\overline{OE}$ access time must be less than 70 ns. When looking at EPROM data sheets, it can be seen that common access time combinations (address access, OE access) are 55,35; 70,40; 90,40; 120,50; 150,65; 200,75; and 250,100 ns.

The 55, 35 device meets both timing requirements for a 33 MHz clock on the DS80C320 and is the recommended selection.

Table 1 shows the slowest EPROM memory speeds recommended for various processor clock frequencies. If for some reason it is desirable to use devices faster than those recommended, this is possible. For this document, the memory speeds available (maximum access times from address and $\overline{OE}$ respectively) are assumed to be those indicated above, however, any combination that meets the two requirements may be used. In Table 1, the memory speeds shown in bold are the recommended configurations.

RECOMMENDED EPROM SPEEDS Table 1

CLOCK (MHZ)	MEMORY SPEED 74F373	MEMORY SPEED 74HCT373
33.0	**55 ns**	–
25.0	**90 ns**	–
20.0	**120 ns**	70 ns
18.432	**120 ns**	90 ns
16.0	150 ns	**120 ns**
14.746	150 ns	**120 ns**
14.318	150 ns	**120 ns**
12.0	200 ns	**150 ns**
11.059	200 ns	**200 ns**
7.37	250 ns	**250 ns**
1.8432 and below	250 ns	**250 ns**

It should be noted that it is often possible to reduce the power consumption of EPROMs by keeping the $\overline{OE}$ pin held active and controlling the device with the $\overline{CS}$ pin. When doing this, however, the access time from $\overline{CS}$ must be more closely considered. The $\overline{CS}$ access time is often nearly the same value as the address access time (i.e., much slower than $\overline{OE}$ access). If power consumption is the prime system consideration, a faster device may be selected, and the $\overline{CS}$ used to select the chip.

DATA MEMORY

Choosing a data memory (RAM) device to interface to the DS80C320 is much easier than choosing an EPROM device because of the flexibility designed into the processor. The DS80C320 provides a unique feature that allows the application software to adjust the speed at which data memory is accessed. The processor is capable of performing a MOVX instruction in as little as two instruction cycles (eight oscillator clocks). However, this value can be "stretched" as needed so that both fast memory and slow memory or peripherals can be accessed with no glue logic. On power up, the DS80C320 defaults to a stretch value of one resulting in a three cycle MOVX instruction. This default condition is a convenience to existing designs that may not have fast RAM in place. For the user who needs maximum performance, a stretch value of zero may be selected by software, resulting in a two machine cycle MOVX instruction. Even in high speed systems, it may not be

necessary or desirable to perform data memory accesses at full speed. Additionally, there are a variety of memory mapped peripherals such as LCD displays or UARTs that are not fast enough to keep up with the full speed DS80C320. This flexibility allows the user to trade some performance for slower data RAMs if so desired.

For maximum performance, i.e., with a two machine cycle data memory access programmed into the processor, the fetch of a MOVX instruction takes one machine cycle leaving one machine cycle for the memory read or write. For the analysis of the data memory's timing requirements that follows, it will be assumed that the recommendations of Table 1 have been followed. This implies that a 74F373 latch is used for clock frequencies above 16.31 MHz. A diagram of a single cycle data memory read is shown in Figure 3 and a single cycle data memory write is shown in Figure 4.

DATA MEMORY READ Figure 3
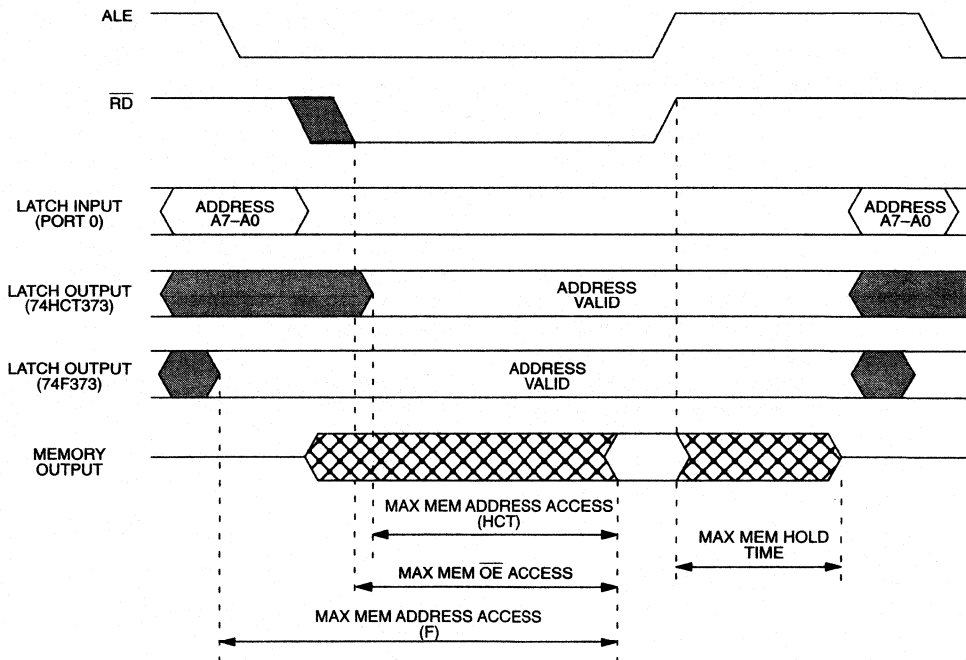
DATA MEMORY WRITE Figure 4
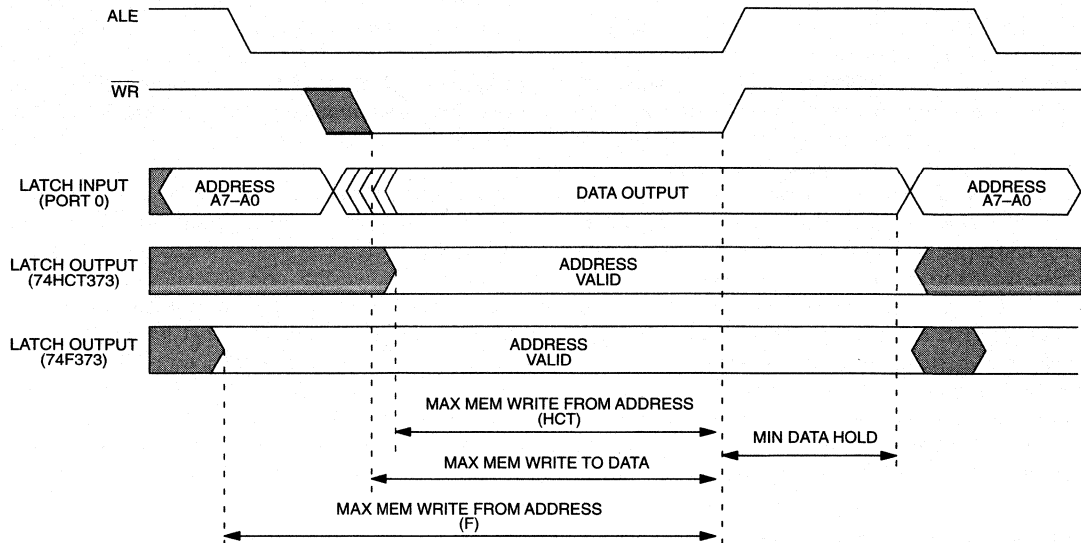
Note that the term t_{MCS} is used in the data sheet and in the equations that follow. It is a term that represents the time interval added for each stretch cycle. For example, if stretch is 0, then t_{MCS} is zero and the equation remains the same. If stretch is one then t_{MCS} equals $2t_{CLCL}$, and the equation is increased by this amount. The value of t_{MCS} is increased by $4t_{CLCL}$ for every additional stretch cycle.

Through analysis and a survey of up–to–date memory products[5], it can be determined that there are four SRAM timing parameters that are necessary and sufficient to meet the timing requirements of the DS80C320 in most situations. For the following discussion of these requirements, the worst case timing conditions, i.e., a 33 MHz clock and zero stretch cycles, have been used. For a data read operation, the DS80C320 expects the time from an address change until valid data is available to be 60 ns ($t_{AVDV1}=3t_{CLCL}-31+t_{MCS}$) or less. If the propagation delay from D to Q of a 74F373 latch (8 ns) is subtracted from this parameter, you obtain a memory address access (t_{AA}) requirement of 62 ns. Also for a read, the DS80C320 expects the time from the $\overline{RD}$ signal going low until valid data is received from the memory to be 35 ns ($t_{RLDV}=2t_{CLCL}-25+t_{MCS}$) or less. Since the processor's $\overline{RD}$ signal is tied to the memory's $\overline{OE}$ pin, the memory must have an output enable access

time (t_{OE}) of less than 40 ns. After the DS80C320 has read the data, the SRAM must relinquish the bus within 25 ns ($t_{RHDZ}=t_{CLCL}-5$). This dictates that the SRAM parameter t_{OHZ} be less than 25 ns. For a write, the processor will provide a minimum write pulse of 49 ns ($t_{WLWH}=2t_{CLCL}-11$), which is equal to the required minimum write pulse width (t_{WP}) of the SRAM. On the basis of these four calculated parameters and assumed SRAM speeds shown in Table 2, the appropriate speed device may be determined for a number of different clock frequencies. A summary of the recommended RAM speeds is given in Table 3.

AVAILABLE RAM PARAMETERS Table 2

t_{AA} (ns)	t_{OE} (ns)	t_{OHZ} (ns)	t_{WP} (ns)
60	35	25	45
70	35	30	45
80	35	30	60
100	50	35	60
120	60	45	70
150	55	40	90
170	80	35	120
200	100	35	150

5. Memory data books from Dallas Semiconductor 1992–1993, Fujitsu 1990, Hitachi #M18, Micron 1992, Mosel 1991–1992, NEC 1989, Sony 1991 were surveyed for suitable RAM products. Internet sites for IDT and Hitachi were also surveyed.

Table 3 illustrates the point that even with a 33 MHz clock, relatively slow SRAM devices may be selected if a single stretch cycle (default condition) is used. If performance is not the primary system consideration, or if it is but data memory accesses are an insignificant part of the overall processing requirements, the use of a stretch cycle may allow a more cost effective solution.

RECOMMENDED RAM CONFIGURATION Table 3

CLOCK (MHz)	LATCH	MEMORY SPEED (zero stretch)	MEMORY SPEED (one stretch)
33.0	F373	60 ns	90 ns
25.0	F373	80 ns	170 ns
20.0	F373	120 ns	200 ns
18.432	F373	120 ns	200 ns
16.0	HCT373	120 ns	200 ns
14.746	HCT373	120 ns	200 ns
14.318	HCT373	120 ns	200 ns
12.0	HCT373	170 ns	200 ns
11.059	HCT373	200 ns	200 ns
7.37	HCT373	200 ns	200 ns
1.8432 and below	HCT373	200 ns	200 ns

ADDITIONAL CONSIDERATIONS

In developing this application note, it was noted that some EPROM devices have extremely long "turn off" times. If the EPROM selected for 33 MHz systems has an "output disable to float" time greater than 25 ns (parameter $t_{PXIZ}=t_{CLCL}-5$), bus contention will occur on the processor's AD7–AD0 bus. In most situations, this simply results in higher power consumption. However, in some situations the address setup time to the memory can be affected necessitating a faster memory. The simplest solution to this problem is to use a device with the required turn–off time, but another possible solution exists. A 74F244 driver can be placed between the EPROM's output and the processor's data bus as shown in Figure 5. The 74F244's outputs turn off within a maximum of 8 ns, thereby releasing the processor's bus almost immediately and eliminating the contention.

All of the timing calculations used in this application note are based on the equations in the DS80C320 data sheet. The timing specifications given in the data sheet assume an approximately equal capacitive load on the signals specified. If the configuration of Figure 1 is used, this is achieved. If, however, any signal is connected to additional loads, then the capacitive loading including the additional devices should be evaluated. If there is a significant difference, additional margins should be used in the critical path analysis, and appropriate memory speeds selected.

For older or otherwise unconventional SRAM devices, it may be wise to confirm other important timing parameters such as data setup before write active. With the devices surveyed, meeting the four parameters discussed above will qualify the device for use.

FAST EPROM TURNOFF Figure 5
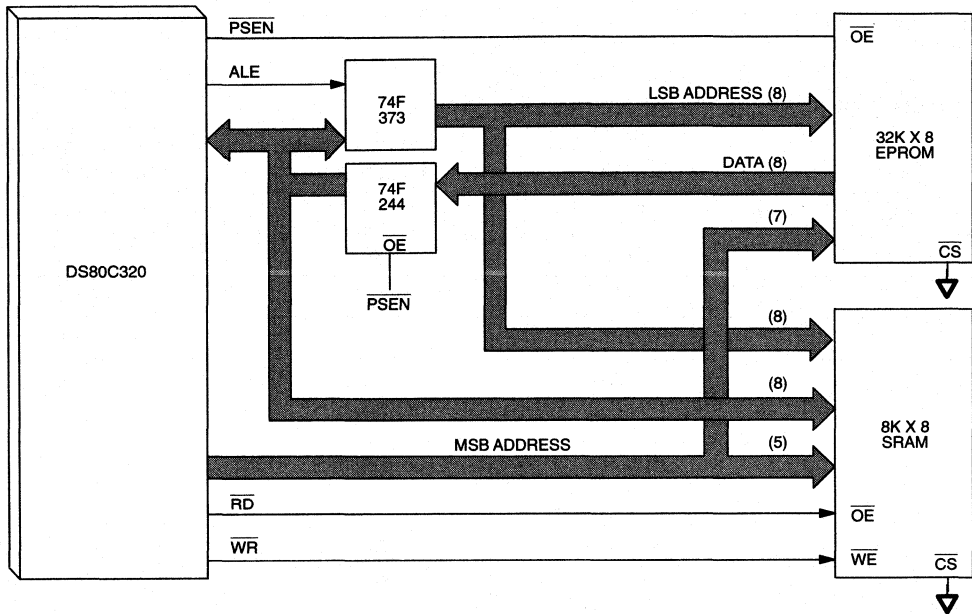
EQUATION SUMMARY

For the user who wishes to calculate the memory speed requirements using a crystal frequency not shown in the preceding tables, the following equations provide a concise summary of the information needed. The memory devices selected must have an address access time (based on the use of an F373 or an HCT373), an $\overline{OE}$ access time, a $\overline{WE}$ time, and a bus release time less than or equal to the calculated values. Note again that t_{CLCL} is the period of the clock.

EPROM Access Times

$\overline{OE} = t_{PLIV}$
$= 2.25t_{CLCL}-20$

F373
address $= t_{AVIV1}-8$
$= 3t_{CLCL}-28$

HCT373
address $= t_{AVIV1}-44$
$= 3t_{CLCL}-64$

Bus Release $= t_{RHDZ}$
$= t_{CLCL}-5$
(same for RAM)

RAM Read Access Times

$\overline{OE} = t_{RDLV}$
$= (2.0t_{CLCL}-20+t_{MCS})$

F373
address $= t_{AVDV1}-8$
$= (3.0t_{CLCL}-20+t_{MCS})-8$

HCT373
address $= t_{AVDV1}-44$
$= (3.0t_{CLCL}-20+t_{MCS})-44$

RAM Write Pulse Time

$\overline{WE} = t_{WLWH}$
$= (2t_{CLCL}-5+t_{MCS})$

DALLAS
SEMICONDUCTOR

Application Note 75
Using the High–Speed Micro's Serial Ports

INTRODUCTION

The High–Speed Micro's serial interfaces are function-ally identical to those found on other lower performance 8051 processors. They are based on a Universal Syn-chronous/Asynchronous Receiver/Transmitter (USART) implementation. As the name implies, a USART converts parallel data to and from a synchro-nous or asynchronous serial bit stream. The parallel data side of the device interfaces with the processor's internal data bus, and the serial side interfaces with the outside world. This application note describes the setup and operation of the most commonly encountered oper-ating modes of this interface.

Because of its universal applicability, the most fre-quently used configuration of the High–Speed Micro's serial channel is its 10–bit asynchronous mode. In this application note, this configuration will be described in detail. A general overview of the port's operation will be provided and a detailed software example will be pres-

ented. Examples illustrating the use of dual serial ports and 11–bit address recognition capability will also be presented. Through these three examples, different aspects of serial port operation will be discussed.

The most frequently encountered serial communication modes are based on asynchronous data transmission. In this mode of transmission, there is no separate clock signal. The classical serial asynchronous data commu-nications format illustrated in Figure 1 provides the nec-essary synchronization without a clock signal. In this format, 8 or 9 data bits are accompanied by one start bit and one or two stop bits. The 9th data bit is frequently used as a parity bit. The start and stop bits provide the necessary synchronization information. The serial bit stream's content in this mode is compatible with the popular RS–232 protocol. However, the signal levels are not. For signal level compatibility, a level translator such as the DS1228 must be used to convert TTL/CMOS levels to/from RS–232 levels.

ASYNCHRONOUS DATA FORMAT Figure 1
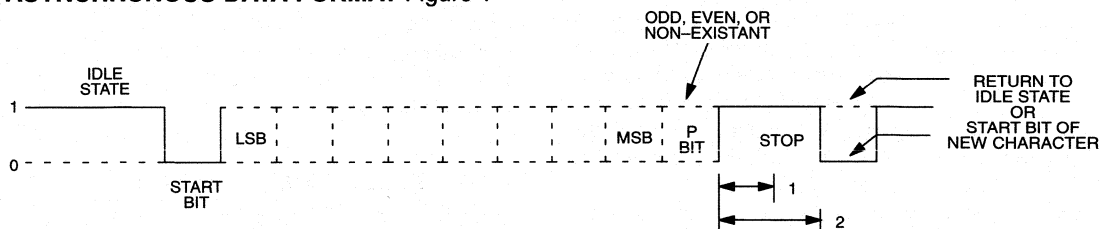
Please note that many members of the High–Speed Microcontroller family have two functionally identical serial ports. As new members are added to the proces-sor family, some may not have both serial ports. Refer to the individual device's data sheet for specific features. This application note will concentrate on the use of a single serial port (port 0), but in the examples, the ideas presented are equally applicable to both ports. One example program demonstrates the use of both serial ports.

BAUD CLOCK SOURCES

While it is not the purpose of this document to discuss details of the High–Speed Micro's internal timers, there must be some discussion of them since they are fre-quently used as the source of the baud rate clock. The High–Speed Micro contains three internal timers; two of which may be used as baud rate generators. The follow-ing sections briefly describe these timers and their modes most commonly used for baud rate generation. Detailed information on the timers' setup will be illus-trated by the software examples that follow.

042695 1/13

On both timers, auto–reload mode is frequently used for baud rate generation because it requires no intervention by the processor. Once initialized, the timers run automatically, producing a baud rate clock based on a value loaded into the timers' reload registers. When in this mode, timer 1 uses SFR registers TL1 to count and TH1 to store a reload value. Software must initialize TH1 with the desired reload value, and if the first time interval is to be correct, TL1 must also be loaded with the same value. Timer 2 uses registers TL2 and TH2 for counting, and registers RCAP2L and RCAP2H for holding the reload values. Again, these registers must be initialized by software. When enabled, the timers will begin counting based on the selected clock source. The clocks possible for timer 1 are oscillator/12 or oscillator/4. For timer 2, oscillator/2 is the only possibility when in baud clock generation mode. When the count registers roll over from their maximum count to 0, they will be automatically reloaded with the value in the reload registers. The reload value remains unchanged unless modified by software. Every time the rollover occurs, a clock pulse is generated. This clock pulse is used by the serial port as a baud rate clock. By changing the reload value, a wide variety of baud rates may be achieved.

POLLED VS. INTERRUPT DRIVEN MODES

The High–Speed Micro's serial channels have flag bits in the SFR address space that indicate the status of the channel. For each serial channel, there is a flag indicating when a character has been received (RI) and a flag indicating when a character has been transmitted (TI). These flags are set whether or not the associated interrupts are enabled, and can therefore be used in a polled mode of operation. Alternatively if the interrupts are enabled, the setting of either of these flags will cause a jump to the serial channel's associated interrupt vector.

Since the serial interrupts are asynchronous based on serial communications with an external device, most applications will benefit from using an interrupt driven communications scheme. In this way, the processor can accomplish other tasks while it is waiting to receive an interrupt. If a polling method were used, time would be consumed continually checking the flag bits to see if one has been set. The examples of this application note illustrate both approaches. The first example demonstrates a typical interrupt driven mode of operation. The second example executes a very structured set of events, so it is well suited to polled operation.

ASYNCHRONOUS 10–BIT MODE EXAMPLE

The asynchronous 10–bit mode of operation is arguably the most frequently used method of serial communication on the 8051 family. This is because this mode is compatible with the familiar RS–232 protocol. While the signal levels are different, the use of a simple level translator will allow communications with the standard serial ports of any personal computer. In fact, all of the software in this applications note was tested using a PC to interface to a DS80C320 test board.

This particular serial mode can use timer 1 to generate baud rates for serial port 1 or timers 1 or 2 for serial port 0. In this example, timer 2 is operated in auto–reload mode to generate the baud rate for serial port 0.

After establishing the timer's mode of operation, the reload value must be stored in the reload register. The contents of the reload registers may be calculated for a desired baud rate and oscillator frequency using the following equation:

$$RCAP2H, RCAP2L = 65536 - \frac{Oscillator\ Frequency}{32 * Baud\ Rate}$$

Using this equation, the reload value for any desired baud rate and oscillator frequency may be calculated. For this software example, an oscillator frequency of 11.0592 MHz is assumed. The table below shows the reload values for several common baud rates based on this crystal frequency. It should be noted that not every crystal value will produce acceptable baud rates. Some evaluation of the above equation may be necessary before selecting the crystal for a system if a specific baud rate is required.

TIMER 2 RELOAD VALUES Table 1

BAUD RATE	RCAP2H	RCAP2L
57600	0FFh	0FAh
9600	0FFh	0DCh
2400	0FFh	090h
1200	0FEh	0E0h

Once a reload value is determined, it must then be loaded into the timer's reload registers to establish the timer's output clock frequency.

After initializing the timer, the serial port may be set up as desired. To establish the correct operating mode for serial port 0, the SM0 and SM1 bits of the SCON register (address 098h) must be set appropriately. The following table shows the possible settings of these bits and the resulting mode. As shown, SM0 and SM1 (SCON0.1 and SCON0.0) must be set to 0 and 1 respectively for 10–bit asynchronous operation.

SERIAL MODE BITS Table 2

SM0	SM1	MODE	FUNCTION	LENGTH	PERIOD
0	0	0	Sync	8 bits	4/12 t_{CLK}
0	1	1	Async	10 bits	Timer 1 or 2*
1	0	2	Async	11 bits	64/32 t_{CLK}
1	1	3	Async	11 bits	Timer 1 or 2*

* Timer 2 is only available for baud rate generation on serial port 0.

Since the serial port will be operated in interrupt driven mode, the interrupt enables must be set appropriately. By setting the ES0 (address 0ACh) and EA (address 0AFh) bits to 1, serial port 0 will generate an interrupt when a character has been transmitted or when a character has been received.

As a final step in initialization, the timer is started by setting bit TR2 (address 0CAh). At this point, serial communications may begin. To transmit a character, the character is written to the SBUF (byte address 099h) and other tasks are performed until an interrupt is received (in this case a tight loop). In receiving a character, no action is required until an interrupt occurs. Since either a transmit or a receive can cause a jump to the same interrupt vector, the interrupt service routine (ISR) must determine which was the cause. This is done by reading the TI (bit address 099h) and RI (bit address 098h) status bits of the SCON register. If TI is set, then a "Transmit Complete" caused the interrupt. If RI is set, then a "Receive" caused the interrupt. If a transmit caused the interrupt, the TI bit may be cleared and the ISR exited. If a receive caused the interrupt, then the RI bit must be cleared and the received character must be read from SBUF.

The software listing for example 1 below illustrates the details of implementing this mode of serial operation.

DUAL SERIAL PORT EXAMPLE

This example demonstrates the use of the two serial ports available on the DS80C320. The main purpose of the example is to illustrate how to initialize and use the second port. As discussed earlier, much of the information regarding serial port 0 applies equally to serial port 1. However, this example will help clarify any confusion about the use of this resource.

In this example, the output of port 1 (TXD1) is connected to its input (RXD1) in a "loop back" configuration. The software is written to create a closed serial loop with port 0 as input and output. A terminal or PC running a terminal emulator is connected to port 0's input (RXD0) and output (TXD0). Initially, software outputs a three line message to the terminal on port 0, and the DS80C320 waits for an input. When the terminal sends a character to the DS80C320, the RI bit is set. When the software recognizes this bit has been set, it reads the received character and transfers it to the transmit buffer of port 1. For illustrative purposes, the character is converted to upper case (if not already) before it is transferred. Since the output of port 1 is tied to its input, the transmitted character automatically ends up in the receive buffer. The software then copies this character to port 0's transmit buffer, which causes it to be transmitted out of the processor. Finally, the character arrives at the terminal as an upper case character, thereby completing the loop.

In this software example, both serial ports are set to run from a baud clock generated from Timer 1. The timer is set for auto–reload mode, and the count and reload registers are loaded with the appropriate value. Timer 1's equation for calculating reload values is different from Timer 2's, and is shown below:

$$\text{Reload} = 256 - \frac{2^{SMOD} * \text{OscillatorFreq.}}{384 * \text{BaudRate}}$$

Using the above equation, the reload value for a desired baud rate and oscillator frequency may be calculated. It can be seen that the reload value is a function of 2 raised to the power of SMOD. Since SMOD can be either 0 or 1, this term can be either 1 or 2 ($2^0 = 1, 2^1 = 2$). Therefore, setting SMOD to 1 has the effect of doubling the baud rate. Again for this software example, an oscillator frequency of 11.0592 MHz is assumed. Table 3 below shows the reload values calculated for several common baud rates based on this crystal frequency.

TIMER 1 RELOAD VALUES

BAUD RATE	SMOD	RELOAD
57600	1	FF
19200	1	FD
9600	0	FD
2400	0	F4
1200	0	E7

As seen in the above equation, the reload value is calculated based on the SMOD baud rate doubler bit's setting. If this bit is zero for the desired baud rate, it is not necessary to clear it in the initialization software because this is its reset default condition. However for clarity, the instructions to clear this bit for both ports are included in the example. Since the SMOD bit is not "bit addressable" you must write to the entire PCON register. The example code shows instructions for clearing and setting the SMOD bit using a single logical instruction. The instruction not used in the program is commented out.

In the example, the timer mode is initialized, the count and reload registers are loaded, the two SMOD bits are cleared, and the timer interrupt is disabled. This completely configures the baud clock generation. After the two serial control registers are set for the desired mode, the timer is started, and serial communication begins.

This example handles serial communication differently than illustrated by the earlier example. This example uses a polled mode to monitor serial status. Since this example is more structured in its operation, this polled approach is appropriate. The basic functions that transfer characters, are GETCH and PUTCH. Both of these functions perform a tight loop waiting for the appropriate flag to be set. When it is, the program continues. This would normally be considered a waste of time, but in this

application and others like it, polled operation makes sense.

The software listing for example 2 illustrates the details of implementing this mode of serial operation.

ADDRESS RECOGNITION EXAMPLE

This example demonstrates the address recognition capability of the High–Speed Micro's serial channel. In addition, it illustrates a method of baud clock generation that does not involve the use of a timer (i.e., serial mode 2).

The address recognition feature of the High–Speed Micro is frequently used for multi–processor communications. Each processor on the bus can be assigned a unique address. When configured, the processors will not recognize any serial communications unless its address is matched. Full details of this mode of operation may be found in the High–Speed Microcontroller User's Guide.

In this example, the address is set to recognize a control C character (03h). Any character received before a control C is ignored. However when a control C is received, a character string is immediately printed, and subsequent characters received on RXD0 are echoed out TXD0.

As mentioned, the illustrated mode of generating a baud rate does not involve a timer (i.e., serial mode 2). The baud rate is selectable as shown in the following equation:

$$\text{baudrate} = \frac{2^{\text{SMOD}} * \text{OscillatorFreq.}}{64}$$

As can be seen in the equation, the selection of crystal frequencies is much more restricted for this serial mode than others if a particular baud rate is desired. In fact, there are relatively few crystal frequencies available that will produce a standard baud rate. In this example, an oscillator frequency of 7.372 MHz is assumed which will result in a rate of 115,200 baud with SMOD cleared to 0. It is quite common to see an oscillator of 12.0 MHz that will produce a rate of 187,500 baud with SMOD cleared to 0.

The software listing for example 3 illustrates the details of implementing this mode of serial operation.

CODE EXAMPLE # 1 : 10–BIT ASYNCHRONOUS MODE

```
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; * * * * * * * * * * * * * * *    Program ASYNC10B.ASM   * * * * * * * * * * * *
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;            This program sets up the DS80C320's Port 0 serial port to operate
;            in 10-bit asynchronous format. Timer 2 is used as the baud rate
;            generator by setting it in auto-reload mode. A crystal value of
;            11.0529 MHz is assumed for the baud rate reload values.
;
;
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
;            Interrupt Vectors
;
        ORG    0000h         ; RESET VECTOR
        AJMP   START         ; Jump to start of program
;
        ORG    0023h         ; SERIAL PORT 0 VECTOR
        AJMP   SERINT        ; Jump to serial interrupt routine
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
;            Initialization
;
        ORG    0100h
;
START:  MOV    IE, #0        ; Disable ALL interrupts
;
;            Set up timer 2
;
        MOV    T2CON, #030h  ; Timer 2 : auto-reload mode for
                             ;    both receive & transmit baud clocks
        MOV    RCAP2L, #T2RL96L    ; Establish 16-bit reload value for 9600
        MOV    RCAP2H, #T2RL96H
        MOV    TL2, #T2RL96L ; Make first timeout correct
        MOV    TH2, #T2RL96H
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
;            Set up Serial Port 0
;
        MOV    SCON, #050h   ; Mode 1, set receive enable
        SETB   ES0           ; Enable Serial Port interrupt
        ;
        SETB   EA            ; Set Global Interrupt enable
        SETB   TR2           ; Start Timer
;
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; * * * * * * * * * * * * *  Test Code to exercise the serial port   * * * * *
;
        MOV    SBUF, #'D'    ; Put character 'D' in buffer (send it)
        CALL   WAIT          ; Wait until flag cleared from interrupt
                             ;    service routine.
        MOV    SBUF, #'S'
        CALL   WAIT
```

```
            MOV     SBUF, #'8'
            CALL    WAIT
            MOV     SBUF, #'0'
            CALL    WAIT
            MOV     SBUF, #'C'
            CALL    WAIT
            MOV     SBUF, #'3'
            CALL    WAIT
            MOV     SBUF, #'2'
            CALL    WAIT
            MOV     SBUF, #'0'
            CALL    WAIT
            MOV     SBUF, #00Dh
            CALL    WAIT
            MOV     SBUF, #00Ah
            CALL    WAIT
            ;
            SJMP    $
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; * * * * * * * * * * * * * *   Subroutine to wait for a serial interrupt   * *
WAIT:       MOV     R1, #0FFh
HERE:       CJNE    R1, #0000h, HERE
            RET
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; * * * * * * * * * * * * * * * * *   Serial Interrupt Service Routine   * * * *
;
;
SERINT:     CLR     ES0             ; Disable serial interrupts
            ;
            JNB     TI, SER_A       ; If TI=0, must be receive complete interrupt
            CLR     TI              ; Else must be transmit complete interrupt
            SJMP    SER_X
SER_A:
            CLR     RI              ; Clear the receive bit
            MOV     A, SBUF         ; Get the character and
            MOV     SBUF, A         ;   echo it
SER_X:
            MOV     R1, #00h        ; Indicate interrupt has been serviced
            ;                       ;   (i.e. exit wait)
            SETB    ES0             ; Re-enable serial interrupts
            RETI                    ; Return from interrupt
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
            END                     ; Program ASYNC10B.ASM
```

CODE EXAMPLE #2 : DUAL SERIAL PORT

```
;  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;  * * * * * * * * * * * * * * * *        Program Tst2Ser       * * * * * * * *
;  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
;        This program demonstrates the simultaneous use of both of the DS80C320's
;        serial ports. Both serial channels run off of timer 1 which is set for
;        9600 KBaud with SMOD = 0, and an 11.059 MHz crystal. The program
;        assumes that the transmit and receive pins of port 1 are connected
;        together (i.e., TXD1 tied to RXD1). When the program starts, the
;        DS80C320 sends a three line message out on TXD0. It then waits for
;        a character to come in on RXD0 from an external terminal. This
;        received character is converted to upper case if it is not already.
;        The upper case character is then transferred to the transmit pin of
;        port 1, TXD1, and the processor then waits for a character to be
;        received on port 1's receive pin, RXD1. The received character is
;        then transferred to port 0's transmit pin, TXD0, which completes the
;        loop with the terminal.
;
;
;  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
;        Initialization
;
         ORG    0100h
START:
         MOV    IE, #0          ; Disable all interrupts
;
;                Set up Timer 1
;
         MOV    TMOD, #020h   ; Timer 1: Mode 2 (8-bit reload)
         MOV    TH1, #REL96   ; Reload value for 9600, SMOD = 0
         MOV    TL1, #REL96   ; Make first time-out correct
         ;
         ANL    (PCON), #07Fh ; Clear SMOD bit for port 0
         ANL    (WDCON), #07Fh        ; Clear SMOD bit for port 1
         ;
         CLR    ET1           ; Disable timer 1 interrupts
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
;                Set up serial ports
;
         MOV    SCON, #050h   ; Timer 1 : Mode 1 (vari baud)
                              ;     and set receive enable
         MOV    SCON1, #050h  ; Timer 2 : Mode 1 (vari baud)
                              ;     and set receive enable
         SETB   TR1           ; Start timer 1
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
;                Configuration Demonstration
;
         ; Output 3 Message Lines
         MOV    DPTR, #MSG1   ; Point to first message
         CALL   PUTS          ;     and call routine to output
         MOV    DPTR, #MSG2   ; Point to second message
```

```
                CALL    PUTS.           ;    and call routine to output
                MOV     DPTR, #MSG3     ; Point to third message
                CALL    PUTS            ;    and call routine to output
                ;
GETCH:
                JNB     RI, GETCH       ; Wait here for received character
                CLR     RI              ; Get ready for next character
                MOV     A, SBUF         ; Put new character in Acc
                ;
                ; If lower case character, convert to upper case
                CJNE    A, #'a', GETCH_A        ; If character
GETCH_A:        JC      GETCH_CONT              ;    is between 'a'
                CJNE    A, #'z'+1, GETCH_B      ;    and 'z' +1
GETCH_B:        JNC     GETCH_CONT              ;    it is lower case
                CLR     C                       ;    so subtract 020h
                SUBB    A, #020h                ;    from it
                ;
GETCH_CONT:
                CALL    PUTCH1          ; Output character to port 1
                ;
GETCH1:
                JNB     RI1, GETCH1     ; Wait here for received character
                CLR     RI1             ; Get ready for next character
                MOV     A, SBUF1        ; Put new character in Acc
GETCH1_CONT:
                CALL    PUTCH           ; Output character to port 0
                ;
                JMP     GETCH           ; Repeat cycle
;
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; * * * * * * * * * * * * * * * * * * * * * *   Subroutines   * * * * * * * *
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
;               ; Output a character on port 0
PUTCH:
                MOV     SBUF, A         ; Copy Acc to SBUF 0
                JNB     TI, $           ; Wait here until TI bit set
                CLR     TI              ; Clear TI for next character
                RET                     ; Return
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
;               ; Output a string on port 0
PUTS:
                CLR     A               ; Make address offset zero
                MOVC    A, @A+DPTR      ; Get next char to output
                JZ      PUTS_A          ; If zero, end of string
                CALL    PUTCH           ; Output character pointed to
                INC     DPTR            ; Point to next character
                JMP     PUTS            ; Repeat procedure
PUTS_A:
                RET                     ; Exit from routine
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
;               ; Output a character on port 1
PUTCH1:
```

```
           MOV    SBUF1, A        ; Copy Acc to SBUF 1
           JNB    TI1, $          ; Wait here until TI bit set
           CLR    TI1             ; Get ready for next character
           RET                    ; Return
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
;                  ; Output a string on port 1
PUTS1:
           CLR    A               ; Make address offset zero
           MOVC   A, @A+DPTR      ; Get next char to output
           JZ     PUTS1_A         ; If zero, end of string
           CALL   PUTCH1          ; Output character pointed to
           INC    DPTR            ; Point to next character
           JMP    PUTS1           ; Repeat procedure
PUTS1_A:
           RET                    ; Exit from routine
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
msg1:  db      'Dallas Semiconductor DS80C320', 0dh, 0ah, 0
msg2:  db      'Serial Port Tests.', 0dh, 0ah, 0
msg3:  db      'Characters typed in after this are converted to uppercase.'
       db      0dh, 0ah, 0
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
           END    ; Program Tst2Ser
```

CODE EXAMPLE # 3 : 11–BIT ADDRESS RECOGNITION MODE

```
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; * * * * * * * * * * * * * * *     Program ADRECMD.ASM   * * * * * * * * * * * * *
;
;        This program sets up the DS80C320's Port 0 serial channel to operate
;        in Multi-Processor Communications mode. In this mode, serial
;        characters received that do not match a masked address are ignored
;        (no interrupt generated). The address and mask are initially set to
;        recognize a <cntl>C character (03h). When a <cntl>C is received, a message
;        is sent out, and characters input are "echoed" out unchanged.
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;              Interrupt Vectors
;
        ORG    0000h          ; RESET VECTOR
        AJMP   START          ; Jump to start of program
;
        ORG    0023h          ; SERIAL PORT 0 VECTOR
        AJMP   SERINT         ; Jump to serial interrupt routine
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;        Initialization
;
        ORG    0100h
START:
        MOV    IE, #0              ; Disable ALL interrupts
        MOV    P1, #0              ;/Clear ports P1
;
;        Clear SMOD for divide by 64
        ANL    (PCON), #07Fh       ; Clear SMOD bit
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
;        Set up Serial Port 0
;
        MOV    SCON, #0B8h  ; Mode 2, set receive enable,
                            ;     set multi-cpu communication, and set
                            ;     TB8 to be a 1 (consistant with input)
        SETB   ES0          ; Enable Serial Port interrupt
        ;
        MOV    SADDR0, #03h ; Set address register to <cntl>C (03h)
        MOV    SADEN0, #0FFh ; Set mask to all 1s (no mask)
        ;
        SETB   EA           ; Global Interrupt enable
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; * * * * * * * * * * * * * * *   Code to exercise the serial port   * * * * * * *
;
LOOP:
        MOV    A, SADEN0    ; If <cntl>C has not been received
        CJNE   A, #00, LOOP ;    continue waiting here.
        CALL   WAIT         ;  <cntl>C will be echoed, wait till
                            ;       transmit complete.
        ; Output a message
        MOV    SBUF, #'D'   ; Put character 'D' in buffer (send it)
        CALL   WAIT         ; Wait until flag cleared from interrupt
                            ;    service routine.
```

```
            MOV    SBUF, #'S'    ; Repeat for 'S'
            CALL   WAIT
            MOV    SBUF, #'8'    ; Repeat for '8'
            CALL   WAIT
            MOV    SBUF, #'0'    ; Repeat for '0'
            CALL   WAIT
            MOV    SBUF, #'C'    ; Repeat for 'C'
            CALL   WAIT
            MOV    SBUF, #'3'    ; Repeat for '3'
            CALL   WAIT
            MOV    SBUF, #'2'    ; Repeat for '2'
            CALL   WAIT
            MOV    SBUF, #'0'    ; Repeat for '0'
            CALL   WAIT
            MOV    SBUF, #00Dh   ; Repeat for <cr>
            CALL   WAIT
            MOV    SBUF, #00Ah   ; Repeat for <lf>
            CALL   WAIT
            ;
            SJMP   $             ; Continuous loop. Only serial
                                 ;    interrupts exit loop. Characters are
                                 ;    echoed.
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; * * * * * * * * * * * * *   Subroutine to wait for a serial interrupt   * * * *
WAIT:       MOV    R1, #0FFh     ; Serial ISR clears R1 when complete
HERE:       CJNE   R1, #0000h, HERE
            RET
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; * * * * * * * * * * * * * * * * * *   Serial Interrupt Service Routine   * * * *
;
SERINT:     CLR    ES0           ; Disable serial interrupts
            ;
            JNB    TI, SER_A     ; If TI=0, must be receive complete interrupt
            CLR    TI            ; Else must be transmit complete interrupt
            SJMP   SER_X
            ;
SER_A:
            CLR    RI            ; Clear the receive bit
            MOV    A, SBUF       ; Get the character
            MOV    SADEN0, #0    ; Clear address mask register
            MOV    P1, A         ; Display character on port 1
            MOV    SBUF, A       ;    and echo it
SER_X:
            MOV    R1, #00h      ; Indicate interrupt has been serviced
                                 ;    (i.e. exit wait)
            SETB   ES0           ; Re-enable serial interrupts
            RETI                 ; Return from interrupt
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
            END  ; Program ADRECMD
```

APPENDIX A : CONSTANT DEFINITIONS

```
;  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;  * * * * * * * * * * * * * * * * * * * *     DEFINITIONS      * * * * * * * *
;           Define Register Addresses
PCON      EQU    087h             ; Contains SMOD baud rate doubler bit
TCON      EQU    088h             ; Control for timer 1 and 0
TMOD      EQU    089h             ; Mode register for timers 1 and 0
TL1       EQU    08Bh             ; Timer 1 low (count) byte
TH1       EQU    08Dh             ; Timer 1 high (reload) byte
CKCON     EQU    08Eh             ; Clock Control register
P1        EQU    090H             ;
SCON      EQU    098h             ; Control for serial port 0
SBUF      EQU    099h             ; Data buffer for serial port 0
IE        EQU    0A8h             ; Interrupt enables
SADDR0    EQU    0A9h             ;
SADEN0    EQU    0B9H             ;
SCON1     EQU    0C0h             ; Control for serial port 1
SBUF1     EQU    0C1h             ; Data buffer for serial port 1
T2CON     EQU    0C8h             ; Timer 2 control register
RCAP2L    EQU    0CAh             ; Timer 2 low byte of reload register
RCAP2H    EQU    0CBh             ; Timer 2 high byte of reload register
TL2       EQU    0CCh             ; Timer 2 low byte of count register
TH2       EQU    0CDh             ; Timer 2 high byte of count register
WDCON     EQU    0D8h             ; Watchdog Control register
;
;           Define bit addresses
TR1       EQU    08Eh             ; Timer 1 enable bit
RI        EQU    098h             ; Serial port 0 receive interrupt flag
TI        EQU    099h             ; Serial port 0 transmit interrupt flag
REN       EQU    09Ch             ; Serial port 0 receive enable bit
SM2       EQU    09Dh             ; Serial port 0 mode bit 2
SM1       EQU    09Eh             ; Serial port 0 mode bit 1
SM0FE     EQU    09Fh             ; Serial port 0 mode bit 0
;
RI1       EQU    0C0h             ; Serial port 1 receive interrupt flag
TI1       EQU    0C1h             ; Serial port 1 transmit interrupt flag
REN1      EQU    0C4h             ; Serial port 1 receive enable bit
SM21      EQU    0C5h             ; Serial port 1 mode bit 2
SM11      EQU    0C6h             ; Serial port 1 mode bit 1
SM0FE1    EQU    0C7h             ; Serial port 1 mode bit 0
;
TR2       EQU    0CAh             ; Timer 2 enable bit
;
ET1       EQU    0ABh             ; Timer 1 interrupt enable
ES0       EQU    0ACh             ; Serial port 0 interrupt enable
ES1       EQU    0AEh             ; Serial port 1 interrupt enable
EA        EQU    0AFh             ; Global interrupt enable
;
;           Define Constants
; Timer 1 Reload Values
REL576    EQU    0FFh             ; Reload value for 57600 baud, SMOD = 1
REL96     EQU    0FDh             ; Reload value for 9600 baud, SMOD = 0
REL24     EQU    0F4h             ; Reload value for 2400 baud, SMOD = 0
REL12     EQU    0E7h             ; Reload value for 2400 baud, SMOD = 0
; Timer 2 Reload Values
T2RL57H   EQU    0FFh             ; 16-bit Reload value for 57600 baud
T2RL57L   EQU    0FAh
T2RL96H   EQU    0FFh             ; 16-bit Reload value for 9600 baud
```

```
T2RL96L    EQU    0DCh
T2RL24H    EQU    0FFh            ; 16-bit Reload value for 2400 baud
T2RL24L    EQU    070h
T2RL12H    EQU    0FEh            ; 16-bit Reload value for 1200 baud
T2RL12L    EQU    0E0h
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

APPENDIX B: MAXIMUM BAUD RATES

It is frequently asked, what the maximum possible baud rate is for a particular serial mode of the DS80C320. Assuming a 25 MHz crystal, the following table shows the maximum possible baud rates for the four primary serial modes.

MODE	MAXIMUM BAUD, 25 MHz	MAXIMUM BAUD, 33 MHz
0	6,250,000	8,250,000
1	390,625	515,625
2	781,250	1,031,250
3	390,625	515,625

DALLAS
SEMICONDUCTOR

OVERVIEW

Power management is critical in battery–powered applications. Differences of microamperes can translate into months or years of operating life, which can make or break a product in the marketplace. The high level of integration of Dallas Semiconductor microcontrollers make them ideal for portable or battery–operated applications which demand low power consumption. By combining the processor and peripherals onto a single die, redundant hardware is eliminated, and power savings are achieved. In addition, power management features designed into the DS87C5x0 High–Speed Microcontrollers further reduce power consumption.

Dallas Semiconductor microcontrollers are manufactured with a Complementary Metal Oxide Semiconductor (CMOS) process which makes them inherently low power devices. Unlike other technologies, CMOS devices have an infinitesimal quiescent current and only draw significant currents when switching logic states. This means that without further intervention, the user has already designed a low power system. The power management features of the DS87C5x0 family of microprocessors allow the system designer to effect an even greater reduction in power. The maximum current consumption in both operating and halted states of the microcontroller is shown below.

Maximum Current, 25 MHz

	Slowest Operating Speed (PMM2)	Osc. Halted
DS87C520	1.2 mA	1 µA

A number of factors can affect power consumption that are generally beyond the ability of the device to control during operation. The single largest factor in power consumption of a microcontroller is clock frequency. The power consumed by a microprocessor is directly proportional to its operating speed, so it follows that a device operating at the lowest possible frequency will produce the maximum power savings. The speed chosen depends on the system requirements, most notably interrupt service time. Temperature can also affect power consumption. Semiconductor devices draw

greater power at lower temperatures. If the system under development is being designed for cold temperatures, the designer should expect higher than typical power consumption values. System design also has a direct bearing on power consumption, and driving large external loads will increase power consumption.

This application note covers the DS87C520 and DS87C530 High–Speed Microcontrollers. Their power management features are explained, and techniques are presented for minimizing power consumption. Because the power management feature operates in conjunction with many of the peripheral functions, especially the interrupt and serial functions, the user is urged to become familiar with the overall operation of the processor before beginning this section.

The DS87C5x0 incorporates a number of new features specifically designed for power management. They were designed to reduce power consumption without sacrificing throughput or responsiveness to external events. These features are listed below:

DYNAMIC CLOCK SPEED CONTROL

The High–Speed Microcontrollers support four clock management modes: Stop, PMM1 (Power Management Mode 1), PMM2 (Power Management Mode 2), and Idle. The DS87C5x0 can dynamically switch between these modes, allowing the user to optimize the speed of the device while minimizing power consumption. The Stop mode has been improved over standard 8051 capabilities, and now supports resume from external interrupts as well as reset sources.

SWITCHBACK

The DS87C5x0 incorporates an automatic "switchback" feature to allow a device operating in a Power Management Mode (PMM) to switch into "high gear" upon receipt of an external interrupt or serial port transmission. This enables a device in power–saving mode respond quickly to external events and/or operate its

serial ports. Traditional 8051–based devices without the switchback feature lose the ability to service interrupts quickly without running the device at high speed, and higher power consumption, constantly.

SELECTABLE CLOCK SOURCE

The crystal oscillator is a large consumer of power on any microcontroller, especially during low power operation. The DS87C5x0 ring oscillator, used for quick starts from Stop mode, can also be used to provide an approximately 3 to 4 MHz clock source during normal operation. Although a crystal oscillator is still required at power–up, once the crystal has stabilized, device operation can be switched to the ring oscillator, realizing a power savings of as much as 25 mA.

BAND–GAP REFERENCE DISABLING

The DS87C5x0 gives the user the option of disabling the band–gap reference, which is used to detect a power failure while in Stop mode. Stop mode current can be reduced from 80 μA to 1 μA by using this feature.

ENHANCED STATUS REPORTING

Although the ability to dynamically switch the internal clock speed is a benefit, if performed at the wrong time it can seriously interfere with the operation of timing–dependent functions. The Status register (STATUS;C5h), new to the DS87C5x0 devices, contains information about the status of both serial ports, the crystal oscillator, and high priority, low priority, and power fail interrupts. The software can delay or cancel a planned speed change based on the information in this register.

CLOCK SPEED CONTROL

Description

The operating frequency of a microcontroller is the single biggest factor in determining power consumption. The DS87C5x0 family of microcontrollers supports four clock speed management modes which conserve power by slowing or stopping the internal clock. These modes allow the system designer to maximize power savings with a minimum impact on performance.

PMM1

Power Management Mode 1 (PMM1) allows the user to run the DS87C5x0 at a reduced speed to save power. Setting the clock divider rate bits (PMR.7–6) will force the part from its default 4 clocks per machine cycle

(divide by 4) to 64 clocks per machine cycle (divide by 64). The external crystal continues to operate at full speed. All peripherals and instructions will operate at this reduced speed. The DS87C5x0 can resume divide by 4 operation by setting the appropriate clock divider rate bits or by utilizing the switchback feature.

PMM2

Power Management Mode 2 (PMM2) allows the user to run the DS87C5x0 at an even slower speed to improve power savings. Setting the clock divider rate bits (PMR.7–6) will force the part from its default 4 clocks per machine cycle (divide by 4) to 1024 clocks per machine cycle (divide by 1024). The external crystal continues to operate at full speed. All peripherals and instructions will operate at this reduced speed. The DS87C5x0 can resume full–speed (divide by 4) operation by setting the appropriate clock divider rate bits or by utilizing the switchback feature. This mode permits an even greater power savings over PMM1.

STOP MODE

The Stop mode is the lowest power state available to the DS87C5x0. It is initiated by setting the Stop bit (PCON.1). While in this mode the crystal oscillator is stopped, and all internal clocking, including the Watchdog Timer, is halted. The real time clock on the DS87C530 is unaffected by Stop mode. The Stop mode is exited by an external interrupt, real–time clock interrupt, an external reset via the RST pin, or a power–on reset. Each interrupt will cause the device to vector to the corresponding interrupt routine to resume execution.

The DS87C5x0 incorporates a ring oscillator to allow for a fast resumption from Stop mode. This provides an instantaneously available 4 MHz clock source for the device to start operation. It can function until the crystal has stabilized, or can continue to be used as the clock source. The ring oscillator does not exhibit as much stability as an external clock, and the device should not perform timing measurements requiring high accuracy or serial port data transfers while operating from the ring oscillator. For more information concerning the use of the ring oscillator, please consult the Clock Source Control section of this document.

IDLE MODE

The Idle mode halts operation of the DS87C5x0 processor core but leaves internal clocks, serial ports, and timers running. This mode is invoked by setting the IDL bit (PCON.0), and can be exited by an interrupt or external reset via the RST pin. Use of this mode is not recommended on new designs, as lower power operation can be achieved by placing the part in PMM2 and executing NOPs. Its inclusion in the DS87C5x0 provides backward software compatibility.
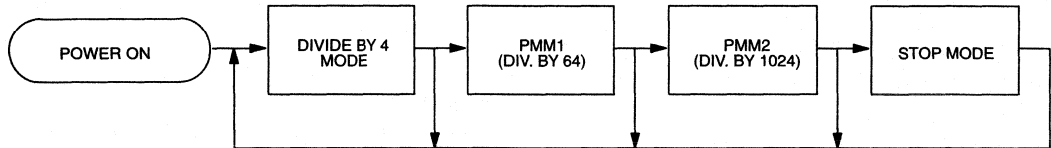
POWER MANAGEMENT MODES

The greatest power savings come from utilizing the power management modes associated with the DS87C5x0. Unlike other techniques, Power Management Modes 1 and 2 (PMM1 and PMM2) allow the user to reduce power consumption without sacrificing performance. Although the power management features are an important part of a power efficient design, a thorough understanding of the microprocessor will allow the system designer to achieve maximum power savings.

The clock speed management modes are designed to be part of a progressive level of power reduction, based on external activity and performance needs. PMM1 and PMM2 provide the lowest level of power consumption while still permitting full computational and peripheral operation. Figure 1 demonstrates the progression of clock management modes. As explained later, transitions between PMM1 and PMM2 must be made through divide by 4 mode.

PROGRESSION OF CLOCK SPEED MODES Figure 1

ENTERING AND EXITING POWER MANAGEMENT MODES

Software invokes the desired power management mode using bits in the Power Management Register (PMR). Stop mode is invoked by setting the STOP bit (PCON.1). The device speed is selected by the clock divider rate bits CD1, CD0 (PMR.7–6), shown below.

CLOCK DIVISOR RATE BIT SETTINGS

CD1	CD0	MACHINE CYCLE RATE
0	0	Reserved
0	1	4 clocks (default)
1	0	64 clocks (PMM1)
1	1	1024 clocks (PMM2)

PMM1 and PMM2 can be exited by configuring the clock divider rate bits CD1, CD0, or by the switchback function. Entry to or exit from either PMM can only be via the divide by 4 mode. For example, to go from PMM1 (divide by 64) to PMM2 (divide by 1024) mode, it is necessary to first switch from PMM1 to divide by 4 mode, and then from divide by 4 to PMM2. Attempts to execute an illegal speed change will be ignored and the bits will remain unchanged. It is the responsibility of the software to test for serial port activity using the Status register (STATUS;C5h) before attempting to change speed. Changing speed during an asynchronous serial port operation will corrupt the serial transmission.

When PMM is invoked, the external crystal will continue to operate at full speed, and the DS87C5x0 will still execute four internal states per machine cycle. In PMM the device performs an internal divide of the external clock, by 16 for PMM1 (16 x 4=64) or 256 for PMM2 (256 x 4=1024) to achieve the desired frequency, as opposed to actually performing 64 or 256 internal states per instruction. For example, operations that occur during C2 will still do so. Most applications will not find it necessary to attend to this much detail, but the information is provided for calculating critical timings.

INTERNAL TIMING RELATIONSHIPS IN PMM1 Figure 2
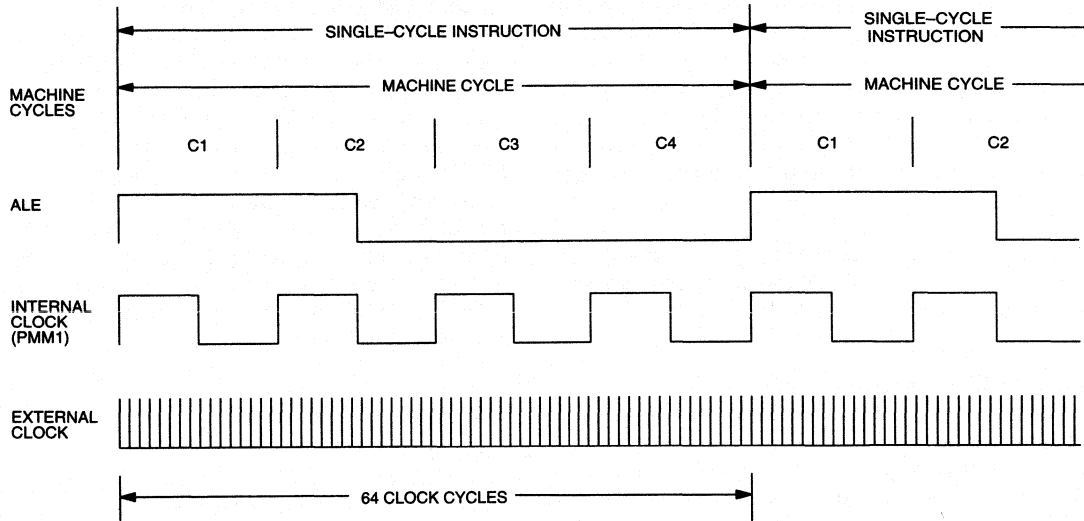
Note that changing the clock divisor, either manually or through a switchback, will not affect Timed Access procedures. Timed Access operates in relation to internal machine cycles, not an absolute time reference.

TIMERS AND PMM

Timers 0, 1, and 2 will default on power–up to a 12 external clocks per timer tick to remain compatible with the original 8051/8032 specifications. The timers can be individually configured to run at a rate of four external clocks per timer tick when the device is operating in divide by 4 mode by setting the relevant bits in the Clock Control Register (CKCON;8Eh). During PMM timers 0, 1, and 2 operate at correspondingly reduced clock rates, because the timers derive their time base from the internal clock. This will also affect the operation of the serial ports in PMM as the timers are used to generate baud rates. Table 2 shows the effect of the clock divide rate on timer operation.

EFFECT OF CLOCK MODES ON TIMER OPERATION Table 2

CD1	CD0	OSC. CYCLES PER MACHINE CYCLE	OSC. CYCLES PER TIMER 0/1/2 CLOCK		OSC. CYCLES PER TIMER 2 CLOCK, BAUD RATE GEN.		OSC. CYCLES PER SERIAL PORT CLOCK MODE 0		OSC. CYCLES PER SERIAL PORT CLOCK MODE 2	
			TxM=1	TxM=0	T2M=1	T2M=0	SM2=0	SM2=1	SMOD=0	SMOD=1
0	0	Reserved								
0	1	4	12	4	2	2	12	4	64	32
1	0	64 (PMM1)	192	64	32	32	192	64	1024	512
1	1	1024 (PMM2)	3072	1024	512	512	3072	1024	16,384	8192

The watchdog timer runs off the same time base as the internal clocks; i.e. if the device is in PMM1, the watchdog will also be running in a divide by 64 mode. This keeps the watchdog timer synchronized with the operation of the processor when switching between PMM and divide by 4 mode. Applications that use the watchdog timer as an additional timer should take this into account if PMM is used.

The real time clock is independent of the PMM setting. It uses the external 32 KHz crystal as its reference, and can be accessed in any mode.

MANUALLY EXITING PMM

In addition to the switchback feature, it is also possible to manually exit a PMM by configuring the clock divider rate bits. Entry to or exit from either PMM can only be via the divide by 4 mode, and attempts to execute an illegal speed change will be ignored and the bits will remain unchanged. If a timing–dependent operation may be in progress, the Status register (STATUS;C5h) should be interrogated to determine if it has been completed before switching out of or into PMM.

RESET SENSITIVITY IN PMM

While in PMM, the method used to detect an external reset pulse differs from that used in divide by 4 mode. In divide by 4 mode, (and standard 8051 architecture) a reset must be high for two machine cycles to be detected. If this were true in PMM, reset would have to be asserted for a minimum of 128 or 2048 clock cycles. To avoid this, devices operating in PMM employ a positive edge–detect sensor, rather than a one machine cycle qualification to detect a reset signal. This means that devices which use PMM are more susceptible to noise and additional care must be taken to keep the reset signal noise–free.

SWITCHBACK FEATURE

Description

The switchback feature allows the user to quickly restore the DS87C5x0 to a higher speed when an event occurs. When enabled, a qualified event causes the device to automatically switch from divide by 64 (PMM1) or divide by 1024 (PMM2) to divide by 4 operation without software intervention. This allows the device to

respond to high priority or interrupt driven events with a minimum delay. The following sources can trigger a switchback:

> external interrupt 0/1/2/3/4/5,
> serial start bit detected, Serial Port 0/1,
> transmit buffer loaded, Serial Port 0/1,
> watchdog timer reset,
> power–on reset,
> external reset.

Because of the intimate relationship between PMM, switchback, external interrupts, timer/counters, and the serial ports, it is highly recommended that the system designer become familiar with these features before attempting to use the switchback feature.

Status Register

A Status register (STATUS;C5h) has been added to the DS87C5x0 to aid the software in determining whether a speed change is appropriate. The Status register provides information on the status of both serial ports, and high priority, low priority, and power fail interrupts, allowing the device to determine whether or not the device should be switched into PMM.

The benefits of the Status register become apparent when using the switchback feature to exit or enter PMM. A device executing an interrupt service routine in PMM will not execute a switchback in response to an interrupt of equal or lower priority. The Status register can be used to test for an interrupt service routine in progress, and can hold off entering PMM until finished, or take another course of action.

Enabling/Initiating Switchback

Automatic switchback is enabled by setting the SWB bit (PMR.5). When a qualified switchback event occurs, the device will exit either PMM and return to the default operating mode of 4 clocks per machine cycle. Clearing the SWB bit will disable the ability of external interrupts and serial ports to cause future switchbacks, but will not affect the current speed of the DS87C5x0. Five conditions must be met for an external interrupt to cause a switchback:

1. The device must currently be in PMM1 or PMM2.

2. The SWB bit (PMR.5) must be set.

3. Global Interrupts must be enabled by setting the EA bit (IE.7).

4. The specific interrupt must be enabled.

5. The specific interrupt occurs and is acknowledged.

Switchbacks via the serial port are slightly different. In general, switchbacks are caused by interrupts. In the case of the serial ports, this introduces a problem as they generate interrupts only upon receipt or transmission of a complete word. For the serial port to properly receive or transmit a word at standard baud rates, it must be operating at full speed. If the DS87C5x0 is operating in PMM, it would never complete a reception to initiate an interrupt, or the corresponding switchback.

The DS87C5x0 solves this problem by initiating a switchback, if enabled, upon the receipt of a falling edge on the RX pin, not the receiver interrupt. This switches the device back to full speed on the next internal machine cycle, in time to capture the start bit, and the rest of the transmission. Note that the ability of the serial port to initiate a switchback is not dependent on the Enable Serial Port Interrupt bits (IE.4 or IE.6), only the specific Receiver Enable bit (SCON0.4 or SCON1.4). Four conditions must be met for a serial port reception to cause a switchback:

1. The device must currently be in PMM1 or PMM2.

2. The SWB bit (PMR.5) must be set.

3. The specific serial port must be enabled by setting the specific Receiver Enable bit (SCON0.4 or SCON1.4).

4. A falling edge is detected on the specific RX pin.

The switchback feature also works in conjunction with the transmit function. If the appropriate conditions are met, a device operating in a PMM will automatically return to divide by 4 mode when a serial port buffer (SBUF0;99h or SBUF1;C1h) is loaded. This removes the need for the user to manually set the speed to divide by 4 before initiating the transmission. The transmitter interrupt can be used to signal when the transmission is complete so that software can return the device to the appropriate PMM. Three conditions must be met for a serial port transmission to cause a switchback:

1. The device must currently be in PMM1 or PMM2.

2. The SWB bit (PMR.5) must be set.

3. A serial port transmission must be initiated by loading the specific serial port buffer (SBUF0;99h or SBUF1;C1h).

Although both the serial port transmit and receive functions are possible in PMM, it is not possible to configure the baud rate generator to any standard rate (300, 1200, 2400, etc.) in these modes, making it impossible to communicate with a standard peripheral. The use of the switchback feature is strongly recommended if serial port activity and PMM are to be used in a design.

CONSIDERATIONS WHEN USING SWITCHBACK

Switchback Timing

One of the primary considerations when using the switchback procedure is the time required to return the device to full speed from a PMM. This is a factor in calculating the latency associated with servicing an interrupt. Switchbacks will occur at the C1 cycle of the first instruction following the event initiating the switchback. If the current instruction in progress is a write to the IE, IP, EIP or EIE register, interrupt processing will be delayed until the completion of the following instruction. Figure 3 demonstrates the timing relationship between interrupts and switchbacks during a two–cycle instruction.

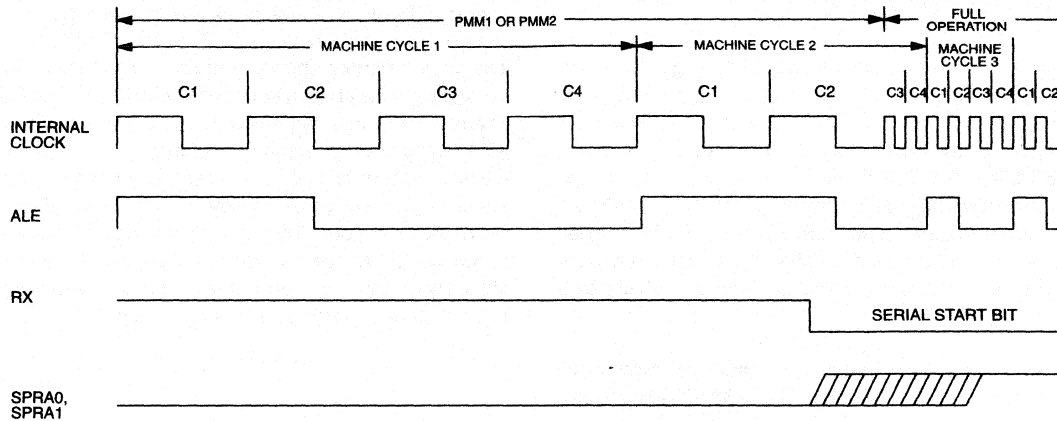## INTERRUPT–DRIVEN SWITCHBACK Figure 3
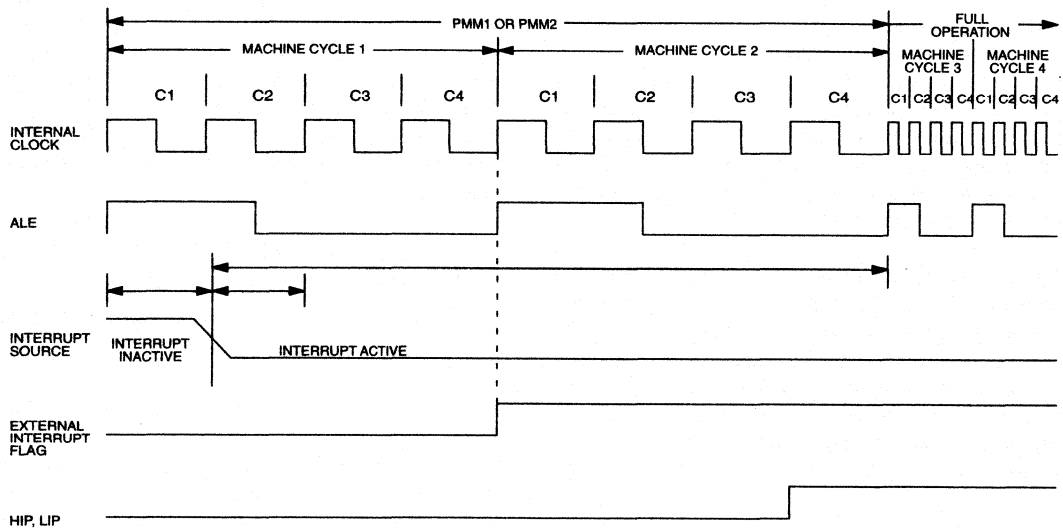
NOTES:

1. Internal clock cycle is 16 external clock cycles for PMM1 and 256 external clock cycles for PMM2.

2. Polarity of interrupt varies with respect to external interrupt number.

3. Example shows two–cycle instruction. Execution of the interrupt and switchback will occur at the end of the last machine cycle of the instruction in which the interrupt is acknowledged.

One exception to the above timing relationship is that serial port switchbacks will occur immediately upon receipt of a falling edge on an enabled serial port receiver. The switchback will occur at the start of the next internal clock cycle following the falling edge. Figure 4 demonstrates the timing relationship between serial port activity and switchbacks.

SERIAL PORT DRIVEN SWITCHBACK Figure 4

NOTES:

1. Internal Cx cycle is 16 external clock cycles for PMM1 and 256 external clock cycles for PMM2.

2. SPRA0 and SPRA1 will change within 1 machine cycle of the falling edge RX.

3. Example shows single–cycle instructions. Execution of the interrupt and switchback will occur at the end of the last machine cycle of the instruction in which the interrupt is acknowledged.

Interrupt Priority

Because the switchback feature uses interrupts to qualify execution, it is affected by interrupt priorities. The external interrupts initiate a switchback upon the start of the interrupt service routine. If a higher priority interrupt is in progress, the associated switchback will remain pending. It is not possible to enable or disable the switchback function for individual interrupt sources, except by enabling or disabling the specific interrupt.

The following example will illustrate how a problem could occur if the priority of the interrupt sources is not taken into account. Assume that the user is employing both Timer 0 and External Interrupt 1 in a design which utilizes PMM. While operating in PMM, a Timer 0 interrupt occurs, and the device begins executing the interrupt service routine (ISR). During the Timer 0 ISR, an External Interrupt 1 occurs, signaling an external event that needs to be serviced quickly. Because both interrupts have the same priority, External Interrupt 1 will remain pending until completion of the Timer 0 ISR. Although such interrupt priorities are a normal consideration in any design, the reduced operating speed in PMM will further increase the latency associated with servicing External Interrupt 1. This could be avoided by specifying External Interrupt 1 as a high priority interrupt and leaving Timer 0 as a low priority interrupt.

A serial port–initiated switchback does not utilize the interrupt structure, and is therefore not affected by interrupt priorities. Serial port–initiated switchbacks are enabled or disabled via the specific receiver enable bit (SCON0.4 or SCON1.4). The ability of a serial port to initiate a switchback is not dependent on the Enable Serial Port Interrupt bits (IE.4 or IE.6).

Serial Port Activity and PMM

Because the function of PMM is to change the internal clock frequency of the DS87C5x0, timing–dependent peripherals such as the serial ports can be affected. The user must be sure that the serial ports are not receiving or transmitting when switching into PMM. The simplest way to do this is to interrogate the serial port activity bits in the Status register (STATUS.3–0).

During a receive operation, the falling edge of the start bit will activate the switchback, if enabled. The serial port activity monitor bit will be set, and the serial port will then check for a valid start bit. If the start bit is received, the serial reception will continue normally, and generate an interrupt when the entire word is received, if enabled. To minimize power consumption, PMM may be enabled again at the start of the serial port interrupt service routine if no further processing is needed in divide by 4 mode.

It is possible to experience a "spurious" switchback caused by a noisy serial port. The DS87C5x0 initiates a switchback on the first falling edge on the RX pin, and begins looking for a valid start bit. If a valid start bit is not received, the system will abort the serial activity, clearing the activity bit, and no serial port interrupt will be executed. The switchback has already been initiated, however, and the device is now operating at full speed. To return the device to PMM, it will be necessary for the user to manually reset the clock rate divider bits.

The code fragment shown in Figure 5 illustrates one possible test for an invalid return to divide by 4 mode. This test can be inserted into the main code loop where it will be periodically executed, or it could be made part of a timer interrupt routine. If no interrupts or serial ports are active, it is likely that the device should be in PMM, not divide by 4 mode. This code should be customized according to the specific configuration, i.e. if a PMM should be allowed in a low priority interrupt, then mask out that bit when testing the Status register.

INVALID SWITCHBACK TEST EXAMPLE Figure 5

```
MODETEST: PUSH  A            ;Save the current value of the accumulator.
          MOV   A, PMR       ;Move the data to a bit-addressable register.
          JB    E7, PMM_ON   ;If bit 7 is set, device is already in PMM.
CHK_STAT: MOV   A, STATUS    ;Check status register for active interrupts.
          AND   A, #0EFh     ;Check for user-defined activity.
          JNZ   CHK_STAT     ;If activity, loop until complete.
            .                ;(Code can either loop until the condition
            .                ; clears or abort attempt to reenter PMM.)
ENA_PMM:  OR    PMR, #0C0h   ;Status okay for return to PMM. Set Clock Rate
                             ; Divider bits (example shows return to PMM2)
PMM_ON:   POP   A            ;Restore accumulator.
```

Multiprocessor Communications in PMM

The effectiveness of PMM and the switchback feature is affected if multiprocessor communications protocols are used. The DS87C5x0 includes features that will support multiple processors on the same serial port. In serial port modes 2 and 3 it is possible to use the SM2 flag (SCON0.5 or SCON1.5) to signify that the received byte is an address. The slave address recognition registers (SADDR0;A9h, SADDR1;AAh, SADEN0;B9h, SADEN1;BAh) can be programmed to ignore a transmission (not cause a receiver interrupt) when a received address does not match a user defined pattern.

The implication of multiprocessor communications for power management is that a switchback is generated by the detection of the first falling edge on a serial port, not the generation of a valid interrupt. As a result, an invalid address which should be ignored by a particular processor will still generate a switchback. Normally, the part could be returned to PMM at the start of the serial port interrupt service routine. Unfortunately, in the above mentioned case no interrupt will be generated. To alleviate this problem, one should avoid using a multiprocessor communication scheme in conjunction with PMM. If the system power considerations will allow for an occasional erroneous switchback, the polling scheme shown in Figure 5 can be used to place the device back into PMM.

CLOCK SOURCE CONTROL

Description

The DS87C5x0 incorporates a number of features to control the clock source to the device. By controlling the source of the system clock, the system designer can simultaneously achieve higher performance and decreased power consumption. To provide maximum flexibility, the DS87C5x0 will operate from three clock sources:

1. External Crystal (using internal crystal oscillator),
2. External Clock Oscillator,
3. Internal Ring Oscillator.

EXTERNAL CRYSTAL

The most common clocking source for the DS87C5x0 is an external crystal. The DS87C5x0 incorporates a crystal amplifier which is designed to drive industry standard crystals over the operating range of the device. External crystals provide a highly accurate clock source for timing–dependent peripherals such as internal timers and serial ports, as well as device operation. The DS87C5x0 requires a fundamental–mode, parallel–resonant (also called anti–resonant) AT cut crystal. Crystal oscillators have significant start up times, however, which can delay the operation of the device when powering on or resuming from the Stop mode. The DS87C5x0 must start operation with an external crystal or external clock source after a power–on reset.

EXTERNAL CLOCK SOURCE

If a clock oscillator is already present in the system, it can be used as the external clock source for the DS87C5x0. External clock oscillators suffer from the same start–up delays as the internal crystal oscillator, and do not provide any special benefits. Either an external crystal or external clock source can be used to clock the device following a power–on reset or power–fail reset.
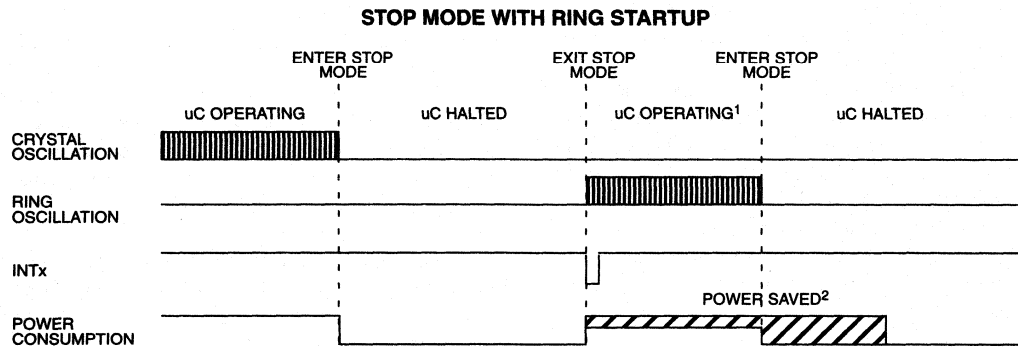
RING OSCILLATOR

The ring oscillator is a low power digital oscillator internal to the DS87C5x0. It can be used as the primary system clock, instead of an external crystal or clock oscillator, at any time except immediately following a power–on reset. When enabled, it provides a 4 MHz clock source for device operation. This is typically a lower frequency than the system clock, and this provides a significant power savings. Once the device has switched to the ring oscillator as the clock source, the external crystal amplifier can be disabled, further reducing power consumption.

In addition, the ring oscillator eliminates the delay associated with a crystal oscillator and provides an almost instantaneous startup from Stop mode. When used to restart the DS87C5x0 from Stop mode, the ring oscillator will function for a minimum of 65,536 clock cycles, at which time it can automatically switch to the external crystal or continue to run from the ring oscillator. The ring oscillator cannot be used to clock the device from a power–on state, however. Because the ring oscillator does not exhibit as much stability as an external clock, the DS87C5x0 should not perform timing measurements or serial port data transfers while using the ring oscillator as the clock source.

The ring oscillator finds wide applicability in applications that only require sporadic bursts of processing. Such a system will occasionally awaken from Stop mode, perform some activity, and then return to Stop mode. The ring oscillator allows the system to quickly switch from the lowest power state, perform an operation, and then return to a low power state, without restarting a halted external crystal. Figure 6 shows the advantages of restarting from Stop mode with and without the ring oscillator.

ADVANTAGES OF RING OSCILLATOR Figure 6

STOP MODE WITHOUT RING STARTUP
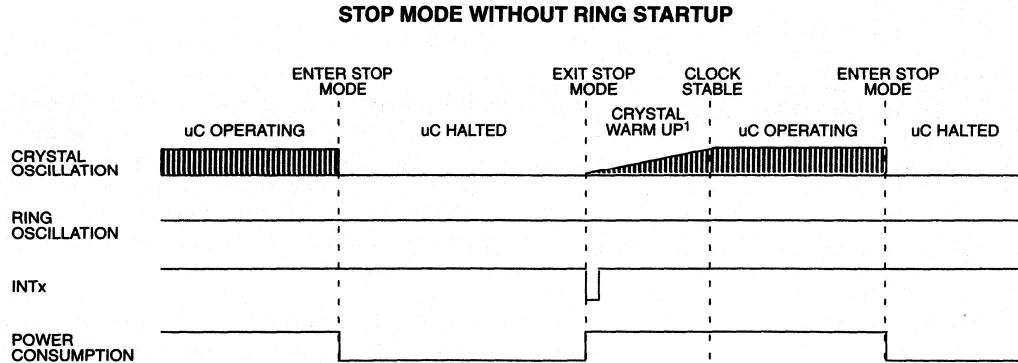
1. Crystal Warm up period is 4 to 10 ms depending on crystal type and speed.

STOP MODE WITH RING STARTUP

1. Diagram assumes that the operation following Stop requires less than 18 ms to complete.

2. Additional power savings due to decreased ring oscillation current compared to crystal amplifier.

Even if timing–dependent functions are necessary shortly after resuming Stop mode, the ring oscillator may be beneficial. Typically there is some processing that is necessary along with a timing routine or serial port transmission. Prior to executing the Stop command, the device should switch to the crystal as the clock source and set the RGSL bit. Upon resuming from Stop, the device can execute code while running from the ring oscillator in preparation for the timing dependent operation. The device can then loop until the RGMD bit has been cleared, indicating that the crystal or external clock source is now the clock source, and timing dependent operations can begin.

CLOCK CONTROL BITS

There are a number of bits that are used to configure the clock management modes of the DS87C5x0. These allow the system to switch between different clock sources, indicate the current status of the clock source, and select how the device will resume from Stop mode. The pertinent bits are shown in Table 3.

CLOCK CONTROL AND STATUS BIT SUMMARY Table 3

BIT NAME	LOCATION	FUNCTION	RESET	WRITE ACCESS
XT/$\overline{\text{RG}}$	EXIF.3	Crystal/Ring Clock Source Select. 0 = Select ring oscillator as clock source. 1 = Select crystal or external clock as clock source.	1	0 anytime; 1 when XTUP=1 and XTOFF=0
RGMD	EXIF.2	Ring Oscillator Mode Status. 0 = Crystal or external clock is current clock source. 1 = Ring oscillator is current clock source.	0	None
RGSL	EXIF.1	Ring Oscillator Select, Stop Mode. 0 = Crystal or external clock will be the clock source when resuming from Stop mode. 1 = Ring oscillator will be the clock source when resuming from Stop mode. Note: Upon completion of crystal warm up period, device will switch to clock source designated by XT/$\overline{\text{RG}}$ bit.	Unchanged except after power–on reset, when it is cleared to 0.	Unrestricted
XTOFF	PMR.3	Crystal Oscillator Disable. 0 = Crystal oscillator is enabled. 1 = Crystal oscillator is disabled. Device is operating from ring oscillator.	0	0 anytime; 1 when XT/$\overline{\text{RG}}$=0
XTUP	STATUS.4	Crystal Oscillator Warm Up Status. 0 = Oscillator warm up still in progress. 1 = Oscillator warm up complete.	1	None

CRYSTAL OSCILLATOR STARTUP DELAY

When power is applied to a crystal oscillator after a period of non–operation, a short period of time is required before the amplitude of the pulse is sufficient to provide a stable clock source. This can result in missed or corrupted clock signals, possibly disrupting processor operation. To ensure a valid clock signal, the DS87C5x0 uses a crystal startup counter to detect 65,536 oscillations of the external crystal or clock oscillator before allowing the device to resume operation. This means that devices utilizing slower crystals will have longer crystal startup times. The crystal startup counter is more sensitive than the internal clock circuitry, and uses the count of both bad and good pulses to determine the warm–up period. The counter value was chosen to allow the majority of crystals enough time to stabilize before releasing the device to run off the external crystal. The counter is reset anytime the XTOFF bit is cleared.

The status of the crystal startup counter can be determined by reading the Crystal Oscillator Warm Up Status Bit, XTUP (STATUS.4). Note that this bit will always be set upon a power–on reset, because the counter must time out before the device will resume operation. For the same reason, this bit will also be set when resuming from Stop mode with the XT/$\overline{\text{RG}}$ bit set to 1. When switching from the ring oscillator to the crystal oscillator, the XTUP bit can be used to tell when the crystal has stabilized. Attempts to switch to the external crystal before the XTUP bit has been set will be disregarded.

SWITCHING BETWEEN CLOCK SOURCES

On occasion the device may wish to switch between the ring oscillator and crystal oscillator. The device can switch to the ring oscillator at any time as there is no start up delay associated with the ring oscillator. Clearing the Crystal Oscillator/Ring Oscillator Select Bit, XT/$\overline{\text{RG}}$ (EXIF.3) will enable the ring oscillator. If there is no expectation that the crystal oscillator will be needed soon, the crystal oscillator can be disabled by setting the Crystal Oscillator Disable Bit, XTOFF (PMR.3). This will provide a significant power savings. Note that clearing the XT/$\overline{\text{RG}}$ bit does not automatically disable the crystal amplifier.

Switching the clock source from the ring oscillator to the crystal oscillator is more involved due to the startup delays inherent in the external crystal. The procedure is as follows:

1. Clear the Crystal Oscillator Disable Bit, XTOFF (PMR.3) to restart the crystal oscillator.
2. Wait for the Crystal Oscillator Warm Up Status bit, XTUP (STATUS.4) to be set, indicating that the external crystal warm up period is complete.
3. Set the Crystal Oscillator/Ring Oscillator Select Bit, XT/$\overline{\text{RG}}$ (EXIF.3) to select the crystal as the clock source.

CLOCK SOURCE AFTER RESET

Following a power–on reset, the RGSL bit is cleared and the XT/$\overline{\text{RG}}$ bit is set. This forces the device to operate from an external crystal or external clock source, regardless of the clock source prior to the event. The crystal startup counter will be reset and begin counting

down, allowing time for the crystal to stabilize before resuming operation.

In the case of external (hardware) and watchdog resets, the XT/$\overline{\text{RG}}$ bit will remain unchanged. This allows the device to continue from the same clock source that was active before the reset event. Regardless of the state of the XT/$\overline{\text{RG}}$ bit, the XTOFF bit is cleared following any reset, which begins the crystal oscillator warm up. If the crystal will not be used, the appropriate reset routines should set the XTOFF bit to disable the crystal oscillator to conserve power.

CLOCK SOURCE AFTER STOP

During Stop mode, internal clocking to the DS87C5x0 is halted. Upon receipt of an external interrupt or reset, the device will use the state of the XTOFF, XT/$\overline{\text{RG}}$, and RGSL bits prior to entering Stop mode to determine the state of the ring oscillator and crystal amplifier. The possible configurations are shown in Table 4.

CLOCK SOURCE AFTER STOP MODE DETERMINATION Table 4

XT/$\overline{\text{RG}}$	XTOFF	RGSL	CLOCK SOURCE WHEN EXITING STOP MODE	CLOCK SOURCE AFTER CRYSTAL WARM–UP PERIOD	STARTUP DELAY WHEN RESUMING?	CRYSTAL OSCILLATOR STATUS
0	0	x	Ring Oscillator	Ring Oscillator	No	Oscillator enabled
0	1	x	Ring Oscillator	Ring Oscillator	No	Oscillator disabled
1	0	0	Crystal Oscillator	Crystal Oscillator	Yes	Oscillator enabled
1	0	1	Ring Oscillator	Crystal Oscillator	No	Oscillator enabled

If the clock source before entering Stop mode is the ring oscillator, the device will resume operation using the ring oscillator and continue running from the ring oscillator after the crystal warm–up period. If the device enters Stop mode running from the crystal oscillator, the Ring Oscillator Select, Stop Mode bit, RGSL (EXIF.1) determines the clock source when resuming from Stop mode. Upon completion of the crystal warm–up period the device may continue to operate from the ring oscillator, or may switch to the external crystal or clock source. This is determined by the state of the XT/$\overline{\text{RG}}$ bit prior to entering Stop mode.

It should be noted that the crystal amplifier will begin its warm–up period automatically if the XT/$\overline{\text{RG}}$ bit was set prior to entering Stop mode. This will happen if the device was running from the external crystal or external oscillator, or if it was running from the ring oscillator but with the crystal amplifier still running. (Although not a logical choice, this is theoretically possible.) When resuming from the ring oscillator with the intent to continue from the ring oscillator, however, starting the crystal warm–up process is unnecessary. To prevent the crystal warm–up, make sure the device is operating from the ring oscillator and the XTOFF bit is set before entering Stop mode.

The ring oscillator is especially useful for systems which require short bursts of processing upon resuming from Stop mode. Operating from the ring allows the system to wake up, perform a short operation, and return to Stop mode in less time that it would require for an external crystal to stabilize. This provides a two–fold power savings: The time out of Stop mode is reduced due to the quick start of the ring oscillator, and the ring oscillator itself typically uses less power than the crystal amplifier.

RING OSCILLATOR CONSIDERATIONS

The ring oscillator used in the High–Speed Microcontroller Family is essentially a chain of inverters with a propagation delay. Although it exhibits fast start up times, it does not carry the stability of a piezo electric quartz crystal oscillator. The ring oscillator will oscillate from 3 to 4 MHz over the temperature and voltage range specified for the device. This variation makes it difficult to generate a stable time base for timers and timing–sensitive operations. Interrupt latencies will also be more difficult to calculate due to the variation of the main system clock.

It is not advised to operate the serial ports in asynchronous mode (Modes 1, 2, or 3) while running from the ring oscillator. The serial ports use internal timers to generate their baud rate, and the resulting frequency is not stable enough to support an asynchronous serial transmission. Synchronous serial transmissions in mode 0 are possible, however, due to the synchronizing clock generated by the host processor.

The use of the ring oscillator does not impair the operation of the real–time clock, watchdog timer, or Timed Access operations. The real time clock incorporated in the DS87C530 is excited by an external 32 KHz crystal which is independent of the system clock. The watchdog timer and Timed Access procedures both function with respect to internal clock cycles, not an absolute time reference, and will operate properly. If an absolute time period is required for the watchdog timer, then an external clock source is recommended.

PERFORMING A "RING–OSCILLATOR SWITCHBACK"

The switchback feature of the DS87C5x0 allows the device to "wake up" for serial port operations when operating in PMM1 or PMM2. Although the device will execute a switchback regardless of the clock source, the device must be operating from a crystal or external clock source for the serial operation to be successful. In most cases, this would preclude the use of the ring oscillator or Stop mode if serial port operations are expected. However, it is possible to "switchback" from the ring oscillator to the crystal upon receipt of a serial transmission.

In the case of a serial transmission, the ring oscillator presents little problem; the system can simply enable the crystal oscillator, wait for the crystal to stabilize, and then begin the transmission. Serial receptions are more difficult. There is no way that a DS87C5x0 operating from a ring oscillator will be able to successfully capture a serial data transfer on the first try. One possible solution would be to employ an a handshaking protocol to confirm that the receiver is ready and the data should be resent. The key in such a scheme is to have the DS87C5x0 detect that a serial operation has been attempted and execute a section of code that will switch over to the crystal source.

The recommended approach utilizes an external interrupt as a serial port activity monitor. If a negative edge triggered interrupt such as $\overline{INT0}$, $\overline{INT1}$, $\overline{INT3}$, or $\overline{INT5}$ is tied to the RX pin, the falling edge of a start bit will generate an interrupt and a switchback. The interrupt service routine will enable the crystal clock source and wait until it is stable, at which time the device will transmit a ready signal back to the originator. The following code example demonstrates one way to do this.

In addition, the delay associated with restarting the crystal can be avoided by keeping the crystal amplifier enabled when running from the ring oscillator. This seems counterproductive at first, as it increases the power consumption slightly when compared to running from the ring oscillator alone. However, when operating the device from a relatively high–speed crystal, the reduced speed of the ring oscillator still results in a net power savings.

PROGRAM EXAMPLE: SOFTWARE RING OSCILLATOR SWITCHBACK

```
;Program RING_SWB
;
;This program shows how the serial ports can be operated in conjunction with
;the ring oscillator. When receiving a byte, the falling edge of the start
;bit will generate an INT1. The INT1 ISR will restart the crystal oscillator.
;When the crystal has stabilized, the system will transmit a ready character
;to the originator to indicate that the receiver is ready.
;
;The core of the program continuously scans Port 1 and records the maximum
;value. The program performs two operations: transmit the maximum
;recorded value, and reset the maximum value to 0. Receipt of an invalid
;command will cause the device to return an error code to the host and
;switch back to the ring oscillator. Invalid codes can also be used to
;intentionally return the device to the ring oscillator at the completion
;of the data transfer. Valid commands will be echoed back to the originator
;to confirm receipt.
;****************************************************************************
;Register Equates
P0          equ     80h      ;Port 0
SP          equ     81h      ;Stack Pointer
PCON        equ     87h      ;Power Control Register
TMOD        equ     89h      ;Timer Mode Control Register
TH1         equ     8Dh      ;Timer 1 MSB (used for baud rate generation)
P1          equ     90h      ;Port 1
EXIF        equ     91h      ;External Interrupt Flag Register
SCON0       equ     98h      ;Serial Port 0 Control Register
SBUF0       equ     99h      ;Serial Port 0 Data Buffer
IE          equ     0A8h     ;Interrupt Enable Register
P3          equ     0B0h     ;Port 3
PMR         equ     0C4h     ;Power Management Register
STATUS      equ     0C5h     ;Status Register
ACC         equ     0E0h     ;Accumulator
;R0 : Command Register.
;R1 : Maximum Value observed on Port 1.

;Bit Equates
RI_0        equ     98h      ;Serial Port 0 Receiver Interrupt Flag
TI_0        equ     99h      ;Serial Port 0 Transmitter Interrupt Flag
IE1         equ     8Ah      ;Interrupt 1 Flag.
TR1         equ     8Eh      ;Timer 1 Run control.
REN_0       equ     9Ch      ;Serial Port 0 Receiver Enable
EX1         equ     0AAh     ;External Interrupt 1 Enable
EA          equ     0AFh     ;Global Interrupt Enable

;String Equates
RDY_CHAR    equ     '!'
ERR_CHAR    equ     '?'

;Interrupt Vector Table.
            cseg    at  0        ;Reset vector.
```

```
                ljmp   START
                cseg   at   13h        ;External Interrupt 1 vector.
                ljmp   EXT_INT1
                cseg   at   23h        ;Serial Interrupt 0 vector.
                ljmp   SER0_INT

                cseg   at 100h                  ;Beginning of code segment.

START:          MOV    SP, #40h        ;Initialize stack pointer.
                MOV    P3, #0Fh        ;Set port pins as inputs.
                MOV    P1, #0FFh       ;Set port pins as inputs.
                CALL   RING_ENA        ;Switch to ring oscillator to conserve power.

                MOV    TH1,   #0FDh    ;Set timer for 19200 baud rate at 11.059 MHz
                MOV    TMOD,  #20h     ;Set Timer as mode 2 for baud rate generation.
                MOV    SCON0, #50h     ;Select Mode 1, enable receiver.
                ORL    PCON,  #80h     ;Set SMOD for 19200 operation.
                SETB   TR1             ;Start Timer 1 for baud rate generation.

                MOV    IE, #94h        ;Enable global, serial 0, and ext. interrupt 1.
                MOV    R1, #0          ;Reset maximum value counter.

CLR_BUF:        MOV    R0, #0          ;This is the reentry point after command
                                       ; completion that clears the command buffer.
                                       ; It then falls through to the main prog loop.

MAIN:           CJNE   R0, #0, COMMAND ;If R0<>0, then service pending command.

PORTSCAN:       MOV    A, P1           ;Get the current port value.
                PUSH   ACC             ;Make a temporary copy of port value.
                CLR    C               ;Compare current value to maximum. If smaller,
                SUBB   A, R1           ; or equal, loop back for next check.
                POP    ACC             ;Restore port value. Note that this does not
                                       ; affect the carry flag from the SUBB inst.
                JC     MAIN            ;If negative number from SUBB, value is not
                                       ; a new maximum, so go on.
                MOV    R1, A           ;We have a new maximum. Store it.
                JMP    MAIN            ;End of main program loop

COMMAND:        CJNE   R0, #'1', CHECK_2     ;If command is not XMIT_MAX, go on.

XMIT_MAX:       MOV    A, STATUS       ;Host is requesting maximum value.  Wait until
                JB     ACC.1, XMIT_MAX ; serial port transmit activity is complete.
                MOV    SBUF0, R1       ;Send maximum value back to host.
                JMP    CLR_BUF         ;Return to main loop to await next command.

CHECK_2:        CJNE   R0, #'2',INVALID ;If command is not RESET_MAX, then an
                                        ; invalid command has been received.

RESET_MAX:      MOV    R1, #0          ;Host is requesting that maximum value
                JMP    CLR_BUF         ; be reset.  Zero value and return
```

```
                                  ; to mail loop to await next command.

INVALID:    MOV     SBUF0, #ERR_CHAR ;An invalid command has been received.
            CALL    RING_ENA         ;Return to ring oscillator and clear the
            JMP     CLR_BUF          ; command buffer. This will also be called
                                     ; intentionally by the originator to return
                                     ; the device to the ring oscillator.


;******************************************************************************
;SER0_INT - This ISR handles serial port 0 interrupts. Receiver interrupts
;           will be caused by receipt of a command byte from the originator.
;           The byte will then be echoed back to confirm receipt.
;
;           Transmitter interrupts are called by the transmission of data
;           or a status character to the originator.
;******************************************************************************
SER0_INT:   JB      TI_0, XMIT_INT   ;Determine source of interrupt.

            MOV     R0, SBUF0        ;Interrupt was serial reception. Save command
            CLR     RI_0             ; byte in R0 for main program loop and clear
                                     ; receiver interrupt flag.
            MOV     SBUF0, R0        ;Echo data to acknowledge receipt.
            RETI


XMIT_INT:   CLR     TI_0             ;Interrupt was caused by transmitter. Clear
            RETI                     ; interrupt flag and return.


;******************************************************************************
;EXT_INT1 - This ISR restarts the crystal in response to a falling edge
;           on the INT0 pin, which is also tied to the serial port 0 RX pin.
;           When crystal has stabilized, it sends a ready signal to the originator.
;           Further INT1s are disabled until the data has been received to prohibit
;           data bits from being mistaken as start bits.
;******************************************************************************
EXT_INT1:   CLR     EX1              ;Disable any more serial restart interrupts
            CALL    XTAL_ENA         ;Start crystal oscillator
            MOV     SBUF0, #RDY_CHAR ;Send Ready signal
            CLR     RI_0             ;Any serial port receiver interrupts at this
                                     ; point are erroneous, so ignore them.
            RETI


;******************************************************************************
;XTAL_ENA - This subroutine checks to see if the crystal is running, and if
;           not, enables it and waits until it has stabilized.
;******************************************************************************
XTAL_ENA:   PUSH    ACC              ;Save accumulator
            ANL     PMR, #0F7h       ;Clear XTOFF bit to restart crystal.

XTALWAIT:   MOV     A, STATUS        ;Check XTUP: Loop until crystal has stabilized.
            JNB     ACC.4, XTALWAIT
```

```
            ORL       EXIF, #08h        ;Switch to crystal as clock source.
            SETB      REN_0             ;Crystal is active, enable receiver.
            POP       ACC               ;Restore Accumulator.
            RET                         ;Device is now running from crystal. Exit.


;*******************************************************************
;RING_ENA - This subroutine checks to see if there is any serial port
;           activity, and if not, switches back to the ring oscillator.
;*******************************************************************
RING_ENA:   PUSH  ACC                   ;Save accumulator


WAIT_SERIAL:
            MOV       A, STATUS         ;Test lower nibble of status reg for serial
            ANL       A, #0Fh           ; port activity.  If serial ports are still
            JNZ       WAIT_SERIAL       ; active, wait before switching to ring.

            CLR       REN_0             ;Ignore any serial port activity while running
                                        ; from ring oscillator.
            CLR       IE1               ;Clear any outstanding serial interrupts that
            SETB      EX1               ; may have been generated by serial data and
                                        ; reenable external interrupt 1 to detect the
                                        ; start of another serial transfer.

            ANL       EXIF, #0F7h       ;Clear XT/RG to enable ring oscillator.
            ORL       PMR, #08h         ;Set XTOFF bit to disable crystal.

            POP       ACC               ;Restore Accumulator.
            RET                         ;Device is now running from ring. Exit.
```

DEVELOPING A POWER MANAGEMENT FRAMEWORK

The Dallas Semiconductor approach to power management allows system designers to reduce power consumption while maintaining maximum performance. To achieve the maximum possible savings, the device operating conditions should be carefully analyzed and a power management scheme developed.

The determination of which power management modes to utilize, when to switch modes, and how to handle high–priority tasks is application dependent. No single approach will suit all possible combinations. In general, the selection of clock speeds and sources depends on the assigned tasks, and the need for timing–dependent operations such as serial ports activity.

There are two basic classes of systems which employ power management. The first is systems which hibernate or spend almost all of their operating time in a standby state, such as Stop or PMM2. These systems are often used in unattended systems to gather data or as environmental monitors. They are characterized by relatively infrequent I/O activity at specific intervals. The second class of systems usually performs a high rate of I/O activity on several devices, or otherwise must be operating constantly. A hibernation approach would be impractical in this instance as the device would spend most of its time simply restarting from the low power state. These two approaches are discussed in more detail in the following sections.

BURST MODE OPERATION

A common mode of operation is to have the device operate in a low power state, perform a brief task, and then place the device back into a low power state until another event occurs. Actions such as a keypad press, or card reader activity fall within this category. Such peripherals typically generate an external interrupt which executes a switchback or resume from Stop mode.

The decision on what to designate as the standby state depends on the type of activity that will initiate a return to the active state. If serial port activity is expected, then the standby state must be one that can receive serial data, such as PMM2. A system which can tolerate longer interrupt latencies can use Stop mode as the low power state.

An examination of the power consumption in the various modes shows that when using a burst-mode approach, the most power savings can be gained by operating in divide by 4 mode, rather than the PMMs. Divide by 4 mode gives 16 times the performance of PMM1, but consumes only four times the current. The higher power/performance ratio of divide by 4 mode means that less total energy will be consumed during the subroutine. As a result, routines that wake from Stop mode to perform short bursts of activity and then return to Stop should do so in divide by 4 mode.

PROGRAM EXAMPLE: REMOTE DATA LOGGER

The following program illustrates a generic scheme for running a remote, battery–powered data–logging device which requires only sporadic use. In this example, a DS87C530 "hibernates" in Stop mode until a keypad is pressed and performs some operation. After the operation is performed, the device will return to Stop mode. This device runs off the ring oscillator during most of its awake state, unless a serial transfer is expected or in progress. Periodically a real–time clock interrupt will cause the device to acquire data from an external source, record it in the on–chip SRAM, and return to Stop mode. Figure 7 shows the state diagram for how the device works. Although this example uses the internal real time clock of the DS87C530, it can be easily modified to work with an internal timer in PMM2 or external real time clock.

REMOTE DATA LOGGER EXAMPLE STATE DIAGRAM Figure 7
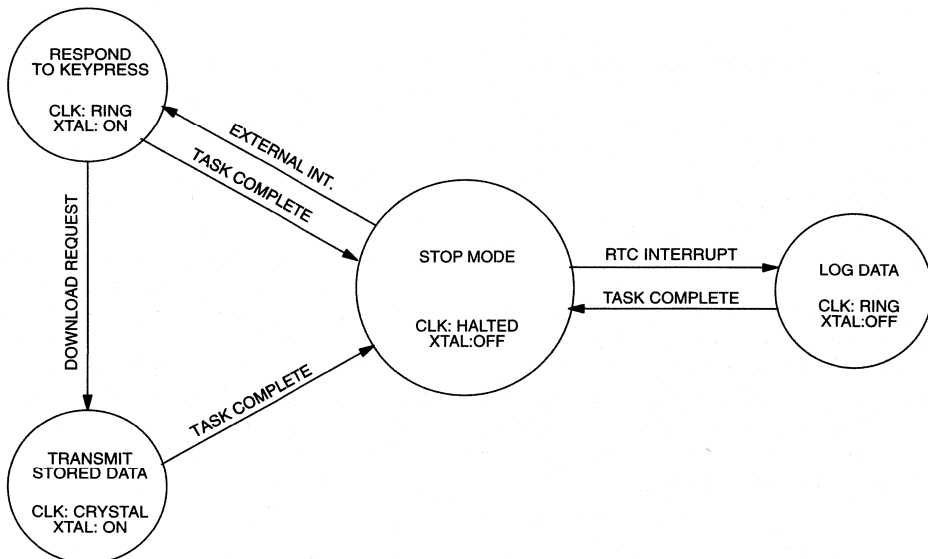
PROGRAM EXAMPLE: BURST–MODE DATA LOGGER

```
;*******************************************************************************
;Program DATA_LOG
;
;This program demonstrates burst mode operation in a data logger device. The
;device remains in Stop mode until a real time clock or external interrupt
;resumes operation. When an operation is complete, the device will switch
;back to the ring oscillator and return to Stop mode.
;
;The real time clock will interrupt the system twice per hour to read a value
;from port 1, which it will store until requested. The interrupt is called
;on the hour to start the data acquisition. The routine presented is simple and
;generic, so the data could be from a D/A converter or a temperature sensor,
;for example.
;
;Upon receiving an external interrupt, the device will start the crystal
;amplifier in expectation of possible serial port activity, but continue to
;operate from the ring oscillator. If serial activity is detected or desired,
;the device will hold operation until the crystal has warmed up, and then
;switch operation to it.
;*******************************************************************************
;Register equate table
P0        equ    80h    ;Port 0 Latch
SP        equ    81h    ;Stack Pointer
DPL       equ    82h    ;Data Pointer 0 Low Register
DPH       equ    83h    ;Data Pointer High Register
DPL1      equ    84h    ;Data Pointer 1 Low Register
DPH1      equ    85h    ;Data Pointer High Register
DPS       equ    86h    ;Data Pointer Select Register
PCON      equ    87h    ;Power Control Register
TCON      equ    88h    ;Timer Control Register
TMOD      equ    89h    ;Timer Mode Register
TH1       equ    8Dh    ;Timer 1 MSB
P1        equ    90h    ;Port 1 Latch
EXIF      equ    91h    ;External Interrupt Flag Register
SCON0     equ    98h    ;Serial Port 0 Control Register
SBUF0     equ    99h    ;Serial Port 0 Data Buffer
P2        equ    0A0h   ;Port 2 Latch
IE        equ    0A8h   ;Interrupt Enable Register
P3        equ    0B0h   ;Port 3 Latch
IP        equ    0B8h   ;Interrupt Priority Register
PMR       equ    0C4h   ;Power Management Register
STATUS    equ    0C5h   ;Status Register
TA        equ    0C7h   ;Timed Access Register
ACC       equ    0E0h   ;Accumulator
EIE       equ    0E8h   ;Extended Interrupt Enable Register
RTASS     equ    0F2h   ;Real Time Alarm SubSecond Register
RTAS      equ    0F3h   ;Real Time Alarm Second Register
RTAM      equ    0F4h   ;Real Time Alarm Minute Register
EIP       equ    0F8h   ;Extended Interrupt Priority Register
RTCC      equ    0F9h   ;Real Time Clock Control
```

```
;Bit equate table
RI_0        equ    98h    ;Serial Port 0 Receiver Interrupt Flag
TI_0        equ    99h    ;Serial Port 0 Transmitter Interrupt Flag
EX1         equ    0AAh   ;External Interrupt 1 Enable bit
F0          equ    0D5h   ;General purpose flag.

;Constant equate table
DATA_TABLE equ     0000h ;Put data log table at start of SRAM

            cseg   at  0          ;Reset vector.
            ljmp   START
            cseg   at  13h        ;External Interrupt 1 vector.
            ljmp   EXT_INT1
            cseg   at  23h        ;Serial Interrupt 0 vector.
            ljmp   SER_INT0
            cseg   at  6Bh        ;Real time clock Interrupt vector.
            ljmp   RTC_INT
;
            cseg   at    100H     ;Beginning of code segment.
START:
            MOV    SP,    #80h    ;Set up stack pointer.

            MOV    P1,    #0FFh ;Set port 1 as inputs.
            MOV    P3,    #0Bh    ;Set RXD0, TXD0 & INT1 as inputs.
            MOV    PMR,   #01h    ;Select on-chip SRAM.
            MOV    RTAM,  #00h    ;Set minute, second, and subsecond alarms.
            MOV    RTAS,  #00h    ; Alarm will wake device every hour on the hour
            MOV    RTASS, #00h    ; to initiate temperature gathering.

            MOV    TA,    #0AAh ;Timed access write to enable minute, second,
            MOV    TA,    #55h    ; and subsecond compares.
            ORL    RTCC,  #0C1h

            MOV    SCON0, #050h ;Set serial port 0 for Mode 3
            MOV    TH1,   #0E6h ;Timer 1 reload value for 2400 baud at 24 MHz.
            MOV    TMOD,  #20h   ;Set timer 1 to 8-bit auto reload and start it.
            MOV    TCON,  #40h

            MOV    IP,    #10h   ;Serial port 0 and RTC high priority interrupts
            MOV    EIP,   #20h   ; so they can interrupt Ext. interrupt 1 routine.
            MOV    EIE,   #20h   ;Enable Serial port 0, Ext. Int. 1. and
            MOV    IE,    #94h   ; RTC interrupts.

;**************************************************************************
;This is the main program loop. It does nothing but wait for interrupts, and
; when they are complete, switches back to the ring oscillator and puts the
; part back into Stop mode.
;**************************************************************************

MAIN:       ANL    EXIF, #0F7h ;Switch to ring.
```

```
            ORL   EXIF, #02h   ;Enable restart from ring.
            ORL   PMR, #08h    ;Disable crystal.
            ORL   PCON, #02h   ;Set the STOP bit to halt the device.

            JMP   MAIN         ;End of main program loop

;****************************************************************************
;SER_INT0 - This ISR handles serial port 0 interrupts. Serial port interrupts
;           will only be possible when the device is "active" following a
;           keypress.  The primary function of this interrupt is to transmit
;           the next character in the table until all data has been sent.
;****************************************************************************
SER_INT0:   JB    TI_0,XMIT_INT ;Test for transmit or receive interrupt.

            CLR   RI_0         ;This example does not receive serial data.
            RETI               ; This code is included for completeness.

XMIT_INT:   CLR   TI_0         ;Clear transmit interrupt and send next byte.
            MOV   A, DPL1       ;Check to see if we are at the end of the
            CJNE  A, DPL, NOT_END ; data.  If DPTR0 and DPTR1 are same, then
            MOV   A, DPH1       ; then all the data has been sent.
            CJNE  A, DPH, NOT_END
            SETB  F0           ;We have reached the end of the table.
            RETI               ; Set completion flag and exit.

NOT_END:    PUSH  DPS          ;Preserve current data pointer.
            MOV   DPS, #01h     ;Switch to DPTR1 to track data pointer.
            MOVX  A, @DPTR      ;We still have data, so transmit it, restore
            MOV   SBUF0, A      ; data pointer, and return to send next byte.
            POP   DPS
            RETI


;****************************************************************************
;EXT_INT1 - This ISR is generated by activity on a keypad. It causes the
;           device to read a command on Port 0 and take the appropriate action.
;           This simple example performs two functions:
;           1. Download stored data to host through serial port 0.
;           2. Clear the data table by resetting the data pointer.
;           If the command is to download stored data, it will switch to the
;           crystal first. The F0 flag is used to indicate when all the data
;           has been sent. This prevents the software from exiting the ISR
;           and reentering Stop mode before all the data has been transmitted.
;****************************************************************************
EXT_INT1:   ANL   PMR, #0F7h        ;Enable crystal for possible serial activity.

            MOV   A, P0            ;Read the data on Port 0 and take
            CJNE  A, #00h, CHECK_2  ; appropriate action.
            JMP   DNLOAD
CHECK_2:    CJNE  A, #01h, INVALID

CLEAR:      MOV   DPTR, #DATA_TABLE ;Zero all of on-chip SRAM space from 0-3FFh.
```

```
                    MOV     A,#0h                ; Reset data pointer. Use A for faster fill.
    NEXT_LOC:       MOVX    @DPTR, A             ;Fill location & increment to next one.
                    INC     DPTR
                    MOV     R0, DPH              ;If DPH is not 04, then do next location.
                    CJNE    R0, #04h, NEXT_LOC

                    MOV     DPTR, #DATA_TABLE    ;On-chip SRAM has been cleared. Reset
                    RETI                         ; pointer to beginning of table and return.

    DNLOAD:         MOV     A, STATUS            ;Wait until crystal has stabilized.
                    JNB     ACC.4, DNLOAD
                    ORL     EXIF, #08h           ;Switch to the crystal.

                    MOV     A,#'!'               ;Transmit starting character. Remaining
                    MOV     SBUF0, A             ; data will be sent by serial port
                    JNB     F0, $                ;Loop here until entire table is transmitted.
                    CLR     F0                   ;Transmit complete. Clear completion flag.

                    MOV     DPS, #01h            ;Switch to DPTR1 and reset transmit pointer.
                    MOV     DPTR, #DATA_TABLE
                    MOV     DPS, #0h             ;Switch back to DPTR0 to log data.
                    ANL     EXIF, #0F7h          ;Switch back to ring oscillator and return

    INVALID:        RETI                         ; to Stop mode.

;********************************************************************************
;RTC_INT -  This ISR is used to read a value from port 1. It is called every
;           30 minutes, and logs data to the data buffer. When done, it will
;           return to the main loop where it will enter Stop mode again.
;           This simple example assumes that the device will be read before
;           the data overflows the on-chip SRAM, so no error checking is
;           included.
;********************************************************************************
    RTC_INT:        ANL     RTCC, #0FDh  ;Clear RTC Interrupt flag.
                    PUSH    ACC          ;Save accumulator.
                    PUSH    DPS          ;Save data pointer, in case we are interrupting
                    MOV     DPS,#0h      ; data download, and switch to data pointer 0.
                    MOV     A, P1        ;Read data from port 1, store it in data table.
                    MOVX    @DPTR, A
                    INC     DPTR         ;Point to next location.
                    XRL     RTAM,#1Eh    ;Toggles RTC alarm minute register between 0 &
                                         ; 30 to interrupt on hour and half hour.
                    POP     DPS          ;Restore data pointer selector and accumulator.
                    POP     ACC
                    RETI
```
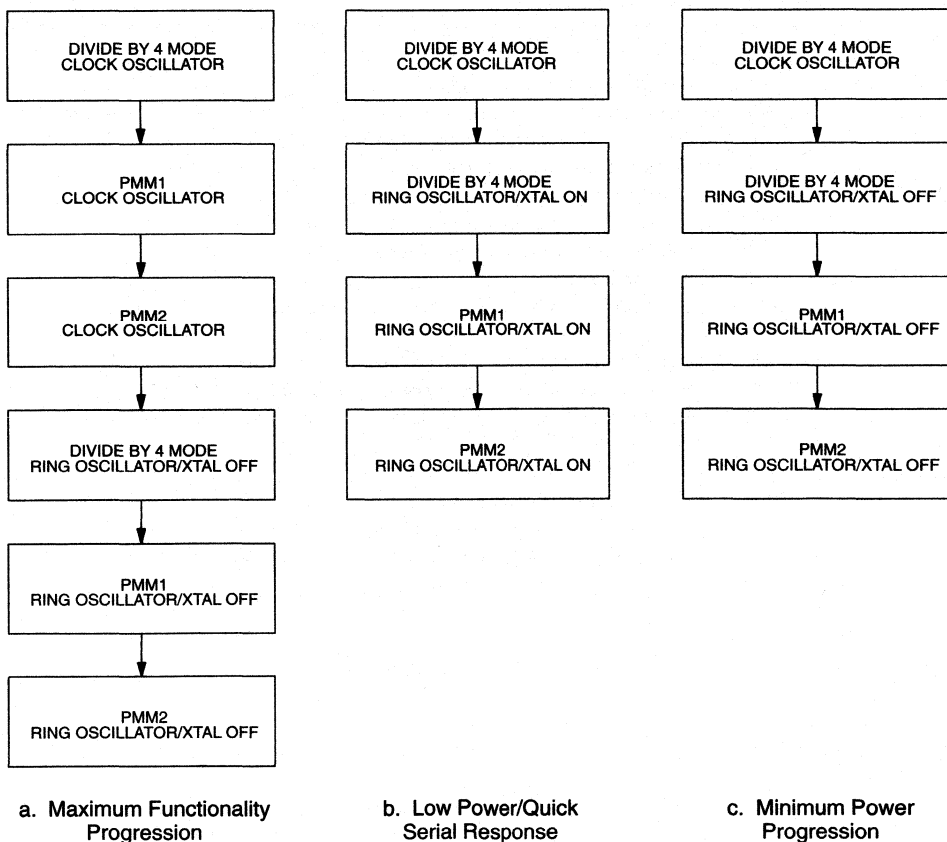
GRADUAL POWER DOWN

Often, a device will need to be operating constantly during periods of little activity. This may be to monitor system status, or to time critical events. While such a system can tolerate some performance degradation for the sake of power savings, it cannot halt operation by using the Stop mode. The Power Management modes used in the DS87C5x0 allow software to gradually reduce system performance based on the amount and type of tasks. This approach is similar to that used by personal computers, where inactivity for a specified period of time will cause the system to reduce its speed to the next lower level of performance.

The decision of when to switch modes, and which modes to switch between is dependent on the user's application. Constructing a "power path" is the simplest way to determine what speeds and clock source are appropriate. For a relatively simple system, only a few states are needed. The power management modes PMM1 and PMM2 were specifically designed to be part of such a gradual power reduction.

Figure 8 demonstrates a number of power paths possible with the DS87C5x0 power management capabilities. Figure 8a shows a relatively complicated scheme which performs a gradual reduction in operating speed, while keeping the clock oscillator operating as long as possible to perform timing–dependent functions. Figure 8b switches to the ring oscillator and gradually reduces the speed of the device, but keeps the crystal amplifier enabled in case the device needs to quickly respond to serial port activity. Figure 8c provides the lowest power consumption by switching to the ring oscillator and disabling the crystal amplifier.

SAMPLE POWER PATHS Figure 8

a. Maximum Functionality Progression

b. Low Power/Quick Serial Response

c. Minimum Power Progression

PROGRAM EXAMPLE: SYSTEM MONITOR

The following program illustrates a basic scheme for operating a device that constantly monitors the state of a system. It operates similar to the way that a personal computer manages its power; if no activity is detected in a specified period of time it switches to the next lower power saving mode.

The program monitors an peripheral on port 1, and updates a display mapped into external memory as needed. The watchdog timer is used to poll the status of the system, which is indicated by the F0 flag. If there is no activity before the timer times out, the device will continue to decrease its speed. Because the watchdog timer period is affected by the speed of the device, the watchdog divide ratio is adjusted to keep as constant an interval as possible. The methods demonstrated could also be used to detect a spurious switchback caused by noise on the serial port and return the device to PMM.

PROGRAM EXAMPLE: GRADUAL POWER DOWN

```
;**********************************************************************
;Program GRADUAL.ASM
;
;This program shows a gradual version of power management. The watchdog timer
;is used to periodically check the F0 flag to see if any activity has
;occurred since the last timer interrupt. If no activity has occurred,
;then the device will switch the clock to the next lower level of operation.
;This example also demonstrates how an external interrupt would be used in
;conjunction with the watchdog timer.
;**********************************************************************
;Register equate table
CKCON        equ    8Eh     ;Clock Control Register
P1           equ    90h     ;Port 1
IE           equ    0A8h    ;Interrupt Enable Register
PMR          equ    0C4h    ;Power Management Register
STATUS       equ    0C5h    ;Status Register
TA           equ    0C7h    ;Timed Access Register
WDCON        equ    0D8h    ;Watchdog Control Register
ACC          equ    0E0h    ;Accumulator
EIE          equ    0E8h    ;Extended Interrupt Enable Register

OLD_VAL      equ    0CDh    ;Previous value from external device.
                           ;Note that this location is normally TH2, but
                           ;we are using it for direct register instruction.

;Bit definition table
RWT          equ    0D8h    ;Reset watchdog timer bit.
F0           equ    0D5h    ;General purpose user flag.
WDIF         equ    0DBh    ;Watchdog interrupt flag.
EWDI         equ    0ECh    ;Watchdog interrupt enable.

;Definition equate table.
DISPLAY      equ    2000h ;External location of display.

             cseg   at     0       ;Reset vector.
             ljmp   START

             cseg   at     63h     ;Watchdog Timer Interrupt vector.
```

```
            ljmp   WDOG_INT
;
            cseg   at     100h    ;Beginning of code segment.

START:      MOV    TA, #0AAh      ;Timed access
            MOV    TA, #55h
            SETB   RWT            ;Reset watchdog timer

            MOV    DPTR, #DISPLAY ;Set data pointer to location of display.

            CLR    F0             ;Clear activity flag
            MOV    CKCON, #0C1h   ;Set watchdog divide ratio to 2**26.
            MOV    EIE, #10h      ;Watchdog Interrupt Enable.
            MOV    IE,  #80h      ;Enable all interrupts.
            MOV    OLD_VAL,P1

MAIN:       MOV    A, P1          ;Examine external parameter.
            CJNE   A, OLD_VAL, DIFF ;Begin display update if different.
            JMP    MAIN

DIFF:       MOV    CKCON, #0C1h   ;Return watchdog divide ratio to slow speed.
            MOV    PMR, #041h     ;Switch back to divide by 4 mode.
            MOV    OLD_VAL, A     ;Save new value.
            MOVX   @DPTR, A       ;Put new value to display.
            SETB   F0             ;Set flag to indicate we have activity.
            SETB   EWDI           ;Reenable watchdog interrupt (in case device
                                  ; was operating in /1024)
            JMP    MAIN           ;End of main program loop

;********************************************************************************
;WDOG_INT - This ISR periodically tests for activity, and if none has been
;           detected, drops the device to the next lower clock speed.
;           Because the watchdog interval is a function of the clock divisor,
;           the watchdog divide ratio is modified in each mode to keep a
;           relatively constant interrupt frequency. The following table
;           shows the frequencies, assuming a crystal speed of 25 MHz.
;           When the device enters /1024 mode, it disables the watchdog
;           interrupt, because there is no slower speed to enter.
;
;           Clock Mode        Divide Ratio   WD1 WD0    Watchdog Timeout
;           Divide by 4       2**26           1   1     2684 mS
;           PMM1 (/64)        2**23           1   0     5368 mS
;********************************************************************************
WDOG_INT:   JB     F0, DONEWDOG   ;If activity has been detected, do not change
                                  ; speed.
            PUSH   ACC            ;Save accumulator.

            MOV    A,  PMR        ;Check current speed.
            JNB    ACC.7, D4TO64  ;If CD1 = 0, then mode is /4, switch to /64.

D64TO1024: MOV    PMR, #041h     ;Speed is now /64. Change to /1024 by first
```

```
           MOV    PMR, #0C1h      ; going from /64 to /4 and then from /4 to /1024.
           CLR    EWDI            ;Since we are now in /1024 there is no need
                                  ; to go slower, so disable watchdog interrupt.

D1024:     POP    ACC             ;Restore Accumulator.
DONEWDOG:  CLR    F0              ;Clear activity flag.
           MOV    TA, #0AAh       ;Timed access to clear Watchdog Interrupt flag.
           MOV    TA, #55h
           CLR    WDIF
           RETI                   ;Exit

D4TO64:    MOV    A, STATUS       ;Speed is now /4. Change to /64. Because we
           ANL    A, #0CFh        ; are entering PMM, test for activity. Check all
           JNZ    D1024           ; bits in Status Register except XTUP, and LIP
                                  ; because the watchdog interrupt is low priority.
                                  ; If any activity bit is set, abort speed change.
           MOV    PMR, #081h      ;There is no activity. Change clock from to /64.
           MOV    CKCON, #81h     ;Change watchdog divide ratio from 2^26 to 2^23.
           JMP    D1024
```

BAND–GAP DISABLING

The band–gap reference, used to detect a power failure, draws approximately 150 µA. During Stop mode, this can be an appreciable amount of the total current drawn. The DS87C5x0 supports the option of disabling the band–gap reference, eliminating the associated current drain. When disabled, the device loses the ability to generate a power–fail interrupt or a power–fail reset. The device will continue to operate until V_{CC} drops below V_{RST}, at which time the device will cease operation. Without the bandgap reference, the device has no way of detecting an imminent power loss, or performing an orderly shutdown. When power resumes, the device will perform a power–on reset.

Setting the Band–Gap Select bit, BGS (EXIF.0) enables the band–gap reference during Stop mode. The default or reset condition is with the bit cleared, and the band–gap disabled during Stop mode. Note that this bit can be only changed using a timed access write. It has no control of the reference during full power, PMM, or Idle modes.

DALLAS
SEMICONDUCTOR

Application Note 79
Using the DS87C530 Real Time Clock

OVERVIEW

The DS87C530 incorporates a real time clock (RTC) and alarm to allow the user to perform real–world timing operations such as time–stamping an event, performing a task at a specific time, or executing very long timing delays. Although software timing loops or internal timers could be used for such measurements, they are crystal dependent, inefficient for long time measurements, and are incompatible with the use of power management modes. Integration of the RTC onto the DS87C530 means that only a 32.768 KHz crystal is required. No load capacitors are required with the RTC crystal. The RTC is controlled by dedicated Special Function Registers (SFRs).

The DS87C530 RTC consists of subsecond, second, minute, hour, day of the week, and two total day count registers. In addition there is an alarm register for the subsecond, seconds, minutes, and hours registers. The subsecond register provides a resolution of 1/256 of a second, and a maximum rollover count of 1 second. The registers and control bits used by the RTC are shown in Table 1. Bits and registers designated as unchanged after a reset may be indeterminate following a no–battery reset. Consult the full bit or register description for complete details.

Both user software and the internal clock directly write and read the RTC time registers (RTCSS, RTCS, RTCM, RTCH, RTCD0, RTCD1). To prevent the possibility of both user software and the internal timer accessing the same register simultaneously, the DS87C530 incorporates a register locking mechanism. Updates to the RTC time registers by the internal timer are temporarily suspended for up to 1 ms during software read or write operations. If a subsecond timer tick should occur in the 1 ms window, it will be processed immediately as soon as either the RTCWE or RTCRE bits are cleared. To prevent the possibility of an accidental write to the RTC time registers, the RTCWE bit should be cleared as soon as the planned modifications are complete. As a protective measure, the device will clear the RTCWE bit automatically after 1 ms if it has not been cleared in software. To allow any pending timer ticks to be processed, software must wait four machine cycles between any successive modifications of the RTCWE or RTCRE bits.

This scheme will not affect the accuracy of the RTC, as any subsecond timer tick that may occur during the read or write window is only temporarily delayed, not discarded. Only the recognition of that single subsecond timer tick is delayed, and subsequent ticks will be synchronized with the clock. The only possible implication with respect to RTC operation occurs if a timer tick that would cause an alarm interrupt occurred during a time register read operation. In that case, the alarm would be delayed a fraction of a millisecond until the RTCRE bit was cleared. As mentioned, the next subsecond timer tick will occur at the proper time, so the long–term clock accuracy will not be affected.

It is critical that the 4 machine cycle setup and 1 ms window timings be observed. Any reads from the time registers before the 4 machine cycle period may return an invalid time. Writes to the time registers before the 4 machine cycle period will be ignored. Similarly, any RTC time register operations outside of the 1 ms window will result in invalid read operations or ignored write operations. For this reason, interrupts should be globally disabled before modifying any RTC register.

101994 1/22

REAL TIME CLOCK CONTROL AND STATUS BIT SUMMARY Table 1

BIT NAME	LOCATION	FUNCTION	RANGE	RESET	READ/WRITE ACCESS
ERTCI	EIE.5	RTC Interrupt Enable		0	Unrestricted
PRTCI	EIP.5	RTC Interrupt Priority		0	Unrestricted
RTASS.7–0	RTASS	RTC Alarm Subsecond	0–FFh	Unchanged	Unrestricted
RTAS.5–0	RTAS	RTC Alarm Second	0–3Bh	Unchanged	Unrestricted
RTAM.5–0	RTAM	RTC Alarm Minute	0–3Bh	Unchanged	Unrestricted
RTAH.4–0	RTAH	RTC Alarm Hour	0–17H	Unchanged	Unrestricted
RTCSS.7–0	RTCSS	RTC Subsecond	0–FFh	Unchanged	Read: only if RTCRE=1. Cannot be written. Cleared when RTCWE 1 $\geq$ 0
RTCS.5–0	RTCS	RTC Second	0–3Bh	Unchanged	Read: only if RTCRE=1. Write: only if RTCWE=1. 1 ms Read/Write window
RTCM.5–0	RTCM	RTC Minute	0–3Bh	Unchanged	
RTCH.4–0	RTCH.4–0	RTC Hour	0–17h	Unchanged	
DOW2–0	RTCH.7–5	RTC Day of Week	0–7h	Unchanged	
RTCD1.7–0 RTCD0.7–0	RTCD1, (MSB) RTCD0, (LSB)	RTC Day	0–FFFFh	Unchanged	
SRCE	RTCC.7	RTC Subsecond Compare Enable		Unchanged	Unrestricted
SCE	RTCC.6	RTC Second Compare Enable		Unchanged	Unrestricted
MCE	RTCC.5	RTC Minute Compare Enable		Unchanged	Unrestricted
HCE	RTCC.4	RTC Hour Compare Enable		Unchanged	Unrestricted
RTCRE	RTCC.3	RTC Read Enable		0	Unrestricted
RTCWE	RTCC.2	RTC Write Enable		0	Read: Unrestricted Write: Timed Access
RTCIF	RTCC.1	RTC Interrupt Flag		0	Unrestricted
RTCE	RTCC.0	RTC Enable		Unchanged	Read: Unrestricted Write: Timed Access
E4K	TRIM.7	External 4096 Hz RTC Signal Enable		0	
X12/$\overline{6}$	TRIM.6	RTC Crystal Capacitance Select		Unchanged	
TRM2–0	TRIM.5 TRIM.3 TRIM.1	RTC Trim Bit 2–0		Unchanged	Read: Unrestricted Write: Timed Access
$\overline{\text{TRM2–0}}$	TRIM.4 TRIM.2 TRIM.0	RTC Inverted Trim Bit 2–0		Unchanged	Read: Unrestricted Write: Timed Access, must be inverse of TRM2–0

STARTING AND STOPPING THE RTC

The operation of the RTC crystal amplifier is controlled by the RTC Enable bit, RTCE (RTCC.0). This bit can only be accessed by a Timed Access procedure, and is unaffected by any operational reset. The state of the RTCE bit is undefined after a no–battery reset, however, and should be initialized. Clearing the RTC Enable bit will halt operation of the crystal amplifier and the clock, but all register values (including the time when the clock was disabled) will be retained. This may be desirable to preserve the life of the backup energy source during periods of storage. When restarting the RTC crystal oscillator, either from a no–battery reset condition or by setting the RTC Enable bit, the crystal start–up time must be observed. There is no direct way to detect when the RTC crystal oscillator has stabilized, and the system software must allow sufficient stabilization time when restarting the RTC. Crystal startup times are specified by the crystal manufacturer, but are usually on the order of 1 second.

After a loss of battery power or when attaching a battery for the first time it will be necessary to initialize the RTC. Although there is no status bit to indicate a no–battery reset, there are several ways to detect when the real time clock has lost power / time. The best way is to monitor a reserved location in on–board memory. Because the DS87C530 on–chip SRAM contents are preserved by the same energy source as the RTC, an unexpected change in a previously loaded memory location can indicate a loss of battery power.

READING THE TIME

Reading the current time from the RTC is accomplished by the following procedure:

1. Disable all interrupts by clearing the EA bit (IE.7),
2. Set the RTCRE bit (RTCC.3),
3. Wait 4 machine cycles,
4. Read the appropriate register(s) within 1 ms of RTCRE being set,
5. Clear the RTCRE bit (RTCC.3),
6. Enable interrupts by setting the EA bit (IE.7).

SETTING THE TIME

The time on the DS87C530 is set by writing to the Clock Registers. The Second, Minute, Hour, Day of the Week, and Day Count can be set by writing to the respective registers. It is not possible to set the Real Time Clock

Subsecond Register (RTCS; FBh). This register is automatically reset to 00h when the RTCWE bit is cleared, either through software or the automatic time–out of the 1 ms write window. The procedure for setting an RTC time register is as follows:

1. Disable all interrupts by clearing the EA bit (IE.7),
2. Perform a Timed Access procedure,
3. Set the RTCWE bit (RTCC.2),
4. Wait 4 machine cycles,
5. Write the appropriate register(s) within 1 ms of RTCWE being set,
6. Perform a Timed Access procedure,
7. Clear the RTCWE bit (RTCC.2),
8. Enable interrupts by setting the EA bit (IE.7).

USING THE RTC ALARM

The RTC alarm function is used to generate an interrupt when the RTC value matches selected alarm register values. An alarm can be triggered by a match on one or more of the following alarm registers: Subsecond (RTASS;F2h), Second (RTAS; F3h), Minute (RTAM; F4h), and Hour (RTAM; F5h). Note that there is no alarm register associated with the RTC Day or Day of Week Registers. If an alarm is desired on a specific date, an alarm can be executed once a day and user software can compare the current date against the Day Register. It is not necessary to set the RTC Write Enable bit when setting the alarm registers.

The alarm can be set to occur on a match with any or all of the alarm registers. An alarm can occur on a unique time of day, or a recurring alarm can be programmed every subsecond, second, minute, or hour. The specific alarm registers to be compared are selected by setting or clearing the corresponding compare enable bits (RTCC.7–4). Any compare bit which is cleared will result in that register being treated as a 'Don't Care' when evaluating alarm conditions. Clearing all the compare enable bits will disable the ability of the RTC to cause an interrupt, and will immediately clear the RTC Interrupt Flag (RTCC.1). Unlike some interrupts, the RTC flag is not cleared by exiting the RTC interrupt service routine and must be done in software.

The general procedure for setting the RTC alarm registers to cause a RTC interrupt is as follows:

1. Clear the RTC Interrupt Enable bit (EIE.5),

2. Clear all RTC Alarm Compare enable bits (ANL RTCC, #0Fh),

3. Write one or more RTC Alarm registers,

4. Set the desired RTC Alarm Compare enable bits.

5. Set the RTC Interrupt Enable bit (EIE.5).

Setting the alarm to cause an interrupt for a single time during a 24–hour period is done by setting all the alarm registers to the desired value and enabling all compare bits. For example, if an alarm was desired at 11:45:00 am, the following configuration would be used:

Alarm Subsecond (RTASS)	00 subseconds	= 00h
Alarm Second (RTAS)	00 seconds	= 00h
Alarm Minute (RTAM)	45 minutes	= 2Dh
Alarm Hour (RTAH)	11 hours	= 0Bh
Clock Control (RTCC)	subsecond compare	= F1h
	second compare	
	minute compare	
	hour compare	
	RTC enable	

A recurring alarm is enabled by disabling the compare enable bits associated with one or more alarm registers. In general, a recurring alarm is set using the next lower time increment. For example, if an alarm once an hour was desired, a compare on the RTAM Register would be performed, because the RTCM register will match RTAM register only once an hour. For example, if an alarm once an hour, on the half hour was desired, the following configuration would be used:

Alarm Subsecond (RTASS)	00 subseconds	= 00h
Alarm Second (RTAS)	00 seconds	= 00h
Alarm Minute (RTAM)	30 minutes	= 1Eh
Alarm Hour (RTAH)	11 hours	= 00h
Clock Control (RTCC)	subsecond compare	= E1h
	second compare	
	minute compare	
	RTC enable	

In the above example, the subsecond, second, and minute registers are programmed and the corresponding compare enable bits are set, even though only a match on the minute register is desired. This is because a don't care is always treated as a match for the purposes of evaluating alarms. If the SSCE and SCE bits were cleared to 0 (don't care) in the above example, then a match (and interrupt) would occur during every subsecond of the minute in which the RTAM register matched. This would cause 15,360 interrupts, which is most likely

not the desired effect. In general, when specifying a recurring alarm all the compare bits below the largest time increment should be enabled and the corresponding alarm registers loaded with 00h or a known value.

Alarms can occur synchronously when the clock rolls over to match the alarm condition or asynchronously if the alarm registers are set to a value that matches the current time. Note that only one alarm may occur per subsecond tick. This means that if a synchronous alarm has already occurred during the current subsecond, software cannot cause an asynchronous alarm in the same subsecond.

While this is a relatively minor point, it can have implications if software expects to use the asynchronous capabilities of the alarm. For example, assume an RTC interrupt occurs as when the alarm registers match the current time a 01:00:00:00 (1 hour, 0 minutes, 0 seconds, 0 subseconds. The RTC interrupt is relatively short, taking much less than one subsecond tick (<4 ms), and execution returns to the main program. Immediately upon exiting the RTC interrupt routine, an event occurs that requires software to cause an alarm on the hour by setting the alarm to match on 00 minutes, 00 seconds, 00 subseconds. Normally, setting this alarm condition with the time at 01:00:00:00 would immediately cause an RTC interrupt to occur; but because we have already had an alarm in this subseconds, the condition will not be recognized. The alarm will be missed because it will not be evaluated until the next subsecond tick, when the time will have changed to 01:00:00:01. The designer should guard against the possibility of using synchronous asynchronous alarms in the same subsecond.

Because an alarm condition can occur asynchronously, care must be exercised that a match is not accidentally enabled while writing to the alarm registers. For example, assume that the current time is 0B:00:00:00 and the current alarm conditions are 00:00:00:00. Suppose that software changes the alarm to 0B:01:00:00. If the hour, second, minute, and subsecond compare enables are enabled and the first instruction is MOV RTCH, #0B0H, an alarm will occur immediately instead of at the intended time. The best way to avoid this is to disable all compare enables before changing the RTC alarm registers.

RTC SOFTWARE TRICKS

There are a number of simple tricks that can be used to simplify software associated with RTC operations. The 4 machine cycle delay can be performed using a CJNE A,A,$ instruction. Compared to using 4 NOPs, this is a single instruction, and is 1 byte shorter.

The DS87C530 RTC allows software to dynamically vary the alarm registers to achieve a wide range of intervals. Often software will want to interrupt regularly on half–increments of time (every 30 seconds, 30 minutes, etc.). This can be easily done using the XRL instruction. For example, if the RTAM register is set to 00h, the instruction XRL RTAM, #1Eh will change the contents to 1Eh. Performing the instruction again will change it back to 00h. Placing this instruction at the start of the RTC interrupt routine will cause the appropriate alarm register to be easily and quickly modified each time the interrupt is called.

PROGRAM EXAMPLE: DATALOGGER

The following program illustrates a generic scheme for operating a remote data logging station. In this example,

a DS87C530 is awoken from Stop mode every 30 minutes to read the temperature from a Dallas Semiconductor DS1620 Digital Thermometer and Thermostat. The DS1620 is addressed via serial port 0, using serial mode 0. When the interrupt is called, the DS87C530 will use the ring oscillator to perform a fast resume from Stop, and signal the thermostat to begin a temperature conversion. It will then reset the RTC alarm to occur again in 1 second. This will allow time for the conversion and the crystal warm–up period to complete, after which the device will automatically switch back to the crystal as the clock source. The DS87C530 will read the temperature and transmit it, along with the hour and minute, back to a host system connected to serial port 1. It will then return to Stop mode to await the next alarm. Figure 1 shows a partial schematic for interfacing the DS87C530 and DS1620. If the DS1620 is to be separated from the microcontroller by a long distance, filtering may be necessary on the clock and data lines to reduce noise.

DS1620 INTERFACE EXAMPLE Figure 1
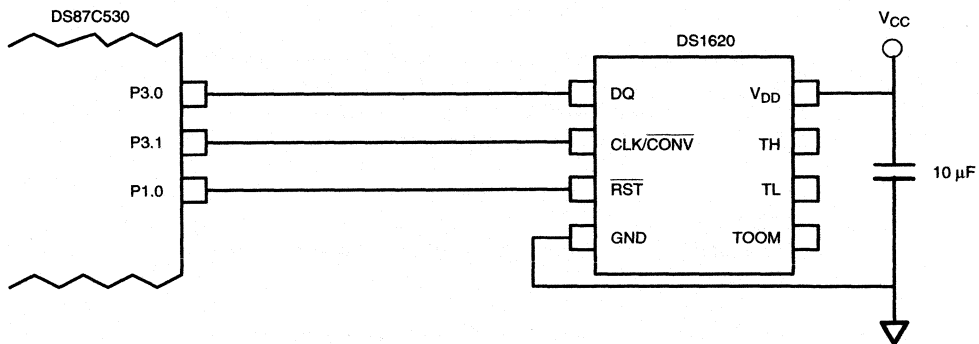

```
;Program DATALOGR.ASM
;
;This program demonstrates how to use the RTC to periodically service an
;external device. The device halts in Stop mode until awoken by an RTC interrupt
;every half hour. It then reads the temperature from a DS1620 Digital
;Thermostat and transmits it, with a time stamp, to the host via serial port 1.

;Register equate table
SP        equ    81h    ;Stack Pointer
PCON      equ    87h    ;Power Control Register
TCON      equ    88h    ;Timer Control Register
```

```
        TMOD       equ    89h    ;Timer Mode Register
        TH1        equ    8Dh    ;Timer 1 MSB
        CKCON      equ    8Eh    ;Clock Control Register
        P1         equ    90h    ;Port 1
        EXIF       equ    91h    ;External Interrupt Flag Register
        SCON0      equ    98h    ;Serial Port 0 Control Register
        SBUF0      equ    99h    ;Serial Port 0 Data Buffer
        P3         equ    0B0h   ;Port 3
        SCON1      equ    0C0h   ;Serial Port 1 Control Register
        SBUF1      equ    0C1h   ;Serial Port 1 Data Buffer
        TA         equ    0C7h   ;Timed Access Register
        WDCON      equ    0D8h   ;Watchdog Control Register
        ACC        equ    0E0h   ;Accumulator
        RTASS      equ    0F2h   ;Real Time Alarm Subsecond Register
        RTAS       equ    0F3h   ;Real Time Alarm Second Register
        RTAM       equ    0F4h   ;Real Time Alarm Minute Register
        RTCC       equ    0F9h   ;Real Time Clock Control
        RTCM       equ    0FCh   ;Real Time Clock Minute Register
        RTCH       equ    0FDh   ;Real Time Clock Hour Register

;Bit equate table
        RI0        equ    098h
        TI0        equ    099h
        REN0       equ    09Ch
        EA         equ    0AFh
        TI1        equ    0C1h
        ERTCI      equ    0EDh


        DS1620_RST equ    090h   ;DS1620 reset pin is tied to DS87C530 P1.0.
        WR_CONFIG  equ    0Ch    ;DS1620 Write Configuration command.
        RD_TEMP    equ    0AAh   ;DS1620 Read Temperature command.
        START_CONV equ    0EEh   ;DS1620 Start Conversion command.


                   cseg   at  0            ;Reset vector.
                   LJMP   START
                   cseg   at  6Bh          ;Real time clock Interrupt vector.
                   LJMP   RTC_INT
        ;
                   cseg   at  100H         ;Beginning of code segment.

        START:     MOV    SP,    #40h      ;Initialize Stack pointer.
                   MOV    EXIF,  #0Ah      ;Enable ring oscillator restart from Stop mode.
                   MOV    P3,    #03h      ;Set P3.1 & P3.0 high to use serial port 0.
                   MOV    SCON0, #20h      ;Set serial port mode 0, 4 tclk.

                   MOV    P1,    #0Ch      ;Set P1.2 & P1.3 high to use serial port 1.
                                           ;Clear P1.0 to reset DS1620.
                   MOV    SCON1, #40h      ;Set serial port mode 1.
                   MOV    TMOD,  #20h      ;Configure timer 1 for 9600 baud
                   MOV    TH1,   #0FDh     ; at 11.0592 MHz.
                   MOV    TCON,  #40h      ;Start timer.
```

```
;Configure the DS1620
            SETB   DS1620_RST      ;Remove DS1620 reset to start operation.
            MOV    A, #WR_CONFIG   ;Send command to address configuration byte.
            CALL   OUT_1620
            MOV    A, #03h         ;Set Configuration byte = CPU & 1-Shot Mode.
            CALL   OUT_1620
            CLR    DS1620_RST      ;Assert DS1620 to end operation.

;Set up the RTC
            MOV    RTAM,  #00h     ;Clear all alarm registers. Alarm will ring
            MOV    RTAS,  #00h     ; on the next hour to start temperature
            MOV    RTASS, #00h     ; conversion.
            MOV    RTCC,  #081     ;Set alarms so we get a reading at start.
            SETB   ERTCI           ;Enable RTC interrupt.
            SETB   EA              ;Global interrupt enable.

MAIN:       ORL    PCON, #02h      ;Set STOP bit to enter Stop mode.
            JMP    MAIN            ;End of main program loop. Program will return
                                   ; here after RTC interrupt is complete.


;********************************************************************************
;RTC_INT - This ISR reads the temp from the DS1620 and outputs the data to
;          serial port 1. The routine starts the conversion, and waits for 1
;          second to allow conversion to complete and crystal to stabilize.
;          When the conversion is complete, the device will read the temperature
;          and send the hour, minute and temperature to the host. The RTAM
;          register will be modified to alarm again in 30 minutes.
;********************************************************************************
RTC_INT:    MOV    RTCC, #081h     ;Clear RTCI flag and second compare enable
                                   ; bit to generate another alarm in 1 second.
            PUSH   ACC             ;Save accumulator.

            SETB   DS1620_RST      ;Remove DS1620 reset to start operation.
            MOV    A, #START_CONV  ;Initiate first temp conversion.
            CALL   OUT_1620
            CLR    DS1620_RST      ;Assert DS1620 to end operation.

            ORL    RTCC, #08h      ;Enable RTC read process, and delay 4 machine
            CJNE   A, ACC, $       ; cycles for time registers to stabilize.
            MOV    R7, RTCM        ;Save minute and hour so we can transmit
            MOV    R6, RTCH        ; them as soon as crystal has stabilized.
            ANL    RTCC, #0F7h     ;Reenable time register updates.

WAIT:       MOV    A, RTCC         ;Wait for RTC interrupt flag to be set,
            JNB    ACC.1, WAIT     ; indicating that conversion is done. The
                                   ; one second delay will be sufficient for the
                                   ; crystal to stabilize, so switch to it now.

            XRL    RTAM, #1Eh      ;Change alarm to ring on next half hour.
            MOV    RTCC, #0E1h     ;Clear RTCI flag, and set compare bits
                                   ; so next alarm will be generated in 30 min.
```

```
            MOV     A, #'!'          ;Transmit start character.
            CALL    OUT_HOST
            MOV     A, R6            ;Transmit the hour.
            CALL    OUT_HOST
            MOV     A, R7            ;Transmit the minute.
            CALL    OUT_HOST

            SETB    DS1620_RST       ;Remove DS1620 reset to start operation.
            MOV     A, #RD_TEMP      ;Conversion is done. Send command to read temp.
            CALL    OUT_1620
            CALL    IN_1620          ;Read LSB of temperature and send it to host.
            CALL    IN_1620          ;Read MSB of temperature and send it to host.
            CLR     DS1620_RST       ;Assert DS1620 to end operation.

            POP     ACC              ;Restore accumulator and go back to sleep.
            RETI


;*********************************************************************
;OUT_HOST - This routine sends data to the host system via serial port 1.
;*********************************************************************
OUT_HOST:   MOV     SBUF1, A    ;Move out byte.
            JNB     TI1, $      ;Wait until data has been transmitted.
            CLR     TI1         ;Clear TI1.
            RET


;*********************************************************************
;OUT_1620 - This routine sends data to the DS1620 via serial port 0.
;*********************************************************************
OUT_1620:   MOV     SBUF0, A    ;Move out byte.
            JNB     TI0, $      ;Wait until data has been transmitted.
            CLR     TI0         ;Clear TI1.
            RET


;*********************************************************************
;IN_1620 - This routine reads a byte from the DS1620 and echoes it back
;          through serial port 1.
;*********************************************************************
IN_1620:    SETB    REN0        ;Enable receiver to clock in data.
            JNB     RI0, $      ;Wait until data has been received.
            CLR     REN0        ;Disable receiver to prevent reception.
            CLR     RI0         ;Clear RI.

            MOV     A, SBUF0    ;Echo data through serial port 1.
            CALL    OUT_HOST
            RET
```

PROGRAM EXAMPLE: RTC INTERFACE

The following program is a general purpose interface routine to set the RTC and display its status. The program communicates through serial port 0, and allows the user to set the time and date, set the alarm registers, and indicates when an alarm has occurred. For the sake of simplicity, the program inputs decimal values of time and outputs hexadecimal values.

```
;*************************************************************************
;Program RTC_UTIL.ASM
;
;This program responds to commands received over the serial port to set
;and read the date, time and alarm information in the DS87C530 Real Time Clock.
;The program initializes the serial port for operation at 28800 baud with an
;11.0592 MHz clock.
;*************************************************************************
;Register equate table
SP        equ    81h    ;Stack Pointer
DPL       equ    82h    ;Data pointer low register
DPH       equ    83h    ;Data pointer high register
PCON      equ    87h    ;Power Control Register
TCON      equ    88h    ;Timer Control Register
TMOD      equ    89h    ;Timer Mode Register
TH1       equ    8Dh    ;Timer 1 MSB
EXIF      equ    91h    ;External Interrupt Flag Register
SCON0     equ    98h    ;Serial Port 0 Control Register
SBUF0     equ    99h    ;Serial Port 0 Data Buffer
P3        equ    0B0h   ;Port 3
TA        equ    0C7h   ;Timed Access Register
ACC       equ    0E0h   ;Accumulator
B         equ    0F0h   ;B Register
RTASS     equ    0F2h   ;Real Time Alarm Subsecond Register
RTAS      equ    0F3h   ;Real Time Alarm Second Register
RTAM      equ    0F4h   ;Real Time Alarm Minute Register
RTAH      equ    0F5h   ;Real Time Alarm Hour Register
EIP       equ    0F8h   ;Extended Interrupt Priority Register
RTCC      equ    0F9h   ;Real Time Clock Control
RTCSS     equ    0FAh   ;Real Time Clock Subsecond register
RTCS      equ    0FBh   ;Real Time Clock Second
RTCM      equ    0FCh   ;Real Time Clock Minute
RTCH      equ    0FDh   ;Real Time Clock Hour
RTCD0     equ    0FEh   ;Real Time Clock Day Register 0
RTCD1     equ    0FFh   ;Real Time Clock Day Register 1

;Bit equate table
RI0          equ    98h    ;Serial Port 0 Receiver Interrupt Flag
TI0          equ    99h    ;Serial Port 0 Transmitter Interrupt Flag
EA           EQU    0AFh   ;Global Interrupt Enable.
ERTCI        equ    0EDh   ;Real Time Clock Interrupt Enable.

;Constant equate table
CR        equ    0Dh
LF        equ    0Ah
```

```
                cseg   at   0          ;Reset vector.
                LJMP   START
                cseg   at   6BH        ;Real time clock Interrupt vector.
                LJMP   RTC_INT

                cseg   at   100H    ;Beginning of code segment.
;Data & string tables.
HEX_TABLE:   DB '0123456789ABCDEF'
NEW_LINE:    DB CR, LF, 0
YES:         DB 'Y  ', 0
NO:          DB 'N  ', 0
COMPARE:     DB CR, LF, 'Compare enabled: ', 0
COMPARE_Q:   DB '  Enable compare (Y/N)? ', 0
ALARM_MSG:   DB CR, LF, 'Alarm: ', 0
TT_BANNER:   DB CR, LF, CR, LF, 'DS87C530 RTC UTILITY'
             DB CR, LF, ' A - Set Alarm, T -Set Time'
             DB CR, LF, ' any other key to show registers'
             DB CR, LF, CR, LF, 'RTC registers:    ', 0
ALM_BANNER:  DB CR, LF,            'Alarm register: ', 0
NEW_BANNER:  DB CR, LF, CR, LF, 'Enter new alarm register settings:',0
SET_BANNER:  DB CR, LF, 'Enter new time:', 0
SS_BANNER:   DB CR, LF, 'Subsecond:    ', 0
S_BANNER:    DB CR, LF, 'Second:       ', 0
M_BANNER:    DB CR, LF, 'Minute:       ', 0
H_BANNER:    DB CR, LF, 'Hour:         ', 0
DW_BANNER:   DB CR, LF, 'Day of Week:     ', 0
DC_BANNER:   DB CR, LF, 'Day Count:       ', 0
DW_STRING:   DB 'Disabled ',0,'Sunday   ',0,'Monday   ',0,'Tuesday  ', 0
             DB 'Wednesday',0,'Thursday ',0,'Friday   ',0,'Saturday ', 0

;Initialize part.
START:       MOV    SP,    #80h    ;Set up stack pointer.

             MOV    P3,    #0Bh    ;Set RXD0, TXD0 & INT1 as inputs.
             MOV    RTAM,  #00h    ;Initialize alarm registers to known values.
             MOV    RTAS,  #00h
             MOV    RTASS, #00h

             MOV    TA,    #0AAh   ;Timed access write to enable RTC.
             MOV    TA,    #55h
             MOV    RTCC,  #01h

             MOV    SCON0, #050h   ;Set serial port 0 for Mode 1, divide by 12.
             MOV    TH1,   #0FEh   ;Timer 1 value for 28800 baud at 11.0592 MHz.
             MOV    TMOD,  #20h    ;Set timer 1 to 8-bit auto reload and start it.
             MOV    TCON,  #40h
             ORL    PCON,  #80h    ;Set SMOD bit to get 28800 baud.

             SETB   ERTCI          ;Enable RTC interrupts.
             SETB   EA
```

```
                LJMP    TELL_TIME       ;Display the time.
;***********************************************************************
;This is the main program loop. It waits for a character on serial port 0,
;and then takes the appropriate action.
;***********************************************************************

CHAR_TEST:  JNB     RI0, $          ;Wait for incoming command character.
            CLR     RI0
            MOV     A, SBUF0        ;Test to see what to do.
CHECKT:     CJNE    A, #'T', CHECKA ;T - set time.
            LJMP    SET_TIME
CHECKA:     CJNE    A, #'A', TT_JUMP ;A - set alarm.
            LJMP    SET_ALARM
TT_JUMP:    LJMP    TELL_TIME       ;else display time.


;***********************************************************************
;SET_TIME sets the current time.
;***********************************************************************
SET_TIME:   MOV     DPTR, #SET_BANNER ;Display set time banner.
            CALL    OUT_STRING
            MOV     DPTR, #H_BANNER   ;Get hour & save temp copy.
            CALL    OUT_STRING
            CALL    IN_TIME
            ANL     A, #1Fh          ;Make sure day of week bits are 0.
            MOV     R4, A
            MOV     DPTR, #M_BANNER   ;Get minute & save temp copy.
            CALL    OUT_STRING
            CALL    IN_TIME
            MOV     R5, A
            MOV     DPTR, #S_BANNER   ;Get second & save temp copy.
            CALL    OUT_STRING
            CALL    IN_TIME
            MOV     R6, A
            MOV     DPTR, #DC_BANNER  ;Get day count(2 bytes) & save temp copies.
            CALL    OUT_STRING
            CALL    IN_TIME
            MOV     R2, A
            CALL    IN_TIME
            MOV     R3, A
            MOV     DPTR, #DW_BANNER  ;Get day of week value and add it on to
            CALL    OUT_STRING        ; the upper 3 bits of the hour register.
            CALL    IN_TIME
            SWAP    A
            RL      A
            ANL     A, #0E0h
            ORL     A, R4
            XCH     A, R4
            MOV     DPTR, #NEW_LINE   ;Add a blank line for esthetics.
            CALL    OUT_STRING
```

```
        MOV     TA, #0AAh          ;We have all the values, now save them.
        MOV     TA, #055h          ;Perform a timed access to write to
        ORL     RTCC, #04h         ; set new time & date.
        CJNE    A, ACC, $          ;Delay 4 machine cycles.
        MOV     RTCSS, R7
        MOV     RTCS,  R6
        MOV     RTCM,  R5
        MOV     RTCH,  R4
        MOV     RTCD0, R3
        MOV     RTCD1, R2
        MOV     TA,  #0AAh          ;Clear RTCWE bit to prevent accidental
        MOV     TA,  #055h          ; changes to time registers.
        ANL     RTCC, #0FBh

        LJMP CHAR_TEST              ;Return and wait for another event.

;*************************************************************************
;TELL_TIME displays the current time, alarm registers, and alarm status.
;*************************************************************************
TELL_TIME:  MOV     DPTR, #TT_BANNER    ;Display current time.
            CALL    OUT_STRING
            CALL    OUT_TIME

            MOV     DPTR, #ALM_BANNER   ;Display alarm registers.
            CALL    OUT_STRING
            MOV     R7, RTASS
            MOV     R6, RTAS
            MOV     R5, RTAM
            MOV     R4, RTAH
            CALL    DISP_TIME

            MOV     DPTR, #COMPARE      ;Now display the compare bits.
            CALL    OUT_STRING
            MOV     A, RTCC

            CALL    DISP_COMP           ;Display Hour compare bit.
            RR      A
            CALL    DISP_COMP           ;Display Minute compare bit.
            RR      A
            CALL    DISP_COMP           ;Display Second compare bit.
            RR      A
            CALL    DISP_COMP           ;Display Subsecond compare bit.

            ORL     RTCC, #08h          ;Set the read bit to stop RTC update.
            CJNE    A, ACC, $           ;Delay 4 machine cycles.
            MOV     R4, RTCH            ;Read the hour register.
            MOV     R3, RTCD0           ;Read the day count registers.
            MOV     R2, RTCD1
            ANL     RTCC, #0F7h         ;Clear the read bit to restart RTC.
```

```
            MOV     DPTR, #DW_BANNER     ;Output Day of Week banner.
            CALL    OUT_STRING           ; the upper 3 bits of the hour register.
            MOV     A, R4                ;Day of week is stored in upper 3 bits
            SWAP    A                    ; of hour register.  Move it to bits 2-0
            RR      A                    ; and multiply by 10 to get location
            ANL     A, #07h              ; within day of week table to start.
            MOV     B, #0Ah
            MUL     AB
            MOV     DPTR, #DW_STRING     ;Now add offset to starting address
            ADD     A, DPL               ; of data table to calculate new
            JNC     NO_INC               ; data pointer location.
            INC     DPH
NO_INC:     MOV     DPL, A
            CALL    OUT_STRING

            MOV     DPTR, #DC_BANNER     ;Output day count banner.
            CALL    OUT_STRING
            MOV     A, R2                ;Send both registers of day count.
            CALL    OUT_DIGIT
            MOV     A, R3
            CALL    OUT_DIGIT
            MOV     DPTR, #NEW_LINE      ;Add a blank line for aesthetics.
            CALL    OUT_STRING

            LJMP    CHAR_TEST            ;Return and wait for another event.

;This routine displays the status of the compare enable bit.
DISP_COMP:  JNB     ACC.4, NO_COMP      ;Display the hour compare bit.
            MOV     DPTR, #YES
            JMP     OUT_COMP
NO_COMP:    MOV     DPTR, #NO
OUT_COMP:   CALL    OUT_STRING
            RET
;This routine outputs the current time.
OUT_TIME:   ORL     RTCC, #08h          ;Set the read bit to stop RTC update.
            CJNE    A, ACC, $           ;Delay 4 machine cycles.
            MOV     R7, RTCSS           ;Grab the current time / date and store
            MOV     R6, RTCS            ; them temporarily in working registers.
            MOV     R5, RTCM
            MOV     R4, RTCH
            ANL     RTCC, #0F7h         ;Clear the read bit to restart RTC.

DISP_TIME:  MOV     A, R4               ;Output hour.
            ANL     A, #01Fh            ;Mask off day of week bits.
            CALL    OUT_DIGIT

            MOV     A, R5               ;Output Minute.
            CALL    OUT_CDIGIT

            MOV     A, R6               ;Output second.
            CALL    OUT_CDIGIT
```

```
                MOV     A, R7               ;Output subsecond.
                CALL    OUT_CDIGIT
                RET


;**********************************************************************
;SET_ALARM sets the alarm registers.
;**********************************************************************
SET_ALARM:      CLR     ERTCI               ;Disable RTC interrupt and clear flag
                ANL     RTCC, #0Fh          ; during this section so that alarms will
                                            ; not be called while enables are changing.
                MOV     DPTR, #NEW_BANNER
                CALL    OUT_STRING
                MOV     DPTR, #H_BANNER
                CALL    OUT_STRING
                CALL    IN_TIME             ;Get hour & save temp copy.
                MOV     R4, A
                CALL    QUERY
                JNC     ASK_M
                ORL     RTCC, #10h          ;Enable hour compare

ASK_M:          MOV     DPTR, #M_BANNER
                CALL    OUT_STRING
                CALL    IN_TIME             ;Get minute & save temp copy.
                MOV     R5, A
                CALL    QUERY
                JNC     ASK_S
                ORL     RTCC, #20h          ;Enable minute compare

ASK_S:          MOV     DPTR, #S_BANNER
                CALL    OUT_STRING
                CALL    IN_TIME             ;Get second & save temp copy.
                MOV     R6, A
                CALL    QUERY
                JNC     ASK_SS
                ORL     RTCC, #40h          ;Enable second compare

ASK_SS:         MOV     DPTR, #SS_BANNER
                CALL    OUT_STRING
                CALL    IN_TIME             ;Get subsecond & save temp copy.
                MOV     R7, A
                CALL    QUERY
                JNC     ASK_X
                ORL     RTCC, #80h          ;Enable subsecond compare.

ASK_X:          MOV     DPTR, #NEW_LINE
                CALL    OUT_STRING

                MOV     RTASS, R7           ;Save new alarm values.
                MOV     RTAS, R6
                MOV     RTAM, R5
                MOV     RTAH, R4
```

```
                    ANL     RTCC,   #0FDh       ;Clear the RTCI flag in case it was
                                                ; accidentally set while we were
                                                ; manipulating compare bits.
                    SETB    ERTCI               ;Reenable RTC interrupt.
                    LJMP    CHAR_TEST


QUERY:              MOV     DPTR, #COMPARE_Q
                    CALL    OUT_STRING
                    JNB     RI0, $
                    CLR     RI0
                    MOV     A, SBUF0
                    CALL    OUT_CHAR            ;Echo it.
                    CJNE    A, #'Y', NO_ENABLE  ;If user wants compare, set flag.
                    SETB    C
                    RET
NO_ENABLE:          CLR     C                   ;User does not want compare, clear flag.
                    RET
;****************************************************************
;Output routines.
;****************************************************************
;This subroutine outputs an ASCII string.  The starting point of the string
;is in DPTR, and the terminating character is '0'.
OUT_STRING: PUSH    ACC                         ;Save accumulator.
CHAR_LOOP:  CLR     A                           ;Clear accumulator for next instruction.
            MOVC    A, @A + DPTR                ;Get the next character from the
            JNZ     NXT_CHAR                    ; string, and if 0, exit.
            POP     ACC                         ;Restore accumulator.
            RET
NXT_CHAR:   CALL    OUT_CHAR                    ;Next character is valid, so transmit
            INC     DPTR                        ; it.  Increment the data pointer
            JMP     CHAR_LOOP                   ; to the next position and loop
                                                ; back to send character.

;This subroutine outputs a leading colon for the minute, second, and subsecond
; when displaying the time. When done, it falls through to OUT_DIGIT.
OUT_CDIGIT: MOV     SBUF0, #':'                 ;Display a colon.
            JNB     TI0, $
            CLR     TI0
;This subroutine outputs a hex number in ASCII format through serial port 0.
OUT_DIGIT:  MOV     DPTR, #HEX_TABLE
            MOV     R0, A                       ;Make another copy of value
            SWAP    A                           ;Do high nibble fist
            ANL     A, #0Fh                     ;Clear unused nibble
            MOVC    A, @A+DPTR                  ;Get character from table
            CALL    OUT_CHAR                    ;Transmit the character.

            MOV     A, R0                       ;Now do low nibble.
            ANL     A, #0Fh                     ;Clear unused nibble
            MOVC    A, @A+DPTR                   ;Get character from table
```

```
                CALL    OUT_CHAR             ;Transmit the character.
                RET                          ;Done


OUT_CHAR:       MOV     SBUF0, A             ;Transmit the character out the serial
                JNB     TI0, $               ; port and wait until complete.
                CLR     TI0
                RET


;*****************************************************************
;Input routines.
;*****************************************************************
;IN_TIME takes two decimal characters from the serial port, and formats them
; as a hexadecimal number.
IN_TIME:        CALL    IN_CHAR              ;Get tens digit.
                MOV     B, #0Ah              ;Multiply first digit by 10 and save to
                MUL     AB                   ; add to ones digit.
                XCH     A, B
                CALL    IN_CHAR              ;Get ones digit and add it.
                ADD     A, B                 ;Acc now has hex value of 2 decimal digit
                RET                          ; number.  Exit.


IN_CHAR:        JNB     RI0, $               ;Wait for character.
                CLR     RI0
                MOV     A, SBUF0
                CALL    OUT_CHAR             ;Echo character back.
                PUSH    ACC                  ;Save copy of A.
                ANL     A, #0F0h             ;If bits 7-4 are not 3h, then character
                CJNE    A, #30h, IN_CHAR     ; is not 0-9. Get another character.
                POP     ACC                  ;Restore A.
                ANL     A, #0Fh              ;Acc now contains 0-9
                RET


;************************************************************************
;RTC_INT -  This ISR notifies the user that an alarm has occurred, and gives
;           the time of the alarm.
;************************************************************************
RTC_INT:        ANL     RTCC, #0FDh          ;Clear RTC Interrupt flag.
                MOV     DPTR, #ALARM_MSG     ;Display alarm message and time of alarm.
                CALL    OUT_STRING
                CALL    OUT_TIME
                RETI                         ;Return
```

RTC CRYSTAL CONSIDERATIONS

The most important factor in the accuracy of the RTC (or any oscillator) is the characteristics of the oscillator crystal. The DS87C530 is rated for an accuracy of ±2 minutes per month over the full operating range of the device. Even higher accuracy can be obtained by controlling the temperature of the device and using the RTC calibration procedures described later. The DS87C530 has been designed to operate with 32.768 KHz RTC crystals with a load capacitance (C_L) of 6 pF or 12.5 pF.

Unlike some crystal amplifiers, no external load capacitors are needed with the RTC crystal.

Dallas Semiconductor products are compatible with industry standard crystals. Table 2 shows a number of common 32.768 KHz crystals. This list is by no means exhaustive, and the inclusion or exclusion of any vendor from this list is in no way a comment on the suitability of a specific crystal in a customer's application.

STANDARD 12.5 PF AND 6 PF RTC CRYSTALS Table 2

MANUFACTURER	MODEL	C_L	PACKAGE
Epson Crystal Corp.	MC–306 32.768K E MC–306 32.768K A	6.0 pF 12.5 pF	SMT SMT
KDS America	DT–26S 32.768 KHz DT–26S 32.768 KHz DMX–26 32.768 KHz DMX–26 32.768 KHz	6 pF 12.5 pF 6 pF 12.5 pF	Cylinder Cylinder SMT SMT
AVX/Kyocera	KF–38G–12P5200 KS–309G–12P5200	12.5 pF 12.5 pF	Cylinder SMT

SELECTING LOAD CAPACITANCE

The value of C_L has the most bearing on the long–term accuracy of the RTC. This parameter specifies the capacitive load that the crystal needs to "see" across its pins to oscillate at its rated frequency. Note that C_L is not the capacitance of the crystal itself, but rather the capacitance of the oscillator circuit and any capacitors connected to the crystal. Using a crystal that has a different C_L than the actual load capacitance of the circuit will affect the frequency of the oscillator. In general, using a crystal with a C_L that is larger than the load capacitance of the oscillator circuit will cause the oscillator to run faster than the specified nominal frequency of the crystal, and vice versa.

The DS87C530 defaults to a mode which makes it compatible with a 12.5 pF crystal, but can be switched to 6 pF by clearing the RTC Capacitance Select bit X12/6 (TRIM.6). Although both crystal types will remain within the specified accuracy, each has a different advantage. The reduced loading of a 6 pF crystal will reduce the power consumption of the RTC crystal oscillator by 25 to 50 percent, increasing the life of the backup battery. A 12.5 pF crystal, however, is less affected by noise and will maintain a higher accuracy over an extended time. Changing the capacitance of the RTC crystal amplifier has no effect on the system clock crystal attached to the X1 and X2 pins.

FINE–TUNING THE OSCILLATOR FREQUENCY

Although the DS87C530 RTC is designed to oscillate at exactly 32.768 KHz, variations in the device, crystal, temperature, and board layout can produce minor timing variations. By adjusting the RTC Trim Bits located in the RTC Trim Register (TRIM;96h), the internal capacitance of the RTC circuitry can be slightly adjusted to improve timing accuracy beyond the minimum specified. Although the trim bits do not correspond to an absolute value of capacitance or frequency shift, they provide a relative adjustment.

Please note that under normal circumstances, adjusting the RTC Trim Bits is not necessary. Upon a no–battery reset, the DS87C530 will reset its internal capacitance to a default value which will guarantee the minimum accuracy specified. If you do not require accuracy better than 2 minutes per month, please skip this section.

To aid the user in determining the true frequency of the RTC, a 4096 Hz signal derived from the 32.768 KHz crystal is available on the P1.7 pin by setting the E4K bit (TRIM.7). This can be measured with a frequency counter to determine the RTC frequency. Do not attempt to measure the frequency of the RTC at the leads of the crystal. The capacitance of oscilloscope probes will distort the operation of the crystal and report erroneous val-

ues. The error of the RTC in minutes per month can be calculated from the following formula:

(P1.7 Frequency – 4096.000 Hz)*(10.547) [minutes/month]

Note that this error is calculated at a specific temperature and voltage. Crystal characteristics change over temperature, and the designer is advised to characterize the error over the system's range of expected operating conditions.

The trim register features extensive protection to avoid accidental corruption. All of the bits of the trim register require a Timed Access procedure to modify them. In addition, writes to the trim register must be done in complementary pairs. Each of the three trim bits has a complement bit which must be set simultaneously. This is to ensure that any writes to the TRIM register are intentional. If an invalid bit sequence is written to the trim bits, the TRIM register will reset to 0x100101 binary. This is the no–battery reset value, except that the X12/6 bit will remain unchanged. The settings of the TRIM bits do not correspond to an absolute value of capacitance or frequency, and are only used to provide a relative adjustment.

To adjust the RTC trim bits, place the device into the target system with the selected crystal and remove any sources of loading from P1.7. Then attach a frequency counter to P1.7 and perform the following procedure.

1. Perform a Timed Access procedure,
2. Set TRIM.7, E4K, and modify the TRMx bits, writing their complements to the TRMx\ bits in the same instruction. This will enable the external 4096 Hz signal. on P1.7,
3. Record the frequency,
4. Repeat steps 1–3 eight times until all combinations of TRM0, TRM1, TRM2 have been measured.

After all the measurements have been taken, the measurement closest to 4096.00 Hz is the most accurate setting of the TRMx bits. Program this value into the TRIM register for the maximum accuracy. An example program is provided below.

PROGRAM EXAMPLE: RTC CALIBRATION

The following program example is provided to assist system designers in calibrating their RTC for maximum accuracy. It demonstrates how to set the RTC trim bits and pause the program to allow time to read the frequency output on P1.7.

```
;*****************************************************************************
;Program RTC_CALB.ASM
;
;This program configures the DS87C530 so that the internal RTC frequency can
;be measured. A 4 KHz signal, derived by dividing the 32.768 KHz RTC by 8,
;will be asserted on pin P1.7. The device will step through the 8 settings of
;the RTC trim bits, displaying the current contents of the trim register on
;port 3. A delay of approximately 15 seconds (at 25 MHz) is inserted between
;each setting to allow time to record the frequency.
;
;To calibrate the RTC capacitance, connect a frequency counter to pin P1.7 and
;execute this program. Record the frequency from the counter and the trim bit
;settings as shown on port 3 as it steps through the 8 possible trim settings.
;The setting that produces a frequency closest to 4096 Hz is the most accurate
;setting of RTC capacitance.
;*****************************************************************************


RTCC       equ      0F9h          ;Real Time Clock Control
TA         equ      0C7h          ;Timed Access Register
TRIM       equ      96h           ;RTC Trim Register
P3         equ      0B0h          ;Port 3 Latch
```

```
;These definitions are for 6 pF crystal calibration.
TRIM0      equ     95h          ;First trim bit setting    (6 pF)
TRIM1      equ     96h          ;Second trim bit setting   (6 pF)
TRIM2      equ     99h          ;Third trim bit setting    (6 pF)
TRIM3      equ     9Ah          ;Fourth trim bit setting   (6 pF)
TRIM4      equ     0A5h         ;Fifth trim bit setting    (6 pF)
TRIM5      equ     0A6h         ;Sixth trim bit setting    (6 pF)
TRIM6      equ     0A9h         ;Seventh trim bit setting  (6 pF)
TRIM7      equ     0AAh         ;Eighth trim bit setting   (6 pF)

;These definitions are for 12.5 pF crystal calibration.
;TRIM0     equ     0D5h         ;First trim bit setting    (12.5 pF)
;TRIM1     equ     0D6h         ;Second trim bit setting   (12.5 pF)
;TRIM2     equ     0D9h         ;Third trim bit setting    (12.5 pF)
;TRIM3     equ     0DAh         ;Fourth trim bit setting   (12.5 pF)
;TRIM4     equ     0E5h         ;Fifth trim bit setting    (12.5 pF)
;TRIM5     equ     0E6h         ;Sixth trim bit setting    (12.5 pF)
;TRIM6     equ     0E9h         ;Seventh trim bit setting  (12.5 pF)
;TRIM7     equ     0EAh         ;Eighth trim bit setting   (12.5 pF)
;
cseg       at      0            ;Reset vector.
           LJMP    START

cseg       at      100H         ;Start of program

;
START:     MOV         P3, #0AAh        ;I'm alive message.

           MOV         TA, #0AAh        ;Timed access.
           MOV         TA, #55h
           MOV         RTCC, #01h       ;Start RTC and clear RTC interrupt flag.

           LCALL       HALFSEC          ;Delay to give RTC oscillator time to
                                        ; warm up.
; End of initialization. Now step through all the settings of the trim bits.
           MOV         R0, #TRIM0       ;Trim setting 0
           LCALL       NEXT_SETTING

           MOV         R0, #TRIM1       ;Trim setting 1
           LCALL       NEXT_SETTING

           MOV         R0, #TRIM2       ;Trim setting 2
           LCALL       NEXT_SETTING

           MOV         R0, #TRIM3       ;Trim setting 3
           LCALL       NEXT_SETTING

           MOV         R0, #TRIM4       ;Trim setting 4
           LCALL       NEXT_SETTING
```

```
                MOV        R0, #TRIM5       ;Trim setting 5
                LCALL      NEXT_SETTING


                MOV        R0, #TRIM6       ;Trim setting 6
                LCALL      NEXT_SETTING


                MOV        R0, #TRIM7       ;Trim setting 7
                LCALL      NEXT_SETTING


                MOV        P3, #0FFh        ;Turn on all port 3 pins to signal
DONE:           JMP        DONE             ; we're done.


;*************************************************************************
;NEXT_SETTING - This subroutine writes the new setting to the RTC trim register,
;               displays the value of the trim register on port 3 for reference,
;               and delays for a period to give time to record the data
;*************************************************************************
NEXT_SETTING:
                MOV        TA, #0AAH        ;Timed access.
                MOV        TA, #55h
                MOV        TRIM, R0         ;Set E4K and new trim setting.
                NOP
                MOV        P3, TRIM         ;Output value of trim register.

SEC30:          MOV        R3, #30          ;15 second delay with 25 MHz crystal.
SECLOOP:        LCALL      HALFSEC
                DJNZ       R3,SECLOOP
                RET


;*************************************************************************
;HALFSEC - This subroutine generates a delay of approximately 0.5 second with
;          a 25 MHz crystal.
;*************************************************************************
HALFSEC:        MOV        R0,#25
OUTER:          MOV        R1,#125
MIDDLE:         MOV        R2,#249
INNER:          NOP
                DJNZ       R2,INNER
                DJNZ       R1,MIDDLE
                DJNZ       R0,OUTER
                RET
```

NOISE AND CRYSTAL LAYOUT GUIDELINES

The crystal inputs of the DS87C530 RTC (RTCX1, RTCX2) have a very high impedance. Unfortunately, this can cause the leads to the crystal to function as antennae, coupling high frequency signals into the RTC circuitry from the rest of the system. This can lead to a distortion of the crystal oscillator signal, resulting in extra or missed clock edges. In most situations high frequency noise will present the greatest problem, causing the clock to run fast.

The following procedure can be used to determine if noise is the cause of the inaccuracy of a RTC:

1. Power the system up and synchronize the RTC to a known, accurate clock,

2. Remove V_{CC} to the device (but maintain V_{BAT}),

3. Wait for a long period of time (24 hours),

4. Apply V_{CC}, read the RTC, and compare to the known, accurate clock,

5. Resynchronize the RTC to the known, accurate clock,

6. Keep system powered up and wait for the same period of time in step 3,

7. Read RTC and compare to the known, accurate clock.

The above procedure allows the designer to measure the inaccuracy of the clock both when the system is operating and when it is powered down. If the clock appears less accurate when powered up, the most likely culprit is system noise. If the inaccuracy remains whether the system is on or off, then the cause is most likely not system noise.

Because the crystal pins are highly susceptible to coupling noise, care must be taken when locating the external crystal on the PCB and when routing traces. The following guidelines are presented to reduce the effect of external noise on the RTC.

1. Place the crystal as close as possible to the RTCX1 and RTCX2 pins. Short traces reduce stray capacitance and noise coupling.

2. Keep the use small crystal bond pads and short traces to the RTCX1 and RTCX2 pins. Larger pads and longer traces are more likely to couple noise from adjacent circuits.

3. Place a ground guard ring around the crystal. This helps isolate the crystal from adjacent signals.

4. Avoid routing signals beneath the crystal or RTCX1/RTCX2 traces. This helps isolate the crystal from adjacent signals. It is especially important to keep high frequency signals and devices as far away from the crystal as possible.

5. Place a local ground plane directly beneath the ground guard ring. This helps isolate the crystal from signal layers below the crystal.

Figure 2 shows the recommended placement of the RTC crystal, guard ring, and ground plane. The illustration shows one common orientation for a 4-pin surface mount crystal, but pin orientations will vary between manufacturers and package types.

EXAMPLE CRYSTAL PLACEMENT ON PCB Figure 2

LOCAL GROUND
PLANE (LAYER 2)

GND2

RTCX2

RTCX1

V_{CC2}

PIN 1

PIN 4

GND2

V_{CC2}

CRYSTAL PADS

PIN 2

PIN 3

DALLAS
SEMICONDUCTOR
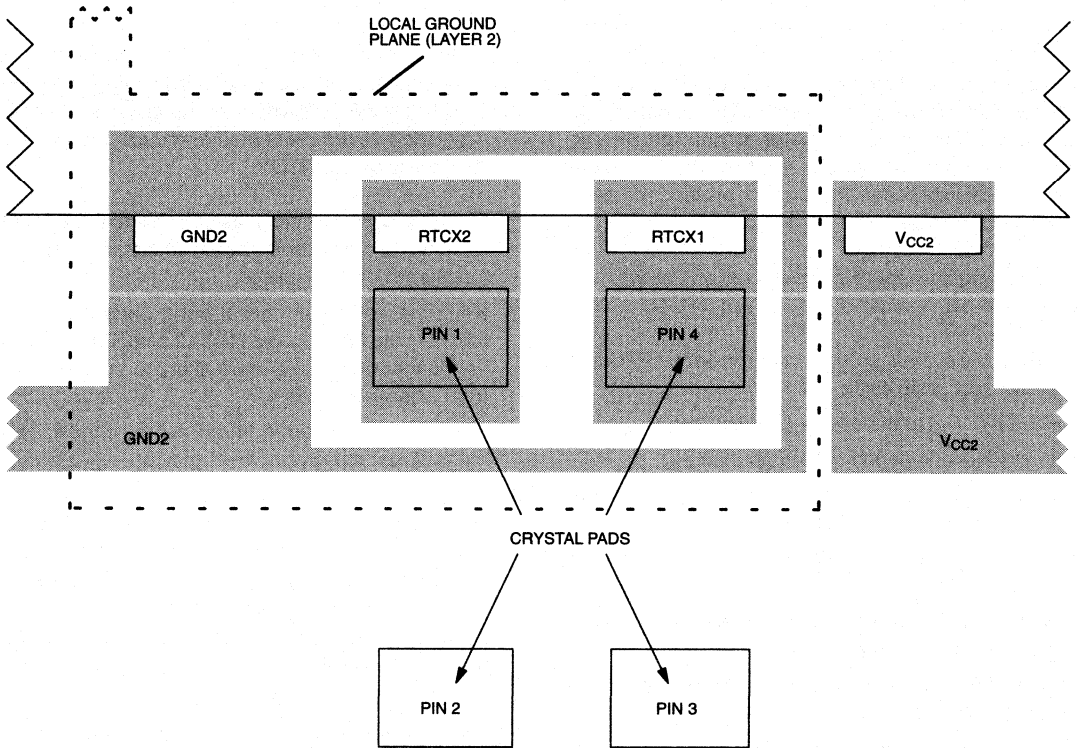
Application Note 80
Using the High–Speed Micro's Watchdog Timer

INTRODUCTION

Today, microcontrollers are being used harsh environments where electrical noise and electromagnetic–magnetic interference (EMI) are abundant. In environments like this, it is beneficial if the system contains resources to help ensure proper operation. In many systems, a commonly used technique for verifying proper operation is the incorporation a watchdog timer.

A watchdog timer is fundamentally a time measuring device that is used in conjunction with, or as part of, a microprocessor and is capable of causing the microprocessor to be reset. In a properly designed system, the watchdog will cause a reset when the microprocessor is not operating correctly, thereby eliminating the faulty condition. In a typical application, the watchdog timer is configured to reset the processor after a predetermined time interval. If the processor is operating correctly, it will restart the watchdog before the end of the interval. After being restarted, the watchdog will begin timing another predetermined interval. If the watchdog is not restarted by the processor before the end of the interval, a watchdog time–out occurs. This results in the processor being reset. If the system software has been designed correctly, and there has been no hardware failure, the reset will cause the system to operate properly again. Of course, the reset condition must be a safe state. For instance, it would not be wise to have the reset state of a disk drive controller enabling the write head.

Many systems have been designed using an external watchdog timer. The need for this additional external component is eliminated, however, with the DS80C320. The DS80C320 contains its own very capable, internal, watchdog timer. The features and the use of this watchdog timer are the subject of this application note.

GENERAL USE OF A WATCHDOG TIMER

The primary application of a watchdog timer is as a system monitor (as discussed in detail in the section below). With a watchdog timer, a system can be designed to be very good at detecting and correcting an out of control microprocessor. A system using a watchdog timer is particularly well suited to detecting bit errors. Momentary bit errors can be caused by such things as soft memory failures and electromagnetic discharges into memory devices and their interfaces. These can cause temporary bit polarity flipping of data into and out of the processor. When this occurs while fetching program information, the microprocessor will begin executing erroneous code. Potentially, the processor could begin executing operands instead of op–codes. When the processor begins executing this bad code, it will not properly execute the code that restarts the watchdog. After the time–out interval, the watchdog will cause a processor reset. In a properly designed system, the reset will correct the error.

Regardless of how capable a watchdog timer might be, it cannot resolve all reliability issues. There are certain failures that cannot be corrected by a reset. For instance, a watchdog timer cannot prevent the corruption of data. In its basic form, the watchdog restart is dependent on proper program execution, and generally, does not depend on the values in data memory. Unless corruption of data affects program flow, or some extra measures are taken, data corruption will not cause a watchdog time–out. Of course, self–diagnostic software can be written in such a way as to make restarting the watchdog contingent on verification of data memory. While this approach can be very effective and is quite common, it is beyond the scope of this document to discuss in detail.

Also note that a watchdog timer cannot detect a fault instantaneously. By definition, the watchdog timer must reach the end of a predetermined time interval before it resets the processor. This fact explains why a minimum possible time–out interval should be selected. In this way, a minimum amount of time expires before an out of control condition is corrected.

THE WATCHDOG AS A SYSTEM SUPERVISOR

The most common use of the High–Speed Micros watchdog timer is as a system supervisor. While it can be used in a number of different ways, some of which will be discussed in this document, system supervisor is the most common application. In system supervisor mode, the timer is restarted periodically by the processor as described above. If the processor runs out of control, the watchdog will not be restarted, it will time–out, and subsequently will cause the processor to be reset.

In the High–Speed Micro, the watchdog timer is driven by the main system clock that is supplied to a series of dividers. The divider output is selectable, and determines the interval between time–outs. When the time–out is reached, an interrupt flag will be set, and if enabled, a reset will occur 512 clocks later. The interrupt flag will cause an interrupt to occur if its individual enable bit is set and the global interrupt enable is set. The reset and interrupt are completely discrete functions that may be acknowledged or ignored, together or separately for various applications.

When using the watchdog timer as a system monitor, the watchdog's reset function should be used. If the interrupt function were used, the purpose of the watchdog would be defeated. To explain, assume the system is executing errant code prior to the watchdog interrupt.

The interrupt would temporarily force the system back into control by vectoring the CPU to the interrupt service routine. Restarting the watchdog and exiting by an RETI or RET would return the processor to the lost position prior to the interrupt. By using the watchdog reset function, the processor is restarted from the beginning of the program, and thereby placed into a known state.

This is not to say that the DS80C320 watchdog's interrupt function is not useful for the system monitor application. Since the reset occurs 512 clocks after the interrupt, a short interrupt service routine can be used to store critical variables before the reset occurs. This may allow the system to return to proper operation in a state that more closely resembles the conditions before the failure. Of course, if the data is the source of the error, storing it without correction would be of no benefit. For any specific system, the approach taken is a function of the system and the level of reliability required.

As mentioned above, the watchdog timer in the DS80C320 is driven by the main system clock that is passed through a series of dividers. The divider output may be selected by the user, allowing a time–out of 2^{17}, 2^{20}, 2^{23}, or 2^{26} clocks. If enabled, a reset of the processor will occur 512 clocks later. Table 1 shows the reset time intervals associated with different crystal frequencies.

DS80C320 WATCHDOG RESET TIME INTERVALS Table 1

CLOCKS	@1.832 MHz	@11.059 MHz	@12 MHz	@25 MHz
$2^{17} + 512$	71.83 ms	11.90 ms	10.97 ms	5.26 ms
$2^{20} + 512$	572.6 ms	94.86 ms	87.42 ms	41.96 ms
$2^{23} + 512$	4.58 s	758.6 ms	699.1 ms	335.6 ms
$2^{26} + 512$	36.63 s	6.07 s	5.59 s	2.68 s

As can be seen, there is a range of time–out intervals available. The interval selected should be based on several issues. The first objective is to select an interval that represents the maximum time the processor can be allowed to run out of control. For example, a system that issues a position command to a robotic arm every 500 milliseconds, ideally, would not use a time–out interval greater than this. Keeping the time–out interval shorter, ensures that there will be at most one bad command issued to the arm.

The other primary concern in setting the watchdog time–out interval is the ability to locate the restart commands within the system software. This can be a very complicated issue depending on the nature of the system software. The most desirable approach is to have a single location within a single main loop of the system software that restarts the watchdog timer. The time required to pass through the main program loop will determine the required time–out interval.

The above approach assumes that the system software flow is linear enough to allow it. Some programs are too convoluted and their flow is too non–linear to allow this approach. With a program structure like that, it is difficult to locate the correct points for watchdog restarts. One possible solution to this problem is to use the DS80C320's watchdog timer itself to assist in determining the appropriate restart locations. This method uses the watchdogs interrupt capability, and is described in detail in a section below.

In some systems, the software is too complex or the program flow is too variable to allow a complete and thorough analysis. It may be impossible to determine that all program paths have been covered by a watchdog restart. In this case, a different approach may be used. In this case, diagnostic software may be developed to test the system. This diagnostic software will be called at periodic intervals, perhaps using the interrupt feature of the watchdog timer. If the diagnostics pass, the watchdog is restarted. If not, the watchdog times out and the processor is reset. Of course in this case, the test must be thorough enough to be effective. The exact approach used in a given system may be any of the above or some combination of each, as appropriate for the application.

WATCHDOG RESET EXAMPLE

A short program illustrating most of the basic watchdog timer functions is shown below. This program illustrates how to initialize the watchdog timer so that when it times out, it will cause a reset.

The program illustrates one of the DS80C320 watchdog timer's unique features. Software that changes the watchdog's operation must perform a timed access operation. A timed access operation is a sequence of steps that must be executed together, in sequence, otherwise the access fails. The example program shows the timed access being used for restarting the watchdog and enabling its reset. As can be seen, the value 0AAh is first written to the timed access register (TA). Next, the value 055h is written to the TA register. Finally, the protected bit is modified. These instructions must be executed in the sequence shown with no interruption to gain access to the protected bit. Further details on timed access operation may be found in the High–Speed Micro User's Guide. The watchdog timer bits that are protected by the timed access procedure are the Enable Watchdog Timer Reset (EWT = WDCON.1) bit, the WatchDog Interrupt Flag (WDIF = WDCON.3) bit, and the Restart Watchdog Timer (RWT = WDCON.0) bit.

```
;  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *
;                          WD_RST.ASM Program
;
;       This program demonstrates the use of the watchdog timer in
;       the DS80C320. It uses the timer's reset capability. When
;       running, the program sets port 1's pins low to indicate
;       the processor is idle waiting for the watchdog to time-out. When
;       the watchdog times out, the processor is reset causing the port
;       pins to return high. A delay is written into the program so that
;       the port pins will be high long enough to be seen if attached to
;       LEDs.
;
;  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *
;
;       Reset Vector
;
        ORG             00h
        SJMP            START
;
;  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *
;
;       Main program body
;
        ORG             080h
        ;
```

```
START:    ORL           CKCON, #080h          ; Set Watchdog time-out period 2**23
                                  ; (approximately 758 mS @ 11.059 MHz)
          ;
          ; In a real application, the next three lines would be placed
          ; at various locations in the program to restart the watchdog
          ; before it times out.
          ;
          MOV           TA, #0AAh ; Restart Watchdog timer
          MOV           TA, #055h ;   using timed
          SETB          RWT       ;   access.
          ;
          ;
          MOV           TA, #0AAh ; Enable Watchdog timer reset
          MOV           TA, #055h ;   using timed
          SETB          EWT       ;   access.
          ;
          ;
          MOV           R1, #0FFh ; Create a delay loop so the port
LOOP:     MOV           R2, #0FFh ;   pins are high long enough after
          DJNZ          R2, $     ;   a reset to be seen.
          DJNZ          R1, LOOP
          ;
          MOV           P1, #00   ; P1 = 0, Reset causes P1 = 1
          ;
          MOV           PCON, #01h; Go  to idle mode waiting for reset
          SJMP          $
;
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
          END
```

THE WATCHDOG TIMER AS A LONG INTERVAL TIMER

A slightly different application of the High–Speed Micro's watchdog timer is as a long interval timer. In this application, the interrupt is enabled using the Enable Watchdog Timer Interrupt (EWDI=EIE.4) bit and the reset is left disabled. When the time–out occurs, the Watchdog Timer will set the WDIF bit (WDCON.3), and an interrupt will occur if the global interrupt enable bit (EA=IE.7) is set. The Watchdog Interrupt Flag will indicate the source of the interrupt, and must be cleared by software. As shown in the table above, intervals from 5.26 ms to 2.68 seconds are available with a 25 MHz crystal. This interval is significantly longer than any possible using the standard 16–bit timers.

Another short program illustrating features of the watchdog timer is shown below. This program demonstrates how the watchdog timer and interrupts must be initialized for so that a time–out causes an interrupt. A short interrupt service routine is included.

```
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
;
;                          WD_INT.ASM Program
;
;         This program demonstrates the use of the watchdog timer of
;         the 80C320. It uses the timer's interrupt generating capability.
;         For purposes of demonstration, the program toggles Port 1's pins
```

```
;             each time the watchdog's Interrupt Service Routine is entered.
;
;
$MODS320
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
;        Reset Vector
;
         ORG              00h
         SJMP             START
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
;        Watchdog Interrupt Vector
;
         ORG              063h
         ;
         MOV              TA, #0AAh ; Restart watchdog timer
         MOV              TA, #055h ;    using timed
         SETB             RWT       ;    access.
         ;
         MOV              TA, #0AAh ; Clear watchdog interrupt flag
         MOV              TA, #055h ;    using timed
         CLR              WDIF      ;    access.
         ;
         CPL              A         ; Complement port 1 to show the
         MOV              P1, A     ;    interrupt routine was entered.
         ;
         RETI                       ; Return from interrupt.
;
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
;        Main program body
;
         ORG              080h
         ;
START:
         ORL              CKCON, #040h; Set Watchdog time-out period 2**20
                                   ; (approximately 94.8 mS @ 11.059 MHz)
         ;
         MOV              TA, #0AAh ; Restart Watchdog timer
         MOV              TA, #055h ;    using timed
         SETB             RWT       ;    access.
         ;
         SETB             EWDI      ; Enable Watchdog Interrupt and
         SETB             EA        ;    set global interrupt enable
         ;
Here:    MOV              PCON, #01 ; Go to Idle mode and wait
         SJMP             Here      ; After interrupt, go back to idle
```

```
;
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
        END
```

THE WATCHDOG TIMER AS AN AID IN LOCATING RESTART INSTRUCTIONS

As discussed above, locating the watchdog restart instructions in the system software can sometimes be difficult. The structure of the system software and the complexity of its flow, determines the task's level of difficulty. In the DS80C320, the watchdog timer itself can be used to assist in this activity. The general approach for this is to allow the watchdog to cause an interrupt, and from within the service routine, determine where in the code the interrupt occurred. By placing the watchdog restart instructions prior to this point, you can be assured that the watchdog will be restarted before the time–out (when the software flow follows this particular branch). This process is repeated until no more watchdog interrupts occur. If the program flow is linear and not data dependent, the system will function as desired.

The previous software example provides most of the software necessary to perform this function. As a first step however, the desired maximum time–out interval should be determined, and the code modified for this value. As always, the time–out selected is a function of the system and how long the micro can be allowed to run out of control. After modifying the software to initialize the desired watchdog time–out interval, the following instructions should be added to the Interrupt Service Routine. They will cause the processor to display the address of the instruction that would have been executed if the interrupt had not occurred. If this display mechanism is not convenient for the system implementation, the address can be converted to ASCII and output on one of the serial ports.

```
MOV   R0, SP      ; Get SP contents
MOV   P3, @R0     ; Display high address byte
DEC   R0          ; Point to low address byte
MOV   P1, @R0     ; Display low address byte
SJMP  $           ; Stop here
```

The instructions above, move the contents of the Stack Pointer to R0 that is then used to point to the data

pushed onto the stack when the interrupt was acknowledged. This address reflects the next instruction that would have been executed if the interrupt had not occurred. The high byte of the address is displayed on Port 3 pins, and the low byte of the address is displayed on Port 1 pins. If instructions to restart the watchdog timer are placed before this address, the watchdog will never reach time–out.

SUMMARY

When designing a system using a watchdog as a monitor, there are a number of considerations that must go into an effective design. First, the maximum time that the processor can run out of control will determine the maximum watchdog time–out period. Once the time–out period is determined, the system software must be analyzed to determine where to locate the watchdog restart instructions. For an effective design, the number of watchdog restarts should be kept to a minimum, and some consideration should be given to the likelihood of incorrectly executing a restart. As mentioned previously, some system software is too convoluted or data dependent to ensure that all software flow paths are covered by a watchdog restart. This may dictate that a self–diagnostic software approach might be required. If there is an expected failure mechanism such as a periodic EMI burst or power supply glitch, the watchdog time–out should consider this period.

For the watchdog reset to be an effective error correction mechanism, the reset state of the processor must be safe. In some applications, the watchdog's interrupt capability might be used to manipulate data or the stack before the reset to ensure that the processor will function properly after the reset.

By carefully considering the above aspects, a system can be designed using a watchdog timer that will operate in very harsh environments.

DALLAS
SEMICONDUCTOR

Application Note 81
Memory Expansion with the
High–Speed Microcontroller Family

OVERVIEW

All members of the High–Speed Microcontroller Family are designed to directly address up to 64KB of program and data memory. Occasionally, however, an application will require more memory than is either available on–chip or through the use of the 64KB memory map. The High–Speed Microcontroller Family includes many features which make it easy to address program and/or data memory greater than 64KB. Bit–addressable I/O ports allow single instruction modification of control lines, which can be used to bank switch or page between multiple memory devices. The ROMSIZE feature allows easy memory resizing for devices with on–chip memory.

This application note discusses the expansion of both program and data memory. It is subdivided into three main categories: expanding program memory of ROM-less devices beyond 64KB, using the ROMSIZE feature to expand on–chip program memory up to and beyond 64KB, and expanding data memory. It begins with an introduction to bank switching and software support techniques.

BANK SWITCHING THEORY

Expanded memory access beyond 64KB is most frequently done through bank switching. This technique uses one or more general purpose I/O lines as decode lines to address more memory. If a single large–capacity memory device is used, the additional signals can be used directly as address lines. If several smaller capacity memory devices are used, the signals can be used as chip selects. The basic unit of memory switched by the decode logic is called a bank or page. For example, if an I/O line was used to switch between two 64KB EPROMs, the memory would consist of two 64KB pages.

Probably the biggest obstacle in using a paged memory scheme is the placement of the interrupt vector table. During most of the device operation, software can perform an orderly switch between pages. When an interrupt occurs, however, the device will immediately jump to the appropriate vector address, below 0070h. The software has no control over the bank configuration at this point, and the device will attempt to jump to the low end of the cur rent bank to seek the vector table.

There are two approaches to solving this problem. The simplest is to duplicate the interrupt vector table at the low end of each page. In this way, the interrupt vector table will be available at all times, regardless of the current memory configuration. There are a number of disadvantages to this approach, however. It is an inefficient use of program memory as the interrupt vector table (approximately 120 bytes) and often the interrupt service routines, must be duplicated on every page. Also, some compilers do not directly support duplication of data across pages, complicating program generation. A more efficient approach is to reserve some lower portion of memory, which includes the interrupt vector table, so that it is not paged. This "common area" is directly accessible from any expanded bank without modification of the bank selection mechanism. Any time the processor performs a code fetch at this common area, hardware forces the memory to access this area, regardless of the current page. Careful design will allow the previous bank address to be saved, so that the device will return automatically when the operation in the common memory is complete. Interrupt service routine execution times can be reduced by locating them along with the interrupt vector table in the common area located at the low end of memory. Most of the examples in this application note designate the lower region of memory as the common area.

072996 1/13

SOFTWARE SUPPORT FOR MEMORY EXPANSION

For a paged memory scheme to work, it is necessary to fragment the code into pages and provide a means for the software to switch between pages. Care must be exercised when switching pages that the instruction flow will not be disturbed. There are two main approaches to this. The first is to switch pages "on the fly." This is demonstrated in the simple page expansion example which follows. This approach causes program execution to jump directly from one expanded page to another expanded page. One must exercise caution so that the bank switch will occur at a location that corresponds to the start of the next instruction in the next bank. Failure to correctly align the instructions may cause the next opcode fetch to occur in the middle of a multi–byte instruction, resulting in complete loss of program control.

A better approach is to change banks from a location that will be unaffected by the change. This is most often a common or unpaged location in memory, such as a reserved location where the interrupt vector table is located. Any access to lower memory is automatically switched into the common memory by hardware. This eliminates the code alignment difficulties with the simple page expansion above, and the need to duplicate interrupt vectors and/or interrupt service routines on each memory page.

Many compilers and linkers directly support bank switching, and many of them include library functions for page switching. The documentation accompanying your compiler will provide information concerning its expanded memory support. Brief examples of assembly language support are listed after some of the examples below.

Care must be exercised if multiple pages are programmed into a single EPROM. Many EPROM programmers calculate program offsets using the address specified in the file, which can lead to misplaced code. For example, suppose that a paging scheme involves pages of code mapped into program space from 8000h to FFFFh. If the designer wished to locate a page at 10000h in the EPROM, he or she would normally select an offset of 10000h in the EPROM when loading the file into the programmer. All addresses in the hex file begin at 8000h, however, which the device programmer would add to 10000h. This would inadvertently place the page at 18000h, which was not the intended result. Various device programmers implement offsets in different ways, and the designer is advised to consult the documentation accompanying the device programmer for the best solution.

ROMLESS PROGRAM EXPANSION

The absence of on–chip program memory makes expanding the program memory of the DS80C320 relatively simple. Three approaches to expanding program memory are presented here. The first involves expansion of relatively small amounts of memory by duplicating vector tables and overlapping pages. The second example uses a common bank for interrupt vectors and interrupt service routines, and pages memory using several general purpose I/O lines. The last example uses a latched address to address large amounts of memory without using additional general purpose I/O lines.

Simple Page Expansion

This example shows the simplest way of adding relatively small amounts of program memory. A single general purpose I/O line is used to provide up to 128K bytes of program memory. A single 27C010 128K byte EPROM is used, and is divided into two overlapping memory blocks. One general purpose I/O line, P1.0 in this case, is used to provide bank switch control. It is latched by a 74F74, which is clocked on the rising edge of the $\overline{PSEN}$ signal. This synchronizes the bank switch with the memory cycle. This approach can be extrapolated to add even greater amounts of memory by using additional I/O lines. The hardware configuration for this example is shown in Figure 1.

SIMPLE PAGE EXPANSION EXAMPLE HARDWARE Figure 1
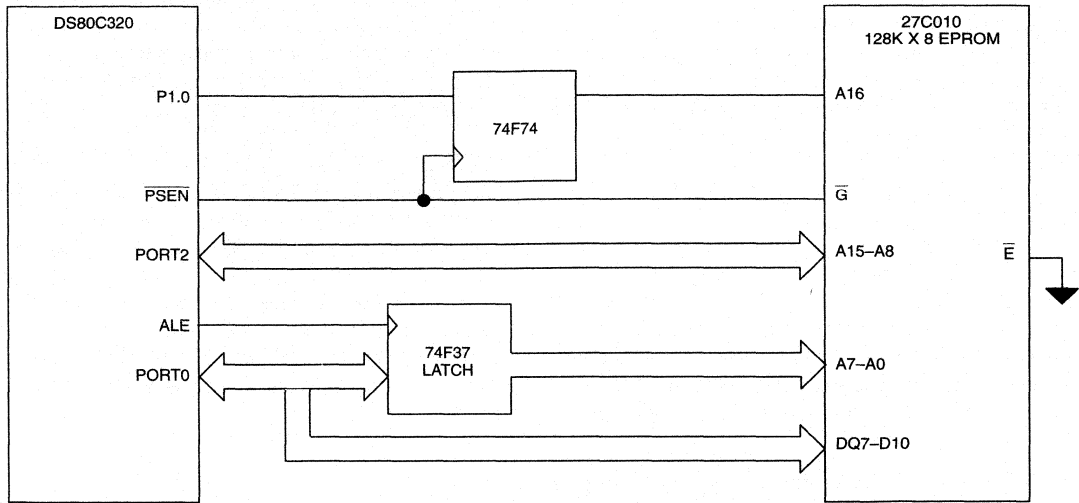
The simplicity of the hardware comes at the cost of some software complexity, however. This example uses two banks, both of which contain the interrupt vector tables in the lower portion of memory. This is necessary because when the device is reset, P1.0 will be high, forcing the reset vector address to 10000h. Also, an interrupt may occur while executing code from either page, so the interrupt vectors must be available without software intervention. The interrupt vector table consumes approximately 115 bytes from locations 00000h to 00070h, and 10000h to 10070h. Additional space may be required if duplication of interrupt service routines is desired on each page.

SIMPLE PAGE EXPANSION EXAMPLE MEMORY MAP Figure 2
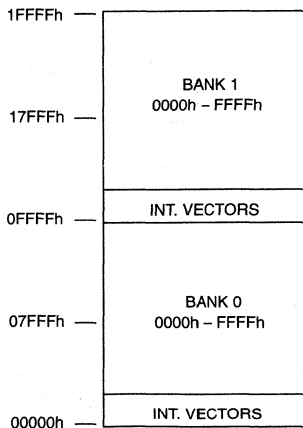
This approach is most effective when page switching is kept to a minimum, i.e. executing straight runs of code. Code efficiency will be improved if the interrupt routines (not just the vectors) are small enough to be duplicated on each page as well. Data tables or strings accessed with MOVC instructions should be located on the same page as the instruction.

This approach directly modifies the page without modifying the program counter. This means that the starting location in the new page will be the same at the end of the bank switch routine in the new page. Consequently, instruction location between pages is critical. Figure 3 shows the timing relationship between a MOV P1, A instruction and the bank select signal. In this example, the MOV instruction is located at location 5A10h, and the first instruction on the new page is located at 5A14h.

The port pin which controls the bank selection will change during the first cycle following MOV instruction. The 74F74 latch causes the new bank selection to be valid on the prefetch during the execution of the second NOP. The first instruction on the new page must be at the address following the second NOP.

There are other ways of modifying port pins, and this scheme will work with 2 cycle instructions such as MOV direct, direct, and 1 cycle instructions such as SETB bit. There are only two requirements on the bank switching instructions. The first instruction on the new page must be at the address following the second NOP. Also, the instruction proceeding the MOV instruction must not be a MOVX. The MOVX instruction timing is variable because of stretch cycles, and could disturb the instruction flow.

BANK SWITCHING TIMING DIAGRAM Figure 3
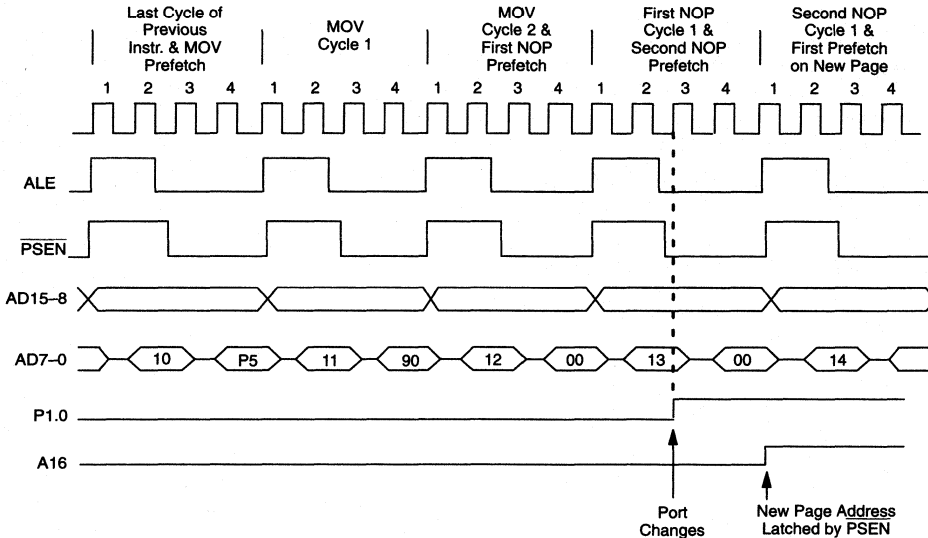
Common-Page Expansion Using I/O

The following two examples use a common–block approach in dealing with the problem of interrupt vector placement. This allows faster interrupt service times, and simplifies code construction. Both examples employ a paging scheme with 16 pages of 32KB each. Page 0, mapped at 0000h to 7FFFh, is the common area and contains the interrupt vector table and interrupt service routines. Address line A15 determines whether the common block, or one of the 15 expanded pages is addressed.

This example shows how to address up to 512KB of ROM using the general purpose I/O lines on port 1 or 3 as bank switching controls. Bank 0 is a single 32KB page from 0 to 7FFFh. This common area will contain the interrupt vectors and commonly used subroutines. Expanded memory will be contained in 15 pages of 32KB mapped from 8000h to FFFFh. Bank control is provided by 4 general purpose I/O pins. The memory map is shown in Figure 4.

COMMON-PAGE EXPANSION MEMORY EXAMPLE MEMORY MAP Figure 4

EPROM Address	Code Address and P1.3-0	EPROM Address	Code Address and P1.3-0	EPROM Address	Code Address and P1.3-0	EPROM Address	Code Address and P1.3-0
1FFFFh	BANK 3 8000h – FFFFh P1=x3h	3FFFFh	BANK 7 8000h – FFFFh P1=x7h	5FFFFh	BANK 11 8000h – FFFFh P1=xBh	7FFFFh	BANK 15 8000h – FFFFh P1=xFh
17FFFh	BANK 2 8000h – FFFFh P1=x2h	37FFFh	BANK 6 8000h – FFFFh P1=x26	57FFFh	BANK 10 8000h – FFFFh P1=xAh	77FFFh	BANK 14 8000h – FFFFh P1=xEh
0FFFFh	BANK 1 8000h – FFFFh P1=x1h	2FFFFh	BANK 5 8000h – FFFFh P1=x5h	4FFFFh	BANK 9 8000h – FFFFh P1=x9h	6FFFFh	BANK 13 8000h – FFFFh P1=xDh
07FFFh	BANK 0 0000h – 7FFFh P1=xxh	27FFFh	BANK 4 8000h – FFFFh P1=x4h	47FFFh	BANK 8 8000h – FFFFh P1=x8h	67FFFh	BANK 12 8000h – FFFFh P1=xCh
00000h		20000h		40000h		60000h	

The hardware configuration is shown in Figure 5. Bank control is provided by P1.0–3, decoded by 4 AND gates, requiring only a single IC package. When A15 is low, the device is forced to access only the lower 32KB of memory. This removes the need for software intervention when accessing the interrupt vector table in low memory. This example uses an 27C040 512KB EPROM.

DS80C320 EXPANDED MEMORY EXAMPLE HARDWARE CONFIGURATION Figure 5
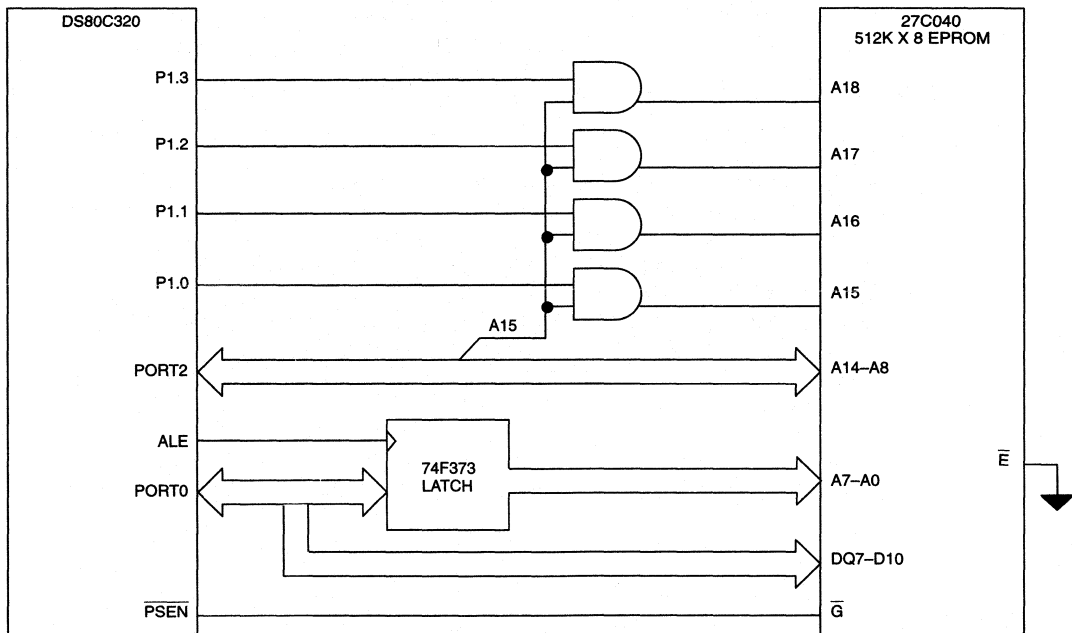
The following software example shows an assembly language routine to jump to a new location in any bank using I/O lines as shown in Figure 5. Before calling the bank switch subroutine, software pushes the new address and bank number onto the stack. It then calls a subroutine that pops the new bank address from the stack and places it on P1.0–3. The stack is then modified so that the subsequent RET instruction will return to the new program location. This is a simple demonstration of one of many possible ways to effect a bank–switch in assembly code.

PROGRAM EXAMPLE: JUMPING BETWEEN BANKS USING I/O

```
;*************************************************************************
;Program BANKJMP1.ASM
;
;This program demonstrates one possible way to jump between banks in a common-
;page memory expansion configuration using I/O. Four general purpose I/O pins
;at P1.0-3 are used to control external bank selection.
;
;Software pushes the address in the new bank, as well as the new bank address
;onto the stack. The subroutine BANKJUMP, located at a non-paged address,
;modifies the page selection, and then repositions the stack pointer to the
;new address which was previously pushed onto the stack. The RET is then
;executed, resuming operation from the new address in the new page. Note that
;BANKJUMP can be called from any area in program memory.
;*************************************************************************
;Equate table
BANKMASK        equ     0F0h            ;P1.3-0 are used for bank selection.
NEWSUB_BANK     equ     03h             ;Subroutine NEWSUB is located on page 03h.

NEWSUB          equ     0F000h          ;Address of subroutine NEWSUB. If a
                                        ;compiler is used, it may supply this
;                                       ;automatically.

cseg            at      0               ;Reset vector.
                LJMP    START
cseg            at      100H            ;Start of program
START:          MOV     P1, #0F0h       ;Default to bank 0.

                MOV     A, #low NEWSUB  ;Low byte of NEWSUB address.
                PUSH    A
                MOV     A, #high NEWSUB ;High byte of NEWSUB address.
                PUSH    A
                MOV     A, #NEWSUB_BANK ;Bank address of NEWSUB.
                PUSH    A

                LJMP    BANKJUMP        ;Call subroutine to switch operation
                                        ;to new bank. This will transfer execution

                                        ; to location NEWSUB in bank NEWSUB_BANK.

;*************************************************************************
;BANKJUMP -      This subroutine pulls the new jump and page address off the stack.
;               It modifies P1.0-3, and then modifies the stack pointer to point
;               to the new address. It then RETurns to the new jump location.
```

```
;                  This subroutine must be placed in a common, unpaged memory area.
;********************************************************************************************
BANKJUMP:     CLR     EA                  ;Disable all interrupts.
              POP     A                   ;Get the new page address off the stack
              ANL     P1, #BANKMASK       ;and modify only P1.0-3 to new bank
              ORL     P1, A               ;address.
              SETB    EA                  ;Reenable interrupts.
              RET                         ;Stack pointer is now at new return
                                          ;address. Program will begin executing
                                          ;from new bank.
```

Common–Page Expansion Using Latched Data

One drawback to the above design is that it requires I/O lines as bank controls. Some I/O intensive applications may not be able to spare port pins for bank switching. The following example uses a Lattice Semiconductor GAL26V12 programmable logic device (PLD) to latch the bank select signals, rather than using dedicated I/O pins. This approach uses the same memory map as the previous example, with the exception that data memory from FFE0h to FFFFh is not accessible. These 32 bytes are inaccessible because A5 through A0 are not decoded, allowing a smaller, lower cost PLD to be used. Decoding more address lines would reduce the amount of inaccessible data memory, but would require a more complicated decoding mechanism.

The GAL26V12 performs the bank switching function based on a write to MOVX data memory. Any write to data memory from FFE0h to FFFFh is decoded, and the lower four bits of the data written to that address are used to configure the bank switch select lines. The hardware configuration is shown in Figure 6.

The source file for the PLD follows the illustration. It scans for any address in the selected range and latches the lower nibble of the data onto A15 – A18 of the memory device. Addressing locations between 80000h will temporarily clear the bank select lines forcing the EPROM to read from bank 0 (0000h – 7FFFh). As soon as the lower memory operation is complete, accesses to the upper half of memory (8000h – FFFFh) will automatically return to the previous bank because the bank address is still latched in the registered outputs of the PLD.

Although a variety of PLDs are suitable for this application, any device used must reset its outputs to 0 on power–up. This is necessary because upon power–up the device must be able to access the reset vector located at 0000h in bank 0. When selecting a PLD, the designer should be aware that many standard programmable logic devices are designed so that their outputs go high upon reset.

DS80C320 LATCHED ADDRESS MEMORY HARDWARE EXAMPLE Figure 6
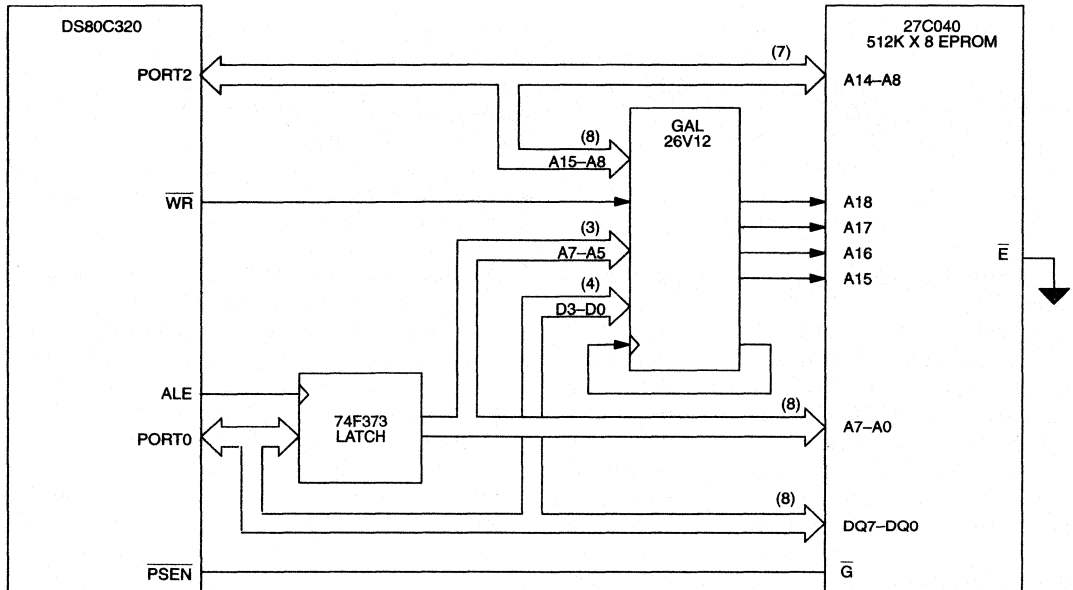
The following software example shows an assembly language routine to jump to a new location in any bank using latched data as shown in Figure 6. Before calling the bank switch subroutine, software pushes the new address and bank number onto the stack. It then calls a subroutine that pops the new bank address from the stack and writes it out to location FFFFh, where it is latched as the new bank address. The stack is then modified so that the subsequent RET instruction will return to the new program location.

PROGRAM EXAMPLE: JUMPING BETWEEN BANKS USING LATCHED ADDRESSING

```
;********************************************************************
;Program BANKJMP2.ASM
;
;This program demonstrates one possible way to jump between banks in a common
;page memory expansion configuration using latched addressing.
;
;Software pushes the address in the new bank, as well as the new bank address
;onto the stack. The subroutine BANKJUMP, located at a non-paged address,
;modifies the page selection, and then repositions the stack pointer to the
;new address which was previously pushed onto the stack. The RET is then
;executed, resuming operation from the new address in the new page. Note that
;BANKJUMP can be called from any area in program memory.
;********************************************************************
;Equate table
LATCH_ADR      equ      0FFFFh      ;Address of bank select latch.
NEWSUB_BANK  equ      03h          ;Subroutine NEWSUB is located on page 03h.
NEWSUB        equ      0F000h      ;Address of subroutine NEWSUB. If a compiler is
;                                   ; used, it may supply this automatically.
```

```
cseg        at          0                   ;Reset vector.
            LJMP        START


cseg        at          100H                ;Start of program

START:      MOV         A, #low NEWSUB      ;Low byte of NEWSUB address.
            PUSH        A
            MOV         A, #high NEWSUB     ;High byte of NEWSUB address.
            PUSH        A
            MOV         A, #NEWSUB_BANK     ;Bank address of NEWSUB.
            PUSH        A

            LJMP        BANKJUMP            ;Call subroutine to switch operation
                                            ; to new bank.

;********************************************************************
;BANKJUMP - This subroutine pulls the new jump and page address off the stack.
;            It writes the new page address out to the PLD, where it is latched
;            as the new page address. The RET then causes execution to the address
;            specified in the function that called this routine. This subroutine
;            must be placed in a common, unpaged memory area.
;********************************************************************
BANKJUMP:   CLR         EA                  ;Disable all interrupts.
                                            ; point to page address.
            POP         A                   ;Get the new page address off the stack.
            MOV         DPTR, #LATCH_ADR    ;Write page address out to latch.
            MOVX        @DPTR, A.
            SETB        EA                  ;Restore interrupts.
            RET                             ;Stack pointer is now at new return
                                            ; address. Program will begin executing
                                            ; from new bank.
```

The source file for the GAL26V12 PLD is presented to aid the designer in developing his or her own devices. The file was written in the CUPL language, but can be easily be modified to work with other assemblers.

PROGRAM EXAMPLE: GAL PROGRAM FILE

```
Name  MEM_EXP;
Device g26v12;
/********************************************************************/
/** This CUPL file will program a GAL26V16 as a bank switching    **/
/** latch to address up to 512 kbytes of program memory.  Any     **/
/** MOVX write to addresses FFE0h through FFFFh will latch D0 - D3 **/
/** as the new bank selection.                                    **/
/**                                                               **/
/** Any access to 0000h through 7FFFh in program memory will clear **/
/** the bank select lines for that operation only. This allows the **/
/** device to jump to interrupt vectors with no intervention, and **/
/** return to the same bank when finished.                        **/
/********************************************************************/
/*
/** PIN DEFINITIONS: These definitions are for 28 pin PLCC and DIP  **/
```

```
/** Input pins **/
PIN 1   = CLKIN;      Clock input for GAL latches.
PIN 2   = A15;        Microcontroller address lines for address decode.
PIN 3   = A14;
PIN 4   = A13;
PIN 5   = A12;
PIN 6   = A11;
PIN 8   = A10;
PIN 9   = A9;
PIN 10  = A8;
PIN 11  = A7;
PIN 12  = A6;
PIN 13  = A5;
PIN 14  = D3;         Microcontroller data lines to set bank selection.
PIN 15  = D2;
PIN 16  = D1;
PIN 17  = D0;
PIN 28  = WR_STROBE; Microcontroller write strobe

/** Output pins **/
PIN 20  = CLKOUT;     Qualified microcontroller write strobe. It is fed
;                     back into the GAL 26V12 CLK input. This
;                     signal must be assigned to pin 20 or 22 because
;                     it requires 12 product terms.
PIN 18  = A18_LATCH;  Temporary latch of A18 memory signal.
PIN 19  = A17_LATCH;  Temporary latch of A17 memory signal.
PIN 22  = A16_LATCH;  Temporary latch of A16 memory signal.
PIN 23  = A15_LATCH;  Temporary latch of A15 memory signal.

PIN 24  = MEMADR_18;  Output to memory device pin A18.
PIN 25  = MEMADR_17;  Output to memory device pin A17.
PIN 26  = MEMADR_16;  Output to memory device pin A16.
PIN 27  = MEMADR_15;  Output to memory device pin A15.

;PIN 7  = VCC
;PIN 21 = GND

/** Qualify write strobe to detect valid bank latch write. **/
BANK_SEL = A15 & A14 & A13 & A12 & A11 & A10 & A9 & A8 & A7 & A6 & A5
CLKOUT = !WR_STROBE & BANK_SEL

/**   BANK SELECT 0 GENERATION **/
A15_LATCH.D = D0
A15_LATCH.OEMUX = 0   ;Disable external expression of this signal.
MEMADR_15 = A15_LATCH.Q & A15

/**   BANK SELECT 1 GENERATION **/
A16_LATCH.D = D1
A16_LATCH.OEMUX = 0   ;Disable external expression of this signal.
MEMADR_16 = A16_LATCH.Q & A15
```

```
/**  BANK SELECT 2 GENERATION **/
A17_LATCH.D = D2
A17_LATCH.OEMUX = 0   ;Disable external expression of this signal.
MEMADR_17 = A17_LATCH.Q & A15

/**  BANK SELECT 3 GENERATION **/
A18_LATCH.D = D3
A18_LATCH.OEMUX = 0   ;Disable external expression of this signal.
MEMADR_18 = A18_LATCH.Q & A15
```

USING THE ROMSIZE FEATURE

The ROMSIZE feature allows software to dynamically reconfigure program memory size, permitting a portion of program memory to be switched between on– and off–chip. It provides an easy way to increase program memory to 64KB plus on–chip memory. In addition, it simplifies the task of building a boot–loader for external programmable memory, such as FLASH, EEPROM, or Nonvolatile SRAM (NV SRAM).

Using the ROMSIZE feature is very straightforward. Bits RMS2, RMS1, RMS0 (ROMSIZE.2–0) select the maximum amount of on–chip memory. The ROMSIZE select bits are Timed Access protected to ensure maximum software reliability. Any program memory accesses outside of the range defined by the ROMSIZE register will automatically be fetched externally via ports 0 and 2. External code fetches on devices with the ROMSIZE feature are performed in the same way as on all members of the High–Speed Microcontroller Family. The designer is reminded that if ports 0 and 2 will be used for external memory access, they should not be used as general purpose I/O ports.

The modification of the ROMSIZE register must be followed by a 2 machine cycle delay, such as executing 2 NOP instructions, before jumping to the new address range. Interrupts must be disabled during this operation, because a jump to the interrupt vector during the changing of the memory map can cause erratic results. The procedure to reconfigure the amount of on–chip memory is as follows:

1. Jump to a location in program memory that will be unaffected by the change,
2. Disable interrupts by clearing the EA bit (IE.7),
3. Write AAh to the Timed Access Register (TA;C7h)
4. Write 55h to the Timed Access Register (TA;C7h)
5. Modify the ROM Size Select bits (RMS2–0),
6. Delay 2 machine cycles (2 NOP instructions),

7. Enable interrupts by setting the EA bit (IE.7).

There are a number of software considerations when using the ROMSIZE feature to switch between on– and off–chip memory. Modification of the ROM Size Select register must be made from a program memory location that will be valid both before and after the on–chip memory configuration. Care must be exercised when assembling or compiling the program so that all the modules are located at the correct starting address, including the interrupt vector table.

If the 0k byte on–chip memory option is selected, extra precautions must be taken. It is necessary to duplicate the interrupt vector table in off–chip memory when switching the lower 1KB of program memory from on–chip to off–chip. In general, applications will find it most useful to reduce the on–chip memory no smaller than 1KB. This will maximize the addressable external memory range, while keeping the interrupt vectors on–chip. The 0KB option is most useful when the on–chip memory is only used as a boot loader.

EXPANDING MEMORY BEYOND 64KB WITH THE ROMSIZE FEATURE

Addressing more than 64KB of external memory in conjunction with the ROMSIZE feature is done similar to the ROMless method. The primary difference is that the ROMSIZE feature allows the designer to use the on–chip program memory as the "common" block. This simplifies the construction of external hardware, as the common block memory signal (the A15 signal in the examples presented) does not have to be decoded.

The key to designing with the ROMSIZE feature is to incorporate the on–chip memory into the memory map with the most efficient memory utilization and simplest decoding method. There are many approaches to this problem, but only one will be presented here. This example uses 16KB of on–chip memory, plus eight 48KB pages of expanded memory located in a 27C040

512KB EPROM. This provides a total program memory of 400KB. The interrupt vectors and service routines are contained in the on–chip memory for fast access.

Figure 7 shows one possible memory map for use with the DS87C520 incorporating 16KB of on–chip EPROM. Note that program memory from 0000h to 3FFFh on each external page is not used. This greatly simplifies the design of the memory decode, requiring no external logic and one less I/O line. Figure 8 shows how to use three I/O lines to directly decode the uppoer address

lines of the device. Software for this configuration is similar to that presented in previous examples.

It is also possible to use the ROMSIZE feature in conjunction with a latched bank address on the MOVX bus. Similar to the ROMless example, this approach does not require dedicated I/O pins for bank switching. Because the use of on-chip program memory allows for a simpler decode circuit, less expensive PLDs may be used.

ROMSIZE FEATURE COMMON–PAGE EXPANSION MEMORY MAP Figure 7

27C040 EPROM
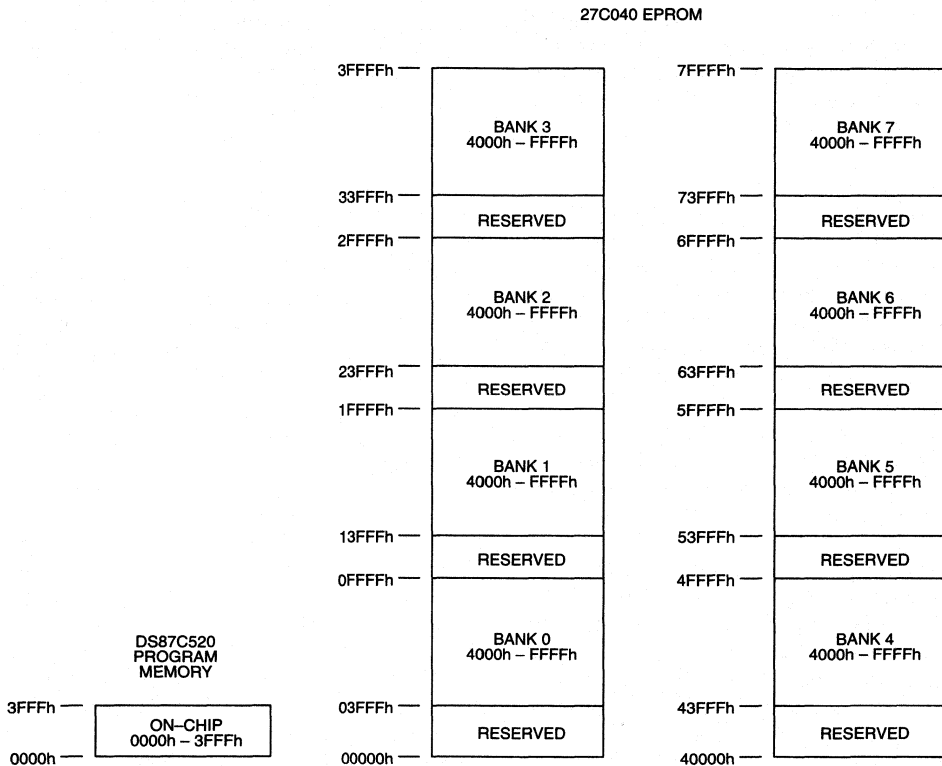
ROMSIZE FEATURE COMMON–PAGE I/O EXPANSION Figure 8
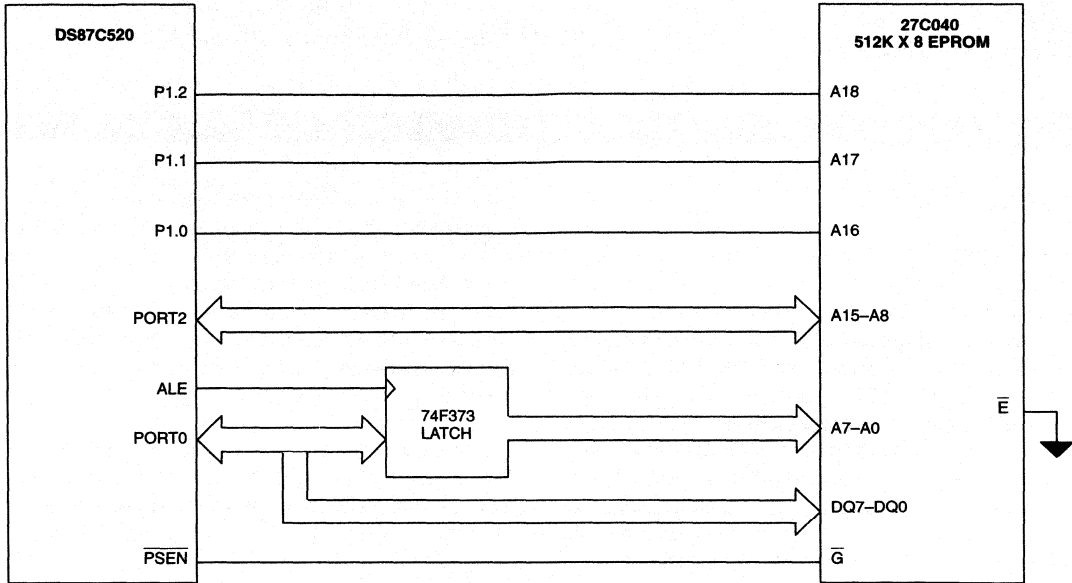
Expanding the amount of data memory used by the microcontroller is the easiest form of memory expansion. Because there is no possibility of interfering with program execution, timing is not as critical. General purpose I/O lines can be connected directly to the address lines or chip enables of the memory device(s). The appropriate port pin can be directly modified to access the correct page prior to the memory operation. If the application requires all available I/O lines, then a latched bank address scheme demonstrated in the above examples can be used.

DALLAS
SEMICONDUCTOR

Application Note 89
High–Speed Micro Memory Interface Timing

INTRODUCTION

The DS80C320 microcontroller was the first member of the new High–Speed Micro Family from Dallas Semiconductor. Since its introduction in 1993, there have been a number of new micros introduced which include new features and peripherals. The family now includes the DS80C310, DS87C520, and DS87C530, among others. The DS80C320 is unique in the family because it has slightly different external memory interface timing than the other members. When the DS80C320 was introduced, an application note (#57) was published to describe the speed requirements for the memories that interfaced to it. This application note serves the same purpose for the more recent family introductions.

One major difference in the timing of the new members of the family and the DS80C320 is that the original DS80C320 had a maximum clock rate of 25 MHz. All of the more recent 5 volt micros were introduced at a maximum clock rate of 33 MHz. Obviously, this affects the timing of the external memory interfaces significantly. The analysis that follows is based on a "worst case" timing using a 33 MHz clock, but will also identify memory speeds required for other frequencies as well.

A common configuration for a High–Speed Micro based system is shown in Figure 1. In this example, both program (EPROM) and data (SRAM) memory devices are included in the system. Of course with an EPROM based part such as the DS87C520, it is likely that no other program storage will be required outside the processor. However, for the purposes of this discussion, it will be assumed that external program storage will be used. If the application dictates the use of both on–board and external program memories, some additional decoding logic (not shown) may be required so that the two memory spaces do not overlap.

As with all 8051 external memory interfaces, Port 0 lines (P0.7–P0.0) of the processor carry both address and data, and a latch is used to demultiplex the information. The specific part number for the latch will be determined by the speed requirements of the system as discussed below. However, in general, the latch will be a '373 or '573 type. The main difference between these two part types is the pinout. On the '573, all inputs are on one side of the package and all outputs are on the other. This allows a more convenient board layout.

The number of lines connected from the processor's Port 2 address bus (P2.7–P2.0) to the memories' address bus is shown in the figure as 'N'. The exact value of 'N' is determined by the storage size of the memory devices used.

LATCH REQUIREMENTS

Due to the high speed of the Port 0 (AD7–AD0) bus, some consideration must be given to the choice of latches used for address demultiplexing. By examining the High–Speed Micro data sheet, it can be seen that some timing constraints are placed on the latch. For instance, the CPU parameter t_{AVLL} (Port 0 Address Valid to ALE Low) determines the minimum setup time (t_{SU}) the latch will actually have. The parameters t_{LHLL} and t_{LLAX} also affect the timing requirements of the latch. Table 1 shows the CPU parameters for 33 MHz operation, and the requirements placed on various latch families. For the parameters in the table, the CPU parameters must be greater than the latch parameters. It can be seen that the minimum required setup and hold time are violated for the HC latch family (highlighted). For this reason, this family cannot be used for 33 MHz operation.

The other relevant property of the latch is its propagation delay from input to output. Since the latch is in the address path, this parameter has a direct and significant impact on the memory timing requirements. This parameter will be discussed in the following section.

TYPICAL HIGH–SPEED MICRO SYSTEM Figure 1
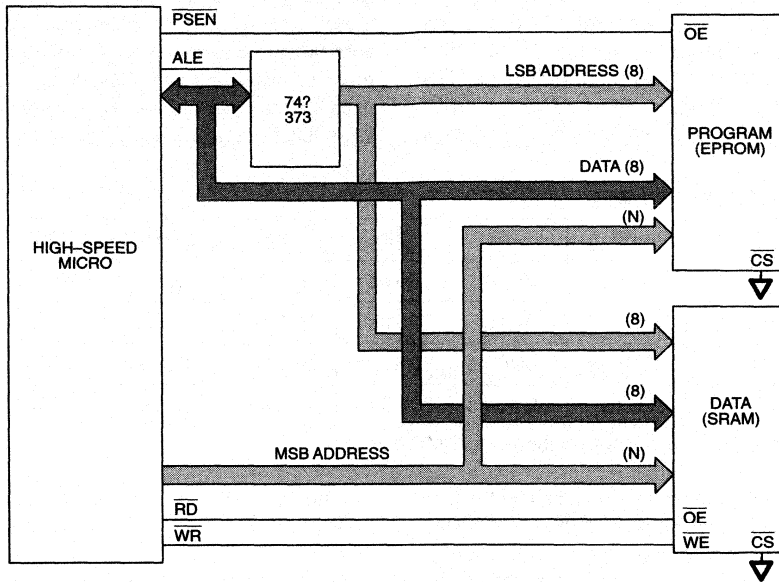
LATCH PARAMETERS Table 1

CPU PARAMETER	@33 MHz	LATCH PARAMETER	AC FAMILY	F FAMILY	HC FAMILY
t_{LLHL} min	40 ns	t_W	4.5 ns	6.0 ns	20.0 ns
t_{AVLL} min	10 ns	t_{SU}	6.0 ns	2.0 ns	**15.0 ns**
t_{LLAX} min	10 ns	t_H	1.0 ns	3.0 ns	**13.0 ns**
		t_{PROP}	11.5 ns	8.0 ns	38.0 ns

PROGRAM MEMORY

As shown in Figure 1, the program memory (typically EPROM) interfaces to the processor's LSB address through the address demultiplexing latch. The EPROM also interfaces directly with the processor via the MSB address bus, and the Program Store ENable ($\overline{PSEN}$) signal. The processor always outputs the MSB address before the LSB address, so this interface can be ignored. However, each of the two remaining interfaces must be examined for the most critical timing. The relevant timing parameters for EPROM devices may be found by evaluating their data sheets. Table 2 summarizes the main parameters for several speed grades of EPROMs from two different manufacturers[1].

EPROM PARAMETERS Table 2

PARAMETER	–55	–70	–90	–120	–150	–200	–250
t_{ACC} min	55 ns	70 ns	90 ns	120 ns	150 ns	200 ns	250 ns
t_{CE} min	55 ns	70 ns	90 ns	120 ns	150 ns	200 ns	250 ns
t_{OE} min	25/35 ns	30/40 ns	30/40 ns	35/50 ns	65 ns	75 ns	100 ns
t_{DF} min	25 ns	30/25 ns	30/25 ns	35/30 ns	30 ns	30 ns	30 ns

Note: X/Y indicates difference between the Atmel/AMD devices.

Upon evaluation of the timing specifications for the processor, the latch and the EPROM, it can be seen that the most critical timing path is through the LSB address bus. The address must appear on this bus, pass through the latch, address the EPROM, and the EPROM must output valid data in less time than the CPU parameter t_{AVIV}. Since the latch is in the path, the timing of this bus can be expressed by the following equation: $t_{PROP} + t_{ACC} < t_{AVIV}$. The DS87C520 data sheet shows that t_{AVIV} is a function of clock speed (denoted t_{CLCL}), and is given by: $t_{AVIV} = 3t_{CLCL} - 20$ ns. Solving these equations for 33 MHz operation using an F type latch, it can be seen that an EPROM access time of less than 63 ns is required. Therefore for full speed operation, an EPROM with 55 ns address access time must be used because this is the slowest speed grade that meets the requirement of 63 ns.

The equation above shows that the latch speed directly impacts the required speed of the EPROM. Since a fast latch is less expensive than a fast EPROM, an F type latch is recommended at clock speeds that would necessitate the use of an EPROM of 120 ns or faster.

Solving for the clock speed in the equation, an F type latch is recommended for clock speeds greater than 19.8 MHz. Based on the same argument, an AC type latch is recommended for clock speeds greater than 16.8 MHz.

Table 3 shows the EPROM speeds and latch types recommended for various CPU clock speeds. The suggested speed grade is based on the above equation and the EPROM and latch timing parameters. Further evaluation shows that the EPROM parameter t_{DF} may also be a critical parameter at some high CPU clock speeds. This parameter must always be less than the CPU parameter t_{PXIZ}. As indicated in Table 2, t_{DF} varies for the same speed grade device from different manufacturers. Therefore in Table 3, AMD is the recommended manufacturer where a CPU frequency of 29.4912 MHz is used. If the Atmel device were used, the t_{PXIZ} parameter would be violated. This exception applies to any clock frequency between 30.61 MHz where the switch of a 70 ns EPROM is made and 28.47 MHz where processor requirement for t_{PXIZ} becomes 30 ns.

---

1. EPROM devices from AMD and Atmel were considered.

RECOMMENDED EPROM SPEEDS Table 3

CLOCK FREQUENCY (MHz)	SPEED WITH 'F373 LATCH	SPEED WITH 'AC373 LATCH	SPEED WITH 'HC373 LATCH
33.0	**55**	55	N/A
29.4912	**70***	70	N/A
25.0	**90**	70	55
22.1184	**90**	90	70
20.0	**120**	90	90
19.8	120	**120**	90
18.432	120	**120**	90
16.8	150	120	**120**
16.0	150	150	**120**
14.746	150	150	**120**
14.318	150	150	**150**
12	200	200	**150**
11.059	200	200	**200**
7.37	250	250	**250**
1.8432 and below	250	250	**250**

* : Should be AMD 70 ns device because of t_{DF}.

DATA MEMORY

There are a number of factors that make interfacing data memories (SRAMs) to the High–Speed Microcontroller family extremely easy. First, SRAM devices are generally faster, and more readily available in higher speed grades. In fact, it is sometimes difficult to find a slow SRAM. A more significant factor is that all High–Speed Micro Family members have the ability to insert stretch cycles into the MOVX instructions. This provides a convenient means of supporting both high and low speed devices on the same data bus without requiring external support hardware. All High–Speed Micro Family members default to the use of one stretch cycle for MOVX instructions. To obtain maximum throughput, application software can write to certain Special Function Register (SFR) bits and cause the MOVX instructions to operate with zero stretch cycles. This default condition is a convenience to existing designs that may not have fast RAM in place. Even in high speed systems, it may not be necessary or desirable to perform data accesses at full speed. Additionally, there are a variety of memory

mapped peripherals such as LCD displays or UARTs that are not fast enough to keep up with the full speed high–speed micro. This flexibility allows the user to trade some performance for slower data RAMs if so desired. For the discussion that follows, a worst case timing scenario of zero stretch cycles will be assumed.

For maximum performance, i.e., with a zero stretch cycle data memory access programmed into the processor, a MOVX instruction requires two machine cycles. The fetch of the instruction takes one machine cycle leaving one machine cycle for the memory read or write. In the analysis of the data memory's timing requirements that follows, it will be assumed that the recommendations of Table 3 have been followed. Specifically, this means that an "F" family latch is used for CPU clock frequencies greater than 19.8 MHZ, an "AC" family latch is used for frequencies greater than 16.8 MHZ, and an "HC" family latch is used for lower frequencies.

Through analysis and a survey of memory products[2], it can be determined that four SRAM timing parameters are necessary and sufficient to meet the processor's timing requirements for most situations. These parameters and their values for various speed grades are shown in Table 4. During a data read operation, the processor expects the time from an address change until valid data is available to be 71 ns ($t_{AVDV1} = 3t_{CLCL} - 20$) or less. If the propagation delay from D to Q of an F373 latch (8 ns) is subtracted from this parameter, you obtain a memory address access (t_{AA}) requirement of 63 ns. Also for a data read operation, the time from the $\overline{RD}$ signal going low until valid data is received from the memory device must be 41 ns ($t_{RLDV} = 2t_{CLCL} - 20$) or less. Since the processor's $\overline{RD}$ signal is tied to the memory's $\overline{OE}$ pin, the memory must have an output enable access time (t_{OE}) of less than 41 ns. After the processor has read the data, the SRAM must relinquish the bus within 25 ns ($t_{RHDZ} = t_{CLCL} - 5$). This dictates that the SRAM parameter t_{OHZ} be less than 25 ns. For a write, the processor will provide a minimum write pulse of 56 ns ($t_{WLWH} = 2\ t_{CLCL} - 5$), which is equal to the minimum required write pulse width (t_{WP}) of the SRAM. On the basis of these four calculated parameters and assumed SRAM speeds shown in

Table 4, the appropriate speed device may be determined for a number of different clock frequencies. A summary of the recommended RAM speeds is given in Table 5. It should be noted that the critical timing parameter is not always the access time. Because of the high speed of the processor and variations in memory parameter relationships, all four parameters must be checked for any specific clock speed.

SRAM PARAMETERS Table 4

t_{AA} (ns)	t_{OE} (ns)	t_{OHZ} (ns)	t_{WP} (ns)
35	20	15	25
55	30	25	35
70	35	30	45
80	35	30	60
100	50	35	60
120	60	45	70
150	55	40	90
170	80	35	120
200	100	35	150

RECOMMENDED RAM SPEEDS Table 5

CLOCK (MHz)	LATCH	MEMORY SPEED (zero stretch)	MEMORY SPEED (one stretch)
33.0	F373	55 ns	150 ns
29.4912	F373	55 ns	170 ns
25.0	F373	80 ns	170 ns
22.1184	F373	100 ns	200 ns
20.0	F373	120 ns	200 ns
19.8	AC373	120 ns	200 ns
18.432	AC373	120 ns	200 ns
16.8	HC373	120 ns	200 ns
16.0	HC373	120 ns	200 ns
14.746	HC373	120 ns	200 ns
14.318	HC373	150 ns	200 ns
12	HC373	170 ns	200 ns
11.059	HC373	200 ns	200 ns
7.37	HC373	200 ns	200 ns
1.8432 and below	HC373	200 ns	200 ns

2. Memory data books from Dallas Semiconductor 1992–93, Fujitsu 1990, Hitachi #M18, Micron 1992, Mosel 1991–92, NEC 1989, Sony 1991 were surveyed for suitable SRAM products.

ADDITIONAL CONSIDERATIONS

All of the timing calculations used in this application note are based on equations found in the DS87C520 data sheet. These specifications assume an approximately equal capacitive load on the signals specified. If the configuration of Figure 1 is used, this is achieved. If, however, any signal is connected to additional loads, then the capacitive loading including the additional devices should be evaluated. If there is a significant difference, additional margins should be used in the critical path analysis, and appropriate memory speeds selected.

For older or otherwise unconventional SRAM devices, it may be wise to confirm other important timing parameters (such as data setup before write active). However, on the devices surveyed, meeting the four parameters discussed above will qualify the device for use.

EQUATION SUMMARY

For the user who wishes to calculate the memory speed requirements using a crystal frequency not shown in the preceding tables, the following equations provide a concise summary of the information needed.

EPROM Equations

$\overline{\text{PSEN}}$ Access

$$t_{OE} = t_{PLIV}$$
$$= 2t_{CLCL} - 20$$

Address Access

$$t_{ACC} = t_{AVIV} - \text{latch delay}$$
$$= 3t_{CLCL} - 20 - \text{latch delay}$$
$$(\text{same for RAM})$$

Bus Release

$$t_{DF} = t_{PXIZ}$$
$$= t_{CLCL} - 5$$

RAM Equations

Read Access

$$t_{OE} = t_{RLDV}$$

(zero stretch)
$$= 2t_{CLCL} - 20$$

(one stretch)
$$= 4t_{CLCL} - 20$$

Write Pulse

$$t_{WP} = t_{WLWH}$$

(zero stretch)
$$= 2t_{CLCL} - 5$$

(one stretch)
$$= 4t_{CLCL} - 10$$

Bus Release

$$t_{OHZ} = t_{RHDZ}$$

(zero stretch)
$$= t_{CLCL} - 5$$

(one stretch)
$$= 2t_{CLCL} - 5$$

DALLAS
SEMICONDUCTOR

<div align="right">

Application Note 91
Microcontroller Design Guidelines for Reducing ALE Signal Noise

</div>

OVERVIEW

The 8051 architecture allows for external program and data access through the use of Port 0 and Port 2 as an external memory interface. The 8051 architecture multiplexes the data and LSB of address on Port 0, requiring a 74373 latch for demultiplexing. This means that Port 0 will be directly connected to at least two devices. More devices may be placed on the bus if an external data SRAM or memory–mapped peripherals are used.

Because Port 0 must switch quickly between address and data, it requires strong current drive characteristics. The need to quickly switch many loads requires strong drive characteristics on Port 0. Unfortunately, the high instantaneous current requirements of quickly switching all the pins of Port 0 can induce noise on the ALE signal. In some instances, this noise can interfere with program and data accesses by causing the external hardware to latch an incorrect address. This is a relatively rare occurrence, and most designers will not encounter it. The magnitude of this problem is directly related to several issues associated with both the system and software. Devices which do not access external memory via Port 0 and Port 2 will not experience this problem.

This application note will discuss ways the system designer can reduce the effects of Port 0 switching on device operation. It is applicable to any ROMless 8051 microcontroller which accesses external memory via Port 0 and Port 2, including the DS80C310 and DS80C320. It is also applicable to any microcontroller with internal program memory that accesses external memory.

ALE NOISE GENERATION

Under certain system conditions, noise induced on ALE can cause an incorrect LSB address to be latched when using the multiplexed address/data bus. The noise, as seen in Figure 1, is generated by the high speed switching of Port 0 when the processor stops driving a memory address and begins driving data during a MOVX write. The noise pulse can, under the right conditions, rise above the V_{IH} input threshold of TTL, LS, FS and HCT logic. In this case, the 74373 latch may be falsely triggered, latching an incorrect address and disturb the LSB address of the MOVX write.

DATA MEMORY WRITE (IDEAL TIMING) Figure 1
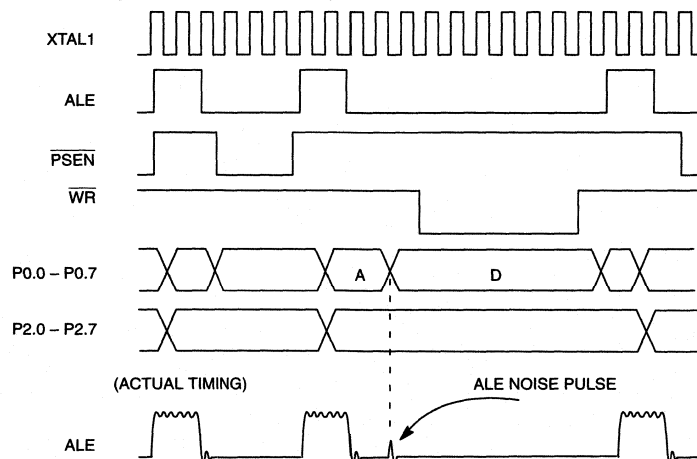
Figure 2 shows a system diagram of how the noise pulse is generated. The noise pulse is produced when the processor drives a Port 0 pin with a high address (see "A" in Figure 1) followed by a low for data (see "D" in Figure 1). The device must sink a relatively large amount of current on each pin (I_B) to take the line from a high to low state. It is obvious that the more pins which change from a high to a low, the larger the noise. The worst case will be during a MOVX write instruction with an LSB address of FF (Hex) and a data byte of 00 (Hex). Because all eight port pins are switching simultaneously, the maximum amount of current will be drawn into the microcontroller. The combined inductance and resistance both inside the processor and in the system result in the processor internal ground rising above the system ground. This in turn induces the noise seen on ALE. The case of a MOVX read does not involve the sinking of current by the processor and should not induce significant noise on the ALE signal. System elements which have a direct relationship to the magnitude of the noise are:

1. Port 0 bus capacitance
2. System ground inductance (L2) and resistance (R2)
3. System supply voltage (V_{CC})

ALE NOISE SOURCE Figure 2
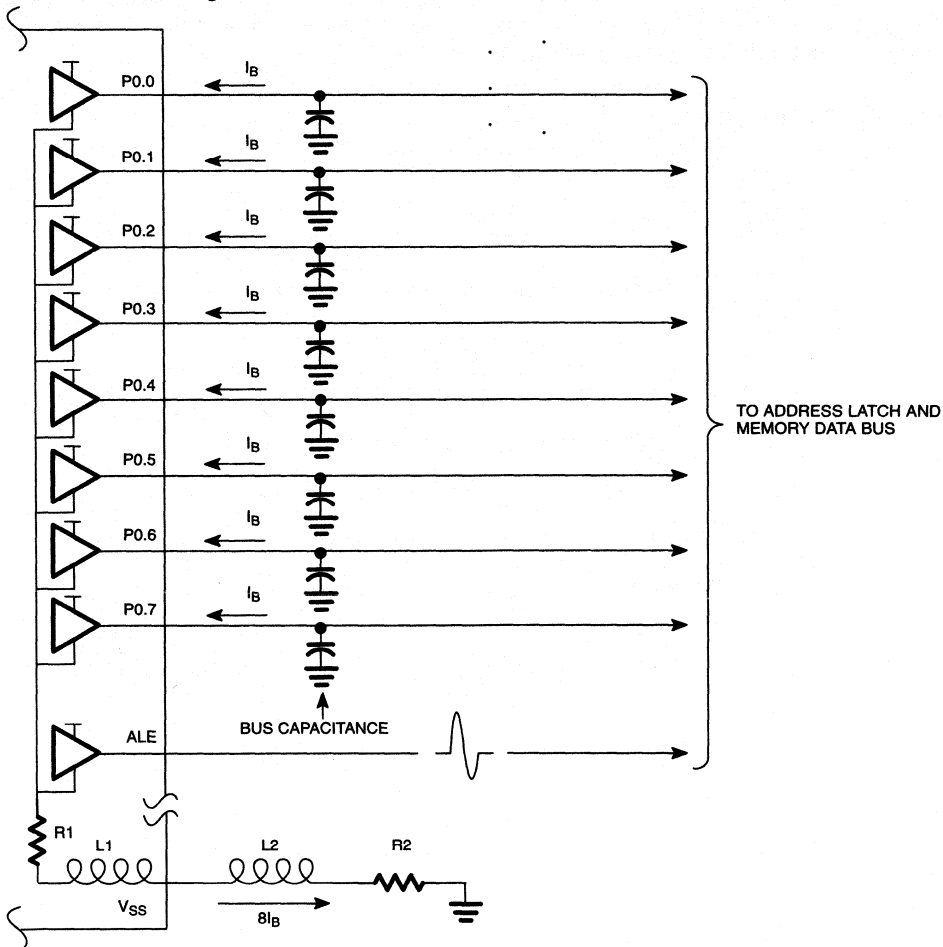
NOISE REDUCTION

There are several techniques that can be used to minimize the effect of Port 0 switching on ALE noise. Reducing bus capacitance reduces the energy required to be discharged which results in lower peak currents and reduced peak voltages in the noise pulse. Reducing the external ground resistance and inductance also reduces the noise level, by reducing the resistive and inductive voltage drop.

The supply voltage is also directly proportional to the voltage level of the noise pulse. Maintaining V_{CC} within recommended specifications will limit the noise voltage level.

Adding low impedance resistors in series with Port 0, as seen in Figure 3, reduces the noise level by limiting the peak current drawn into the microcontroller. Care must be taken to verify that these resistors do not adversely effect the slew rate or final input voltage level to the memory as the processor writes to external memory. Values in the range of 50Ω to 150Ω can generally be used without disturbing write cycle times. Actual values for the series resistance should be verified in the end system.

Use of a capacitor on the ALE signal line will also significantly reduce the noise pulse. Again, values must be verified in the system, with care used in not reducing the slew rate of the ALE signal to a point that memory access is no longer valid. Generally a capacitance of between 10 and 30 pF is sufficient to reduce the noise level without effecting normal system operation.

NOISE REDUCTION Figure 3

P0.0
P0.1
P0.2
P0.3
P0.4
P0.5
P0.6
P0.7
ALE

TO ADDRESS LATCH AND MEMORY DATA BUS

CONNECT TO CMOS LOGIC FOR HIGHER V$_{IH}$ THRESHOLDS, TO IMPROVE NOISE IMMUNITY

BUS DAMPING RESISTORS (50Ω TO 150Ω)

ALE DAMPING CAPACITOR (10 pF TO 30 pF)
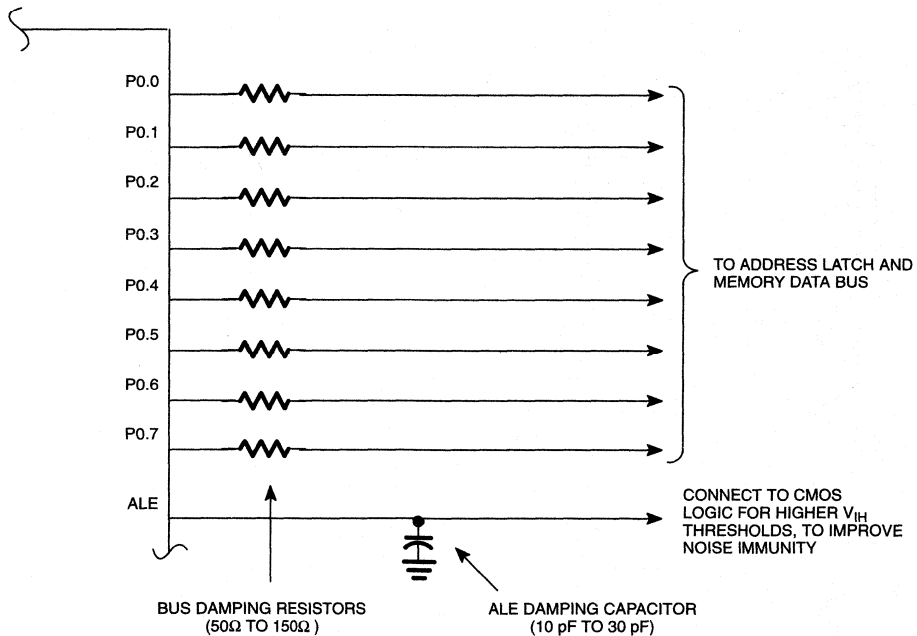
INPUT THRESHOLDS

The simplest and most reliable method of eliminating the address latch related noise is to select a logic family with a high input threshold. Standard TTL, LS, FS, and HCT logic parts have a V_{IH} threshold of approximately 2.0 volts. HC (High–Speed CMOS) or AC (Advanced CMOS) logic, on the other hand, has a V_{IH} of approximately 3.5 volts at a supply voltage of 5 volts. The higher threshold level of the HC or AC CMOS logic increases the noise immunity by approximately 1.5 volts. This is generally all that is needed to prevent the undesired latching by ALE.

One disadvantage of using CMOS logic is that it is slower than other logic families. Propagation delays through CMOS logic are generally in the range of 18 ns for HC and 10 ns for AC, compared with 2 to 4 ns for FS logic when using a supply of 5 volts. For slower micro-controllers such as the DS5000, DS5001, and DS5002, the propagation delay is usually not an issue because of the slow clock rate. Faster microcontrollers such as the High–Speed Microcontrollers should carefully consider the timing effects of slower logic. In any event, testing should be done in the final application to verify the effects using slower CMOS logic.

DALLAS
SEMICONDUCTOR

Application Note 92
DS5002FP Memory Expansion Techniques

OVERVIEW

All members of the Soft Microcontroller Family are designed to directly address up to 64KB of program and data memory. Occasionally, however, an application will require more memory than is available through the 64KB memory map. The Soft Microcontroller Family includes many features which make it easy to address program and/or data memory greater than 64KB. Bit-addressable I/O ports allow single instruction modification of control lines, which can be used to bank switch or page between multiple memory devices. This application note discusses the expansion of both program and data memory. It begins with an introduction to bank switching and software support techniques.

BANK SWITCHING THEORY

Expanded memory access beyond 64KB is done through bank switching. This technique uses one or more general purpose I/O lines as decode lines to address more memory. If a single large–capacity memory device is used, the additional signals can be used directly as address lines. If several smaller capacity memory devices are used, the signals can be used as chip selects. The basic unit of memory switched by the decode logic is called a bank or page. For example, if an I/O line was used to switch between two 64KB EPROMs, the memory would consist of two 64KB banks.

One difficulty in using in implementing a bank switching scheme is the placement of the interrupt vector table. During most of the device operation, software can perform an orderly switch between banks. When an interrupt occurs, however, the device will immediately jump to the appropriate vector address, below 0030h. The software has no control over the bank configuration at this point, and the device will attempt to jump to the low end of the current bank to seek the vector table. This means that the interrupt vector table must be duplicated in each 64KB bank, so it will be available regardless of the current memory configuration. Additional space may be required if duplication of interrupt service routines is desired on each bank.

Program efficiency is maximized when bank switching is kept to a minimum, i.e., executing straight runs of code. Code efficiency will be improved if the interrupt routines (not just the vectors) are small enough to be duplicated on each bank as well. Data tables or strings accessed with MOVC instructions should be located on the same bank as the instruction.

DESIGNING A BANK SWITCHING SCHEME

The approach described in this application note selects a new memory bank without modifying the program counter. This means that the starting location in the new bank will be relative to the location of the bank switch routine in the "old" bank. Consequently, instruction location between banks is critical. Table 1 shows the timing relationship between a MOV P1, #data instruction and the switching of the bank select signal. In this example, the MOV instruction is located at location 1000h, and the first instruction on the new bank is located at 1004h. The port pin which controls the bank selection will change during the last cycle of the MOV instruction. During this cycle, the DS5002FP will already be prefetching the NOP instruction at location 1003h from the same bank which contains the MOV instruction. The first fetch from the new bank will be at the address following the NOP, 1004h.

There are other ways of modifying port pins, and this scheme will work with 2 cycle instructions such as MOV *direct, direct,* and 1 cycle instructions such as SETB *bit.* The only requirement on the bank switching instructions is that the first instruction on the new bank must be at the address following the NOP as described above.

Current Bank		New Bank	
Address	Instruction	Address	Instruction
1000h	MOV P1, #01h	1000h	??
1001h	(second byte)	1001h	??
1002h	(third byte)	1002h	??
1003h	NOP	1003h	NOP
1004h	??	1004h	[First instr.]

For bank switching to work, it is necessary to fragment the code into banks and provide a means for the software to switch between banks. The software techniques presented here show how to switch program banks "on the fly." This approach causes program execution to jump directly from one bank to another without modifying the program counter. One must exercise caution so that the bank switch will occur at a location that corresponds to the start of the next instruction in the next bank. Failure correctly align the instructions may cause the next opcode fetch to occur in the middle of a multi–byte instruction, resulting in loss of program control.

Immediately following a reset, all port pins will go to a logic one state. This means that upon a hardware reset the device will begin operating from the bank which is selected when the port pins are high. Make sure that this bank contains the correct startup code to properly initialize your application.

Many compilers and linkers directly support bank switching, and many of them include library functions for bank switching. The documentation accompanying your compiler will provide information concerning its expanded memory support.

BANK LOGIC SELECTION Table 1

P1.0	P1.1	P1.2	P1.3	Memory Selected
0	1	1	1	Program Bank 0
1	1	1	1	Program Bank 1 (default)
0	0	0	1	Program Bank 2
1	0	0	1	Program Bank 3
0	0	1	0	Program Bank 4
1	0	1	0	Program Bank 5
0	X	X	X	Data Bank 0
1	X	X	X	Data Bank 1

PROGRAMMING

The serial bootloader is designed to program one 64 KB bank of memory at a time. Multiple banks can be programmed by using the serial bootloader software to manipulate the general-purpose port pins which control bank switching. In terminal mode, this is done via the "P" command, and via the "PUT_PORT" command using the K2.exe software available to load the DS5002FP using the DS5000TK Evaluation Kit. To properly load the program memory, the MSL bit must be cleared via the bootloader. This is done by the K2.EXE software with the RANGE 128 command or by clearing the MSL bit in terminal mode. To initialize data memory, the MSL bit should be set before doing a load operation. The procedure to load multiple banks is as follows:

1. Start the bootstrap loader.

2. Make sure the memory configuration is correct.

3. Modify the appropriate port pins to select the appropriate 64KB bank.

4. Load the software for that bank.

5. Exit the bootstrap loader.

INTERFACING TO BATTERY–BACKED SIGNALS

The memory expansion method described in this paper uses port pins in a simple fashion as bank selection controls. Please note that the port pins are not battery backed as such special care must be used when interfacing non–battery backed signals to battery backed circuits. This is especially important when external logic such as NAND and NOR gates are used in the address or chip select decode circuitry. The low leakage current of 74HCXX logic makes it appropriate for use in battery–backed circuits, as long as all gates of the 74HCXX logic are driven to either a high or low state during the battery backed mode. This is because the input stage of CMOS logic can draw very high current if the input is between the V_{IL} and V_{IH} levels.

The problem of floating CMOS inputs often arises in memory expansion circuits because the DS5002FP general purpose port pins (Ports 0, 1, 2, and 3) are used as inputs to battery backed logic. During battery backed mode, the port pins are tri–stated, and will float to an indeterminate state. Analog CMOS transmission gate logic can be used to isolate battery backed and non–battery logic since the source and drain of the transmis-

sion gate do not connect to the gates in the 74HCXX logic. Controls to these transmission gates, however, must be connected to battery backed signals.

A related problem can arise if battery–backed signals are connected to unpowered logic. During data retention mode, the $\overline{CE1}$, $\overline{CE2}$, $\overline{CE3}$, and $\overline{CE4}$ signals are driven to their inactive (high) state. These signals should not be connected to unpowered logic during data retention mode, or the DS5002FP will source an excessive amount of current and shorten battery life.

BANK SWITCHING EXAMPLES

Two examples are presented on the following pages. Both utilize 128KB SRAMs. This is the most cost–efficient method of implementing large amounts of memory. In this configuration, $\overline{CE3}$ serves as the A15 signal to the SRAMs, and $\overline{CE2}$ serves as A16, which is active during data memory operations. The first exam-

ple is a relatively simple scheme which employs 128KB of program and 128K of data memory. Two general purpose port pins are used to select the appropriate 64KB bank in program and data memory. The second example expands this to 192KB of program and 128KB of data memory. The more complicated memory configuration of the second example requires a 74HC4053 analog switch latch to isolate the bank select signals (P1.0–3) in data retention mode.

Figure 1 shows the memory map for the first example. The P1.0 signal is used to select one of two 64KB blocks of program memory. Note that immediately following a hardware reset, bank 1 will be selected because the P1.0 signal will be high. This bank should contain the code to be executed immediately following reset. P1.1 selects which of two 64KB blocks of data memory will be used by MOVX instructions. Figure 2 shows the interconnection scheme of the hardware.

SIMPLE BANK SWITCHING EXAMPLE MEMORY MAP Figure 1
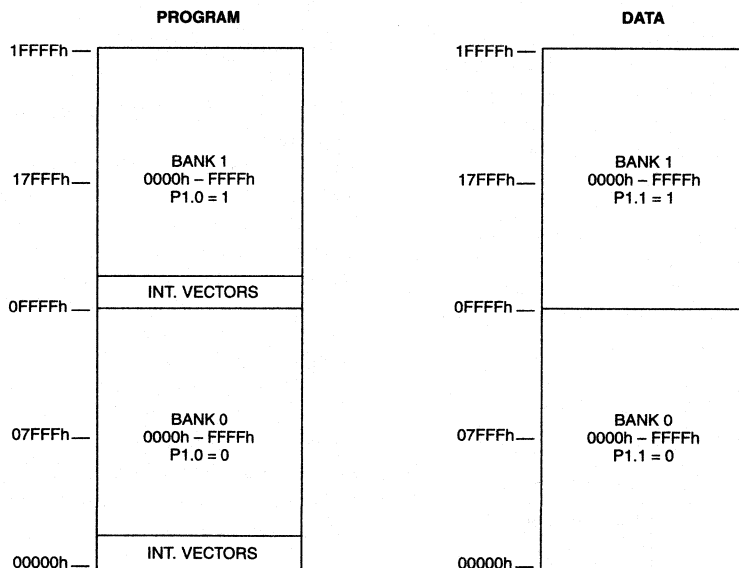
Figure 2
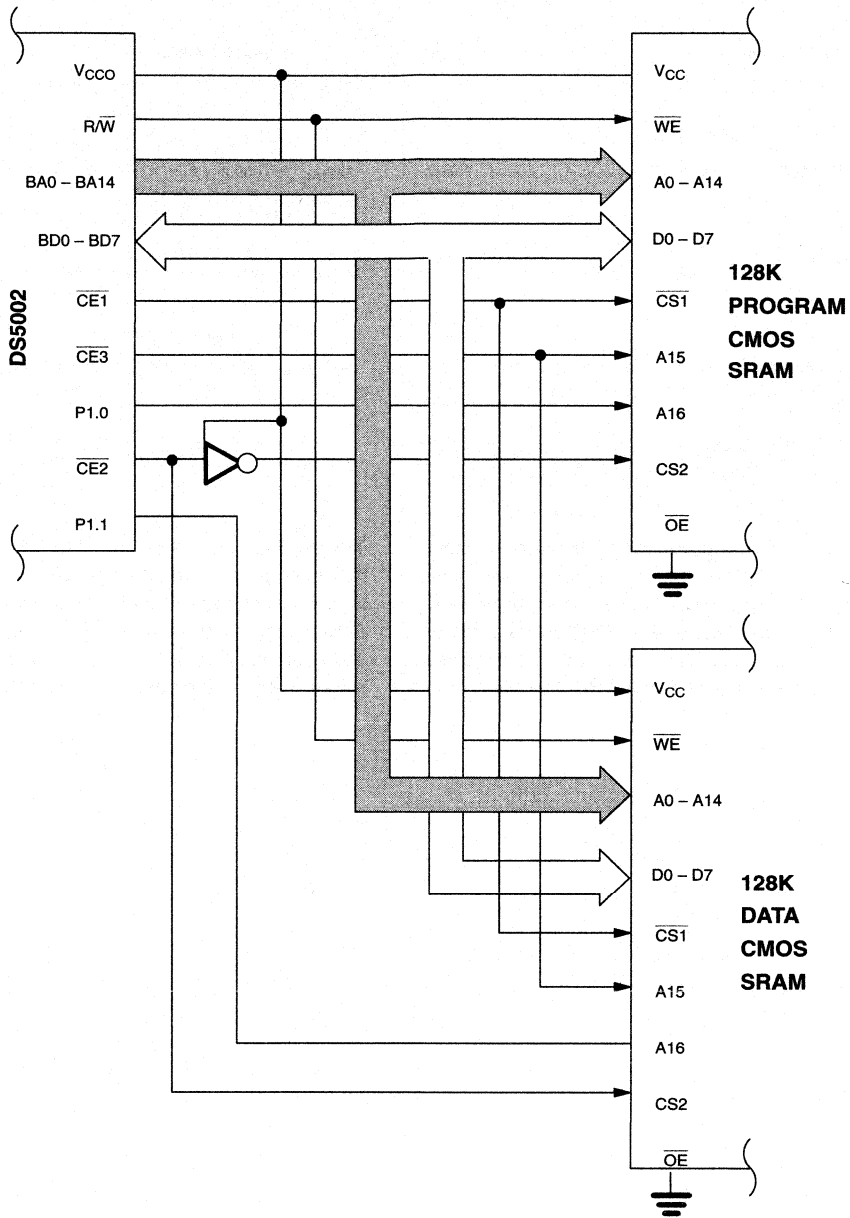
ADVANCED BANK SWITCHING EXAMPLE MEMORY MAP Figure 3
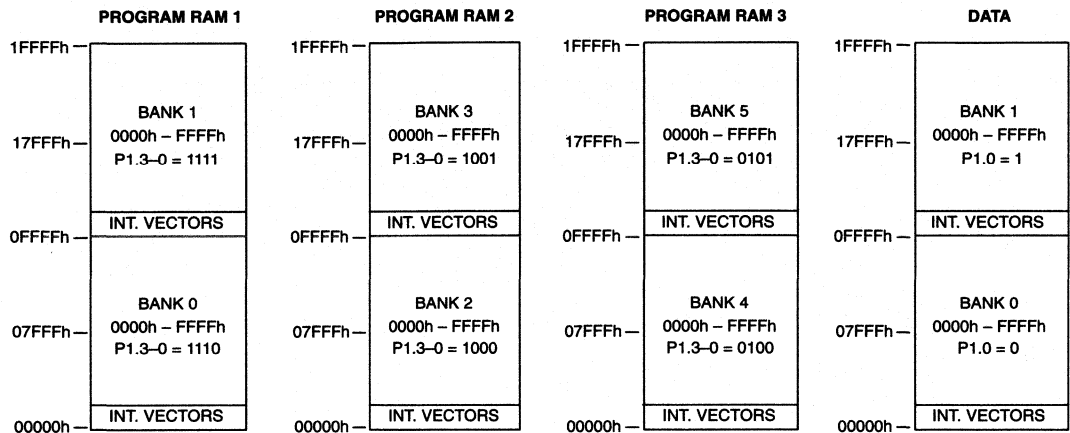
Figure 3 shows the memory map for the second example. The P1.0–3 signals are used to select one of six 64KB blocks of program memory. Note that immediately following a hardware reset, bank 1 will be selected because all port pins will be high. This bank should contain the code to be executed immediately following reset.

The decode logic through the 74HC4053 is designed so that each port pin selects one device. The bank select logic is described in Table 1. P1.0 selects which of two 64KB blocks of data memory will be used by MOVX instructions. Figure 4 shows the interconnection scheme of the hardware.

Figure 4

082495 6/6

DALLAS
SEMICONDUCTOR
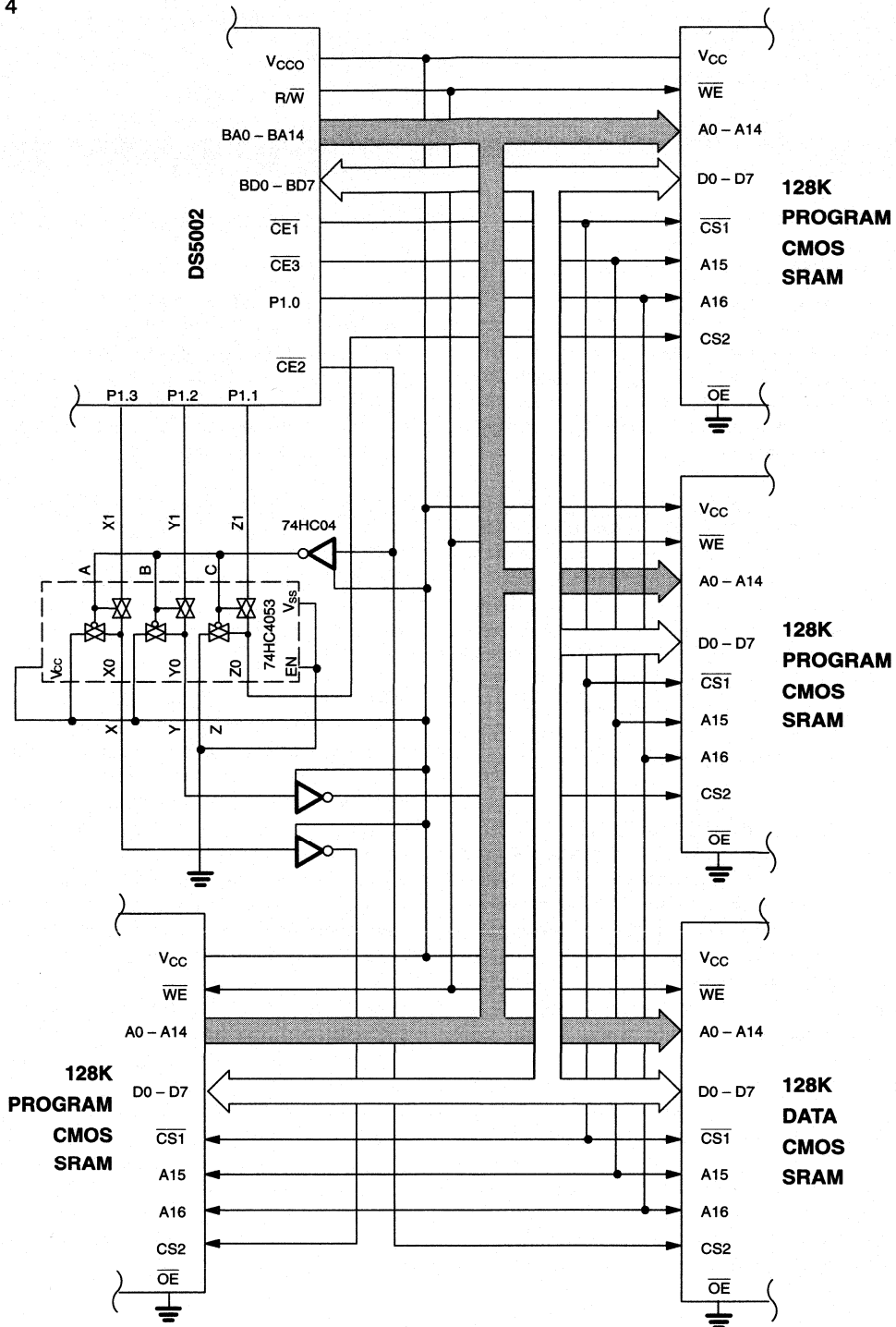
OVERVIEW

Microcontrollers which incorporates battery–backed, nonvolatile SRAM (NV RAM) have found wide acceptance in the embedded marketplace. Unlike Flash memory or EEPROM technology, nonvolatile SRAM has no write limitations, which makes it ideal for real-time data logging applications. In products such as the Dallas Semiconductor Secure Microcontroller family, NV RAM can be used to provide in–system reprogrammable program memory.

This application note discusses design guidelines for microcontroller products which incorporate NV RAM. A number of design suggestions are presented to improve the reliability of microcontrollers which incorporate NV RAM. It should be stressed that battery–backed memory is as reliable as nonvolatile memory as long as standard CMOS design guidelines are used. This application note is applicable to the Secure Microcontroller family of products (DS5000, DS5001FP, DS5002FP, DS2250(T), DS2251T, DS2252T) as well as the DS87C530 High–Speed Microcontroller.

OUT–OF–TOLERANCE VOLTAGE SPIKES

The real world is a harsh place; electrostatic discharge (ESD), electrical noise, etc. can enter into a system. Many of these phenomena can induce negative voltages on one or more device pins. CMOS design guidelines require that no pin be taken above V_{CC} or below V_{SS}. Violation of this guideline can result in a hard failure (damage to the silicon inside the device) or a soft failure (unintentional modification of memory contents).

Negative voltage spikes are a particular problem for CMOS devices. When a negative voltage spike is experienced, one or more parasitic diodes inside the device can become forward–biased. This will cause the device to consume a large amount of current, and can cause the device to latch–up. In general, the only way to reverse CMOS latch–up is to remove power from the device. If the supply to the device is not limited, the excessive current draw can cause irreparable damage to the device.

Less severe, but just as troublesome, is the effect this can have on nonvolatile memory. When a parasitic diode forms, a high–current path will form. This will drain current away from the device, and may cause the voltage to sag internally. If the internal voltage drops below the minimum needed to maintain the memory, soft errors may be experienced.

In a practical application, it is often difficult or impossible to remove all voltages above V_{CC} or below V_{SS}. Realistically, overshoots or undershoots of 0.3V can be tolerated by most devices. The following cases discuss common causes of such conditions and ways to minimize their effect on systems.

PROTECTING I/O PINS

The most common way that a device experiences negative voltage spikes is through its general–purpose I/O pins. These are usually the only contact that the microcontroller has with the "outside world." Devices such as electrical motors and keypads can generate large amounts of electrical noise. The shock that one experiences when touching a doorknob, for example, can be as much as 30KV. If this voltage is transmitted from a keypad into a microcontroller system, it can seriously damage electronic components.

Figure 1 demonstrates a diode protection scheme that can be used to protect the I/O pins of a microcontroller. The scheme relies on the use of Schottky diode and current limiting resistor to reduce the effect of current spikes on the device. When the voltage approaching the device pin exceeds V_{CC} or V_{SS} by more than 0.1V – 0.2V, the Schottky diodes will become forward biased, conducting the excess voltage away from the device pins. The current limiting resistors will also help dampen the effect of the voltage spike on the microcontroller.

The protection diodes shown in Figure 1 must be chosen carefully. Most Schottky diodes, such as the popular 1N5817, are suitable at room temperature. At high temperatures, however, their reverse bias leakage will begin to load the I/O line, making it difficult for the device to drive a logic one. For example, a DS87C520 can drive a logic one with a maximum current of 50 µA. At room temperature, the reverse leakage of a typical 1N5817 is 30 µA. At 70° C, however, the reverse leakage is greater than 100 µA. The microcontroller will be unable to source enough current to maintain a CMOS logic high of 2.4V.

Any Schottky diode used should have a leakage well below the maximum drive current of the sourcing device. Diodes with low reverse leakage currents are available from many vendors. Examples include the SC311 and SC015 diodes from Fuji Semiconductor.

EXTERNAL I/O PROTECTION SCHEME Figure 1
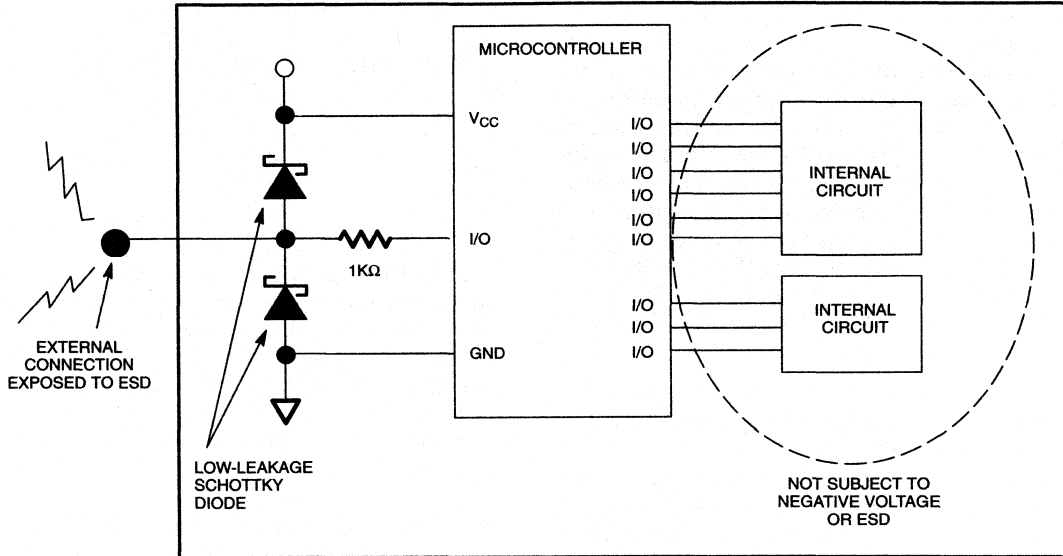
USER APPLICATION

NEGATIVE POWER SUPPLY TRANSIENTS

Another source of voltage spikes is the power supply. Many simple voltage regulators, such as a 7805 5V regular are notorious for generating voltage spikes as they power up. Care must be exercised when designing the power supply so that it does not generates excessive noise when powering up or when switching loads in and out of the system.

The problem of power supply transients can be lessened by using a reverse biased Schottky diode across V_{CC} and GND, similar to that used for protecting I/O pins. Figure 2 shows one such scheme.

VOLTAGE REGULATOR PROTECTION SCHEME Figure 2

DALLAS
SEMICONDUCTOR

Application Note 94
Using the Secure Microcontroller with EPROM/ROM
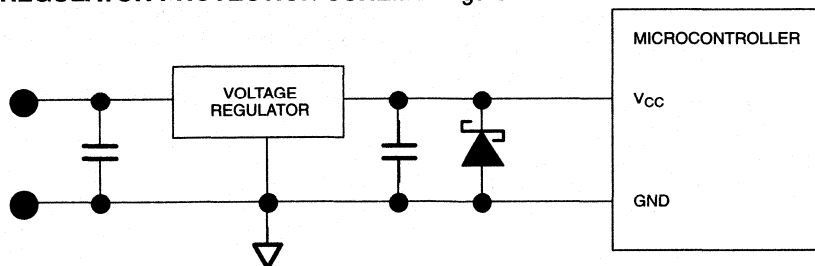
OVERVIEW

The Soft Microcontroller family is designed to take advantage of the many features provided by NV RAM technology. Occasionally a designer may wish to use an EPROM or other non–battery backed technology for program storage. While this sacrifices features such as the Bootstrap loader and program encryption, it will still allow a designer to use Soft Microcontroller features such as the power–fail monitor, watchdog timer and the I/O pins made available by the use of the embedded byte–wide bus. This application note addresses considerations when designing such a system using the DS5000FP and DS5001FP Soft Microcontrollers. This application note is not applicable to the DS5002FP Secure Microcontroller as it must use NV RAM to support the encryption features.

USING THE SOFT MICROCONTROLLER WITHOUT A BATTERY

The Soft Microcontroller will work reliably without a battery if a few design precautions are taken. The V_{BAT} input must be tied to V_{SS} if a battery will not be used in the design. A floating V_{BAT} input will cause unreliable device operation and/or a spontaneous reset.

Note that on DS5000FP devices revision D5 or earlier, the power–fail interrupt and power–fail reset will not work properly if V_{BAT} is tied to V_{SS}. On these devices, the V_{BAT} signal is used as a voltage reference to determine the reset trip point. The revision level is marked on each device as follows: xxxxRR–16, where RR is the revision level.

The lack of a battery means that registers that are typically nonvolatile will default to their no V_{LI} reset state. The registers shown in Table 1 should be reinitialized as part of a power–on reset routine to ensure that they are properly set. Note that upon a no V_{LI} reset the DS5001FP will be configured for a partition of 64KB and a range of 32KB. Although this is an invalid setting, the device will operate correctly below the 32KB boundary. This will allow the application software to properly configure the memory map before further execution.

NONVOLATILE REGISTERS Table 1

PCON	87h
MCON	C6h
Encryption Key	N/A
RPCTL	D8h
CRC	C1h

MIXING NV RAM AND ROM MEMORY

Many of the memory interface signals output from the Soft Microcontrollers are battery–backed in data retention mode. This ensures that the memory is write protected in the battery–backed state. While this is beneficial when using NV RAM, it can cause a problem if these signals are connected to memory which will be powered off in data retention mode.

The DS5001FP has a special version of the $\overline{CE1}$ signal that is not battery backed in data retention mode. This signal, $\overline{CE1N}$ can be directly connected to an external memory such as EPROM that will not be battery backed during data retention mode. In addition, $\overline{PE3}$ and $\overline{PE4}$ are not battery backed and can be attached to any kind of peripheral. During data retention mode the address and data lines of the microcontroller are driven to a low state, and so will not be affected by external logic.

Occasionally a design may require the connection of battery–backed signals to logic that will be unpowered during data retention mode. This is often the case if external decode logic is used to bank switch memory devices. In such instances an analog switch such as a 74HC4053 can be used to isolate the signals.

051295 1/1

430

DALLAS
SEMICONDUCTOR

Application Note 101
Using the Secure Microcontroller Watchdog Timer

OVERVIEW

Microcontrollers are often used in harsh environments where power supply transients, electromagnetic–magnetic interference (EMI), and electrostatic discharge (ESD) are abundant. Program corruption caused by bus corruption and electromagnetic discharges can cause a microprocessor to execute erroneous instructions. In these environments, a watchdog timer is a useful peripheral that can help catch and reset a microcontroller that has gone "out–of–control".

A watchdog timer is a simple countdown timer which is used to reset a microprocessor after a specific interval of time. In a properly operating system, software will periodically "pet" or restart the watchdog timer. After being restarted, the watchdog will begin timing another predetermined interval. When software or the device is not functioning correctly, software will not restart the watchdog timer before it times out. When the watchdog timer times out, it will cause a reset of the microcontroller. If the system software has been designed correctly, and there has been no hardware failure, the reset will cause the system to operate properly again. The reset condition must be a "safe" state. For instance, it would not be wise to have the reset state of a magnetic stripe card reader enabling the write head.

Many systems have been designed using an external watchdog timer. The Secure Microcontroller family eliminates the need for external components by incorporating an internal watchdog timer. By moving the watchdog timer inside the microcontroller, the number of devices in the system is reduced, increasing the overall system reliability. The watchdog timer can take advantage of the high–precision crystal oscillator used by the microcontroller, rather than the imprecise RC oscillator used by most independent watchdog timers. The operation of the watchdog timer is independent of the microcontroller, unless specifically addressed via the Timed Access procedure. The possibility of an out–of–control microcontroller accidentally disabling the watchdog timer is less than 1 in 7.2×10^{16}. This application note describes the features and use of the Secure Microcontroller's watchdog timer.

GENERAL USE OF A WATCHDOG TIMER

The primary application of a watchdog timer is as a system monitor to detect and reset an "out–of–control" microprocessor. When program execution goes awry it will not properly execute the code that restarts the watchdog. In such a case the watchdog timer will time–out and cause a microcontroller reset. In a properly designed system, the reset will correct the error.

Regardless of how capable a watchdog timer might be, there are certain failures that cannot be corrected by a reset. For instance, a watchdog timer cannot prevent or detect the corruption of data memory. Unless corruption of data affects program flow, or some extra measures are taken, data corruption will not cause a watchdog time–out. Of course, self–diagnostic software can be written in such a way as to make restarting the watchdog contingent on verification of data memory. While many applications implement such a data verification scheme, it is beyond the scope of this document.

It should be remembered that a watchdog timer cannot detect a fault instantaneously. By definition, the watchdog timer must reach the end of its time out interval before it resets the processor. The system designer should be aware of the maximum time interval that can occur between the execution of a bad instruction and the watchdog timer reset.

PLACING THE RESTART INSTRUCTIONS

In the Secure Microcontroller family, the watchdog timer is driven by the main system clock. The time–out interval is fixed at 122,800 machine cycles (1,473,600 external clock cycles). When the time–out is reached a reset will occur. Table 1 shows the reset time intervals associated with different crystal frequencies.

061695 1/3

WATCHDOG TIME–OUT INTERVALS Table 1

CLOCK FREQUENCY	TIME–OUT INTERVAL
16.0000 MHz	92 ms
14.7456 MHz	100 ms
11.0592 MHz	133 ms
7.73280 MHz	191 ms
5.52960 MHz	266 ms
1.84320 MHz	800 ms

A primary concern is the location of the watchdog timer reset command (setting the RWT bit) in the software. The most desirable approach is to have a single location within the main loop of the system software that restarts the watchdog timer periodically. The time required to pass through the main program loop must be less than the time–out interval or the device will reset itself during normal operation. In some systems, however, the program flow is not linear enough to allow the placement of a single watchdog timer reset function. Multiple reset functions should be placed in the code, corresponding to the longest software paths.

Often a system will need to know if a watchdog timer reset has occurred. The WTR bit (PCON.4) will be set whenever this occurs, and software can test for this early in the reset sequence if a system fault has occurred. If so, the system may decide to go into a "safe" mode and alert the user to an error condition.

WATCHDOG RESET EXAMPLE

A short program illustrating the initialization and basic function of the watchdog timer is shown below. It illustrates the Timed Access feature, which prevents the accidental modification of the watchdog control bits. A Timed Access operation is a sequence of steps that must be executed together, in sequence, otherwise the access fails. The example program shows the timed access being used for restarting the watchdog and enabling its reset. Further details on Timed Access operation may be found in the Secure Microcontroller User's Guide. The watchdog timer bits that are protected by the Timed Access procedure are the Enable Watchdog Timer Reset (EWT;PCON.2) and Restart Watchdog Timer (RWT;IP.7) bits.

```
;                       WD_RST.ASM Program
;
;    This program demonstrates the use of the watchdog timer.
;    When running, the program counts on port 1 to indicate the device is
;    running and periodically resetting the watchdog timer. After counting
;    to 16, it stops resetting the watchdog timer, simulating a system fault.
;
;    The program begins by checking to see if the WTR bit is set. If so, the
;    reset was caused by the watchdog timer, and the program will execute
;    the FAULT subroutine. Port 1 is set to F0h to indicate this condition.
;    If the WTR bit is not set, the reset was caused by another source and
;    execution should continue normally.
;***********************************
RWT      EQU     0BFh            ;Reset Watchdog Timer bit
TA       EQU     0C7h            ;Timed Access Register
PCON     EQU     87h             ;Power Control Register
ACC      EQU     0E0h            ;Accumulator
P1       EQU     090h            ;Port 1

         ORG     00h             ;Reset Vector
         SJMP    START


;***********************************
         ORG     080h            ;Program starts at 80h in this example.
START: MOV       A, PCON         ;If reset was caused by watchdog time-out,
         JB      ACC.4, FAULT    ; (WTR bit =1) execute fault subroutine.

;***********************************
;A normal power-on reset has occurred. Start initialization sequence.
         MOV     P1, #00h        ;Clear P1 to signal start of program.
```

```
;Watchdog timer initialization sequence
        MOV     TA, #0AAh     ;First restart the Watchdog timer
        MOV     TA, #055h     ;  using timed
        SETB    RWT           ;  access.

        MOV     TA, #0AAh     ;Next enable the Watchdog timer reset
        MOV     TA, #055h     ;  function using timed
        ORL     PCON, #04h    ;  access.


;**********************************
;Main program loop. This simulates a program that is operating
; correctly and then goes awry. After the program has counted to 16
; on Port 1 it will skip over the watchdog timer reset function. This
; will simulate a fault and allow the watchdog timer reset to be asserted.
;**********************************
MAIN:   MOV     R1, #0FFh     ;Create a delay loop. This simulates
LOOP1:  MOV     R2, #0FFh     ; a device actually "doing something."

LOOP2:  JB      P1.4, SKIP_WD_RST ;Have we been through loop 16 times?

        MOV     TA, #0AAh     ;Watchdog timer reset. In a user application it
        MOV     TA, #055h     ; should be placed at strategic locations
        SETB    RWT           ; where it will be executed periodically.

SKIP_WD_RST:
        DJNZ    R2, LOOP2
        DJNZ    R1, LOOP1

        INC     P1            ;Increment counter.
        SJMP    MAIN          ;Go back to main program loop.
;**********************************
;Watchdog time-out fault. This subroutine would normally have special
; routines to be executed in the event of a system fault. In this example,
; it disables the watchdog reset and sets Port 1 to F0h to indicate a fault.
; In a real application, this routine could either clear the fault and
; restart the software, or signal a fault and halt further operation.
;**********************************
FAULT:  MOV     P1, #0F0h     ;Signal a fault
        MOV     TA, #0AAh     ;Disable watchdog timer reset
        MOV     TA, #55h      ; using timed
        ANL     PCON, #0FBh   ; access.
        SJMP    $             ;Halt further operation.
```

SUMMARY

A number of considerations must go into any design that uses a watchdog as a monitor. Once the time–out period is determined, the system software must be analyzed to determine where to locate the watchdog restart instructions. For an effective design, the number of watchdog restarts should be kept to a minimum, and some consideration should be given to the likelihood of incorrectly executing a restart. As mentioned previously, some system software is too convoluted or data dependent to ensure that all software flow paths are covered by a watchdog restart. This may dictate that a self–diagnostic software approach might be required. If there is an expected failure mechanism such as a periodic EMI burst or power supply glitch, the watchdog time–out should consider this period.

DALLAS
SEMICONDUCTOR

Application Note 102
Using the High–Speed Microcontroller as a Bootstrap Loader

OVERVIEW

Some members of the High–Speed Microcontroller Family incorporate internal EPROM or ROM for program storage. Some applications, however, require in–system reprogrammablility of program memory. Such a system can be easily implemented using the Dallas Semiconductor High–Speed Microcontroller with internal program memory to reload an external nonvolatile memory such as Flash memory or NV RAM. Using the internal program memory as a bootloader allows a lower cost solution than could be obtained by using a device with internal NV RAM program memory or a costly boot–block Flash memory.

The most common bootloader configuration incorporates two elements: a microcontroller programmed with the bootstrap loader, or bootloader, and an external memory device such as an NV RAM or Flash memory to hold the user application software. When the system resets, following a power–on or external reset, it will begin executing instructions out of the microcontroller's internal program memory. The bootloader code, inside the microcontroller begins by checking for a "loader/no–loader" stimulus such as a logic low on a specific port pin, serial port character, etc. This allows the system to decide if it should load a new user program, or if execution should begin using the existing user program. If the stimulus is not received, indicating that no load is desired, the device will disable internal program memory via the ROMSIZE feature and begin execution from external memory. If the stimulus is present, the device will execute the bootloader routine, and begin reprogramming the external memory.

This application note will provide examples of how a designer can use an external Flash or NV RAM to add in–system reprogrammability to a High–Speed Microcontroller based design. General hardware and software design guidelines are presented. Software examples to support the techniques described herein are available in electronic format from Dallas Semiconductor via our anonymous FTP site (Internet) or BBS.

INVOKING THE BOOTLOADER

There are several ways to invoke the loader. The simplest is to dedicate a general purpose I/O pin to be sampled as part of the reset routine contained inside the High–Speed Microcontroller. The examples in this application note utilize P1.7, INT5, because it is least likely to interfere with existing 8051 code designs. Following a reset, the device will begin executing code from internal EPROM. The internal program will perform a quick test of the pin to determine if the loader should be invoked. Because this pin defaults to a high state after reset, it is recommended that a low condition on this pin be used as the signal to invoke the loader. Using an interrupt pin also allows the device to invoke the loader at times other than reset via the interrupt service routine. The method used to assert a logic low can be as simple as a dedicated switch, or a more complicated connection via the RS–232 cable from the host that pulls the pin low when connected.

An alternate method involves using the serial port to invoke the bootstrap loader. Upon reset, the device can continuously poll the serial port for a character. If a character is not received in a specified period of time, the program will exit the loader and begin execution from the external memory. This approach has the advantage of not requiring a general purpose port pin. Its primary disadvantage is that the device will experience a fixed delay every time it is reset before running the user application.

EXITING THE BOOTLOADER

After loading the new software to external memory or if loader operation is not desired, the system will need to exit the loader and begin execution from external program memory. The ROMSIZE feature provides a fast and convenient way to do this. The ROMSIZE register allows software to "turn off" internal program memory and force all program execution externally, similar to pulling the $\overline{EA}$ pin low. The software should then execute an LJMP to the reset vector at 0000h.

The ROMSIZE register must be modified from an external memory location that is outside the memory range of internal memory. For example, the DS87C520 contains 16KB of EPROM. The instruction to modify the ROMSIZE register should be located in external memory at an address of 4000h or greater. Failure to do this could result in in code execution failure if the memory map switched in the middle of an instruction.

The simplest way to do this is to map a short routine (~16 bytes) at the high end of memory. If a 64KB memory space is used, this could be at FFF0h. This offers the least chance of interfering with the user application code. Upon completion of the loader routine, software will jump to external memory (location FFF0h in this example), modify the ROMSIZE register to disable internal program memory, and then jump to location 0000h. This simulates a reset of the user application code. It is important to include a NOP or other dummy instruction following the modification of the ROMSIZE register to allow one machine cycle for the memory select circuitry to disable the internal program memory. The following routine is suggested:

```
CSEG at 0FFF0h
    MOV     TA, #0AAh
    MOV     TA, #55h
    MOV     ROMSIZE, #0h
    NOP
    LJMP    0000h
```

BOOTLOADER SOFTWARE
There are many different features that can be included in a bootloader, which vary depending on the specific memory device used. In general, these should include load, verify, and CRC commands. Flash memory devices will require chip erase commands, and an NV RAM may find a fill command useful.

Figure 1 illustrates the basic operation of a bootloader program. The software begins by checking for the signal to start the loader. If present, the device performs user-requested bootloader functions. When complete, or if no loader stimulus was detected, the device will jump to location FFF0h, modify the ROMSIZE register to disable internal EPROM, and jump back to the restart vector. This simulates a device beginning execution at address 0000h following a reset.

A sample bootloader for the High-Speed Microcontroller family is available from Dallas Semiconductor. The assembly language source file HSM_LOAD.ASM incorporates optional include files to support several different memory types, including Flash and NV RAM. The following is an list of some of the commands that are supported:

- Load Intel Hex file to memory

- Verify Hex file against memory

- Erase chip (Flash only)

- Fill range of memory with data (NV RAM only)

- Calculate CRC

- Modify and read port values

- Dump memory contents in Intel Hex format

- Exit loader

BOOTLOADER FLOWCHART Figure 1

INTERNAL EPROM

EXTERNAL
PROGRAM MEMEORY

RESET

IS LOADER
STIMULUS
ACTIVE? → N

Y

PERFORM 1 OR MORE
LOADER FUNCTIONS

EXIT
LOADER?

N

Y

JUMP TO FFF0h

MODIFY ROMSIZE
REGISTER USING
TIMED ACCESS

JUMP TO 0000h

BEGIN USER
APPLICATION AT
ADDRESS 0000h
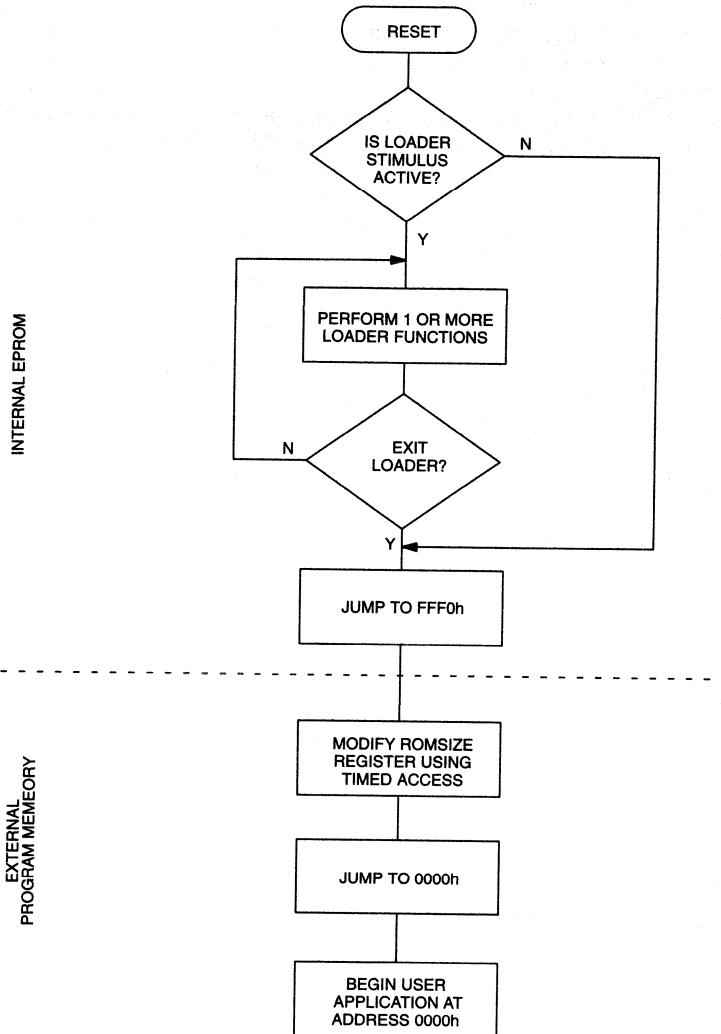
HARDWARE

Figure 2 illustrates the use of a 29F010 5.0V–only Flash memory as part of a bootloader design. Although this configuration was designed to be compatible with the software presented in the file HSM_LOAD.ASM, it can be easily adapted for other Flash memory or NV RAM devices. Some flash memory devices such as the 28Fxxx series require an external 12V V_{PP} for programming. Designs which incorporate these devices will need to include a voltage source.

Although the 29F010 is a 128KB device, this example only uses 64KB by tying the A16 address line low. This will not effect the Flash memory command functions as most devices are designed to ignore A16 during their programming algorithms. To use A16, connect it to an unused general purpose port pin. Be aware that port pins will default to a logic high state, selecting the upper 64KB of Flash memory. An inverter can be used between the port pin and A16 on the Flash device to have it select the lower 64KB on power–up. This will simplify code placement. The "P" command of the bootloader can be used to manipulate the A16 pin of the Flash memory during the loading process to load both the upper and lower 64KB banks of the device. More information on bank switching can be found in Application Note 81, Memory Expansion with the High–Speed Microcontroller Family.

BOOTLOADER HARDWARE Figure 2
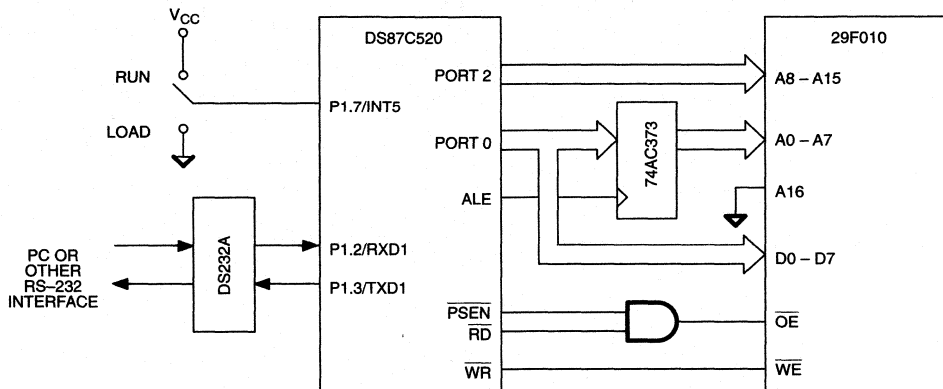
USING NV RAM WITH THE BOOTLOADER

NV RAM has several advantages when used in a bootloader design. It does not require the complicated byte programming algorithm, making it much faster to program. Both data and program can be stored in the same chip, reducing board space while gaining the advantage of data nonvolatility. The use of NV RAMs with internal partitioning, such as the Dallas Semiconductor DS1630 Partitionable 256K NV SRAM or the DS1645 Partitionable 1024K NV SRAM, prevents accidental code corruption. Optional include modules are available for the HSM_LOAD.ASM file to support many Dallas Semiconductor NV RAM products.

ADDENDUM FOR REVISION A4 DEVICES

DS87C520 and DS87C530 devices revision A4 and earlier incorporate an errata pertaining to the $\overline{PSEN}$ signal that requires a minor change to the above hardware. On these devices, the $\overline{PSEN}$ signal toggles regardless of whether the device is operating from internal or external program memory. This will cause a conflict during writes to the Flash memory device. Figure 3 illustrates a temporary workaround. This configuration places the restriction that the Load signal on P1.7 must be removed before beginning operation from external memory. This is a minor point, as most applications will program the Flash memory in a separate step, reset the device, and then begin operation. This errata will be corrected on later revisions of the devices.

BOOTLOADER HARDWARE FOR DS87C520/DS87C530 REVISION A4 Figure 3
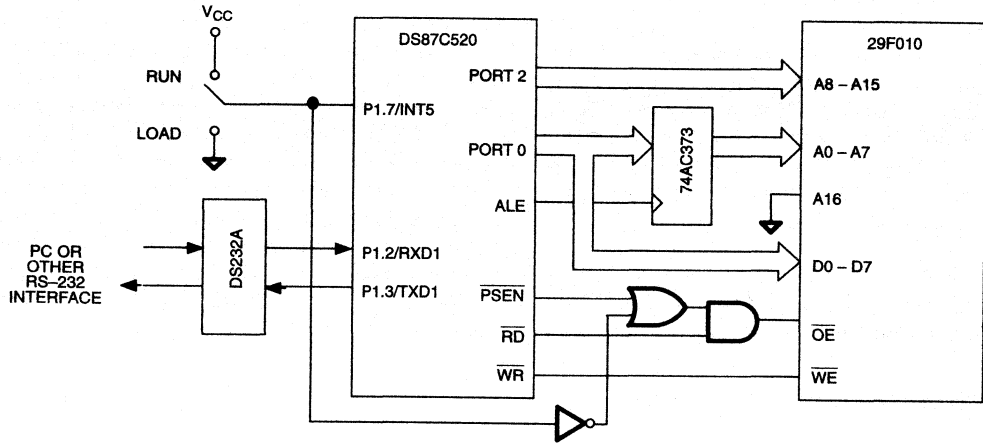
SCSI TERMINATION

DALLAS
SEMICONDUCTOR

The DS2107A is used in SCSI systems to provide active termination for 9 signal lines. In the typical 8 bit–wide data configuration (A cable), two DS2107A's are required to fully terminate the bus (9 control lines + 8 data lines + 1 parity line). In the 16–bit wide data configuration (P cable), three DS2107A's are required to fully terminate the bus (9 control lines + 16 data lines + 2 parity lines). The two packages available are DS2107AS, 16–pin SOIC, and DS2107AE, 20–pin TSSOP (Thin Shrink Small Outline Package).

The DS2107AS and DS2107AE were tested with various cable configurations. The following waveforms are the results of this testing. In all plots the driver (7438) is displayed on top, and the receiver on bottom. The 7438 driver has a V_{OL} of 0.4 Volts. In all plots the voltage scale is 1 volt/division.

For Cases 1 through 8, the time scale is 100 ns/division, and a 50 conductor 28 AWG shielded twisted pair cable was used. The characteristic impedance of the cable is 75 ohms which is within the SCSI–3 Parallel Interface specification of 72 to 96 ohms.

For Case 9, a flat ribbon cable of 100 ohms characteristic impedance was used to depict a typical internal SCSI bus configuration. For this case, the time scale is 20 ns/division for better visibility of leading and trailing edges of the signal. Figure 9a shows the leading and trailing edges of the SCSI bus waveform when one line is active. Figure 9b shows the results of a higher duty cycle, active SCSI bus because the other 8 resistors of the terminator package were grounded, or "pulled to true state." This in effect heats up the termination IC. The DS2107AS and DS2107AE do not adversely affect the SCSI bus under these loaded conditions.

KEY:

△ = 7438 DRIVER

$\boxed{\mathbf{R}}$ = DS2107A ENABLED (RECEIVER)

☐ = DS2107A ENABLED

◯ = DS2107A DISABLED

To disable the DS2107A, the power down pin is grounded.

CASE 1 Figure 1

10 MHz High/Low Pulses

7m
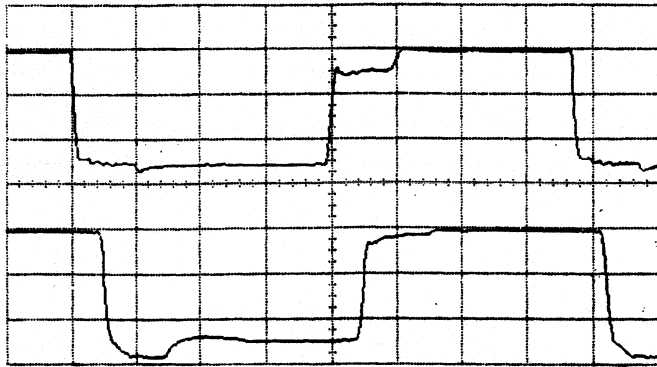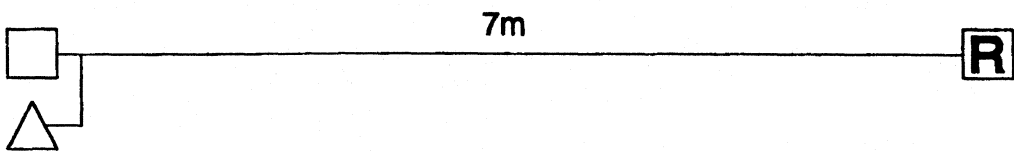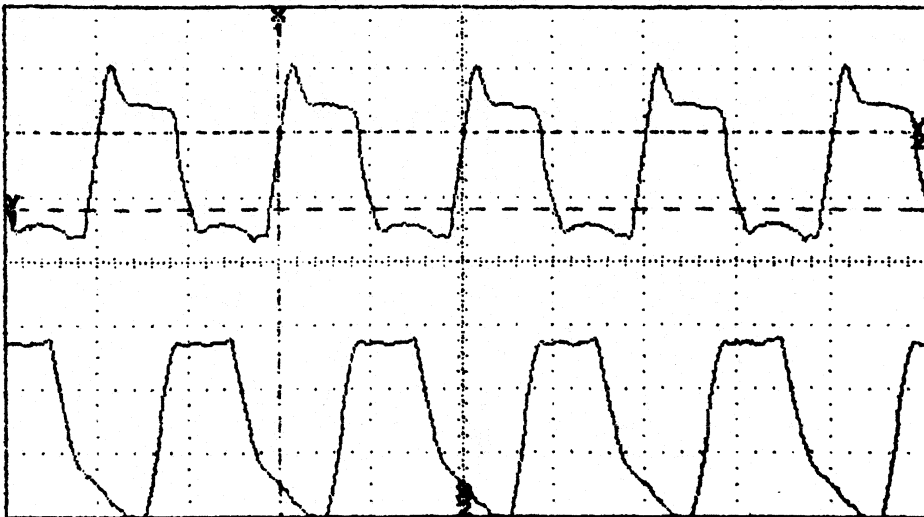
CASE 2 Figure 2

10 MHz High/Low Pulses
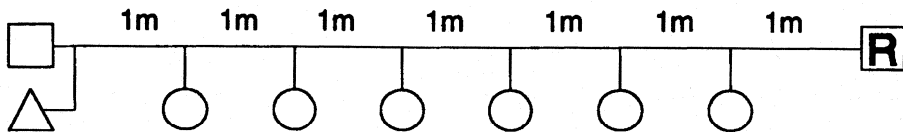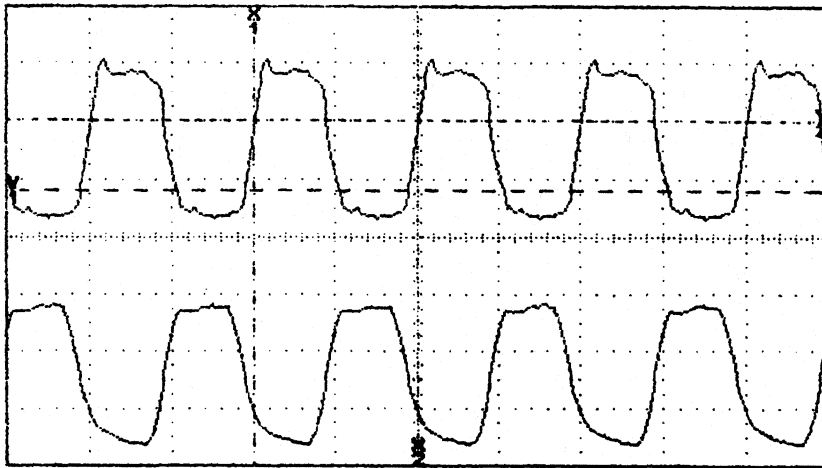
CASE 3 Figure 3
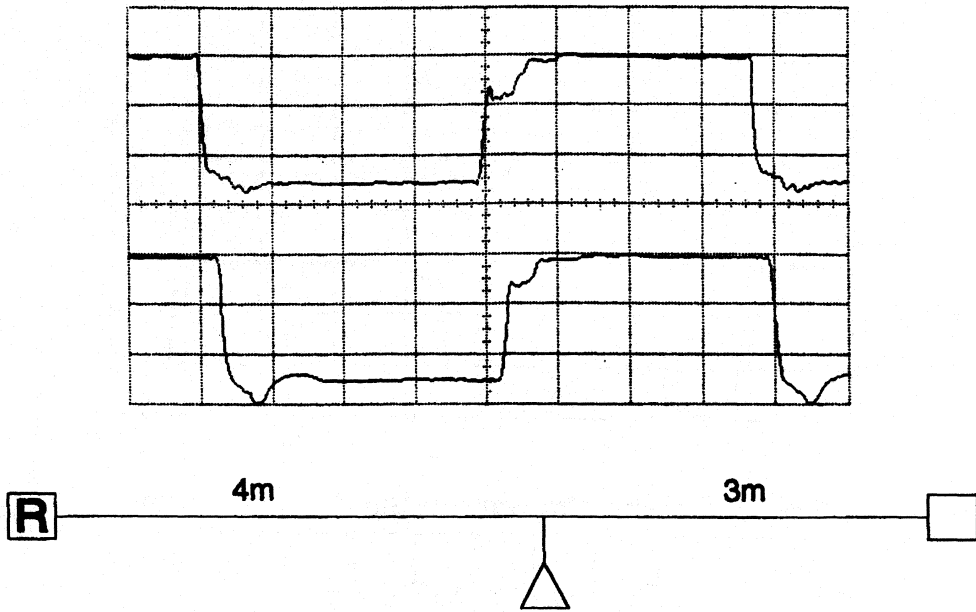
CASE 4 Figure 4
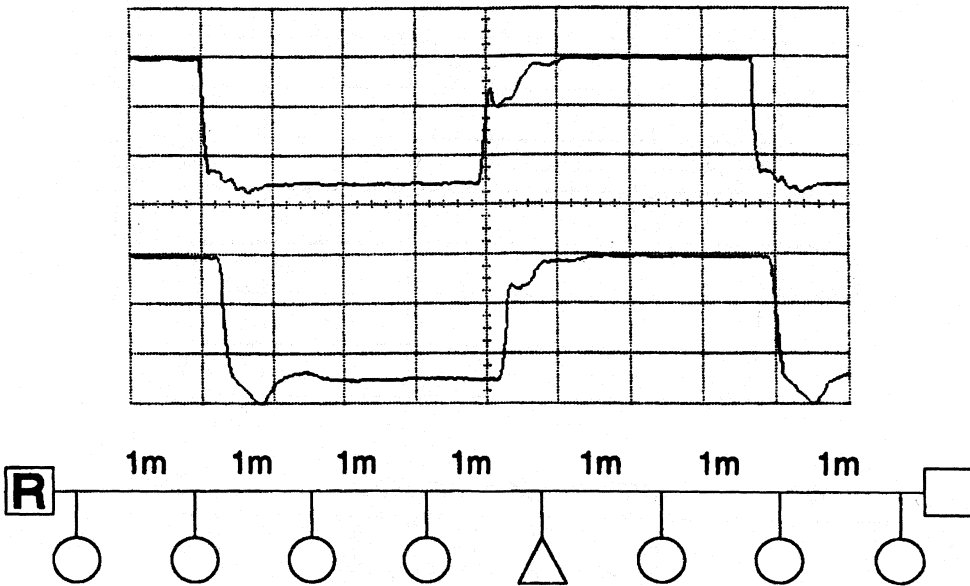
CASE 5 Figure 5
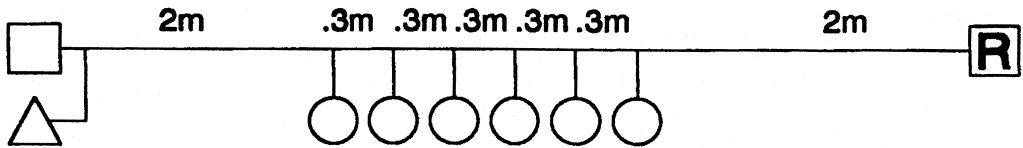
CASE 6 Figure 6
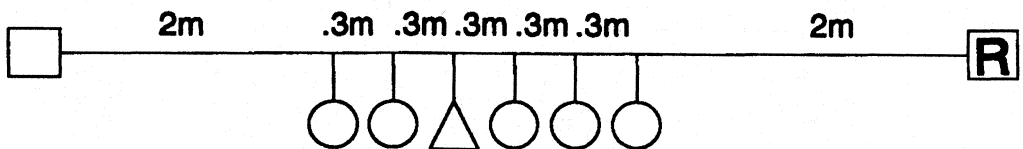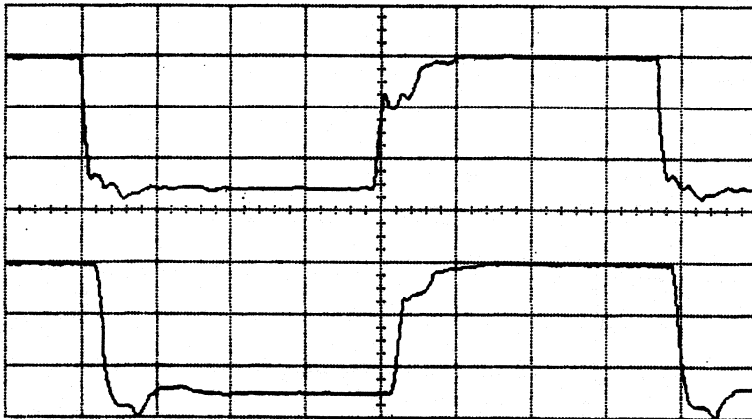
CASE 7 Figure 7
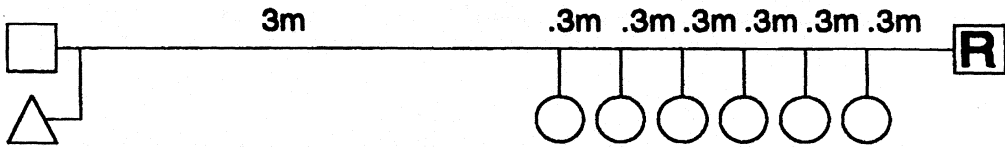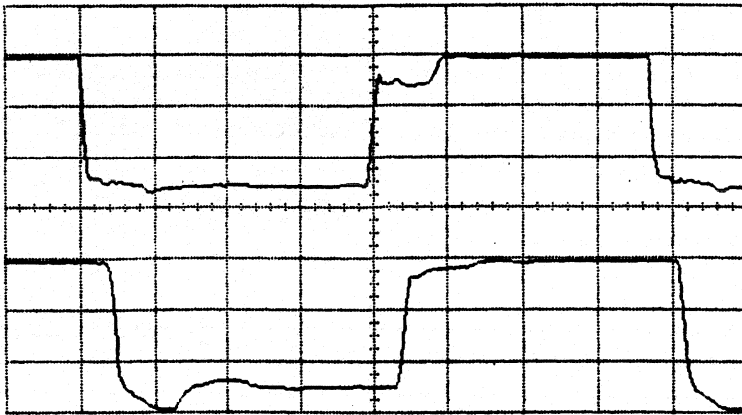
CASE 8 Figure 8

082394 6/8
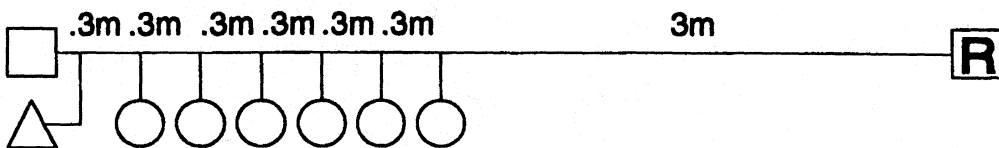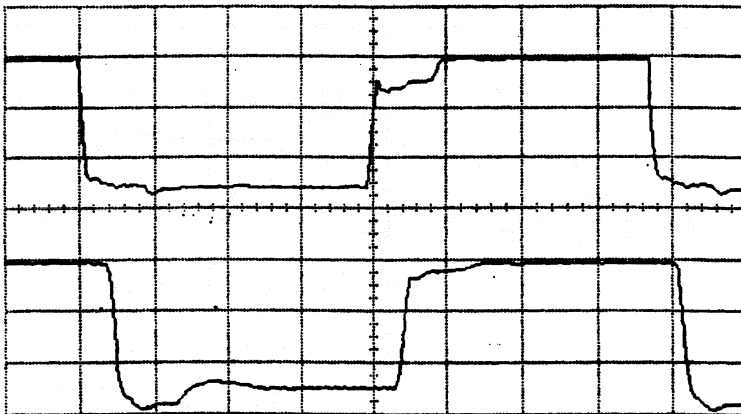
CASE 9 Figure 9a
DS2107A
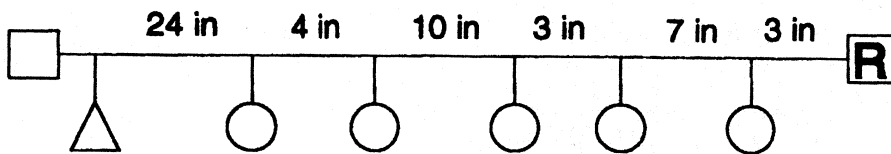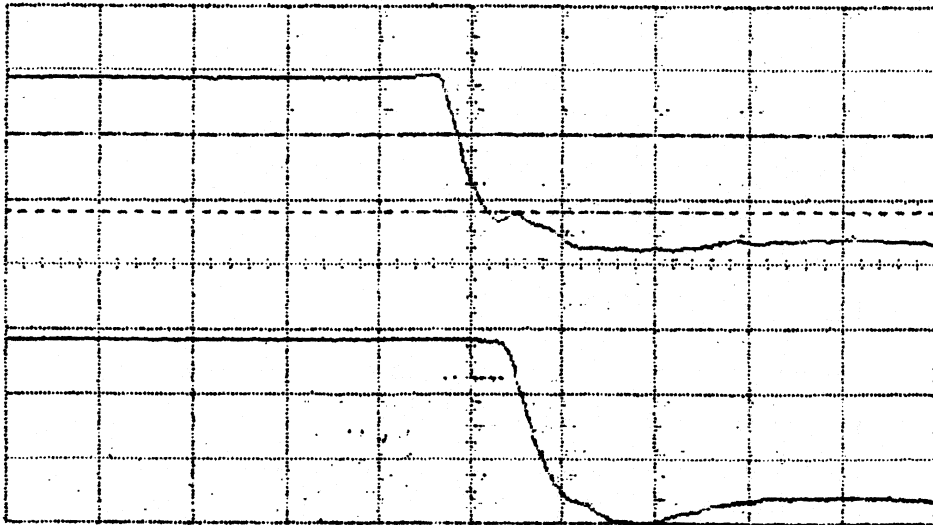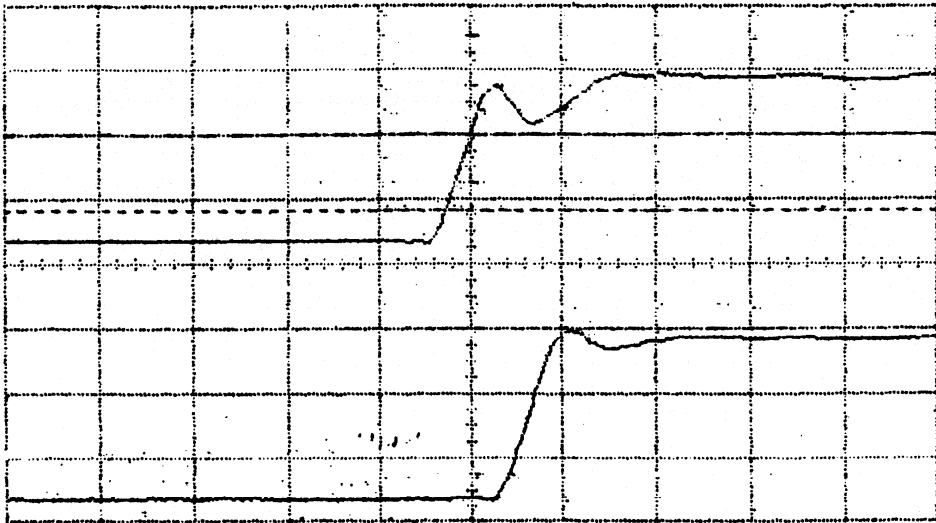
CASE 9 Figure 9b
DS2107A (other 8 lines "true")
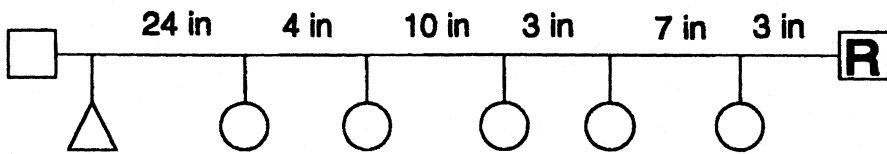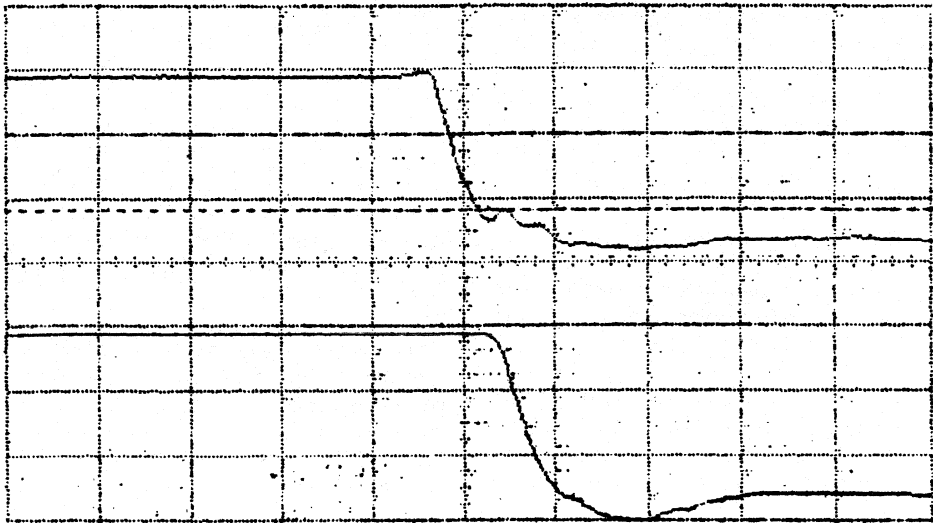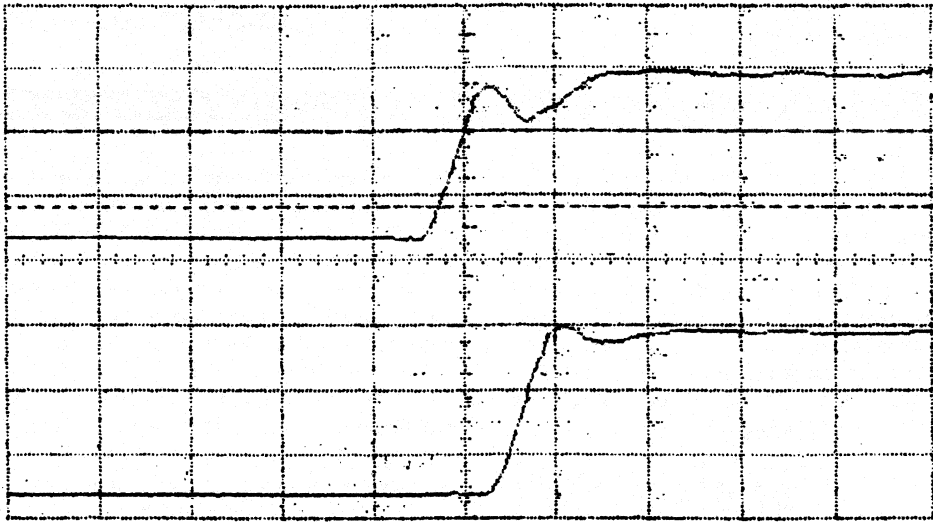
DALLAS
SEMICONDUCTOR

Application Note 71
DS2107A Thermal Considerations
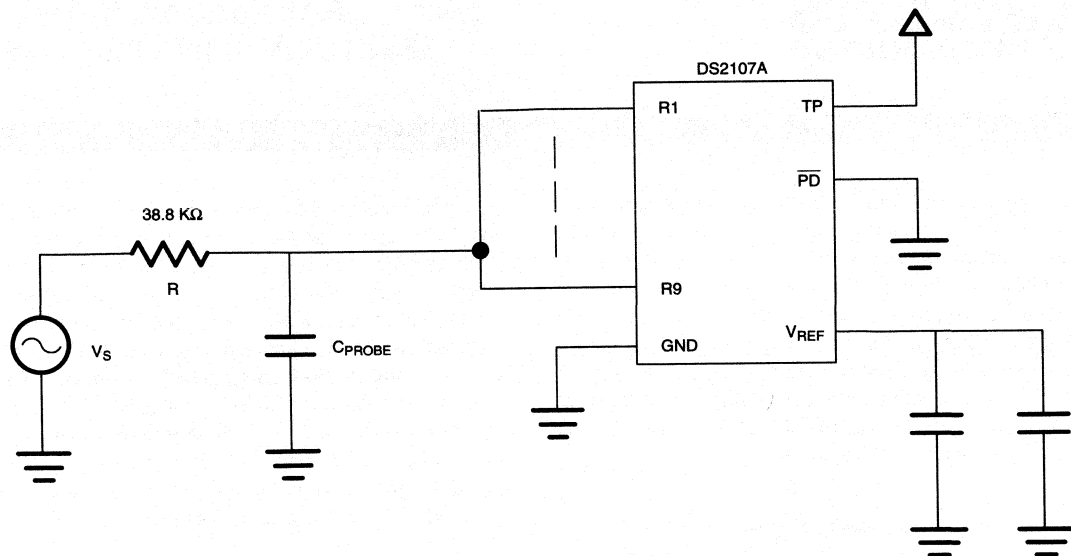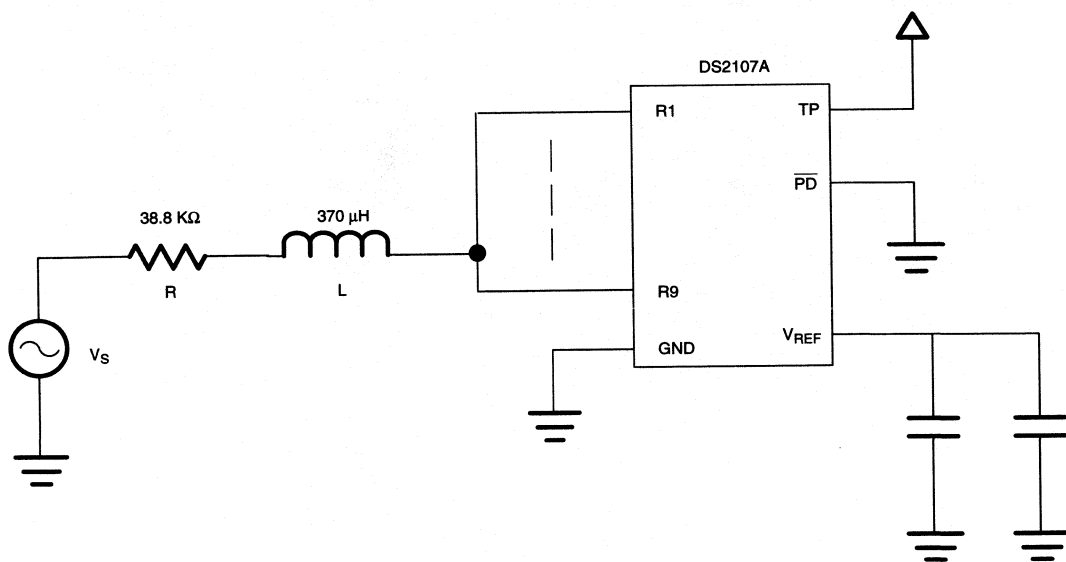
The DS2107A is used in SCSI systems to provide active termination for 9 signal lines. In the typical 8 bit–wide data configuration (A cable), two DS2107A's are required to fully terminate the bus (9 control lines + 8 data lines + 1 parity line). In the 16–bit wide data configuration (P cable), three DS2107A's are required to fully terminate the bus (9 control lines + 16 data lines + 2 parity lines). The two packages available are DS2107AS, 16–pin SOIC, and DS2107AE, 20–pin TSSOP (Thin Shrink Small Outline Package).

Semiconductor die temperature is a key characteristic in semiconductor operation and long–term reliability. Thermal resistance, theta (Θ), is used in determining the heat dissipation capability of a semiconductor package. A low Θ value corresponds to better heat conduction. Junction, case and ambient temperatures are important because these are the areas of power dissipation. Θ_{JA} and Θ_{JC} are the junction–to–ambient and junction–to–case thermal resistances respectively. The die temperature, T_{DIE}, can be calculated as follows:

$$T_{DIE} = T_{AMBIENT} + \Theta_{JA}P_{DISSIPATED}$$
or
$$T_{DIE} = T_{CASE} + \Theta_{JC}P_{DISSIPATED}$$

Where $P_{DISSIPATED}$ is the power dissipated by the semiconductor. An example calculation is shown in Appendix A.

Θ_{JA} and Θ_{JC} were measured on the DS2107AS and DS2107AE using the well known V_{BE} variation of a forward biased diode as a measuring element. This method is preferred for this type of measurement due to

the linear dependence of the forward biased diode voltage on temperature and the small amount of power dissipated in the diode to measure junction temperature (~2mW). The diode V_{BE} was measured over temperature to calibrate the die temperature readings. Next, various resistor combinations of the DS2107A were grounded and allowed to run for 10 minutes to insure stability of the die temperature for the given power level. Finally, the diode V_{BE} was measured and referenced to the calibration plot to determine the die temperature. The Θ_{JA} and Θ_{JC} were then calculated from the known ambient and case temperatures. Results of the testing are shown in Table 1. The values are the averages measured at 1 Watt.

Table 1

	Θ_{JA}	Θ_{JC}
DS2107AS	60°C/W	19°C/W
DS2107AE	93°C/W	19°C/W

Graph 1 shows the results of the measured Θ_{JA} and Θ_{JC} versus the power dissipated. Since the die temperature is more affected by the power dissipation as the power is increased, the most important part of the graph is to the right.

Use of heat sinks in board design improves the thermal characteristics of the DS2107A. The TSSOP package has a custom leadframe connecting the die directly to the heat sink ground pins and all pads are double bonded. This gives the DS2107AE the best thermal performance in the industry for the TSSOP package.

Θ_{JA} AND Θ_{JC} (°C/W) VS. POWER DISSIPATION (W) DS2107AS, DS2107AE

Graph 1
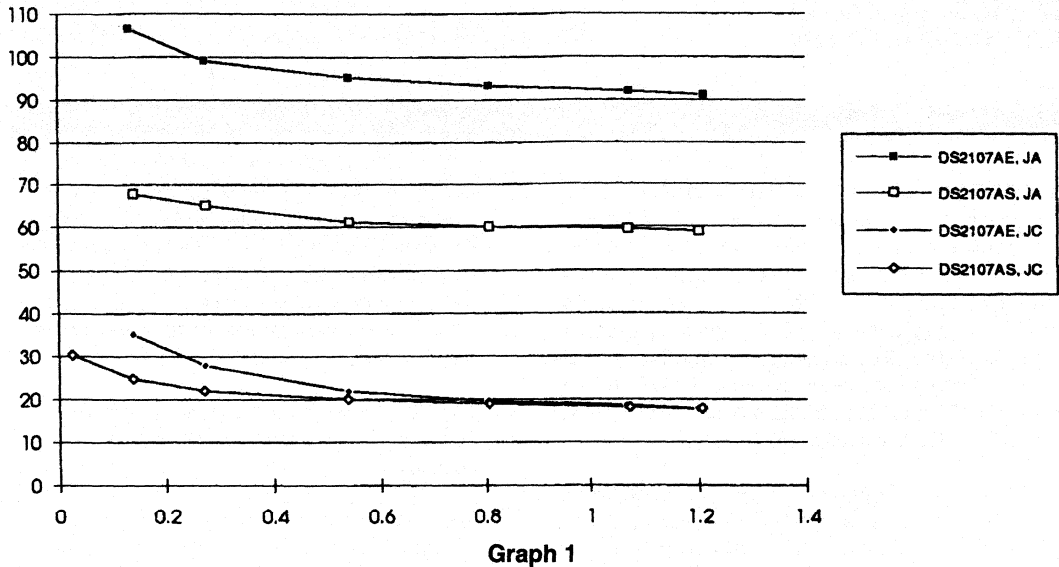
APPENDIX A

Typical or nominal power dissipation can be calculated by using the following five variables:

RTERM   Termination Resistance
TERMPWR   TERMination PoWeR Voltage
VREF   Voltage Regulator Output Voltage
VSL   Voltage on a SCSI signal line when in
                    the zero state
%DC   Percent duty cycle

The formula to calculate the power in Watts dissipated by the DS2107A is:

$$((VREF - VSL) / RTERM) \times 9 \times TERMPWR \times \%DC$$

For example, the following calculations represents the power dissipation under normal conditions.

Parameter	Nominal Value
RTERM	110Ω
TERMPWR	5.0V
VREF	2.85V
VSL	0.5V
%DC	30%

The $P_{DISSIPATED}$ under normal conditions would be:

$$((2.85 - 0.5) / 110) \times 9 \times 5.0 \times 30\% = 0.288 \text{ Watts}$$

DALLAS
SEMICONDUCTOR

Application Note 72
DS2107A Power Down Capacitance

The DS2107A is used in SCSI systems to provide active termination for 9 signal lines. In the typical 8 bit–wide data configuration (A cable), two DS2107A's are required to fully terminate the bus (9 control lines + 8 data lines + 1 parity line). In the 16–bit wide data configuration (P cable), three DS2107A's are required to fully terminate the bus (9 control lines + 16 data lines + 2 parity lines). The two packages available are DS2107AS, 16–pin SOIC, and DS2107AE, 20–pin TSSOP (Thin Shrink Small Outline Package).

The SCSI bus is defined to use termination at the two physical ends of the bus. If a device is connected in the middle of the bus, the total lumped capacitance of the device is limited on each signal line. Section 7.1.4 of the SCSI–3 Parallel Interface specification defines this value as 25 pF. Annex E defines the measurement of the pin capacitance:

"The objective of this procedure is to determine the lumped capacitance imposed on each signal conductor of the bus proper by an SCSI device connected thereto. The model for this procedure assumes the bus in ribbon cable form passing through an insulation–displacement SCSI connector, the mating part that is mounted on an SCSI device controller printed–wire board. The bus connector is removed from the device, along with every source of power.

One or more device connector circuit–common pins are connected together to form an effective circuit–common node. An R–F admittance bridge (or equivalent), operation at 1.0 MHz, is connected successively to each signal pin in the device connector, with reference to the circuit–common node.

The signal applied during the measurement shall be biased to 0.5 Vdc.and shall be 0.4V peak to peak in amplitude.

The characteristics shall be determined in terms of a parallel combination of a conductance and a capacitive susceptance. The corresponding capacitance thus determined is the maximum signal capacitance referred to in clause 7.1.4."

The DS2107A was measured using an HP4194A Impedance Bridge, and then verified using a lowpass RC network and an LC resonance circuit. The power down capacitance using the Impedance Bridge was measured to be 8.1 pF. The RC network shown in Figure 1 yielded 8.0 pF, and the LC resonance circuit of Figure 2 measured 8.6 pF. V_S of both Figure 1 and Figure 2 is comprised of Vdc = 0.5V, Vac = 0.4 p–p, and f = 1 MHz.

Figure 1

Figure 2

030794 2/2

DALLAS
SEMICONDUCTOR

Application Note 73
DS2107A Active Negation

The DS2107A is used in SCSI systems to provide active termination for 9 signal lines. In the typical 8 bit–wide data configuration (A cable), two DS2107A's are required to fully terminate the bus (9 control lines + 8 data lines + 1 parity line). In the 16–bit wide data configuration (P cable), three DS2107A's are required to fully terminate the bus (9 control lines + 16 data lines + 2 parity lines). The two packages available are DS2107AS, 16–pin SOIC, and DS2107AE, 20–pin TSSOP (Thin Shrink Small Outline Package).

WHAT IS ACTIVE NEGATION?
Active negation is a technique used by the newer SCSI–2 and SCSI–3 bus drivers on single–ended SCSI buses to insure clean monotonic transitions of the signal. Drivers employing active negation contain the ability to not only sink current but to source current as well. These drivers usually "yank" the SCSI signal to its negated state (i.e. high) by actively pulling the signal high with a driver capable of sourcing current. The SCSI–3 Parallel Interface specification states: *"Additional benefit may be achieved by using active–negation drivers on the DATA BUS and parity signals when operating in fast synchronous data transfer mode by reducing the skews between the first group of signals (ACK, REQ, ACKQ, REQQ) and the DATA BUS and parity signals."*

452
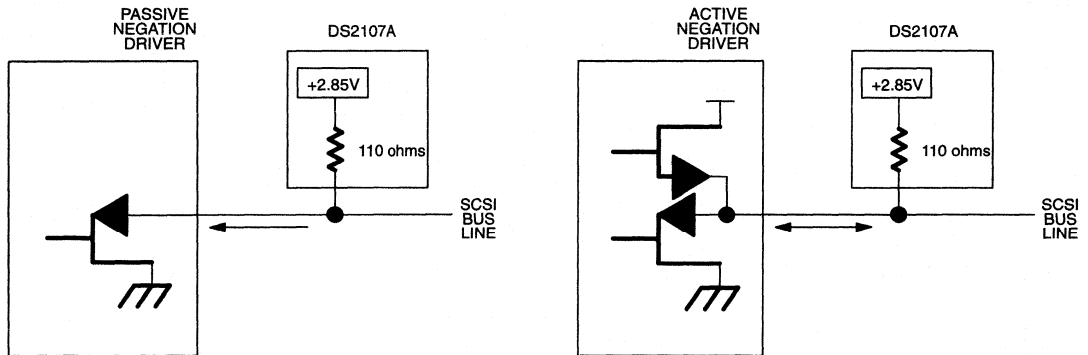
WHY MUST TERMINATORS BE ABLE TO HANDLE ACTIVE NEGATION?

When a driver actively negates an SCSI signal, the bus may be pulled higher than the termination voltage present in the active terminators. If this occurs, current will begin flowing into the regulated 2.85V termination voltage. If the active terminator cannot sink all of the current that is being sourced by the driver, then it will begin to raise above its nominal 2.85V termination voltage which

will result in current greater than the 24 mA being sourced on the other SCSI bus signals being shared by the active terminator (which violates the SCSI specifications). Since drivers can be placed at any point along the SCSI bus and terminators only exist at the physical end of the bus, all active terminators must be able to fully sink all of the current generated by the active–negation drivers.
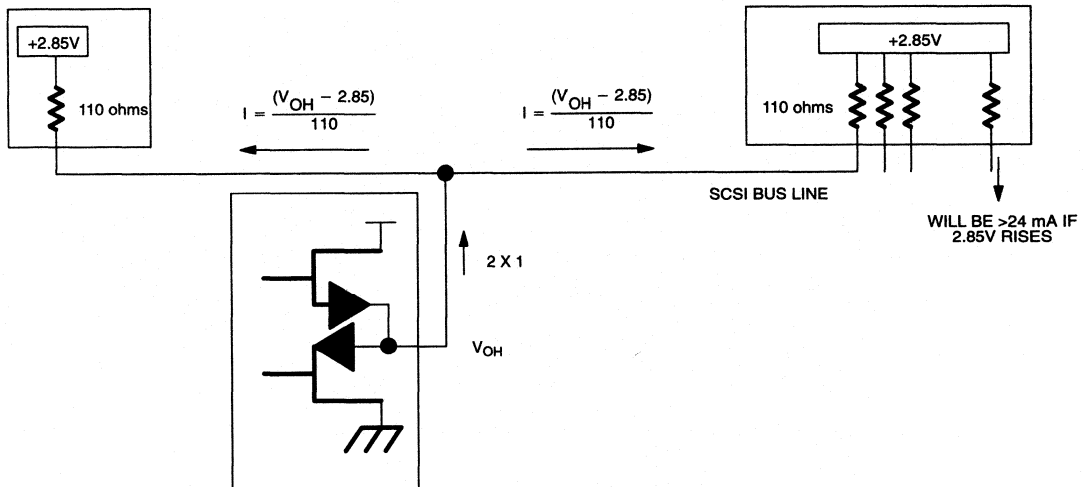
HOW MUCH CURRENT MUST AN ACTIVE TERMINATOR SINK?

The SCSI–3 Parallel Interface specifications allows up to 20 mA sourced current per driver line. On "by nine" termination schemes like the DS2107A, this means the terminator must be able to sink 180 mA (9 lines x 20 mA/line). The DS2107A can sink at least 200 mA, which is within this limit.

DIGITAL POTENTIOMETERS

DALLAS
SEMICONDUCTOR

INTRODUCTION

Dallas Semiconductor manufactures a variety of digital potentiometers. These potentiometers can be used in a variety of applications to replace the traditional mechanical potentiometer. One such application includes LCD contrast control.

For manufacturers of portable computers, workstations, small handheld computers, cellular phones, or video games, the array of flat panel liquid crystal displays (LCDs) grows ever larger. Commonly, all these applications use the mechanical potentiometer to control contrast or brightness.

This application note uses the DS1668/DS1669 Dallastats, as an example, to provide replacement of the standard mechanical potentiometer used in LCDs to control contrast. Other members of the Dallas Semiconductor digital potentiometer family can easily be substituted in place of the DS1668/DS1669 for this purpose.

The DS1668/DS1669 was chosen because it provides the ease of a push–button controlled interface to change potentiometer wiper position. The push–button interface is well suited to provide control of the flat panel LCD contrast and provides a similarity to the mechanical method for tuning traditional potentiometers. Most importantly the DS1668/DS1669 is a nonvolatile device which saves the position of the wiper in the absence of system power. Replacing the mechanical potentiometer with the digital potentiometer can provide added benefits in terms of device control, reliability and manufacturing of the end product.

In addition the DS1669 can be controlled via a CPU, allowing the designer to implement a software controlled potentiometer or even control the potentiometer from the keyboard or keypad of the portable system. This type flexibility is not available when using a mechanical potentiometer.

Because the DS1669 is an integrated circuit, and having no moving parts, it is inherently more reliable than the mechanical potentiometer; since mechanical systems can wear out; especially in high use environments.

For LCD contrast control, the DS1669 can be placed on the PC board away from the push–button on the panel display control section. The DS1669 is available in standard 8–pin DIP and 8–pin SOIC packages and is compatible with a variety of automated assembly techniques.

LCD DISPLAYS

Liquid Crystal Displays can be divided into at least two categories which include character display modules or the larger graphic display modules. Each, depending on power supply requirements, has an LCD driving voltage input which in most configurations can be varied to adjust contrast.

LCD contrast can vary according to driving voltage applied or the ambient temperature. Temperature typically can have an undesirable effect, making the LCD screen contrast extremely dim. A variable resistor is used to adjust the LCD driving voltage; where an increase can overcome the effects of temperature. More importantly, the potentiometer is used to accommodate the desires of the user in terms of screen brightness or screen contrast.

LCD CHARACTER DISPLAY MODULES

LCD Character Display Modules are small in size and support small handheld portable systems. The power supply requirements for such modules consist of one supply of at most 5 volts to power both the LCD and LCD driver module logic. An additional voltage is used to supply the drive requirements for the LCD. This is presented in Figure 1. The LCD drive voltage, V_O, is derived from potentiometer V_R. Typical values specified for V_R range from $10K\Omega$ to $20K\Omega$. These values are in line with the $10K\Omega$ version of the DS1669.

For LCD modules (LCM) having this type power supply configuration, the DS1669 readily fits. Figure 2 presents a DS1669 configuration that works well for the power supply and LCD voltage drive requirements found in Figure 1. The device is connected in single push–button configuration for controlling wiper position movement. A dual push–button configuration could also be used. Both single and dual push–button control

configuration and operation is presented in the "DS1669 Device Operation" section of this application note.

The wiper terminal, R_W, of the DS1669 would connect directly to the LCM drive voltage ,V_O, of the LCD character display module. No other components are necessary for using the DS1669 with LCMs having power supply configurations as shown in Figure 1.

LCD CHARACTER DISPLAY POWER SUPPLY CONFIGURATION Figure 1
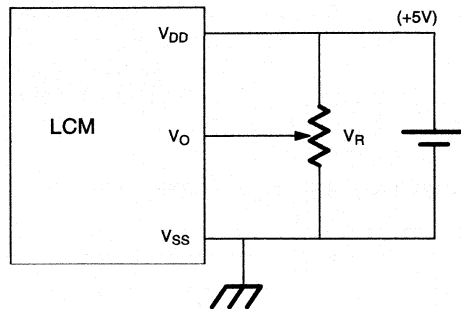
LCD CHARACTER DISPLAY USING DS1669 Figure 2
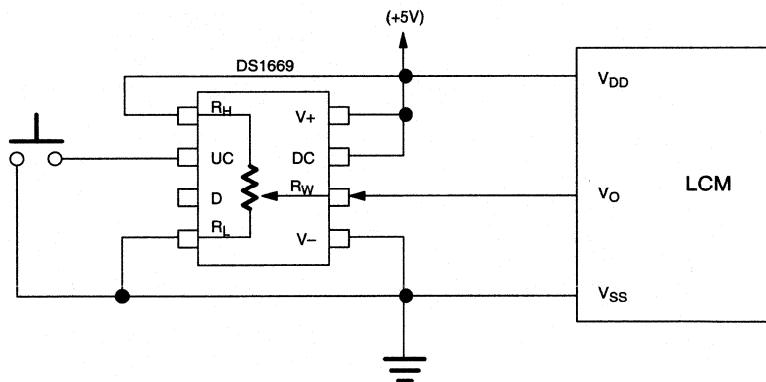
LCD GRAPHICS DISPLAY MODULES

LCD graphics display modules are much larger than the character display modules and use a different power supply configuration. As shown in Figure 3, the power supply requirements consist of a 5 volt supply for logic requirements and a V_{LCD} supply input to power the LCD. The LCD driving voltage, V_O, which controls contrast is derived from a combination of the 5 volt logic supply voltage and the V_{LCD} supply via the potentiometer R. As before, the potentiometer value range is between 10KΩ and 20KΩ. The combination voltage, however, allows V_O to range from 0 to (V_{LCD} + 5V) volts. The voltage V_{LCD} is dependent on LCD used.

The voltage range of V_O precludes the direct use of the DS1669, as well as, other potentiometers presently manufactured by Dallas Semiconductor. To circumvent the wiper current limits and high voltage requirements in this application, the digital potentiometer can be used in conjunction with an Op Amp as shown in Figure 4. The purpose of the Op Amp is to provide a means of supplying the LCD driving voltage, V_O, which is outside the specification for operation using the digital potentiometer. Additionally, the Op Amp limits the amount of current that can flow through the wiper while providing complete coverage of the LCD driving voltage range.

The 5 volt supply which is used to power the LCD logic circuits is used to power the DS1669 (see Figure 4). The potentiometer acts as an attenuator of the 5 volt input signal to the non–inverting Op Amp terminal.

LCD GRAPHICS MODULE POWER SUPPLY CONFIGURATION Figure 3
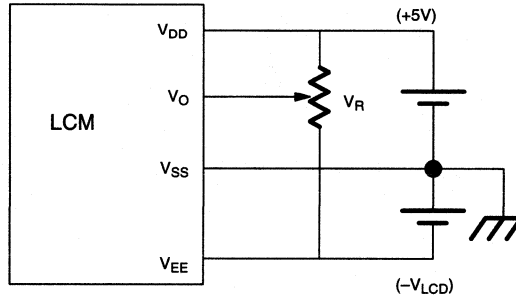
OP AMP CONFIGURATION FOR DRIVING THE LCD GRAPHICS DISPLAY MODULE Figure 4
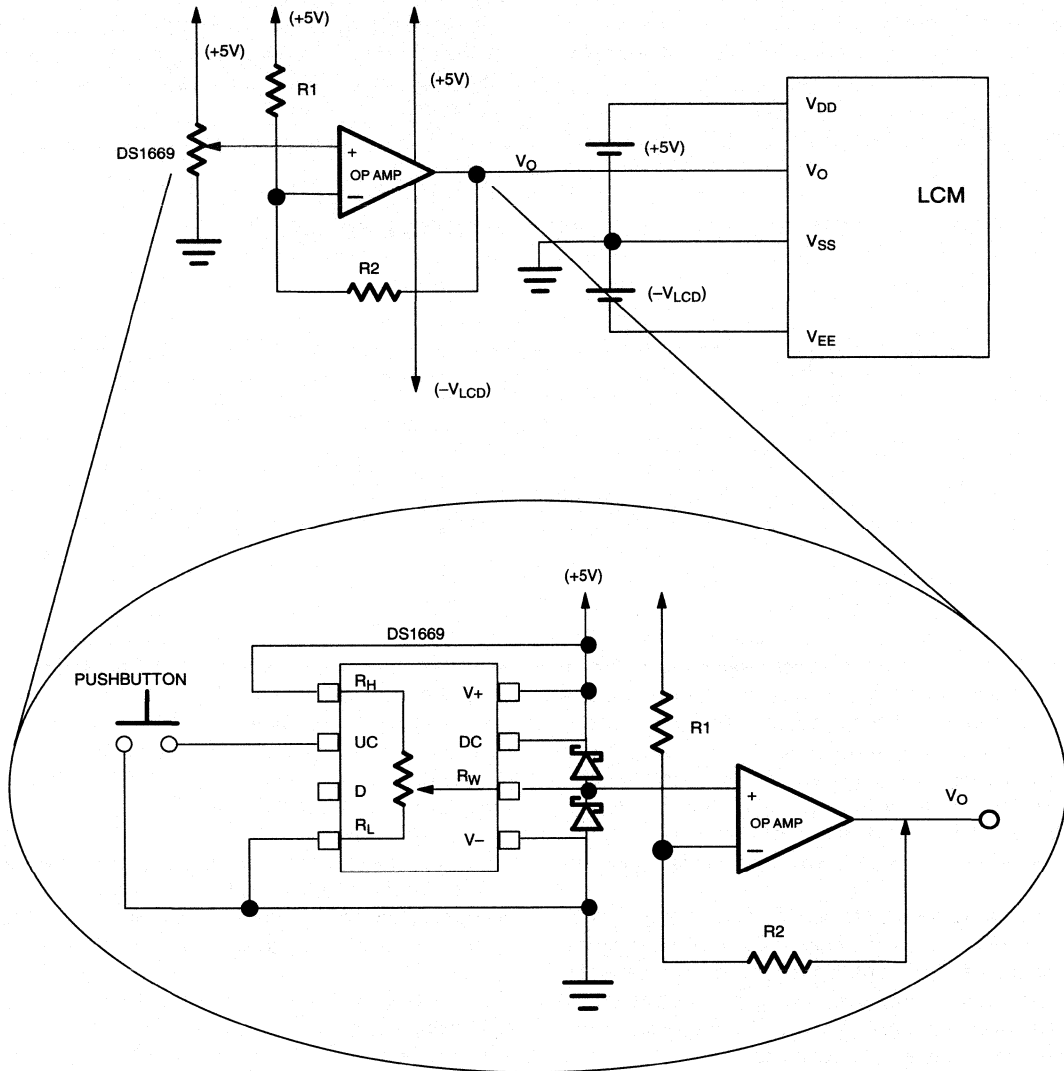
The output of the Op Amp drives the LCD contrast control voltage, V_O, and is given by equation (1):

eq. (1) $V_O = 5n/63*(1+R_2/R_1) - 5(R_2/R_1)$
  $(n = 0..63)$

where n is the position of the wiper. Resistor values R_1 and R_2 are chosen using equation (2):

eq. (2) $R_2/R_1 = V_{LCD}/5$

The choices of R_1 and R_2 allow the Op Amp to drive the complete range of V_O as shown in Figure 4. The wiper terminal R_W would connect directly to the non–inverting terminal of the Op Amp as shown in Figure 4. It is also suggested that the Schottky diode D1 (1N5818 or equivalent) be used to provide additional protection against power surges during power–up or power–down sequences.

DS1669 DEVICE OPERATION

This section is presented to provide the user with a glance of how the DS1669 push–button interfaces operate. A more complete explanation of the device operation can be found in the data sheet.

The DS1669 can be configured to operate from a single contact closure, dual contact closure inputs or driven using a digital source input. Figures 5 and 6 illustrate both configurations, respectively. Contact closure is defined as the transition from a high level to a low level on the up contact (UC), down contact (DC) or, digital source (D) input. These inputs are inactive when in the high state.

DS1669 SINGLE PUSHBUTTON CONFIGURATION (TYPICAL EXAMPLE) Figure 5
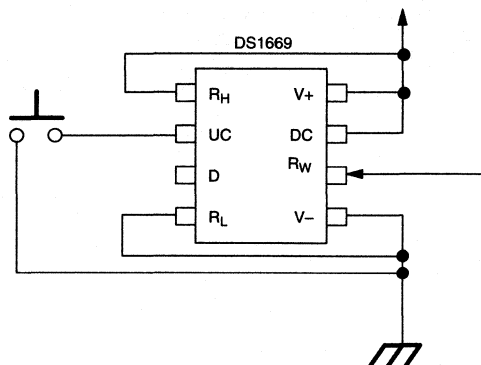
The DS1669 interprets input pulse widths as the means of controlling wiper movement. A single pulse input over the UC, DC, or D input terminals will cause the wiper position to move 1/64th of the total resistance. A transition from a high to low on these inputs is considered the beginning of pulse activity or contact closure. A single pulse is defined as being greater than 1 ms but lasting no longer than a second. This is shown is Figure 8a.

Repetitive pulsed inputs can be used to step through each resistive position of the device in a relatively fast manner (see Figure 8b). The requirement for repetitive pulsed inputs is that pulses must be separated by a minimum time of 1 ms. If not, the DS1669 will interpret repetitive pulses as a single pulse.

Pulse inputs lasting longer than 1 second will cause the wiper to move one position every 100 ms following the initial 1 second hold time. The total time to transcend the entire potentiometer using a continuous input pulse is given by the formula below:

$$1 \text{ (second)} + 63 \text{ X } 100 \text{ ms} = 7.3 \text{ (seconds)}$$

Single contact closure operation allows the user to control wiper movement in either direction from a single push–button input. Figure 5 presents a typical single push–button configuration. The UC input is used to increment and decrement wiper position for single push–button mode of operation. The DC input provides no functionality in this mode but must be connected to the positive supply voltage (V_{CC}). The digital source input (D) can be allowed to float.

On device power–up, the configuration shown in Figure 5 must exist in order to enter the single contact closure mode of operation; especially and specifically, the (DC) input's connection to the positive supply voltage (V_{CC}).

The direction of wiper movement, in single push–button operation, is determined by prior activity; with the direction of wiper movement being opposite to that of the previous activity.

Changing the direction of wiper movement in single push–button mode is accomplish by a period of inactivity on the UC input of a (minimum) 1 second or greater. Also, in single push–button mode, as the wiper reaches the end of the potentiometer range its direction of movement changes. This will occur regardless if the input is a continuous pulse, a sequence of repetitive pulses or a single pulse.

In dual push–button mode, each direction is controlled by the up contact (UC) and down contact (DC) inputs, respectively. No wait states are required to change wiper direction in dual push–button mode. In dual push–button mode, as the wiper position reaches the end of the potentiometer, the direction of wiper movement will not change. Wiper position will remain at the potentiometers' end until an opposite direction input is given.

All contact closure control inputs, UC, DC, and D are internally pulled–up by a 100KΩ resistance. The UC and DC inputs are internally debounced and require no external components for input signal conditioning.

DS1669 DUAL PUSHBUTTON CONFIGURATION (TYPICAL EXAMPLE) Figure 6
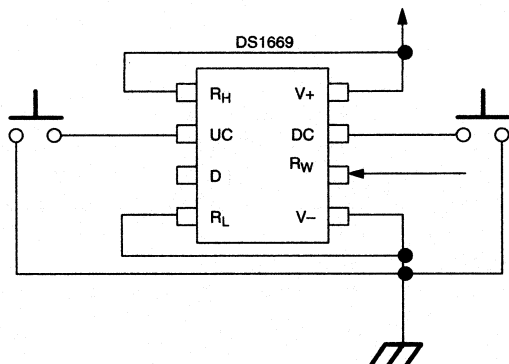
NONVOLATILE WIPER SETTINGS

The DS1669 maintains the position of the wiper in the absence of power. This feature is provided through the use EEPROM type memory cell arrays. During normal operation the position of the wiper is determined by the input multiplexer. Periodically, the multiplexer will update the EEPROM memory cells. The manner in which an update occurs has been optimized for reliability, durability, and performance. Additionally, the update operation is totally transparent to the user. When power is applied to the Dallastat, the wiper setting will be the last recorded in the EEPROM memory cells. If the Dallastat setting is changed after power is applied, the new value will be stored after a delay of 2 seconds. The initial storage of a new value after power–up occurs when the first change is made, regardless of when this change is made.

After the initial change on power–up, subsequent changes in the Dallastat EEPROM memory cells will occur only if the wiper position of the part is moved greater than 12.5% of the total resistance range. Any wiper movement after initial power–up which is less than 12.5% will not be recorded in the EEPROM memory cells. Since the Dallastat contains a 64–to–1 multiplexer, a change of greater than 12.5% corresponds to a change of the fourth LSB.

Changes or storage to the EEPROM memory cells must allow for a 2 second delay to guarantee that updates will occur. The EEPROM memory cells are specified to accept greater that 80,000 writes before a wear–out condition. If the EEPROM memory cells do reach a wear–out condition, the Dallastat will still function properly while power is applied. However, on power–up the device's wiper position will be that of the position last recorded before memory cell wear out.

TIMING DIAGRAMS Figure 7

(a) Single Pulse Inputs

t_{CPW}

V_{IH}

V_{IL}
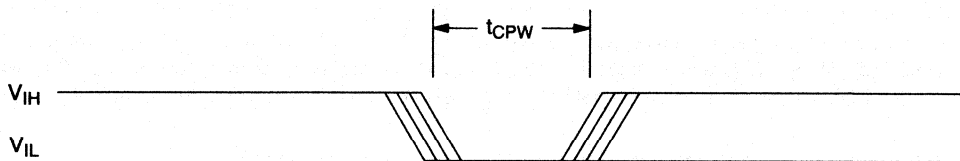
(b) Repetitive Pulse Inputs

t_{CPW}

t_{HPW}

V_{IH}

V_{IL}
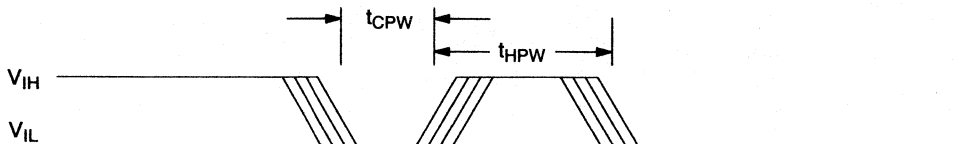
(c) Continuous Pulse Inputs

t_{CCP}

V_{IH}

V_{IL}
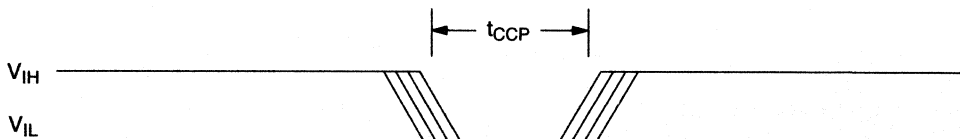
DIGITAL POTENTIOMETER APPLICATION EXAMPLES

As stated in the introductory paragraph, Dallas Semiconductor manufactures a variety of digital potentiometers. These include the DS1267 Dual Digital Potentiometer, the DS1666 Audio Taper, the DS1667 Dual Digital Potentiometer with Op Amps, and the DS1668/DS1669 Dallastats Electronic Rheostats.

Because these potentiometers are digitally controlled and programmable, they lend themselves to applications requiring automated control methods. Applications that traditionally used manual means of tuning device parameters can now take advantage of digitally controlled potentiometers. Applications using fixed gain attenuators, variable gain amplifiers or differential amplifiers can be controlled with more accuracy, more flexibility and more speed if necessary. There are also the benefits to automated manufacturing processes which demand standard packaged integrated circuits for ease of assembly.

Currently, Dallas Semiconductor potentiometers are used in applications ranging from LCD contrast and brightness control, to volume and tone control, and ser-

vo–motor control. These devices are highly used in multi–media applications for the processing of sound including volume and tone. Additionally, these devices are moving into industrial control applications.

Devices such as the DS1267 and DS1667 which can be daisy chained and controlled via one microcontroller or processor provide the user with the ability to control multiple system parameters using a single 3–wire serial data bus. This type of control is unavailable with standard mechanical potentiometers. In addition to the two independent potentiometers available on the DS1667, the device also features two independent Op Amps which can be used in conjunction with the potentiometers or separately.

The DS1666 is a non–linear taper which approximates a logarithmic resistance curve. The device is also control via a 3–wire serial interface. The device works well in audio type applications requiring the log taper feature.

The DS1668 and DS1669 Dallastats Electronic Rheostats are Dallas Semiconductor's newest members of the Digital Potentiometer Family. They are also the only nonvolatile potentiometers currently offered.   The

DS1668 Dallastat is available in a custom 6–pin package which has its own integrated push–button for control of the wiper position. The DS1669 is the integrated circuit version of the DS1668 and is available in an 8–pin DIP and SOIC. The DS1669 allows the user control of the wiper position using an external (user–supplied push–button) contact closure input. The device can be controlled from either a single or dual push–button configuration. In addition to push–button control the DS1668 and DS1669 can be controlled from a digital source such as a CPU. The push–button and digital source control inputs are as near a mechanical method of device control without having to be a mechanical device. This interface has proved successful as a simple method for LCD contrast and volume control in certain applications.

Future versions of Dallas Semiconductors potentiometers will include both the 3–wire serial and push–button interface as well as the nonvolatile wiper feature. Both interfaces will provide the user with a means of easily manipulating the wiper position and knowing, if desired, the exact wiper position.

The following application examples can be used with any of the Dallas digital potentiometers. Three application configurations are provided: Fixed Gain Variable Attenuator, Variable Gain Amplifier, and Programmable Differential Amplifier. The configurations represent the most used methods for device control using Dallas Semiconductor digital potentiometers. And these configurations can be applied to a variety of applications including LCD contrast, volume control, or servo–motor control.

FIXED GAIN ATTENUATOR AMPLIFIER Figure 8

$R_1 >> RW$
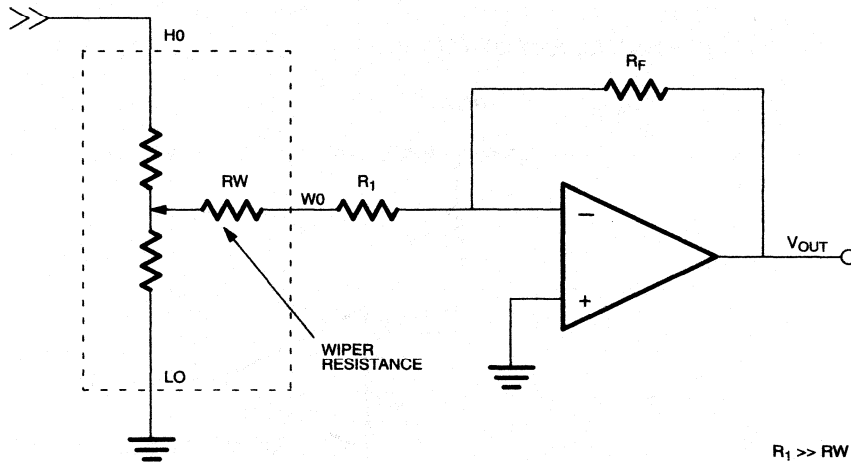
INVERTING VARIABLE GAIN AMPLIFIER Figure 9
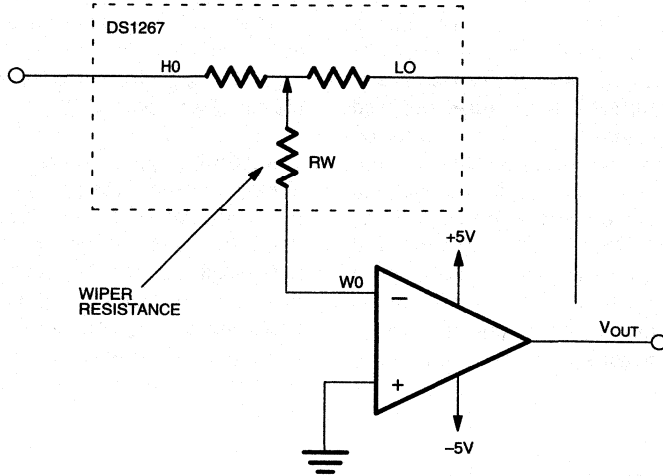
PROGRAMMABLE DIFFERENTIAL AMPLIFIER Figure 10
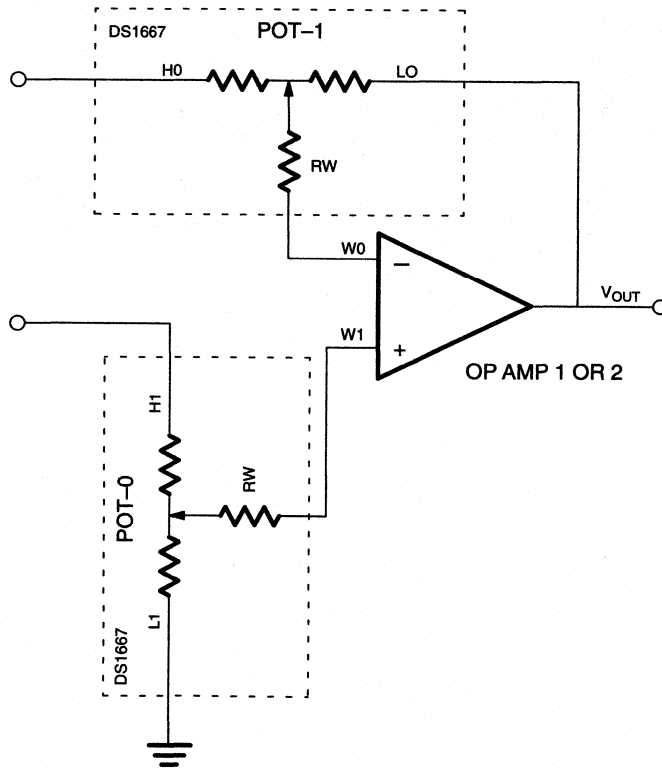
TONE CONTROL APPLICATION Figure 11

DALLAS
SEMICONDUCTOR
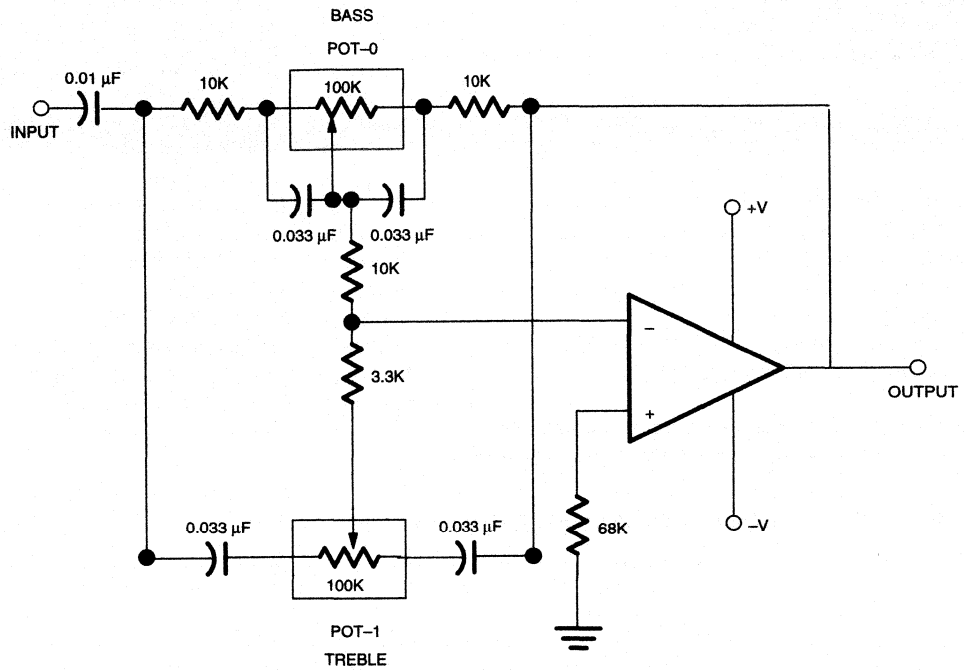
Application Note 87
DS1867 Power Supply Conditioning
for EEPROM Wiper Storage

The DS1867 is the nonvolatile version of the popular DS1267 Dual Digital Potentiometer. The device provides a drop–in replacement for the DS1267 in applications requiring nonvolatile wiper storage during power–off conditions. Design of the DS1867 uses EEPROM memory array cells for storage of the wipers of the device when power is not applied.

Storage of the wipers is accomplished during a predefined power–down sequence for the supply voltage, V_{CC}. Power–down timing is provided in Figure 1 for this sequence. As shown, the required supply voltage must ramp downward. The DS1867 will initiate wiper storage, as the supply voltage crosses 4.5 volts. Complete and correct storage of the wiper will occur if a time, t_{PU}, of 4 ms is maintained for the downward ramp between the 4.5 volt mark and 3 volts or if a 2.5 ms minimum time, t_{PU1}, from 3.9 volts to 3.0 volts is achieved.

POWER DOWN TIMING REQUIREMENTS Figure 1
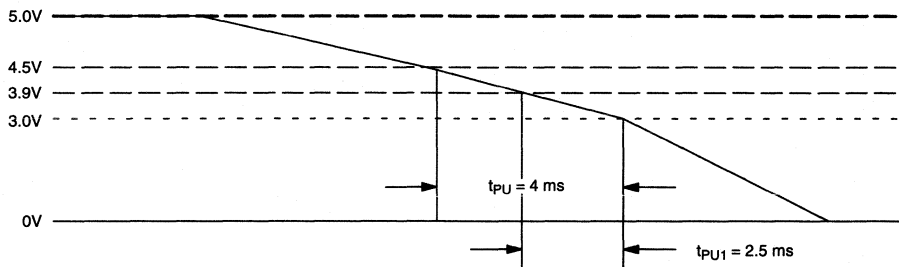
In many cases, the system supply will meet or exceed the power–down requirements for the DS1867 to perform wiper storage. However, if not, then implementing a power supply that will meet the power–down requirements for storage of the wipers can be achieved by using the circuit of Figure 2. In this circuit, two external components are required. These include a 1 μF capacitor and a Schottky diode (1N5818,1N5819, or equivalent). The 1 μF capacitor is used to provide enough power to the DS1867 during power–down to insure proper wiper storage. The Schottky diode is used to isolate the storage capacitor from the remainder of the system insuring all capacitor charge is directed to the DS1867.

Because the Schottky diode has a 0.2 volt drop, care should be taken to insure the voltage (V_{CC}) supplied to the DS1867 is at least 4.5 volts. This requires system power, V_{SYS}, to be 4.7 volts or higher to guarantee proper operation of the DS1867. The substrate bias voltage, V_B of the DS1867 should be connected to the lowest required voltage (not to exceed –5V) or ground.

Complete specifications on operation of the DS1867 Dual Digital Nonvolatile Potentiometer can be found in its data sheet.

POWER SUPPLY CIRCUIT CONFIGURATION Figure 2
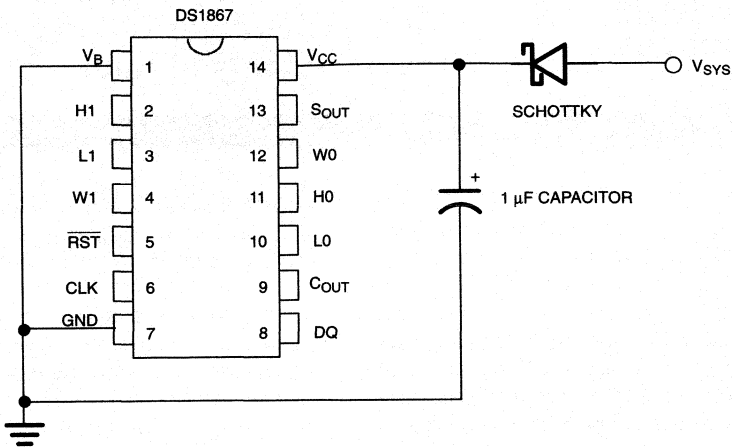
DALLAS
SEMICONDUCTOR

PURPOSE

The purpose of this document is to provide a examination of the DS1802 Dual Digital Potentiometer's audio characteristics. Test circuits were developed with the goal of measuring absolute error between tap points, inter–channel matching or tracking between the two potentiometer wipers, total harmonic distortion (THD), intermodulation distortion (IMD), and cross–talk. Additionally, data is provided for mute levels for each potentiometer, as well as, active and standby current as a function of temperature and voltage. For this report, tests are restricted to the range of frequencies 20 Hz to 20 KHz.

TEST CIRCUIT CONFIGURATION

To evaluate the DS1802's audio performance, the circuit of Figure 1 was constructed. As shown, this circuit

consist of two NE5532 Op Amps connected in a non–inverting unity gain configuration. Inputs to these buffer stages were taken from each wiper of the DS1802.

Several steps were taken to minimize the effects of extraneous noise on measurement data. Shielded cabling was used on all inputs and outputs from (or to) the DS1802 and NE5532 Op Amps. All ground signals were tied to a common point and power supplies were capacitively coupled to ground using 10 μF capacitors. Additionally, both the DS1802 and NE5532 Op Amps used separate power supplies.

Once assembled, this circuit was mounted inside a metal testbox to provide additional protection against coupling of random noise from external sources such as computer monitors, lights, etc. BNC connectors were used for inputs and outputs to test equipment.

TEST CONFIGURATION CIRCUIT Figure 1
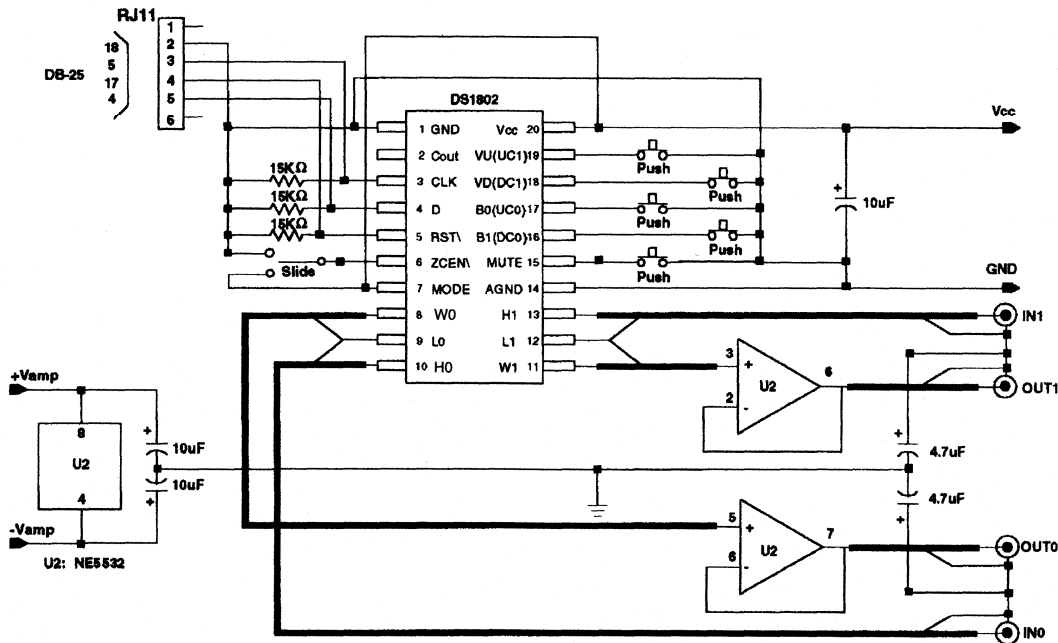
TEST MEASUREMENT EQUIPMENT

Audio measurements were performed using the Audio Precision–System One A version audio analyzer under PC–control. This test equipment provided the means of measuring THD+N, IMD, crosstalk, absolute error, and inter–channel matching.

TEST MEASUREMENTS PERFORMED

As stated in the Scope of this document, device tests were completed on absolute error between tap points, inter–channel matching, THD+N, IMD, cross–talk, mute–levels, and device current characteristics as a function of temperature. These tests and data are presented in the following pages of this section of the report.

Absolute Error and Summary

Absolute error is defined as the difference between the expected voltage output and the measured voltage output at a given wiper position. The DS1802 is specified to provide an absolute tolerance of ±0.5 dB for any given wiper position. For example, wiper position 48 would be expected to provide –48 dB ± 0.5 dB of signal attenuation difference relative to wiper position 0 (or the input signal if connected to the high–end terminal of the potentiometer).

Graphical data is presented in Figures 2 and 3 for a sample set consisting of five units of the DS1802. Figure 2 presents actual data for the five devices tested. Figure 3 further summarizes this data by presentation of the calculated average. The absolute error data taken in these tests indicate the DS1802 performs well within specified data sheet limits of ±0.5 dB.

DS1802 ABSOLUTE ERROR Figure 2
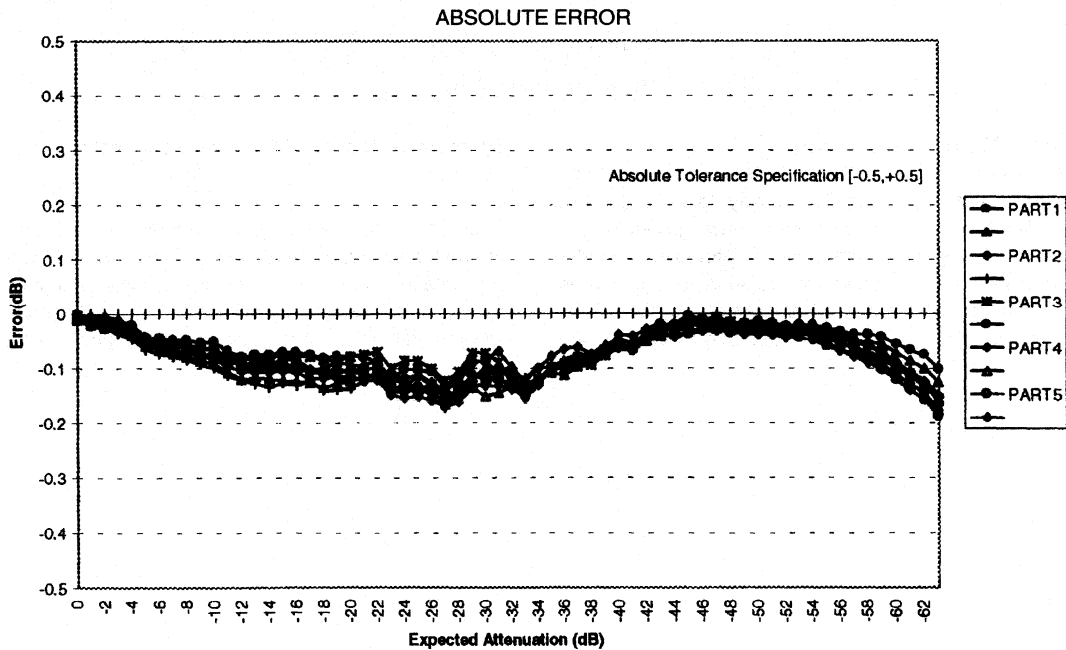
ABSOLUTE ERROR

## DS1802 AVERAGE ABSOLUTE ERROR Figure 3

AVERAGE ABSOLUTE ERROR
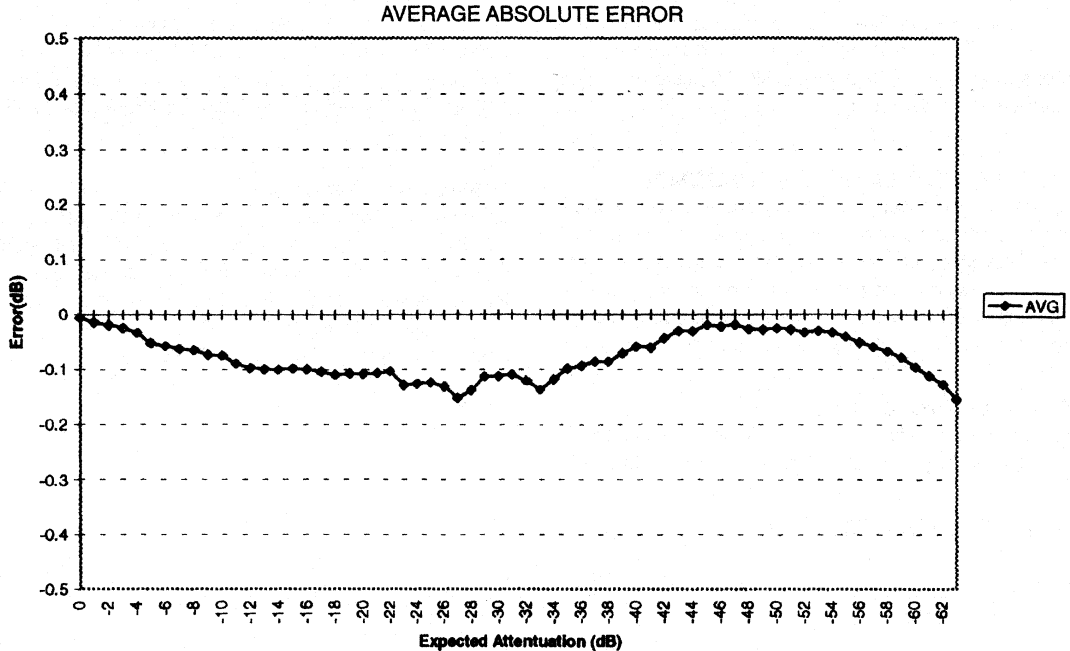
Inter–Channel Matching

The DS1802 has two potentiometers per device. Inter–channel matching provides a way to measure how closely the wiper positions of each potentiometer match. Inter–channel matching is defined as the difference between the same wiper positions of potentiometer 0 and potentiometer 1. For example, wiper position 15 of potentiometer 0 and potentiometer 1 should differ by no more than ±0.25 dB by specification of the DS1802.

Figure 4 provides actual data taken on the units, while Figure 5 shows a summary of the average data. Average data for inter–channel matching indicates that this device characteristic falls within a ±0.05 dB window; which is well within the data sheet specification of ±0.25 dB.

INTER–CHANNEL MATCHING Figure 4

INTERCHANNEL MATCHINGS
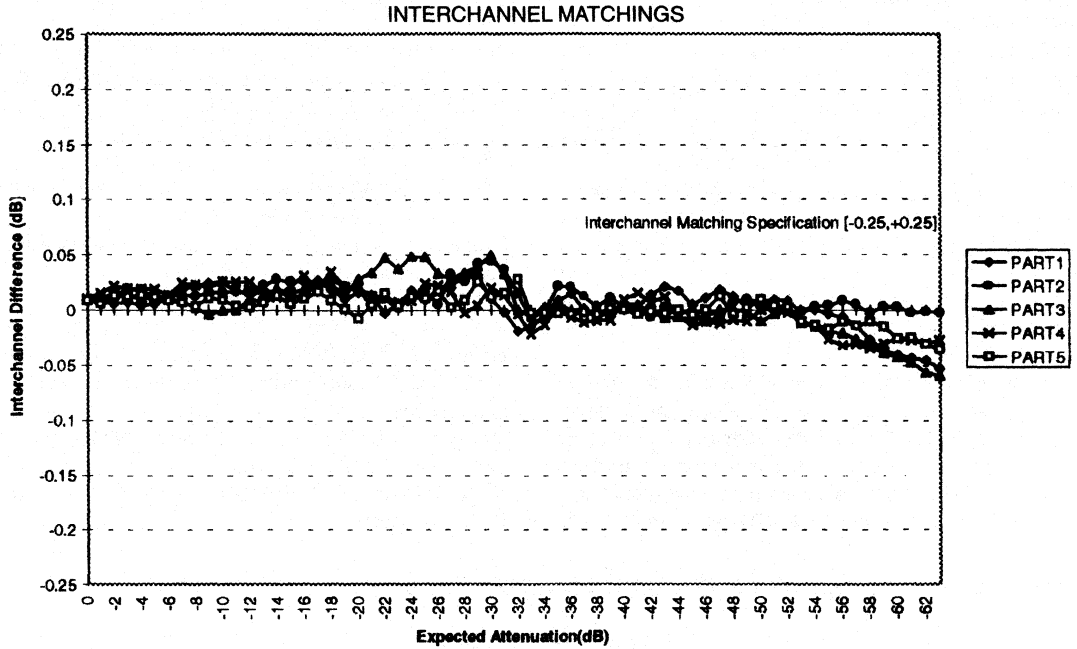
INTER–CHANNEL MATCHING SUMMARY Figure 5

AVERAGE INTERCHANNEL MATCHING
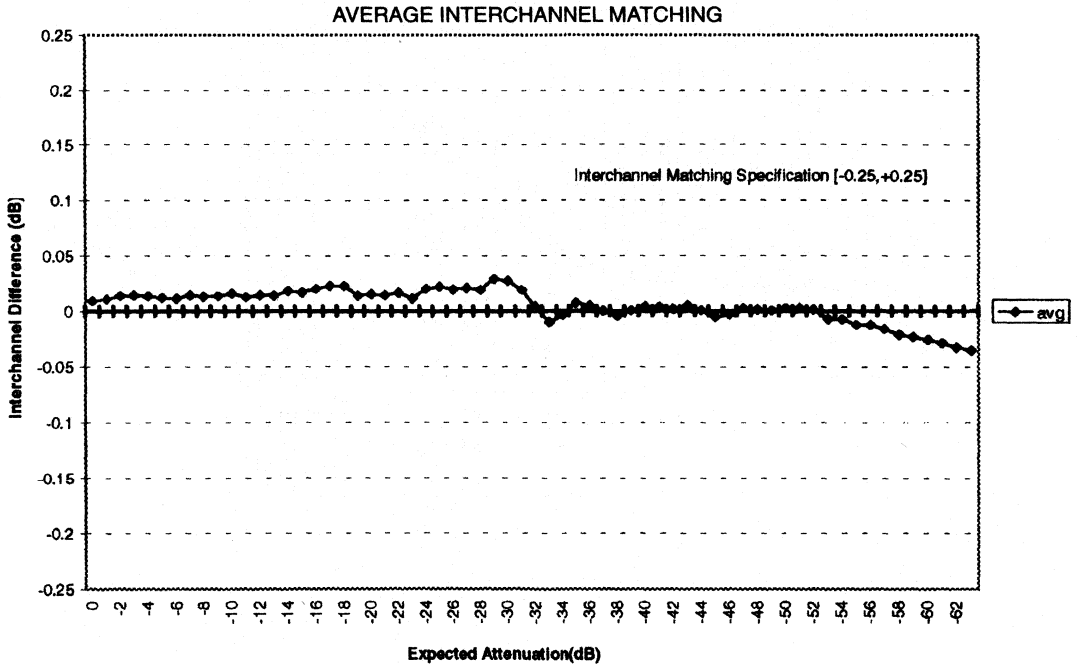
Total Harmonic Distortion + Noise (THD+N) vs. Frequency

Total Harmonic Distortion + Noise as a function of frequency is presented in Figures 6, 7, and 8. Figure 6 presents data for a 80 KHz bandlimited signal having no additional filtering. The data presented in Figure 7 is bandlimited to 30 KHz and includes data filtered using the A–weighting filter. Figure 8 provides data for an input signal bandlimited to 22 KHz and corresponding data when filtered using the A –weighting filter. Device and system configuration for this data is taken with the supply voltage of the DS1802 equal to 5 volts, a 1Vrms input signal, and the wipers of potentiometer 0 and potentiometer 1 set to position 6 (6 dB of attenuation).

By setting the wiper position of the potentiometer at some level other than position–0 allows the potential user of the DS1802 to view distortion of the part as the input signal covers some portion of the resistor array. A position setting of zero allows, primarily, the examination of the input signal source. The position–6 setting used in these tests provides a worst case selection for monitoring %THD+N.
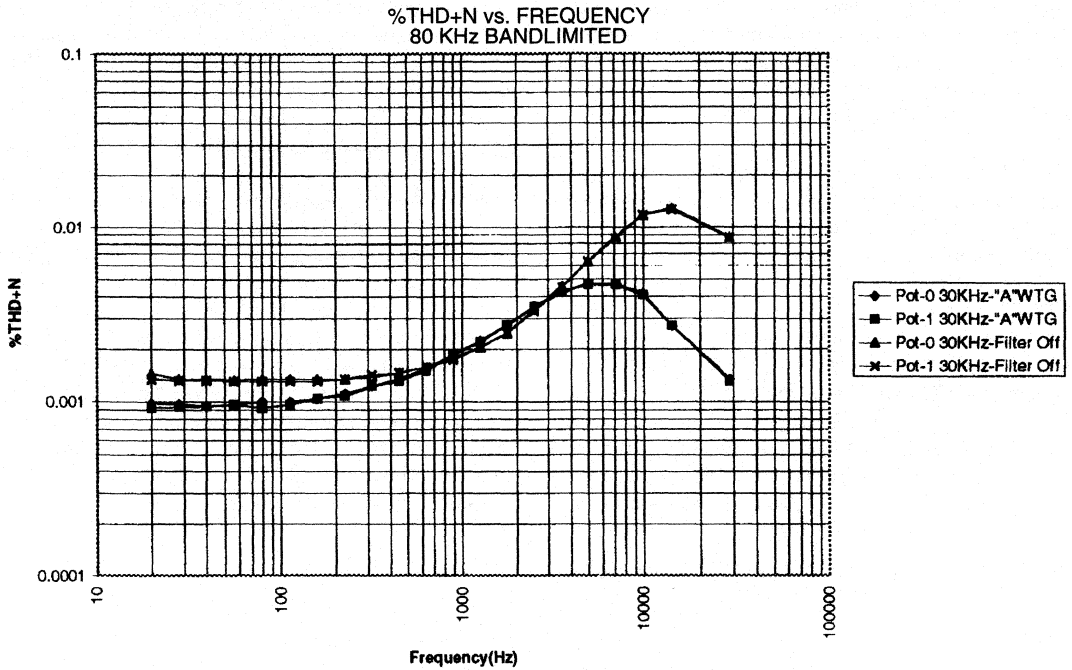
The DS1802 is specified for a %THD+N of 0.002% at 1 KHz. This specification is derived using the 80 KHz bandlimited input signal at 1Vrms, and a position–6 potentiometer setting. Across the 20 Hz to 20 KHz frequency range %THD+ N is less than 0.02% maximum with the 80 KHz bandlimited signal. Additionally, %THD+N across the 20 Hz to 20 KHz frequency range is significantly improved using bandlimited signals in conjunction with the A–weighting filter.

The A–weighting filter is a specific noise weighting filter(ANSI S1.4, IEC Recommendation 179) used to produce noise measurements which correspond well with human observations over the audio frequency range.[1] More detailed information concerning this filter can be found in the ANSI specification stated above or in the Audio Measurement Handbook published by Audio Precision, Incorporated.
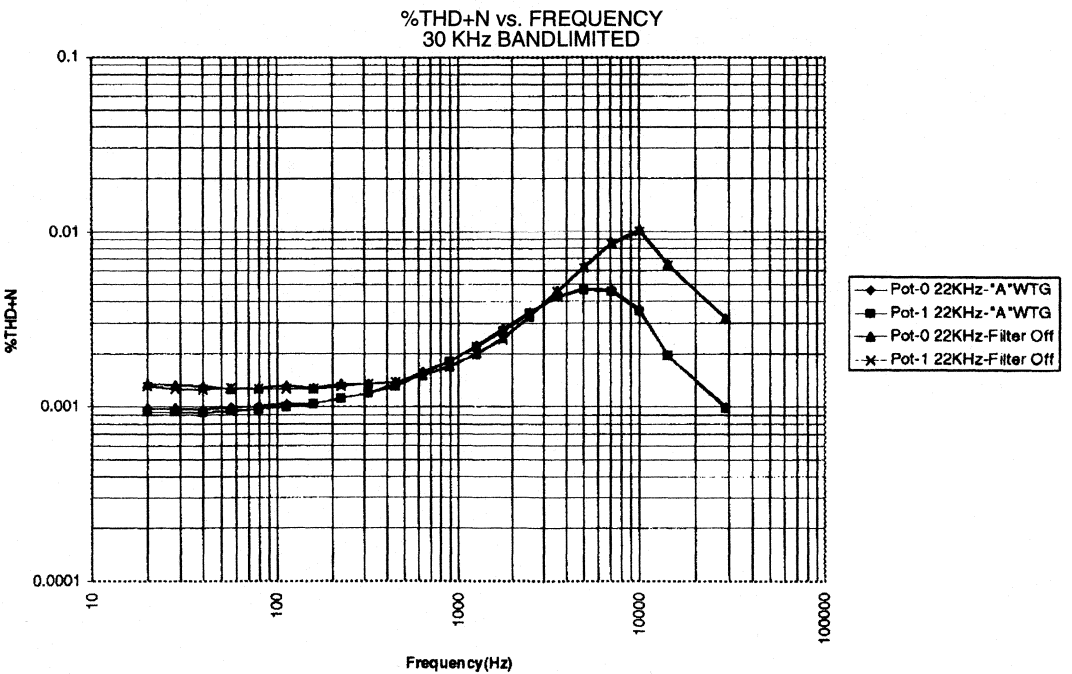
%THD+N vs. FREQUENCY 80 KHz BANDLIMITED Figure 6

%THD+N vs FREQUENCY
80KHz Bandlimited
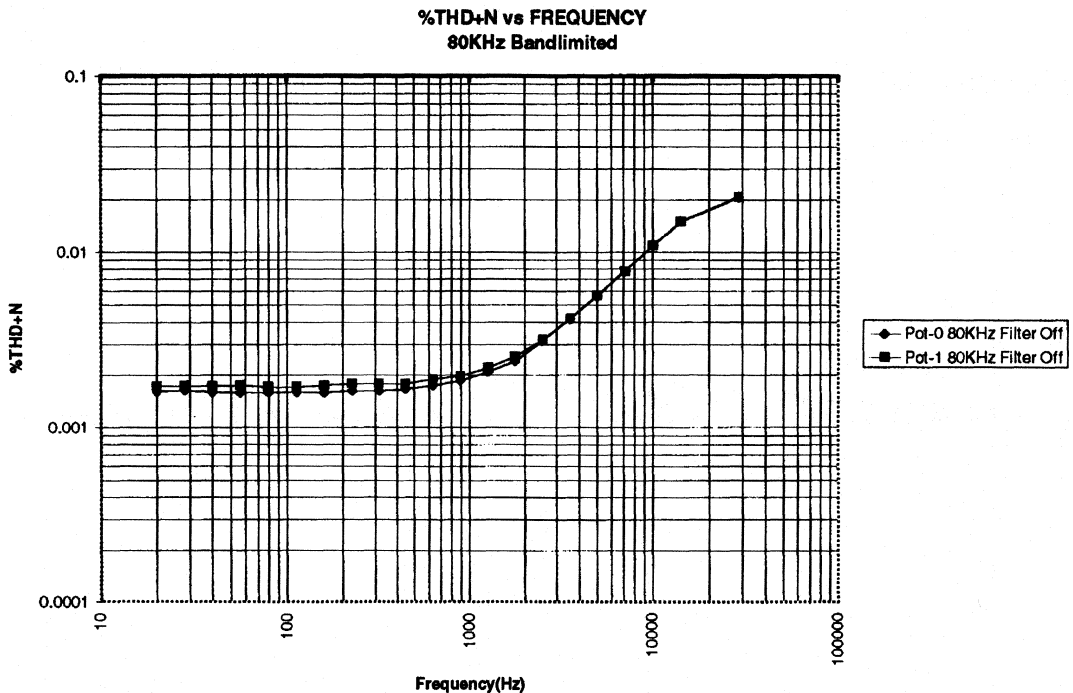
%THD+N vs. FREQUENCY 30 KHz BANDLIMITED Figure 7

%THD+N vs. FREQUENCY
80 KHz BANDLIMITED

Legend:
- Pot-0 30KHz-"A"WTG
- Pot-1 30KHz-"A"WTG
- Pot-0 30KHz-Filter Off
- Pot-1 30KHz-Filter Off

Frequency(Hz)

%THD+N vs. FREQUENCY 22 KHz BANDLIMITED Figure 8

%THD+N vs. FREQUENCY
30 KHz BANDLIMITED

Legend:
- Pot-0 22KHz-"A"WTG
- Pot-1 22KHz-"A"WTG
- Pot-0 22KHz-Filter Off
- Pot-1 22KHz-Filter Off

Frequency(Hz)

Intermodulation Distortion (IMD)

Intermodulation distortion data is provided for the SMPTE and the CCIF intermodulation measurement standards. Data is presented as a function of amplitude for each measurement technique and is shown in Figures 9 (for SMPTE) and 10 (for CCIF)

The SMPTE (Society of Motion Picture and Television Engineers) standard specifies a two–sinewave test signal consisting of a low–frequency high–amplitude tone linearly combined with a high–frequency sinewave at 1/4 the amplitude of the low–frequency tone. The SMPTE specification calls for 60 Hz and 7 KHz as the two sinewaves. When a non–linear device is subjected to a two–tone test signal, intermodulation products appear as sidebands around the high–frequency tone. The percentage intermodulation distortion is defined as the percentage of amplitude modulation which the second and third order pair of sidebands represent of the high–frequency signal. Second order sidebands

around the high frequency tone are spaced at a frequency equal to the low–frequency tone ($F_H \pm F_L$). Third order sidebands are spaced at twice the low–frequency tone or $F_H \pm 2F_L$. (F_H and F_L correspond to the high–frequency and low–frequency tones, respectively.)[2]

CCIF intermodulation distortion testing consists of using two equal–amplitude high frequency signals spaced closely in frequency. Signals used for test data presented include 13 KHz and 14 KHz. The audio analyzer measures only the amplitude of the difference tone or low frequency product resulting from the two high–frequency test signals. Here, the percentage intermodulation distortion is defined as the percentage of amplitude modulation which the low order or difference signal ($F_H - F_L$) represents on the high–frequency signals. Additional information on SMPTE and CCIF can be found in the Audio Measurement Handbook and System One Audio Analyzer User's Manual published by Audio Precision, Incorporated.[3]

SMPTE INTERMODULATION DISTORTION vs. AMPLITUDE Figure 9
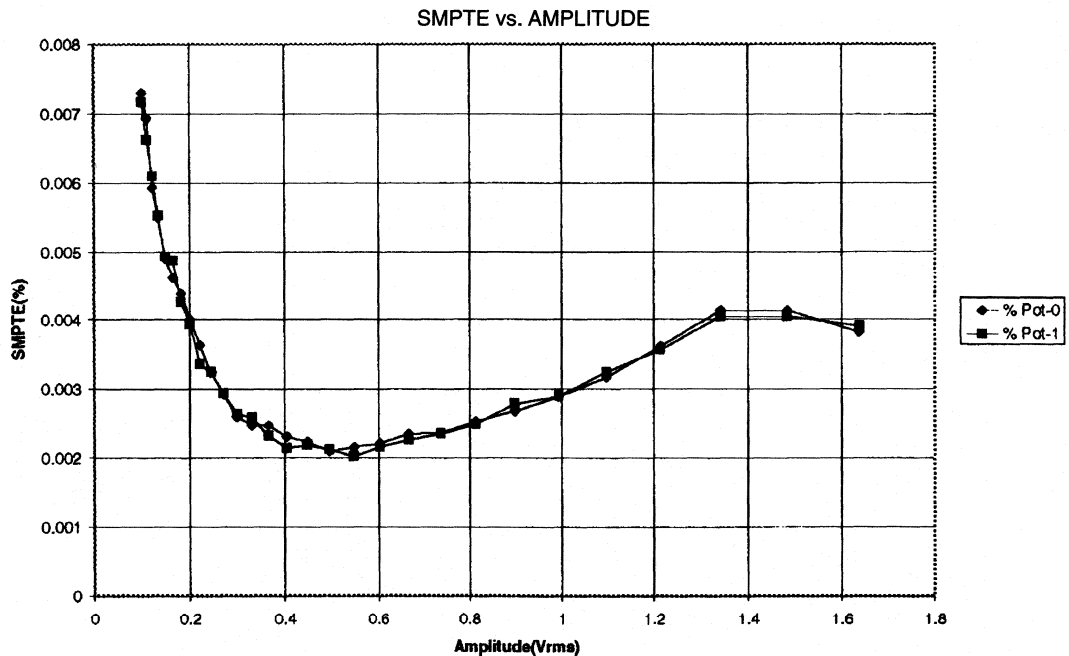
SMPTE vs. AMPLITUDE

CCIF INTERMODULATION DISTORTION vs. AMPLITUDE Figure 10


CCIF(%) vs. AMPLITUDE
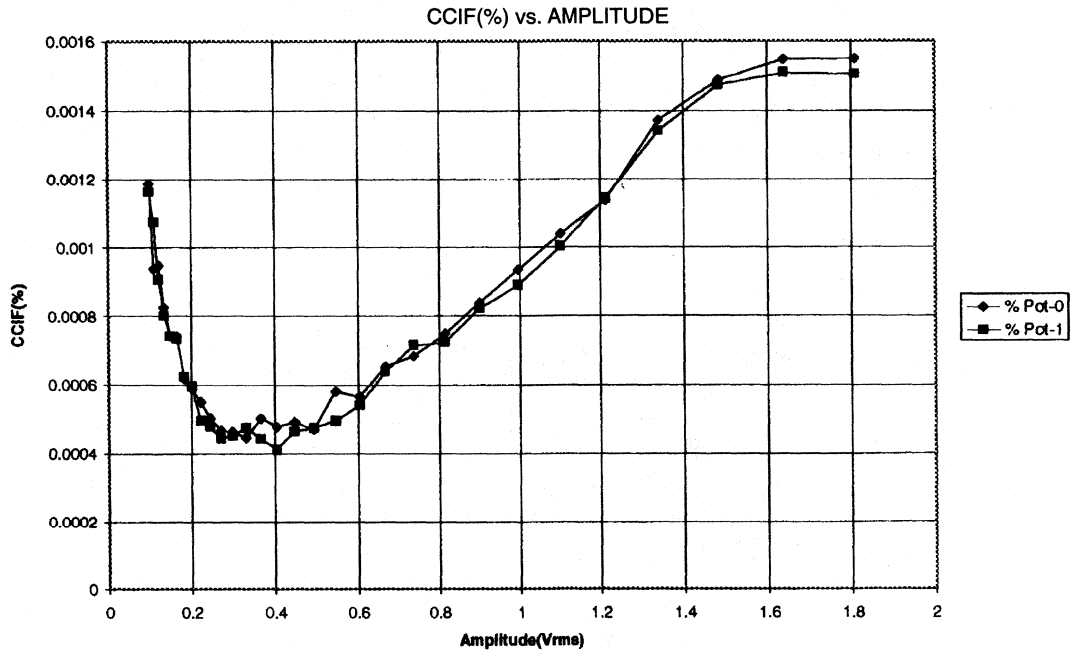
Crosstalk

Crosstalk measurements were performed using the test configuration diagrams of Figures 11 and 12. The crosstalk analysis is performed by evaluating signals taken from both wipers of the DS1802. In this test, potentiometer 0 is supplied a signal from the Audio Precision analyzer. This signal is returned to the analyzer via the wiper of potentiometer 0 (view either Figure 11 or 12). This signal return is connected to the audio analyzer's channel–A input.

Potentiometer 1 and its wiper (primarily the wiper) are used as the comparison point for signal feedthrough.

The high and low terminals, H1 and L1, are either left open circuited, as shown in Figure 11 or grounded as shown in Figure 12. The wiper of potentiometer 1 is connected to the audio analyzer's input via channel–B.

A comparison between the channel–A input and channel–B input of the audio analyzer is then made to determine the level of crosstalk. Graphical data of both the configurations of Figures 11 and 12 are presented in Figure 13.
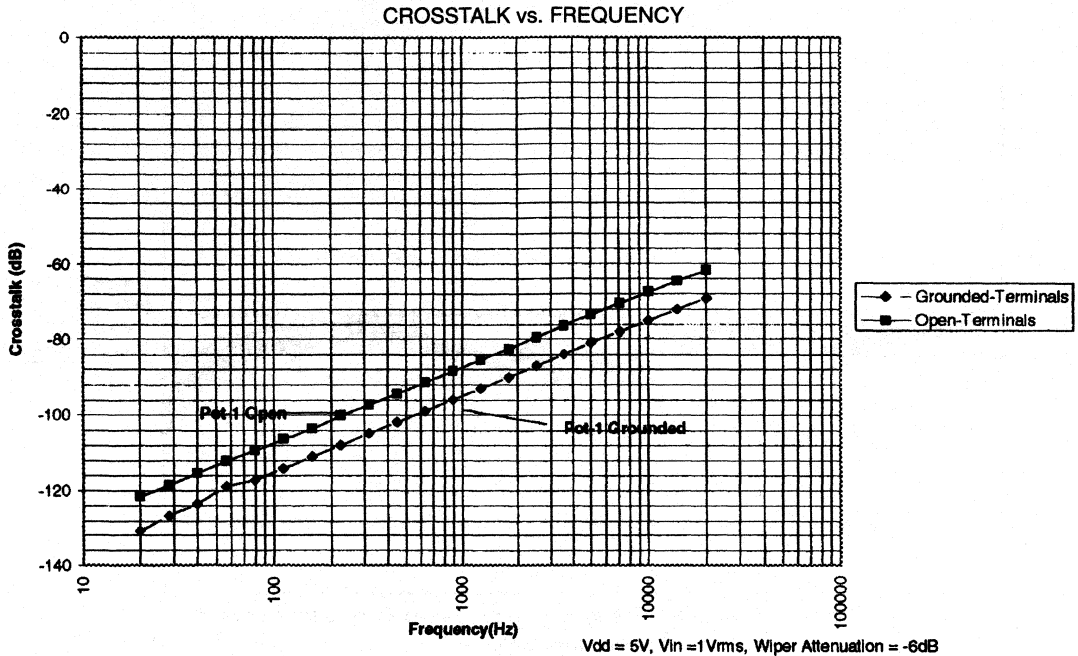
CROSS–TALK CONFIGURATION POTENTIOMETER 1 OPEN Figure 11
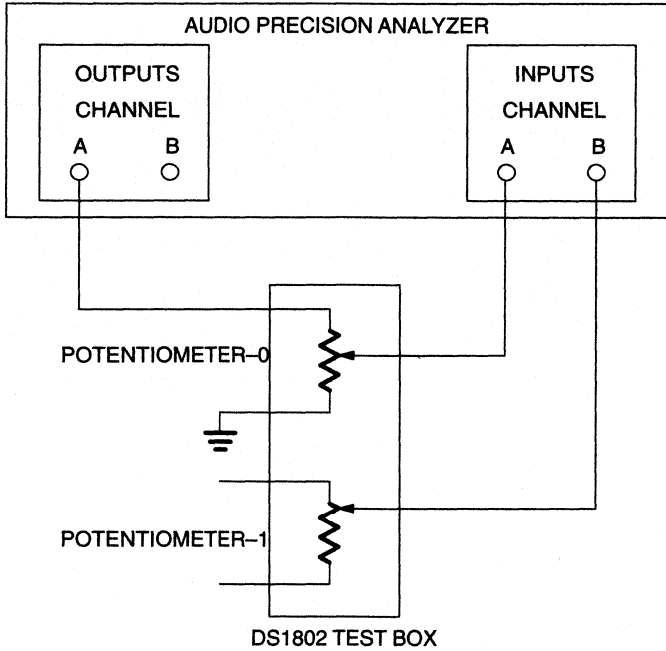
CROSS–TALK CONFIGURATION POTENTIOMETER 1 GROUNDED Figure 12
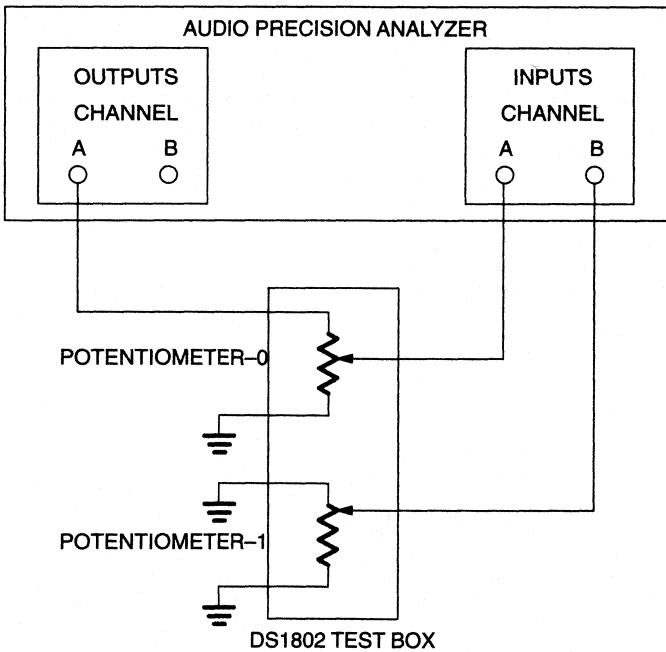
CROSS–TALK DATA Figure 13

CROSSTALK vs. FREQUENCY

Vdd = 5V, Vin =1Vrms, Wiper Attenuation = -6dB

Wiper Position vs. Temperature

Data is provided concerning each potentiometer's wiper position as a function of temperature. Temperatures evaluated include 0°C, 25°C, and 70°C. Data is provided in Figure 14 and is presented as a tolerance or absolute error as a function of wiper position. For example, wiper position 28 should correspond to a tolerance of 28 dB ± 0.5 dB.

As can be seen from graphical data, temperature has little or no effect on absolute tolerance.

WIPER POSITION vs. TEMPERATURE POTENTIOMETER 0 Figure 14


WIPER POSITION vs. TEMPERATURE
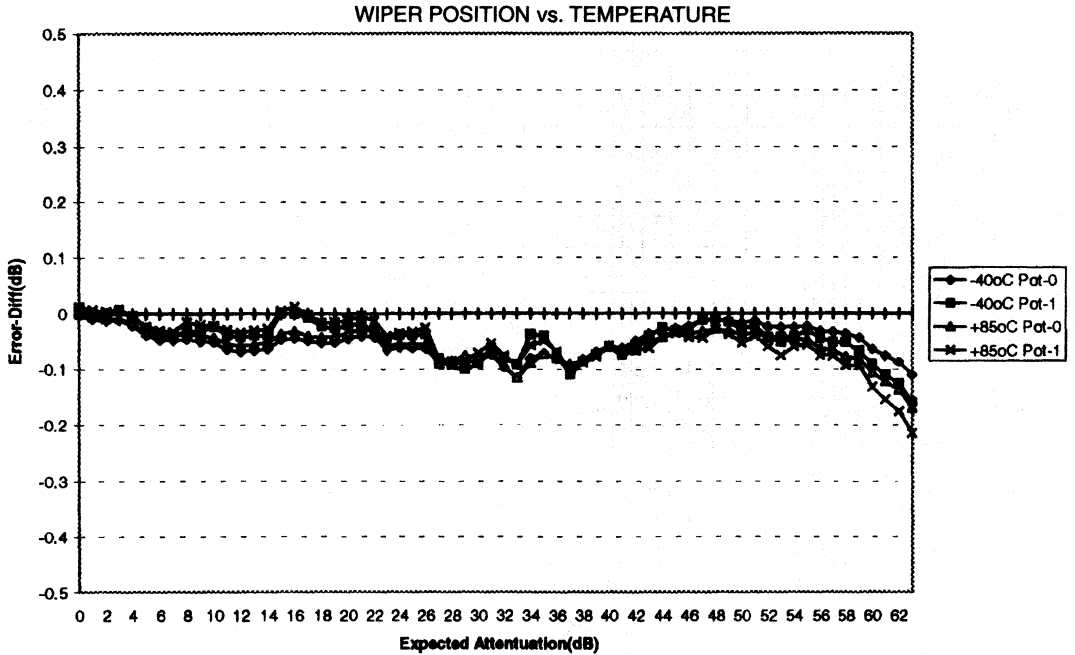
Device Muting Level

The DS1802 provides both a hardware and software mute capability. Data provided in Figures 15 and 16 show mute levels of a sample set of five parts. As shown, mute levels are in excess of –90.0 dB relative to wiper position–0, for both potentiometers over the given sample set. The DS1802 is specified to provide a mute capability of at least –90.0 dB.

DEVICE MUTING LEVEL – POTENTIOMETER 0 Figure 15

MUTE LEVEL VS TEMPERATURE
POTENTIOMETER-0

Mute Control Active: Typ. -90dB
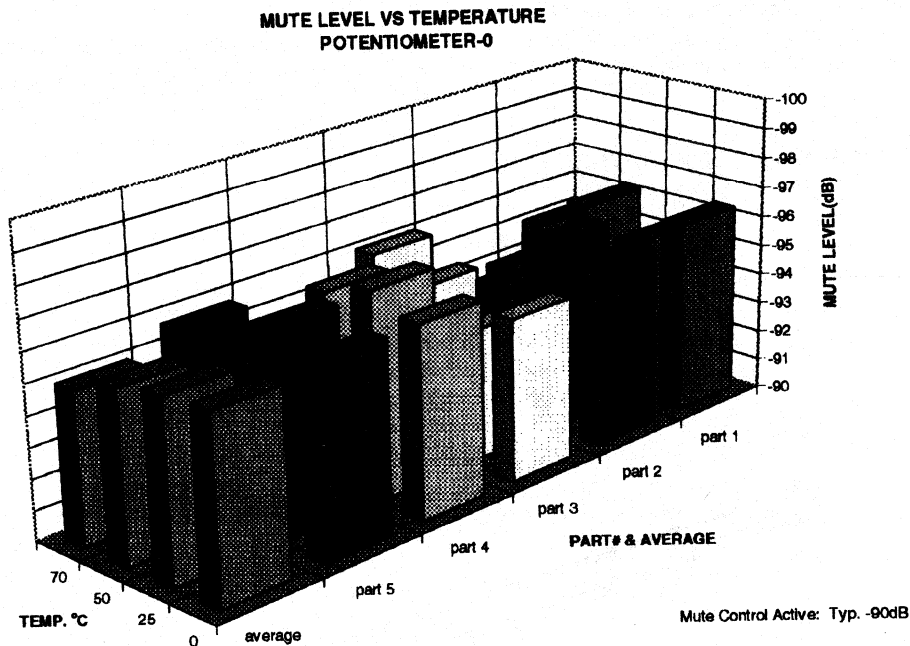
DEVICE MUTING LEVEL – POTENTIOMETER 1 Figure 16

MUTE LEVEL VS TEMPERATURE
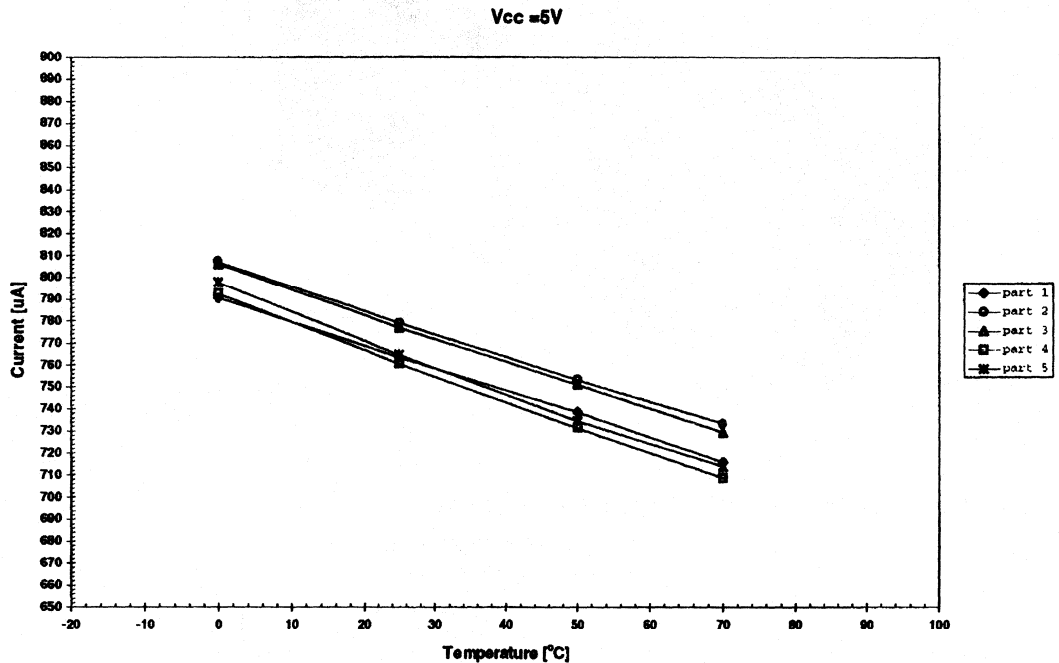**POTENTIOMETER-1**

021395 12/15
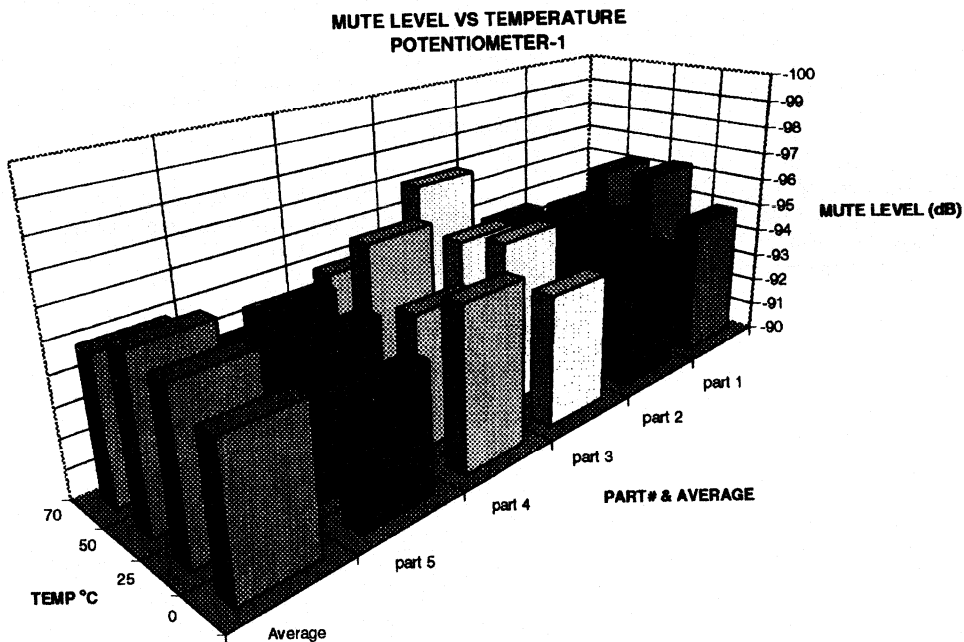
Active Current vs. Temperature

Data for active current is provided for a supply voltage of 5 volts. This data is presented in Figure 17. The DS1802 for this test is configured for zero–crossing detect mode, and pushbutton inputs active. This config-

uration provides worse–case application of the part for active current usage.

For information concerning push–button inputs or zero–crossing detection mode, refer to the data sheet of the DS1802.
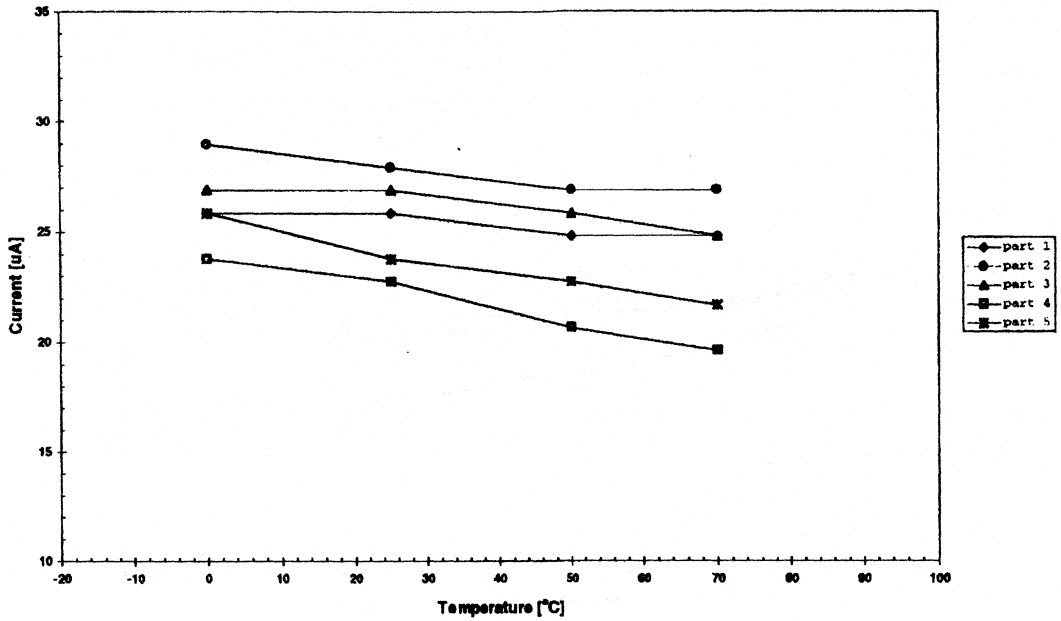
ACTIVE CURRENT vs. TEMPERATURE Figure 17

$V_{CC} = 5V$

Standby Current @ V_{CC}

Standby current data is provided for supply voltages of $V_{CC} = +3.0$ and $+5.0$ volts. The DS1802 configuration for this test has no activity on 3–Wire serial ports, push–

button inputs, or voltage across either potentiometer. Figures 18 and 19 provide graphical data for +3.0 and +5.0 volt operation.

STANDBY CURRENT @ V_{CC} = 3 VOLTS Figure 18

STANDBY CURRENT vs TEMPERATURE
Vcc = 3V

STANDBY CURRENT @ V_{CC} = 5 VOLTS Figure 19

STANDBY CURRENT vs TEMPERATURE
Vcc = 5V
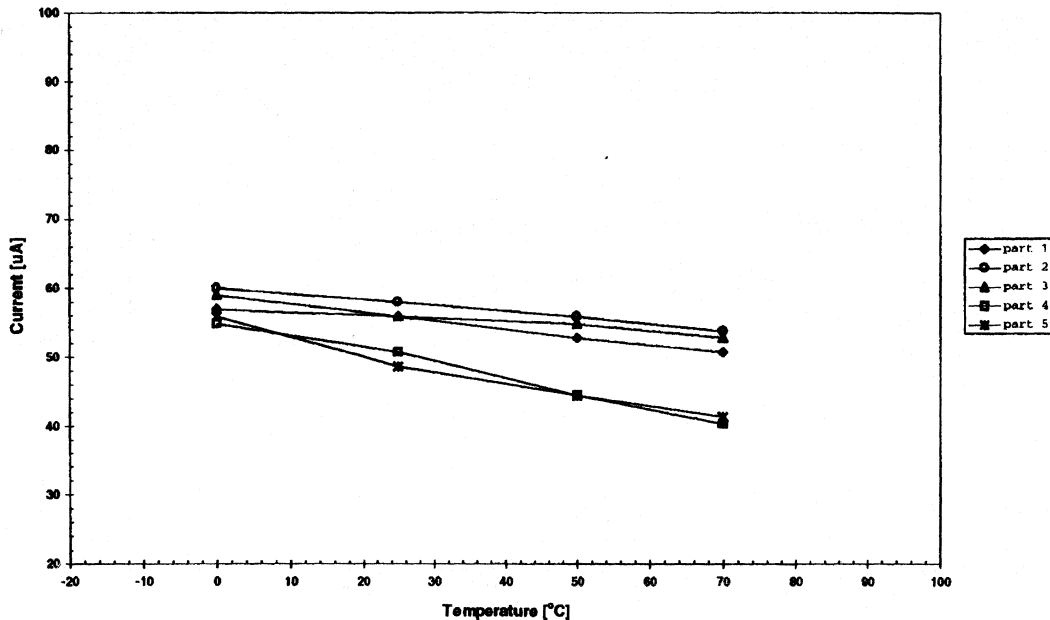
CONCLUSIONS

As stated in the Purpose of this document data presented here represents an examination of the audio characteristics of the DS1802. The DS1802 is specified as a 50 KΩ dual digital potentiometer. The intended use of this data is to provide the audio designer some feel for the performance of the DS1802 in an audio application. Although not a comprehensive report, data provided examines an array of commonly sought after specifications; such as THD+N, intermodulation distortion, and crosstalk.

Questions concerning this report or data provided in this report can be directed to 214–450–8167.

BIBLIOGRAPHY

1. Bob Metzler, Audio Measurement Handbook, 1st edition, (Beaverton, Oregon: Audio Precision, 1993), p. 147.

2. Bob Metzler, Audio Measurement Handbook, 1st edition, (Beaverton, Oregon: Audio Precision, 1993), p. 37–39.

3. Audio Precision, User's Manual Audio Precision System One, 15th Revision, (Beaverton, Oregon: Audio Precision, November 1992), Chapter 16.

I'm Interested. Who Do I Call?

For technical product information, call (972) 371-4448 or FAX us at (972) 371-3715. If calling from overseas, dial (972) 371-4351. We can mail a data book and product literature immediately, or we can have you talk to an applications engineer.

You can order any Dallas Semiconductor product for next-day shipment with a Visa, MasterCard, or American Express. Call our Credit Card Sales service at 1-800-336-6933.

You can also contact our nearest distributor, representative, or sales office.

Literature & Technical Support	(972) 371-4448
Sales & Customer Service	(972) 371-4969
Automatic Data Sheet Faxback	(972) 371-4441
Corporate FTP Site	ftp://ftp.dalsemi.com
Worldwide Web Site	http://www.dalsemi.com
E-mail Format	firstname.lastname@dalsemi.com
Sales Site	sales@dalsemi.com

HEADQUARTERS
Dallas, TX
Voice: (972) 371-4448
FAX: (972) 371-3715
Eastern Area
Duluth, GA
(770) 623-5813
New England/
E. Canada
Chelmsford, MA
(508) 256-4995
Mid Atlantic
Marlton, NJ
(609) 596-1919
Southeast/Florida
St. Petersburg, FL
(813) 527-6454
South Central
Dallas, TX
(972) 788-2197
North Central
Hoffman Estates, IL
(847) 490-5378
Great Lakes Area
Carmel, IN
(317) 573-7613
Western Area
Sunnyvale, CA
(408) 733-8400
Southwest
Irvine, CA
(714) 724-0790

EUROPEAN OFFICE
Birmingham, England
44-121-782-2959

PACIFIC RIM OFFICES
Dallas, TX
(972) 371-4944
Taiwan
Taipei, Taiwan
886-2-698-3940

SOUTH AMERICAN OFFICE
Marlton, NJ
(609) 596-1919

NORTH AMERICAN SALES REPRESENTATIVES
Alabama
Glen White & Associates
Huntsville, AL
(205) 882-6751
Arizona
System Sales of AZ
Mesa, AZ
(602) 464-9989
Arkansas
Bonser-Philhower Sales
Richardson, TX
(972) 234-8438
California
Harvey King, Inc.
San Diego, CA
(619) 695-9300
R.S.V.P. Associates
San Jose, CA
(408) 467-1200

S C Cubed
Tustin, CA
(714) 731-9206
Westlake Village, CA
(818) 865-6222
Canada
InTELaTech Inc.
Mississauga, Ontario
(905) 629-0082
Kanata, Ontario
(613) 599-7330
St. Laurent, Quebec
(514) 343-4877
Vancouver, BC
(604) 434-5699
Colorado
Waugaman Associates
Wheat Ridge, CO
(303) 423-1020
Connecticut
Technology Sales Inc.
Wallingford, CT
(203) 269-8853
Delaware
Trinkle Sales Inc.
Cherry Hill, NJ
(609) 795-4200
Florida
Naltron, Inc.
Davie, FL
(954) 370-9363
Indialantic, FL
(407) 777-3399
Tampa, FL
(813) 287-1433
Plantation, FL
(954) 236-3580

Winter Park, FL
(407) 657-7003
Georgia
Glen White & Associates
Duluth, GA
(770) 418-1500
Norcross, GA
(770) 448-0577
Idaho
Western Technical Sales
Boise, ID
(208) 376-8700
Illinois
Sumer Inc.
Rolling Meadows, IL
(847) 991-8500
Gibb Technology Sales
St. Louis, MO
(314) 890-0443
Indiana
Valentine Associates, Inc.
Carmel, IN
(317) 846-0008
Iowa
Gibb Technology Sales
Cedar Rapids, IA
(319) 363-6511
Kansas
Gibb Technology Sales
Kansas City, MO
(816) 483-6785
Kentucky
Millennium Tech. Sales
Kettering, OH
(513) 435-8650

Louisiana
Bonser-Philhower Sales
Austin, TX
(512) 346-9186
Houston, TX
(713) 782-4144
Richardson, TX
(972) 234-8438
Maine
Mill-Bern Associates
Woburn, MA
(617) 932-3311
Maryland
S-J Chesapeake
Falls Church, VA
(703) 533-2233
Massachusetts
Mill-Bern Associates
Woburn, MA
(617) 932-3311
Michigan
Trilogy Marketing
Auburn Hills, MI
(810) 377-4900
Minnesota
Cahill, Schmitz & Cahill
St. Paul, MN
(612) 646-7217
Mississippi
Glen White & Associates
Huntsville, AL
(205) 882-6751
Missouri
Gibb Technology Sales
St. Louis, MO
(314) 434-4220
Montana
Waugaman Associates
Wheat Ridge, CO
(303) 423-1020
Nebraska
Gibb Technology Sales
Kansas City, MO
(816) 483-6785
Waugaman Associates
Wheat Ridge, CO
(303) 423-1020
Nevada
R.S.V.P. Associates
San Jose, CA
(408) 467-1200
System Sales of AZ
Mesa, AZ
(602) 464-9989

New Hampshire
Mill-Bern Associates
Woburn, MA
(617) 932-3311
New Jersey
Trinkle Sales Inc.
Cherry Hill, NJ
(609) 795-4200
S-J Associates
Rockville Center, NY
(516) 536-4242
New Mexico
System Sales of AZ
Albuquerque, NM
(505) 889-2901
New York
S-J Associates
Rockville Center, NY
(516) 536-4242
Technology Sales, Inc.
Fairport, NY
(716) 223-7500
North Carolina
Glen White & Associates
Huntersville, NC
(704) 875-3777
Raleigh, NC
(919) 848-1931
North Dakota
Cahill, Schmitz & Cahill
St. Paul, MN
(612) 646-7217
Ohio
Millennium Tech. Sales
Dublin, OH
(614) 793-9545
Kettering, OH
(513) 435-8650
Mayfield Village, OH
(216) 461-3500
Oklahoma
Bonser-Philhower Sales
Richardson, TX
(972) 234-8438
Oregon
Western Technical Sales
Beaverton, OR
(503) 644-8860
Pennsylvania
Millennium Tech. Sales
Kettering, OH
(513) 435-8650
Imperial, PA
(412) 695-7661

Trinkle Sales Inc.
Cherry Hill, NJ
(609) 795-4200
Puerto Rico
Atland
Carolina, PR
(809) 752-2160
Rhode Island
Mill-Bern Associates
Woburn, MA
(617) 932-3311
South Carolina
Glen White & Associates
Huntersville, NC
(704) 875-3777
South Dakota
Cahill, Schmitz & Cahill
St. Paul, MN
(612) 646-7217
Tennessee
Glen White & Associates
Huntsville, AL
(205) 882-6751
Texas
Bonser-Philhower Sales
Austin, TX
(512) 346-9186
Houston, TX
(713) 782-4144
Richardson, TX
(972) 234-8438
Utah
Waugaman Associates
Salt Lake City, UT
(801) 261-0802
Vermont
Mill-Bern Associates
Woburn, MA
(617) 932-3311
Virginia
S-J Chesapeake
Falls Church, VA
(703) 533-2233
Washington
Western Technical Sales
Bellevue, WA
(206) 641-3900
Spokane, WA
(509) 922-7600
West Virginia
S-J Chesapeake
Falls Church, VA
(703) 533-2233

Wisconsin
Cahill, Schmitz & Cahill
St. Paul, MN
(612) 646-7217
Sumer, Inc.
Brookfield, WI
(414) 784-6641
Wyoming
Waugaman Associates
Wheat Ridge, CO
(303) 423-1020

NORTH AMERICAN DISTRIBUTORS

ARROW/SCHWEBER ELECTRONICS
Alabama
Hunstville, AL
(205) 837-6955
Arizona
Tempe, AZ
(602) 431-0030
California
Calabasas, CA
(818) 880-9686
Irvine, CA
(714) 587-0404
San Diego, CA
(619) 565-4800
San Jose, CA
(408) 441-9700
Canada
Burnaby, BC
(604) 421-2333
Dorval, Quebec
(514) 421-7411
Mississauga, Ontario
(905) 670-7769
Nepean, Ontario
(613) 226-6903
Colorado
Englewood, CO
(303) 799-0258
Connecticut
Wallingford, CT
(203) 265-7741
Florida
Deerfield Beach, FL
(954) 429-8200
Lake Mary, FL
(407) 333-9300
Georgia
Duluth, GA
(770) 497-1300

Illinois
Itasca, IL
(708) 250-0500
Indiana
Indianapolis, IN
(317) 299-2071
Kansas
Lenexa, KS
(913) 541-9542
Maryland
Columbia, MD
(410) 309-0686
Massachusetts
Wilmington, MA
(508) 658-0900
Michigan
Plymouth, MI
(313) 455-0850
Minnesota
Eden Prairie, MN
(612) 941-5280
Missouri
St. Louis, MO
(314) 567-6888
New Jersey
Marlton, NJ
(609) 596-8000
Pine Brook, NJ
(201) 227-7880
New York
Hauppauge, NY
(516) 231-1000
Melville, NY
(516) 391-1772
Rochester, NY
(716) 427-0300
North Carolina
Raleigh, NC
(919) 876-3132
Ohio
Dayton, OH
(513) 435-5563
Solon, OH
(216) 248-3990
Oklahoma
Tulsa, OK
(918) 252-7537
Oregon
Beaverton, OR
(503) 629-8090
Pennsylvania
Pittsburgh, PA
(412) 327-1130
Texas
Austin, TX
(512) 835-4180

Dallas, TX
(972) 380-6464
Houston, TX
(713) 647-6868
Utah
Salt Lake City, UT
(801) 973-6913
Washington
Bellevue, WA
(206) 643-9992
Wisconsin
Brookfield, WI
(414) 792-0150

FARNELL
Canada
Burnaby, BC
(604) 606-8950
Calgary, Alberta
(403) 273-2780
Concord, Ontario
(905) 738-1071
Nepean, Ontario
(613) 596-6980
Pointe Claire, Quebec
(514) 697-8149
Winnipeg, Manitoba
(204) 786-2589

FUTURE ELECTRONICS
Alabama
Huntsville, AL
(205) 830-2322
Arizona
Phoenix, AZ
(602) 968-7140
California
Agoura Hills, CA
(818) 865-0040
Irvine, CA
(714) 453-1515
San Diego, CA
(619) 625-2800
San Jose, CA
(408) 434-1122
Canada
Calgary, Alberta
(403) 250-5550
Edmonton, Alberta
(403) 438-2858
Mississauga, Ontario
(905) 612-9200
Ottawa, Ontario
(613) 820-8313
Pointe Claire, Quebec
(514) 694-7710

Quebec City, Quebec
(418) 877-6666
Vancouver, BC
(604) 294-1166
Winnipeg, Manitoba
(204) 944-1446
Colorado
Lakewood, CO
(303) 232-2008
Connecticut
Cheshire, CT
(203) 250-0083
Florida
Altamonte Springs, FL
(407) 865-7900
Deerfield Beach, FL
(954) 426-4043
Largo, FL
(813) 530-1222
Georgia
Norcross, GA
(770) 441-7676
Illinois
Hoffman Estates, IL
(847) 882-1255
Indiana
Indianapolis, IN
(317) 469-0447
Kansas
Overland Park, KS
(913) 649-1531
Maryland
Columbia, MD
(410) 290-0600
Massachusetts
Bolton, MA
(508) 779-3000
Michigan
Grand Rapids, MI
(616) 698-6800
Livonia, MI
(313) 261-5270
Minnesota
Eden Prairie, MN
(612) 944-2200
Missouri
St. Louis, MO
(314) 469-6805
New Jersey
Marlton, NJ
(609) 596-4080
Parsippany, NJ
(201) 299-0400
New York
Hauppauge, NY
(516) 234-4000

N. Syracuse, NY
(315) 451-2371
Rochester, NY
(716) 387-9550
North Carolina
Charlotte, NC
(704) 547-1107
Raleigh, NC
(919) 790-7111
Ohio
Beavercreek, OH
(513) 426-0090
Mayfield Heights, OH
(216) 449-6996
Oregon
Beaverton, OR
(503) 645-6454
Texas
Austin, TX
(512) 502-0991
Houston, TX
(713) 785-1155
Richardson, TX
(972) 437-2437
Utah
Salt Lake City, UT
(801) 467-4448
Washington
Bothell, WA
(206) 489-3400
Wisconsin
Brookfield, WI
(414) 879-0244

HAMILTON-HALLMARK
Alabama
Huntsville, AL
(205) 837-8700
Arizona
Phoenix, AZ
(602) 437-1200
California
Irvine, CA
(714) 789-4100
Rocklin, CA
(916) 624-9781
San Diego, CA
(619) 571-7540
San Jose, CA
(408) 435-3500
Woodland Hills, CA
(818) 594-0404
Colorado
Colorado Springs, CO
(719) 637-0055

Englewood, CO
(303) 790-1662
Connecticut
Cheshire, CT
(203) 271-2844
Florida
Ft. Lauderdale, FL
(305) 484-4582
Largo, FL
(813) 547-5000
Winter Park, FL
(407) 657-3300
Georgia
Duluth, GA
(770) 623-4400
Illinois
Arlington Heights, IL
(847) 797-7367
Indiana
Carmel, IN
(317) 575-3500
Kansas
Overland Park, KS
(913) 235-7900
Kentucky
Lexington, KY
(800) 235-6039
Maryland
Columbia, MD
(410) 720-3400
Massachusetts
Peabody, MA
(508) 532-9808
Michigan
Plymouth, MI
(313) 416-5800
Minnesota
Bloomington, MN
(612) 881-2600
Missouri
Earth City, MO
(314) 291-5350
New Jersey
Cherry Hill, NJ
(609) 424-0110
Parsippany, NJ
(201) 515-3000
New York
Hauppauge, NY
(516) 434-7400
Rochester, NY
(716) 475-9130
North Carolina
Raleigh, NC
(919) 872-0712

Ohio
Dayton, OH
(513) 439-6735
Solon, OH
(216) 498-1100ʼ
Worthington, OH
(614) 888-3313
Oklahoma
Tulsa, OK
(918) 254-6110
Oregon
Beaverton, OR
(503) 526-6200
Texas
Austin, TX
(512) 258-8848
Dallas, TX
(214) 553-4300
Houston, TX
(713) 781-6100
Utah
Salt Lake City, UT
(801) 266-2022
Wisconsin
New Berlin, WI
(414) 797-7844

INSIGHT ELECTRONICS
Alabama
Huntsville, AL
(205) 830-1222
Arizona
Tempe, AZ
(602) 829-1800
California
Irvine, CA
(714) 727-3291
San Diego, CA
(619) 587-1100
Sunnyvale, CA
(408) 720-9222
Westlake Village, CA
(818) 707-2101
Colorado
Englewood, CO
(303) 649-1800
Florida
Boca Raton, FL
(561) 997-2540
Clearwater, FL
(813) 524-8850
Georgia
Duluth, GA
(770) 717-8566

Illinois
Schaumburg, IL
(847) 885-9700
Massachusetts
Burlington, MA
(617) 270-9400
Minnesota
St. Louis Park, MN
(612) 525-9999
Ohio
Valley View, OH
(216) 520-4333
Oregon
Beaverton, OR
(503) 644-3300
Texas
Austin, TX
(512) 719-3090
Houston, TX
(713) 448-0800
Richardson, TX
(972) 783-0800
Washington
Kirkland, WA
(206) 820-8100
Wisconsin
Wauwatosa, WI
(414) 258-5338

*INTERFACE
ELECTRONICS*
Massachusetts
Hopkinton, MA
(508) 435-6858
North Carolina
Raleigh, NC
(919) 797-8744

MILGRAY ELECTRONICS
Alabama
Madison, AL
(205) 464-8646
California
Irvine, CA
(714) 753-1282
San Diego, CA
(619) 457-7545
San Jose, CA
(408) 456-0900
Thousand Oaks, CA
(805) 371-9399
Canada
Burnaby, BC
(604) 291-0044
Mississauga, Ontario
(905) 678-0958

Pointe Claire, Quebec
(514) 426-5900
Colorado
Westminster, CO
(303) 657-2750
Connecticut
Milford, CT
(203) 878-5538
Florida
Lake Mary, FL
(407) 321-2555
Georgia
Norcross, GA
(770) 446-9777
Illinois
Palatine, IL
(847) 202-1900
Indiana
Indianapolis, IN
(317) 781-9997
Kansas
Overland Park, KS
(913) 236-8800
Maryland
Columbia, MD
(410) 730-6119
Massachusetts
Wilmington, MA
(508) 657-5900
New Jersey
Mt. Laurel, NJ
(609) 778-1300
Parsippany, NJ
(201) 335-1766
New York
Farmingdale, NY
(516) 391-3000
Pittsford, NY
(716) 381-9700
North Carolina
Raleigh, NC
(919) 790-8094
Ohio
Cleveland, OH
(216) 447-1520
Oregon
Beaverton, OR
(503) 626-4040
Texas
Austin, TX
(512) 331-9961
Dallas, TX
(972) 248-1603
Houston, TX
(713) 870-8101

Utah
Murray, UT
(801) 261-2999

NEWARK ELECTRONICS
Illinois
Chicago, IL
1-800-4NEWARK

STERLING
ELECTRONICS
Arizona
Phoenix, AZ
(602) 437-5565
California
Irvine, CA
(714) 453-7660
San Diego, CA
(619) 560-8097
San Jose, CA
(408) 435-0835
Westlake Village, CA
(818) 865-2333
Colorado
Englewood, CO
(303) 792-3939
Connecticut
Wallingford, CT
(203) 265-9535
Florida
Altamonte Springs, FL
(407) 260-8558
Georgia
Norcross, GA
(770) 441-0449
Illinois
Schaumburg, IL
(847) 303-9900
Kansas
Lenexa, KS
(913) 492-5406
Maryland
Columbia, MD
(410) 290-3800
Massachusetts
Woburn, MA
(617) 938-6200
Minnesota
Eden Prairie, MN
(612) 946-8888
New Jersey
Mt. Laurel, NJ
(609) 273-6420
Parsippany, NJ
(201) 331-0999

New Mexico
Albuquerque, NM
(505) 884-1900
North Carolina
Raleigh, NC
(919) 790-8634
Ohio
Solon, OH
(216) 248-1122
Oklahoma
Tulsa, OK
(918) 663-2410
Texas
Austin, TX
(512) 836-1341
Carrollton, TX
(972) 243-1600
Houston, TX
(713) 627-9800
Utah
Salt Lake City, UT
(801) 972-5444
Virginia
Richmond, VA
(804) 323-5510
Washington
Kirkland, WA
(206) 823-9770

INTERNATIONAL
REPRESENTATIVES &
DISTRIBUTORS
Argentina
Semak S.A.
54-1-381-2108
Australia
Dallas Identification
61-3-9720-5344
Fagan Microelectronics Sys.
Pty. Ltd.
61-3-9699-9899
Veltek Pty. Ltd.
61-03-9574-9300
61-02-9745-1400
61-08-8232-3922
61-09-472-3855
61-07-3216-0770
Austria
Eurodis Electronics
43-1-610620
Belgium
Acal Betea Comp.
32-2-720-5983
Bulgaria
Isomatic Lab Ltd.
359-2-77-4596

China
CET Limited
852-2485-3899
Waslin (H.K.) Limited
852-2-693-6811
Croatia/Bosnia-
Herzegovina
HT-Eurep d.o.o.
385-1-6129610
Czech Republic
HT-Eurep Elec. spol.sr.o.
42-2-663-13053
Denmark
Micronor
45-86-81-65-22
Finland
Memec Finland Oy
358-9836-2600
France
Avnet EMG SA.
33-1-4965-2500
SCAIB S.A.
33-1-6919-8900
Rep' Tronic
(Representative)
33-1-6953-6724
Germany
Atlantik Elektronik
49-89-89505-0
Future Electronics
49-899-57270
Great Britain
D.T. Electronics Ltd.
44-1203-466500
Future Electronics Ltd.
44-1753-763000
Joseph Electronics
44-121-643-6999
Silver Birch Marketing
44-1480-812806
Greece
Drogeta Engineering
30-1-881-0948
Hong Kong
CET Limited
852-2485-3899
Waslin (H.K.) Limited
852-2-693-6811
Hungary
HT-Eurep Elec. Kft.
36-1-270-1002
India
Hynetic Electronics
91-80-620-852

Ireland
SEI-Bloomer Electronics
44-1762-339818
Israel
STG International
972-3-696-5231
Italy
Comprel, S.P.A.
39-3-625-781
Sitel S.p.
39-362-366027
Japan
Microtek Inc.
81-3-5300-5535
Systems Marketing
81-33-254-2751
Korea
Amerix Corporation
82-2-423-9623
Sangsoo Co., Ltd.
82-2-780-5360
Lithuania, Latvia &
Estonia
DanBalt Electronics
45-86846171
Malaysia
Dynamar
60-4-228-1860
The Netherlands
Alcom Electronics BV
31-10-451-9533
New Zealand
Arrow Components NZ Ltd.
64-3-366-2000
Norway
BIT Elektronikk A.S.
47-66-98-1370
Philippines
Dynamar
63-2-818-3550
Poland
HT-Eurep Elec.sp.zo.o.
48-22-621-7704
Portugal
Comelta S.A.
351-941-6507
Romania
HT-Eurep Elec. Kft.
36-1-270-1002
Russia
Software Security, Belarus
7-0172-45-21-03
Singapore
Dynamar
65-542-1878

Slovenia & Croatia
Cadix d.o.o.
386-61-1592577
South Africa
Tarsus Technology Pty. Ltd.
27-11-886-3165
Spain
Comelta, S.A. Barcelona
34-3-582-1991
Comelta S.A. Madrid
34-1327-0614
Sweden
Dipcom Electronics AB
46-87-522-480
Switzerland
Computer Controls AG:
Zurich
41-1-308-6666
Yverdon Les Bains
41-24-423-8200
Taiwan
Landcol Enterprises
886-2-698-3328
Thailand
Dynamar
66-2-376-0132
Turkey
Politeknik Elektronik
90-232-232-7432