Application Note 139
Controlling a DS1803 Digital Potentiometer Using an 8051 Microprocessor to Generate 2-Wire Signals

www.dalsemi.com

Introduction

Dallas Semiconductor's DS1803 is ideal for systems that require the ability to change a biasing resistance, current, or voltage, without the manual intervention required by a mechanical potentiometer. It can also provide a dual Digital to Analog Converter (DAC) if the potentiometers are wired as voltage dividers. Because the DS1803 requires only two I/O pins from a microprocessor to address up to 16 potentiometers, it is perfect for applications with limited I/O capacity and high performance requirements.

This application note presents a simple hardware and software solution for controlling the DS1803 using a Dallas Semiconductor 8051-compatible microprocessor, the DS87C520. A basic schematic shows the necessary connections between the microprocessor and the DS1803, and 8051 assembler code provides a small application for controlling the chip's two digital potentiometers using different write modes. After each write cycle, the application reads back the information from the DS1803 and displays it on a PC via a serial port to demonstrate a properly operational read function.

2-Wire Protocol Overview and Implementation on an 8051

The 2-wire serial protocol utilizes two signals to transmit information: Serial Data (SDA) and Serial Clock (SCL). If both signals are high for the bus free time (1.3us, Fast Mode), then the bus is considered free and any bus master may begin to transmit a signal to any slave on the bus. The transmit signal is initiated by a start condition. A start condition (S) occurs when SDA is brought low while SCL remains high (see Figure 1). After the start condition is generated, the master sends an 8-bit message using SCL and SDA to the slave it wishes to address. In the case of the DS1803, this 8-bit message is a control byte that includes the address of the DS1803 and an indication of whether the master wishes to read from or write to the part. After the slave receives the control byte, it acknowledges the master by pulling the SDA line low.

For a write command, the master transmits another command byte and waits to receive acknowledgment from the slave. After the slave acknowledges the command byte, the master proceeds to transmit the data to the slave in 8-bit blocks, with acknowledgment from the slave each time. After all the data has been transmitted, the master generates a stop condition (labeled P, Figure 1) by setting SDA high when SCL is already high.

Reading is accomplished by sending a different control byte. After the slave acknowledges the control byte, the master continues to clock the SCL line and reads the information that the slave provides on the SDA line. The master must acknowledge the slave after each byte is received, with the exception of the last byte read. After the last byte is read, the master generates a stop condition without acknowledging the slave.

It should be noted that during data transmission, SDA can only be adjusted while SCL is low (otherwise an inadvertent start or stop condition is generated). Further explanation of the 2-wire protocol can be found in the DS1803 data sheet, which can be obtained from the following URL:
http://www.dalsemi.com/datasheets/pdfindex.html

It is relatively easy to write code to communicate with a DS1803 using an 8051 microcontroller -- especially considering that changing a single I/O (such as SCL or SDA) pin requires only a single instruction on this microprocessor. The 2-wire protocol is also nice because it has no specifications for minimum communication rates. Thus, the signals cannot be placed on the bus too fast, but they can be very slow if the processor is required for another more important task. Since each instruction on the 8051 takes a definite number of clock cycles, it is easy to determine the length of time per instruction. This information can be used to determine number of instructions that must be between the modification of either the SCL or SDA pin. If additional delay is required, one or several ***nop*** instructions can be added to stall for the required length of time.

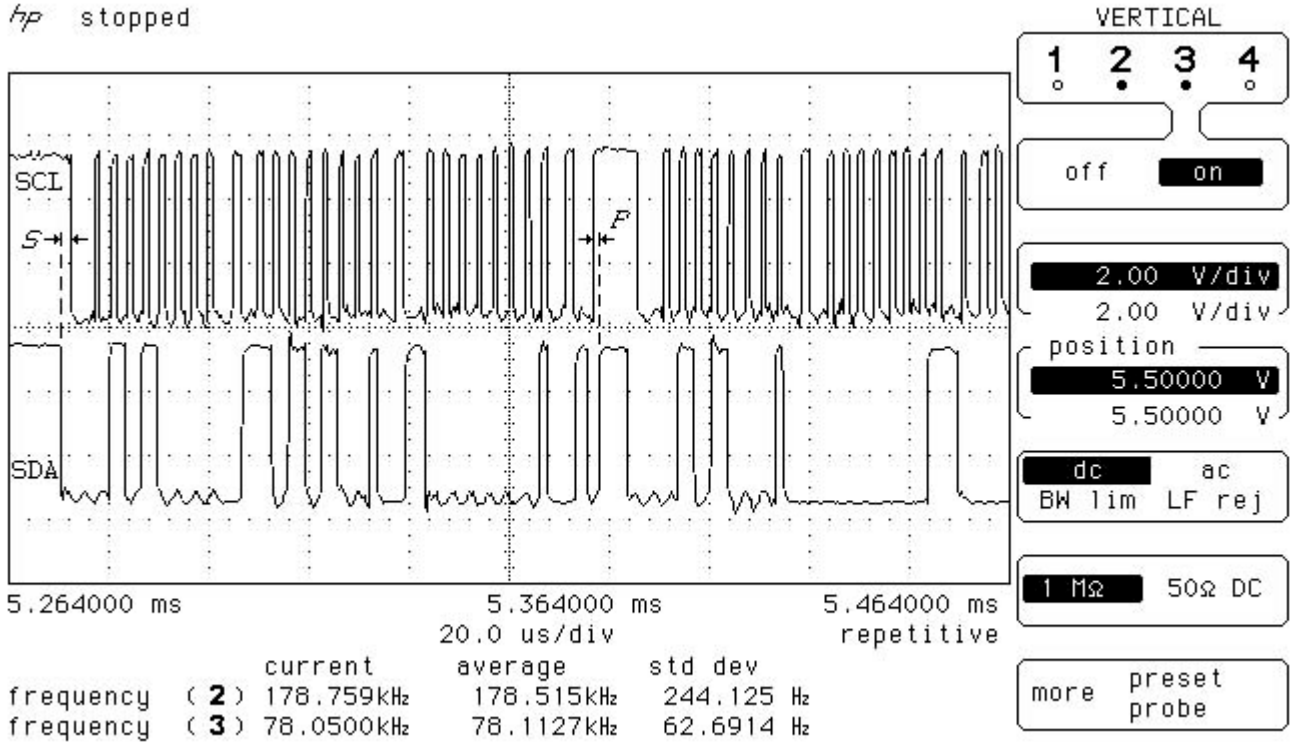

Figure 1 – Start and Stop Conditions on a 2-Wire Bus

Hardware Setup

Because there are myriad 8051 hardware setups that could be used to talk to the DS1803, the diagram below shows only the connections required to make the 8051 work with the DS1803 and the serial port. The serial port is used to demonstrate that the 8051 reads the correct values from the DS1803.

The microcontroller is interfaced to the DS1803 using P1.0 as SCL and P1.1 as SDA. Notice that 4.7 kΩ pull-up resistors are used to pull both the SDA and SCL lines high when the bus is not being driven low by either the 8051 or the DS1803. The DS1803 has been configured to provide two voltage outputs. This is accomplished by attaching HX and LX to Vcc and Gnd, respectively. The address lines of the DS1803 are grounded, so the chip is operating at address zero (0). If a different address is desired, the control bytes in the software must be modified to allow communication to the device.

The third IC shown is the DS232A RS232 Transceiver. It is used to level-shift the 0-5V CMOS signals to and from ±12V signals for RS232 transmission and reception. It uses its 5V Vcc to generate the ±12V supplies needed to perform with the capacitors shown (required for proper operation) on the diagram. The DS232A is a convenient way to provide the hardware for RS232 communication, which allows verification

that the DS1803 design operates correctly. However, if the hardware for serial port 1 is not provided, the code should still work to read and write the digital potentiometer if all of the calls to the ***DisplayPots*** routine are eliminated.

Note: The code as written assumes that the microcontroller is operating on a 22.118 MHz clock.

The clock has to be close to that frequency for RS232 communication at 19200 baud. If the microprocessor is running at a different frequency, the serial port initialization routine may require modifications to the timer 1 auto-reload value, and a new baud rate that is compatible with the microprocessor's clock rate may have to be determined. The microcontroller user's guide should provide some guidelines on serial port setup.

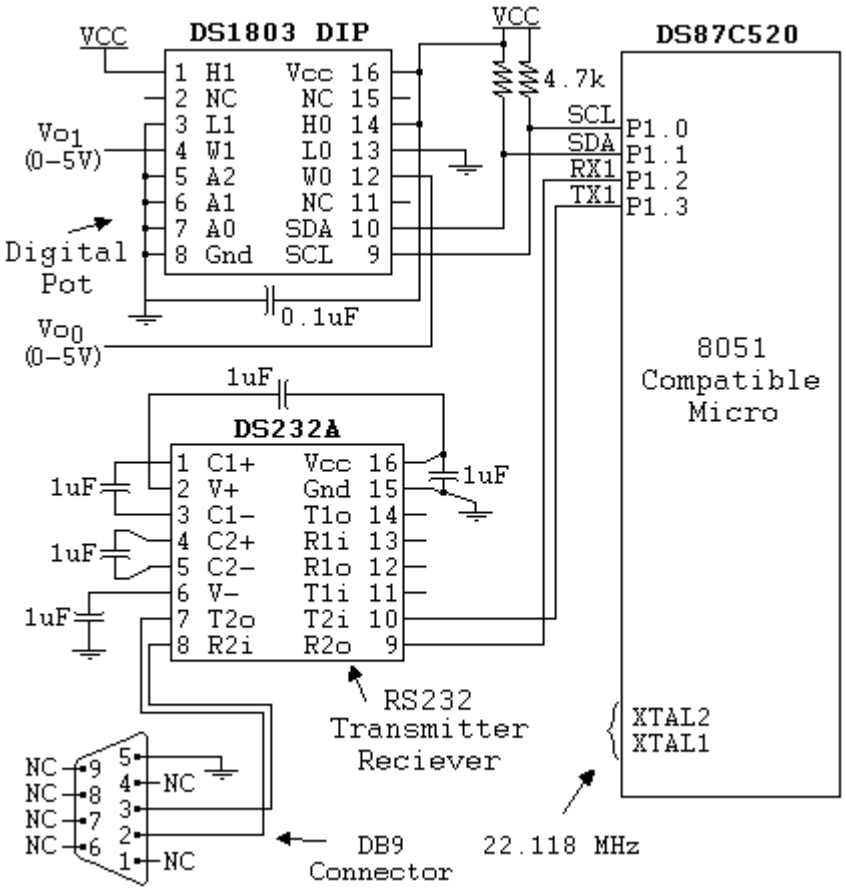

Figure 2 – Hardware Required for Proper Operation of DS1803 8051 Software

Note: All of the delays used for the 2-wire protocol have been generated using ***nop*** instructions.

Therefore, some of the communication code may need modification to increase or decrease the delays if a different clock rate is desired or if a different 8051 microprocessor is used. The DS87C520 processes a ***nop*** instruction in 4 clock cycles. Some 8051 microprocessors may require 12 clock cycles to perform a ***nop***, and hence less ***nop***s are required. It does not hurt to add extra ***nop***s into the code for additional delay if it is believed that the code is not providing proper timing. As mentioned above, there are no limitations on how slow the bus can operate in the 2-wire specification.

If a 22.118 MHz clock is being used, the microcontroller may be connected to the serial port of a PC, and the results of the program's read attempts will be displayed on the PC terminal. The communications port on the PC must be set up as 19200 baud, 1 stop bit, with no parity or flow control.

A boot strap loader system is shown in Appendix B as a practical hardware set-up for development purposes. It is for reference only. It shows the connections to the DS1803 and the DS232A ICs, as well as connections to a DS1075 oscillator to clock the 8051, and connections to a DS1245Y NV-SRAM for storage of both data and code. Appendix B also contains a brief description of how the internal memory of the 8051 is programmed to enable it to operate as a boot strap loader system.

Controlling the DS1803 Using Software Generated Commands on a 2-Wire Bus

This section briefly describes what each routine in the DS1803 software accomplishes. The code associated with the descriptions is in Appendix A.

There are four groups of code: initialization routines, command routines, bit driver routines, and miscellaneous routines. All DS1803 communication occurs in the bit driver routines. These are the functions that actually place data or receive the data on SDA and send the clock signals to the DS1803. The command routines allow transmission of the control and command bytes without the need to type out the same sequences each time in main section of the program.

Initialization Routines

There are three initialization routines in the software that prepare the 8051 to communicate with both the DS1803 and the PC attached to serial port 1. These routines initialize serial port 1, write a welcome message to the PC, and initialize the SDA and SCL lines.

lcall initSP1

Initializes serial port 1 by accomplishing the following: the baud rate doubler is enabled; serial port 1 is placed in asynchronous 10 bit mode; timer 1 is enabled and placed in 8-bit auto reload mode; and timer 0 is on and in 16-bit count mode. The timer 1 auto reload value is also set, and that determines the baud rate will be 19200 baud with the current 22.22 MHz clock.

lcall intro

Displays welcome message by pointing the data pointer **DTPR** to the beginning of the message and calling the ***outstr*** routine.

lcall init2wire

Initializes SDA and SCL by setting both at the start of the program.

Command Routines

These routines transmit the control and command bytes required for operation of the DS1803. All of the command routines that proceed writes to the DS1803 require more information (data) to be sent with subsequent commands as described below. The commands for reading and displaying the values controlling each potentiometer can handle all of the data without need for further processing.

lcall CMDReadPots

Reads the value at which both potentiometers are currently set. Sends a start condition with ***start2wire***, and then writes the control byte to read the potentiometers with ***writebits*** and ***AckSlaveWrite***. Next, the ***readbits*** and the ***AckSlaveRead*** routines are used to place the 8-bits of data from the slave into byte form in the

accumulator. Once received, each byte is placed in the 8051's RAM. Finally, the command is terminated with a stop condition via the ***stop2wire*** command.

Note: The last byte during a read is not acknowledged, as specified in the DS1803 2-wire instructions.

lcall DisplayPots
Displays the values read by ***CMDReadPots*** on a PC via serial port 1. This is accomplished by using the ***binasc*** routine to convert the 8-bit number retrieved from the potentiometer to two ASCII bytes in hexadecimal format. The ***outchar*** routine then transmits the two ASCII bytes to the PC using serial port 1.

lcall CMDWritePot0
Sends a start condition and then writes both the control and command bytes required to set the value of potentiometer 0. After the control and command bytes are written, either 1 or 2 bytes must be written to the part using the ***WriteByte*** routine. Following the data, a stop condition must be generated. The first byte written out following this command sets the value of potentiometer 0, and the second byte (optional) sets the value of potentiometer 1.

lcall CMDWritePot1
Sends a start condition and then writes both the control and command bytes required to change the value of potentiometer 1. One byte of data will always be sent to the potentiometer after this command, and a stop condition will always follow the data byte. This command will write the value of the data byte to potentiometer 1.

lcall CMDWritePot01
Sends a start condition and then writes both the control and command bytes required to set both potentiometers to the same value. One data byte will always be sent to the potentiometer after this command, and a stop condition will always follow the data byte. This command will write the value of the data byte to both potentiometers.

lcall WriteByte
The routine calls the ***writebits*** and the ***AckSlaveWrite*** routines to write the data byte in the accumulator at the time that ***WriteByte*** is called.

Bit-Level Driver Routines
These routines are used to drive the SCL and SDA lines with the proper timing to communicate with the DS1803. The timing is generated using ***nops*** and calls to the ***wait2us*** routine.

lcall writebits
Performs the functions necessary to write a byte, 1 bit at a time over the 2-wire bus. The MSB of the byte in the accumulator is rotated into the carry bit, the carry bit is transferred to SDA, and the bus is clocked. All timing for this routine is done by ***nops***, and it always writes an entire byte.

lcall readbits
Performs the functions necessary to read a byte one bit at a time from the 2-wire bus. Each time through the loop, the clock (SCL) is set and the current bit is read from the SDA line into the carry bit. The carry bit is then rotated into the accumulator (MSB first). After each bit is read, SCL is cleared, completing 1 clock cycle. All timing for this routine is done by ***nops***, and it always reads an entire byte.

lcall AckSlaveWrite
Checks for an acknowledgment from the slave (pull SDA low) after the ***writebits*** routine is called. If the slave fails to acknowledge the data transfer, then the microprocessor sends a message over the serial port to the PC that says "Ack Fail." If the slave does acknowledge the data transfer, then the routine simply returns to the calling function. All timing for this routine is also done with ***nops***.

lcall AckSlaveRead
Pulls SDA low and sends a clock pulse to the slave to acknowledge the data was received during a read operation. All timing is generated in this routine using ***nops***.

lcall start2wire
Waits until the bus is not busy (SDA and SCL both high), and then pulls SDA low to create a start condition. It uses the ***wait2us*** function to generate timing.

lcall stop2wire
Sets SDA high while SCL is already high to create a stop condition. Timing is generated by a combination of ***nops*** and ***wait2us*** calls.

lcall wait2us
Creates a 1.6us delay by using a ***lcall***, single ***nop***, and a ***ret*** instuction.

Miscellaneous Routines
There are three routines used for communicating over the serial port. These routines convert a byte number to two ASCII characters, write a single character to the PC via serial port 1, and write a null terminated string to the PC using serial port 1.

lcall binasc
Converts a byte number to two ASCII characters in a hexadecimal format.

lcall outchar
Moves a single byte into the serial port 1 buffer and waits for the microprocessor serial transmission complete flag to be set before returning to the calling function.

lcall outstr
Sends a null terminated string to the PC via serial port 1. This is accomplished by sending consecutive bytes starting at the location pointed to by the **DTPR** register at the time ***outstr*** is called. When **DTPR** points to the character zero (null character), control is returned to the calling function.

The Main Program
The main program starts at the ***start*** label. It first initializes the program, and then performs the ***CMDReadPots*** (see Figure 3) and ***DisplayPots*** commands. If the part was powered down before these commands, both pots should contain zero as their current value. If the parts have been written to since the last power down, the potentiometers will contain their last values before the read function. Next, 01h is written to potentiometer 0, and the program reads and displays the value of both potentiometers. The values read now should be 01h for potentiometer 0 and 00h (if the part was powered down, see Figures 4 and 5) for potentiometer 1. Next, 02h is written to potentiometer 1, and the value of both potentiometers is again read and displayed. The potentiometer's values should now be 01h and 02h for potentiometers 0 and 1 respectively. The next command demonstrated is ***CMDWritePot01***. The program writes 13h to both pots, and then reads and displays the status of the potentiometers. Finally, it uses the ***CMDWritePot0*** command to write 24h and 25h to potentiometer 0 and 1 respectively. The results are again verified. After the

program is finished, it enters an infinite loop to end execution. The PC output of the program can be seen in Appendix C.

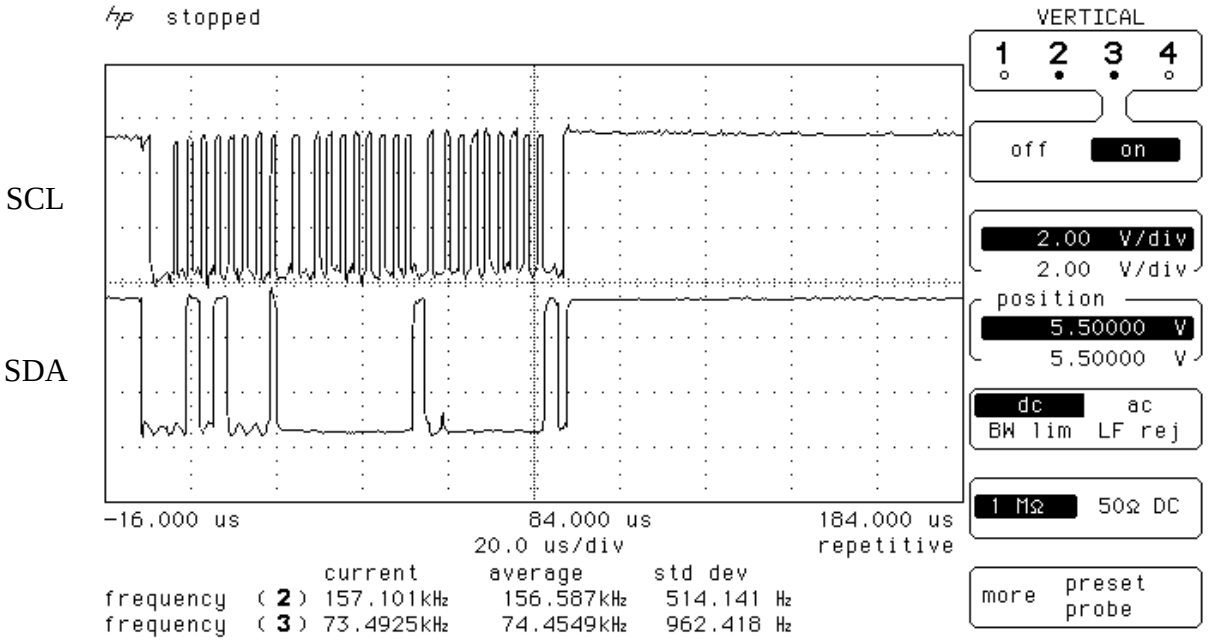

Figure 3 – The First Read Cycle As Seen On An Oscilloscope.

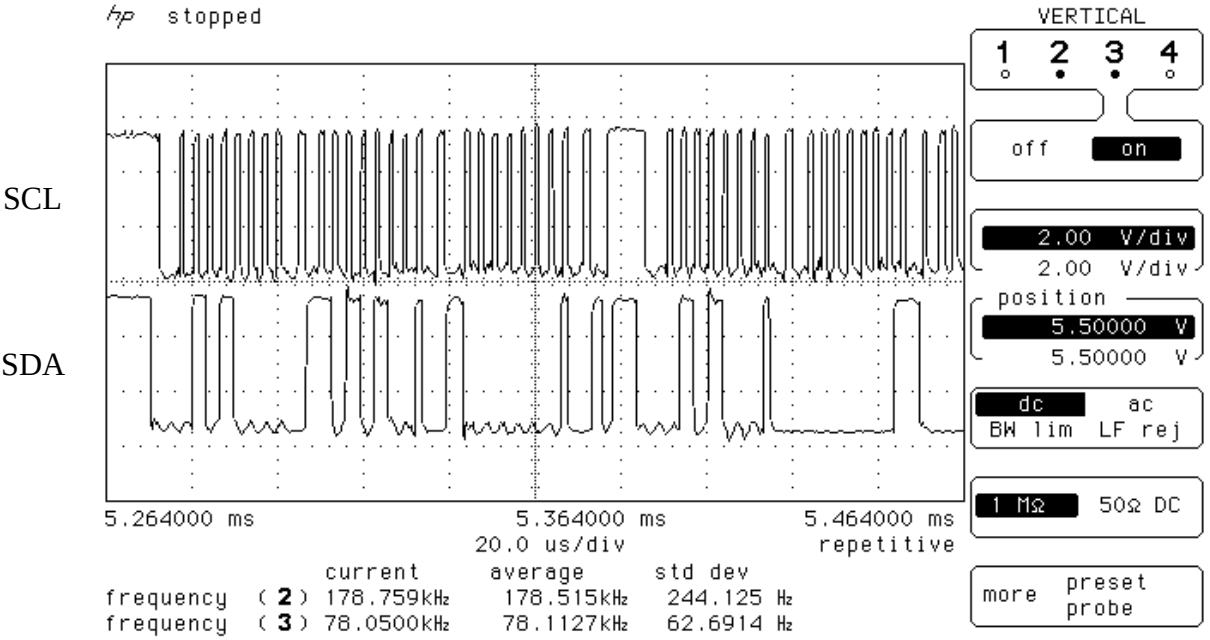

Figure 4 – A Write / Read Cycle As Seen On An Oscilloscope

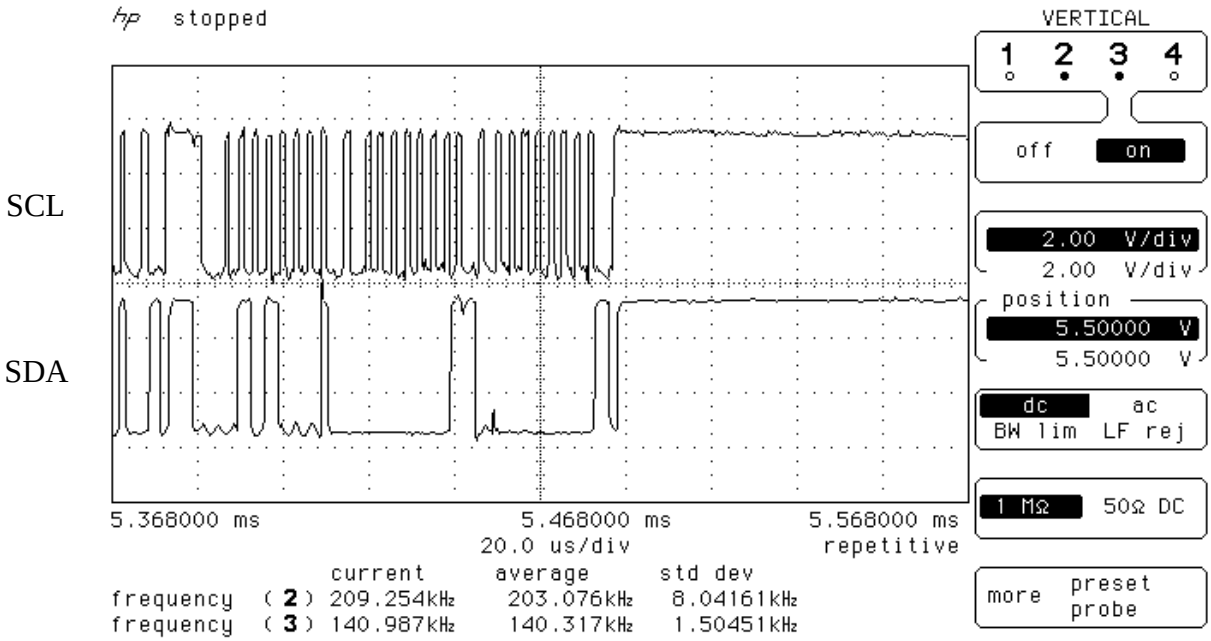

Figure 5 – Continuation of Figure 4, the time in milliseconds can be used to follow the screen shot

Dallas Semiconductor Contact Information

Address
4401 S. Beltwood Parkway
Dallas, TX 75244
Tel: 972-371-4448
Fax: 972-371-4799

Product Literature:
(972) 371-4448

Sales and Customer Service:
(972) 371-4969

World Wide Web Site:
www.dalsemi.com

Ordering Information:
www.dalsemi.com/products/ordering.pdf

FTP Site:
ftp://ftp.dalsemi.com

Datasheets:
www.dalsemi.com/datasheets/pdfindex.html

Package/Mechanical Drawings:
www.dalsemi.com/datasheets/mechdwg.html

Appendix A - Code Used to Generate 2-Wire For The DS1803 Using A DS87C520 Microprocessor

```
;*****************************************************************
;*                                                               *
;*  DS87C520 APPS DEVELOMENT SYSTEM                              *
;*                                                               *
;*  Application: Communication with DS1803                       *
;*                                                               *
;*****************************************************************
;*                                                               *
;*  This program was created to demonstrate using a 2-wire interface *
;*  to talk to a DS1803 Digital Potentiometer.  The program talks to *
;*  a PC using RS232 over Serial Port 1 (19200 Baud).  The micro is  *
;*  operated using DS1075 Oscillator operating at 22.2MHz. Serial    *
;*  Port 0 and most of the resources on the microcontroller are not  *
;*  being used at this time.                                     *
;*                                                               *
;*  Software Revision History                                    *
;*                                                               *
;*  1.0  01/09/01  - First try at operating the DS1803 using         *
;*                   the application's engineering generic 8051      *
;*                   board.                                      *
;*                                                               *
;*  Hardware Description                                         *
;*                                                               *
;*  P1.0 - SCL                   P3.0 - RXD0 - Not used              *
;*  P1.1 - SDA                   P3.1 - TXD0 - Not used              *
;*  P1.2 - RXD1 - PC             P3.2 -                              *
;*  P1.3 - TXD1 - PC             P3.3 -                              *
;*  P1.4 -                       P3.4 -                              *
;*  P1.5 -                       P3.2 -                              *
;*  P1.6 -                       P3.6 - WR\                          *
;*  P1.7 -                       P3.7 - RD\                          *
;*                                                               *
;*  P2.0 -                       P0.0 - SN74F373N                    *
;*  P2.1 -                       P0.1 -     "                        *
;*  P2.2 -                       P0.2 -     "                        *
;*  P2.3 -                       P0.3 -     "                        *
;*  P2.4 -                       P0.4 -     "                        *
;*  P2.5 -                       P0.5 -     "                        *
;*  P2.6 -                       P0.6 -     "                        *
;*  P2.7 -                       P0.7 -     "                        *
;*                                                               *
;*  Window 0 R0 - Used for 2-wire read and write, Do not destroy!    *
;*  Window 0 R3 - Used for binasc routine, Do not destroy!           *
;*                                                               *
;*****************************************************************
```

```
$include (c:\firmware\reg520.inc)  ; SFR register defs for compiler

;*************  Variable Declarations *************
stack    equ    02Fh          ; bottom of stack
                              ; stack starts at 30h
Pot0     equ    040h          ; Value Read from DS1803 for Pot0
Pot1     equ    041h          ; Value Read form DS1803 for Pot1

;*************    SFR Declarations    *************
smod_1    equ    0DFh         ; baud rate doubler bit declared
SCL       equ    90h          ; P1.0 is SCL
SDA       equ    91h          ; P1.1 is SDA

;*************  Constant Declarations *************
cmdWrite0 equ    0A9h         ; command for writing pot0
cmdWrite1 equ    0AAh         ; command for writing pot1
cmdWrt01  equ    0AFh         ; command to write 0&1 to same thing
cntRead   equ    51h          ; control byte, read pot 0 OR 0 & 1
                              ;  at addr=0
cntWrite  equ    50h          ; control byte, write pots at addr=0
```

```
;*********************************************************************
;*  Hardware Interrupt Vectors  (Table on page 95 of DS databook)    *
;*********************************************************************
;*  No Interrupts are enabled in this code.  If interrupts are to be *
;*    enabled they need to have the label initialized here.          *
;*********************************************************************

        org     0000h           ; Power up and Reset
        ljmp    start
        org     0003h           ; External Interrupt 0
        ljmp    start
        org     000Bh           ; Timer 0 Interrupt
        ljmp    start
        org     0013h           ; External Interrupt 1
        ljmp    start
        org     001Bh           ; Timer 1 Interrupt
        ljmp    start
        org     0023h           ; Serial Port 0 Interrupt
        ljmp    start
        org     002Bh           ; Timer 2 Interrupt
        ljmp    start
        org     0033h           ; PowerFail Interrupt (DS Priority 1)
        ljmp    start
        org     003Bh           ; Serial Port 1 Interrupt (DALLAS)
        ljmp    start
        org     0043h           ; External Interrupt 2 (DALLAS)
        ljmp    start
        org     004Bh           ; External Interrupt 3 (DALLAS)
        ljmp    start
        org     0053h           ; External Interrupt 4 (DALLAS)
        ljmp    start
        org     005Bh           ; External Interrupt 5 (DALLAS)
        ljmp    start
        org     0063h           ; Watchdog Interrupt (DALLAS)
        ljmp    start
        org     006Bh           ; Real-Time Clock (DALLAS)
        ljmp    start

;*********************************************************************
;****  Main Program                                                ****
;****   This program talks to a DS1803 using 2-wire, FastMode. It  ****
;****   writes to the part using all of the write modes, and it    ****
;****   reads the information in the part back each time.          ****
;*********************************************************************

;         Main         Program         starts         next         page
```

```
        org   0080h;
start:  clr   EA              ; Disable Interrupts
        lcall initSP1         ; Initialize Ser Port 1 & Timer 1/0
        lcall intro           ; Welcome Message, Serial Port 1
        lcall init2wire       ; Initialize 2-wire Variables

        lcall CMDReadPots     ; Read Data from both pots
        lcall DisplayPots     ; Display data read over Ser. Port 1

        lcall CMDWritePot0    ; Send Command for Write Pot 0
        mov   A, #01h         ; Load Data to send to Pot 0
        lcall WriteByte       ; Write Data in A to Pot 0
        lcall stop2wire       ; Send Stop Condition

        lcall CMDReadPots     ; Read Data from both pots
        lcall DisplayPots     ; Display data read over Ser. Port 1

        lcall CMDWritePot1    ; Send Command for Write Pot 1
        mov   A, #02h         ; Load Data into A to write to Pot1
        lcall WriteByte       ; Write Data
        lcall stop2wire       ; Send Stop Condition

        lcall CMDReadPots     ; Read Data From Both Pots
        lcall DisplayPots     ; Display Data Read over Ser. Port 1

        lcall CMDWritePot01   ; Send Command for Write Pot 0 and 1
        mov   A, #13h         ; Load data into A
        lcall WriteByte       ; Write data to both pots
        lcall stop2wire       ; Send Stop Condition

        lcall CMDReadPots     ; Read Data from both pots
        lcall DisplayPots     ; Display data read over Ser. Port 1

        lcall CMDWritePot0    ; Send command to write pot0
        mov   A, #24h         ; Load data into A
        lcall WriteByte       ; Write Data to Pot0
        mov   A, #25h         ; Load Data into A
        lcall WriteByte       ; Write Data to Pot1
        lcall stop2wire       ; Send Stop Condition

        lcall CMDReadPots     ; Read Data from both pots
        lcall DisplayPots     ; Display data read over Ser. Port 1

endmain:
        sjmp    endmain       ; Waits forever
```

```
;*********************************************************************
;****   2-Wire WriteByte Routine                                  ****
;****      Writes Data out to the part and reads the acknowledge  ****
;****      from the slave                                         ****
;*********************************************************************
;*    requires writebits and AckSlaveWrite routines                  *
;*    transmits data in ACC at time called                           *
;*********************************************************************

WriteByte:
     lcall writebits       ; Writes Data in A to DS1803
     lcall AckSlaveWrite   ; Checks for slave acknowledgment
     ret

;*********************************************************************
;****   2-Wire WritePot0 Command Routine                          ****
;****    Sends a start condition and writes the control and       ****
;****    command bytes out to the pot to write to pot0            ****
;*********************************************************************
;*    requires start2wire, writebits and AckSlaveWrite routines      *
;*********************************************************************

CMDWritePot0:
     lcall start2wire       ; send start condition
     mov   a, #cntWrite     ; load control byte into A
     lcall writebits        ; write control byte
     lcall AckSlaveWrite    ; check for slave acknowledge
     mov   a, #cmdWrite0    ; load command byte into A
     lcall writebits        ; write command byte
     lcall AckSlaveWrite    ; check for slave acknowledge
     ret

;*********************************************************************
;****   2-Wire WritePot1 Command Routine                          ****
;****    Sends a start condition and writes the control and       ****
;****    command bytes out to the pot to write to pot0            ****
;*********************************************************************
;*    requires start2wire, writebits and AckSlaveWrite routines      *
;*********************************************************************

CMDWritePot1:
     lcall start2wire      ; send start condition
     mov   a, #cntWrite    ; load control byte into A
     lcall writebits       ; write command byte
     lcall AckSlaveWrite   ; check for slave acknowledge
     mov   a, #cmdWrite1   ; load command byte into A
     lcall writebits       ; write command byte
     lcall AckSlaveWrite   ; check for slave acknowledge
     ret

;*********************************************************************
```

```
;****   2-Wire WritePot01 Command Routine                          ****
;****    Sends a start condition and writes the control and        ****
;****    command bytes out to the pot required to write the same   ****
;****    byte to both pot0 and pot1                                ****
;*************************************************************************
;*    requires start2wire, writebits and AckSlaveWrite routines          *
;*************************************************************************

CMDWritePot01:
     lcall start2wire      ; send start condition
     mov   a, #cntWrite    ; load control byte into A
     lcall writebits       ; write control byte
     lcall AckSlaveWrite   ; check for slave acknowledge
     mov   a, #cmdWrt01    ; load command byte into A
     lcall writebits       ; write command byte
     lcall AckSlaveWrite   ; check for slave acknowledge
     ret

;*************************************************************************
;****   2-Wire ReadPots Routine                                    ****
;****    Reads the Value of Pot0 and Pot1 and moves the data into  ****
;****    variables called Pot0 (40h) and Pot1 (41h)                ****
;*************************************************************************
;*    requires start2wire, writebits, readbits, AckSlaveWrite,           *
;*             AckSlaveRead, and stop2wire routines                      *
;*************************************************************************

CMDReadPots:
     lcall start2wire      ; send start condition
     mov   A, #cntread     ; load control byte
     lcall writebits       ; write command byte
     lcall AckSlaveWrite   ; check for slave acknowledge
     lcall readbits        ; read data byte from slave
     lcall AckSlaveRead    ; send slave acknowledge
     mov   Pot0, A         ; copy data read to RAM var Pot0
     lcall readbits        ; read data byte from slave

                           ; do not acknowledge slave when you
                           ; read the last byte of data during
                           ; a read sequence

     mov   Pot1, A         ; copy data read to RAM var Pot1
     lcall stop2wire       ; send stop condition
     ret
```

```
;*****************************************************************
;****   Display Pots Routine                                   ****
;****    Displays the value of Pot0 and Pot1 stored in the     ****
;****    variables of the same name using serial port 1        ****
;*****************************************************************
;*    requires binasc and outchar routines                       *
;*****************************************************************

DisplayPots:
     mov   A, Pot0       ; move data read from RAM @Pot0 to A
     lcall binasc        ; convert data from bin to ascii
     lcall outchar       ; send first byte via Ser. Port1
     mov   A, R3         ; move the second byte from conversion
                         ;  from R3 to A
     lcall outchar       ; send second byte via Ser. Port1
     mov   A, #0Dh       ; send Return via Ser. Port1
     lcall outchar
     mov   A, #0Ah       ; send Line Feed via Ser. Port1
     lcall outchar
     mov   A, Pot1       ; Move data read from RAM @Pot0 to A
     lcall binasc        ; convert data from bin to ascii

     lcall outchar       ; write first and second bytes
     mov   A, R3         ; and 2-CRs, 2-LFs out via Ser. Port1
     lcall outchar
     mov   A, #0Dh
     lcall outchar
     mov   A, #0Ah
     lcall outchar
     mov   A, #0Dh
     lcall outchar
     mov   A, #0Ah
     lcall outchar
     ret
```

```
;*********************************************************************
;****   Write Bits Routine                                        ****
;****     Serializes and Transmits the data in the Accumulator at ****
;****     the time the routine is called                            ****
;*********************************************************************
;*    requires no other routines                                     *
;*    Destroys Window 0 R0 register and ACC                          *
;*********************************************************************

writebits:
     mov   R0, #8             ; sets up for transfer of 8 bits
nextwritebit:
     rlc   A                  ; move the MSB of the ACC into C
     mov   SDA, C             ; write C onto SDA line
     setb  SCL                ; set SCL
     nop
     nop                      ; clock high time, 180ns/nop
     nop
     nop
     clr   SCL                ; clear SCL
     nop
     nop                      ; clock low time, 180ns/nop + other
     nop                      ; instructions between last nop and
     nop                      ; next setb SCL
     djnz  R0, nextwritebit   ; if the 8th data bit not sent yet
                              ;    then keep sending data
     ret
```

```
;*****************************************************************
;****   2-Wire Readbits Routine                                  ****
;****     Reads 8-bits of data from the slave device, and stores  ****
;****     the received data in the Accumulator                    ****
;*****************************************************************
;*    requires no other routines                                  *
;*    Destroys Window0 R0 register and ACC                        *
;*****************************************************************

readbits:
     setb  SDA          ; SDA must be set for an open
                        ;   collector read

     mov   R0, #8       ; sets up for transfer of 8 bits
nextreadbit:
     setb  SCL          ; set SCL
     nop                ; clock high time, 180ns/nop + other
     nop                ;   instructions before clr SCL
     mov   C, SDA       ; Place Data on SDA into C
     rlc   A            ; move the C into LSB of A
     clr   SCL          ; clear SCL
     nop
     nop                ; clock low time, 180ns/nop + other
     nop                ; instructions before next setb SCL
     nop
     nop
     djnz  R0, nextreadbit   ; if the 8th data bit not sent yet
                             ;   keep sending data
     ret
```

```
;*****************************************************************
;****   2-Wire Acknowledge Slave Routine for WRITES               ****
;****    Used to acknowledge slave devices DURING WRITES          ****
;*****************************************************************
;*    requires outstr routines                                   *
;*    uses DPTR register                                         *
;*****************************************************************

AckSlaveWrite:
     setb  SDA            ; set   SDA
     nop                  ; wait 180ns/nop
     nop
     setb SCL             ; set SCL
     nop
     nop                  ; wait 180ns/nop + other instructions
     nop                  ;   with clock high
     jb   SDA, Ack_fail   ; if SDA high (acknowledge fails),
                          ;   then jump to error routine
     clr  SCL             ;   else ack passes, set SCL and
     nop                  ; wait 180ns/nop + other instructions
     nop                  ;  for clock to go high
     ret
Ack_fail:
     mov   DPTR, #mess4   ; point to ack fail serial message
     lcall outstr         ; send message out
     clr   SCL            ; clr SCL
     clr   SDA            ; clr SDA
     nop
     nop                  ; clock time low, 180ns/nop + clr
     nop                  ;   SDA instruction
     nop
     nop
     nop
     setb  SCL            ; set SCL
     nop
     nop                  ; clock time high, 180ns/nop
     nop
     nop
     nop
     nop
     nop
     setb  SDA            ; create stop condition
     ret                  ; return to calling procedure
```

```
;*********************************************************************
;****   2-Wire Acknowledge Slave Routine for READS                 ****
;****    Used to acknowledge slave devices DURING READS            ****
;*********************************************************************
;*    requires no other routines                                     *
;*    uses no registers                                              *
;*********************************************************************

AckSlaveRead:
     clr   SDA      ; clear SDA
     nop            ; wait 180ns/nop
     nop
     setb  SCL      ; set SCL
     nop
     nop            ; wait 180ns/nop
     nop
     nop
     clr   SCL      ; clear SCL
     ret            ; return

;*********************************************************************
;****   Wait 2us Function                                          ****
;****    Wastes 1.6us of processor time with call, nop and return  ****
;*********************************************************************
;*    Requires no other routines or registers                        *
;*********************************************************************

wait2us:
     nop  ; 1 nops @4cc each + lcall @16cc + ret @16cc
          ; produces approximately 1.6us of delay with a
          ; 22.22MHz clock
     ret
```

```
;*****************************************************************
;****   2-Wire Start Condition Generator Routine                 ****
;****     Waits until the 2-Wire bus is not busy, the generates a ****
;****     a start condition.  Does not wait for the 2-Wire bus    ****
;****     free time, because this code is not intended to be used ****
;****     in a 2-wire multimaster system.                         ****
;*****************************************************************
;*    requires wait2us routine                                   *
;*    uses no registers                                          *
;*****************************************************************

start2wire:
     jnb   SCL, start2wire ; if SCL low, bus busy, wait
     jnb   SDA, start2wire ; if SDA low, bus busy, wait
     clr   SDA             ; start condition
     lcall wait2us         ; wait 2us
     clr   SCL             ; clear SDL
     lcall wait2us         ; wait 2us
     ret                   ; return to calling function

;*****************************************************************
;****   2-Wire Stop Condition                                   ****
;****    Used to send a stop condition                          ****
;*****************************************************************
;*    requires wait2us routine                                   *
;*    uses no registers                                          *
;*****************************************************************

stop2wire:
        clr   SDA      ; SDA must be low so it can go high while
                       ;    the clock is high to generate the
                       ;    stop condition
     nop               ; kill 180ns/nop, stop setup time
     nop
     nop
     nop
     setb  SCL         ; set clock
     nop
     nop
     nop
     nop
     setb  SDA         ; set SDA generating stop condition
     lcall wait2us
     ret
```

```
;*****************************************************************
;****   2-Wire Initialization Routine                          ****
;****    Inits SCL and SDA to Set Condition                    ****
;*****************************************************************
;*    requires no routines                                       *
;*    Uses no Registers                                          *
;*****************************************************************

init2wire:
     setb SCL             ; start program with SCL set
     setb SDA             ; start program with SDA set
     ret

;*****************************************************************
;****   Initialize Serial Port 1 for PC interface              ****
;****    Set up serial port 1 for use with a 22.1 MHz crystal  ****
;****    Uses timer 1 for 19200 baud, Mode 1                   ****
;*****************************************************************
;*    Uses no other routines or registers                        *
;*****************************************************************

initSP1:
        setb    smod_1       ;enable baud rate doubler
        mov     SCON1, #50h  ;Serial Port 0 asynch, 10 bits
        mov     TMOD,#21H    ;MSB-T1 on and in 8bit autoloadmode
                             ;LSB-T0 on and in 16-bit count mode
                             ;    T0 is free running 2^16cc
                              ; overflow rate (35.59ms)
        mov     TCON, #50H    ;t1/0 enabled, not using ext int
                              ;     edge/level select and detect
                              ;     flag/reg
        mov     TH1,  #0FAH   ;set t1 reset val / baud rate=19200
        ret
```

```
;*****************************************************************
;****   Intro Display Message Routine                          ****
;****    Sends out a greeting message                          ****
;*****************************************************************
;*    Uses outstr function                                       *
;*    Destroys DPTR                                              *
;*****************************************************************

intro:  mov     DPTR, #mess1
        lcall   outstr
        mov     DPTR, #mess2
        lcall   outstr
        mov    dptr, #mess3
        lcall   outstr
        mov     DPTR, #mess2
        lcall   outstr
        ret

;*****************************************************************
;****   Outstring Routine                                      ****
;****    writes a null terminated string to PC via Ser. Port 1 ****
;*****************************************************************
;*    Uses outchar routine                                       *
;*    Destroys dptr and A                                        *
;*****************************************************************

outstr: clr     A
        movc    A,@A+DPTR
        jz      exitstr
        lcall   outchar
        inc     dptr
        sjmp    outstr
exitstr:
        ret

;*****************************************************************
;****   Outchar routine                                        ****
;****    writes character in Acc to the PC via serial port 1   ****
;*****************************************************************
;*    Uses no routines or registers                              *
;*****************************************************************

outchar:
        mov     SBUF1,A
waitchar:jnb    SCON1.1, waitchar
        clr     SCON1.1
        ret
```

```
;*****************************************************************
;****  Binary to Ascii conversion routine                      ****
;****   Converts a binary number in Acc to 2 ascii digits      ****
;****   Leaves results in A (upper digit) and R3 (lower digit) ****
;*****************************************************************
;*    Uses no routines                                           *
;*    Destroys A and R3                                          *
;*****************************************************************
binasc:
     mov  R3, A       ; save number in R3
     anl  A, #0Fh     ; convert least significant digit
     add  A, #0F6h    ; adjust it
     jnc  noadj1      ; if a-f readjust
     add  A, #07h
noadj1:
        add  A, #3Ah  ; make ascii
     xch  A, R3       ; put result in reg2

     swap A
     anl  A, #0Fh     ; convert least significant digit
     add  A, #0F6h    ; adjust it
     jnc  noadj2      ; if a-f readjust
     add  A, #07h
noadj2:
     add  A, #3ah     ; make ascii
     ret


;*****************************************************************
;****   MESSAGES                                              ****
;*****************************************************************

org 8000h
mess1:  db  ' Jason''s Proto-board, Rev. 0.1',0Dh,0Ah
        db  ' Now uses DS1075 for a clock, DS1803',0Dh,0Ah
        db  ' added for 2-Wire Communication Demo.',0

mess2:  db  0Dh,0Ah,0DH,0AH,0

mess3:  db  ' This program talks via a 2-wire interface',0Dh,0Ah
        db  ' to a DS1803, and uses serial port 1 to',0Dh,0Ah
        db  ' communicate with the user',0Dh,0Ah, 0

mess4:  db  0Dh,0Ah,'Ack Fail', 0

 END   ;End of program
```

Appendix B – Practical Hardware Setup for a Bootstrap Loading Board to Communicate with a DS1803

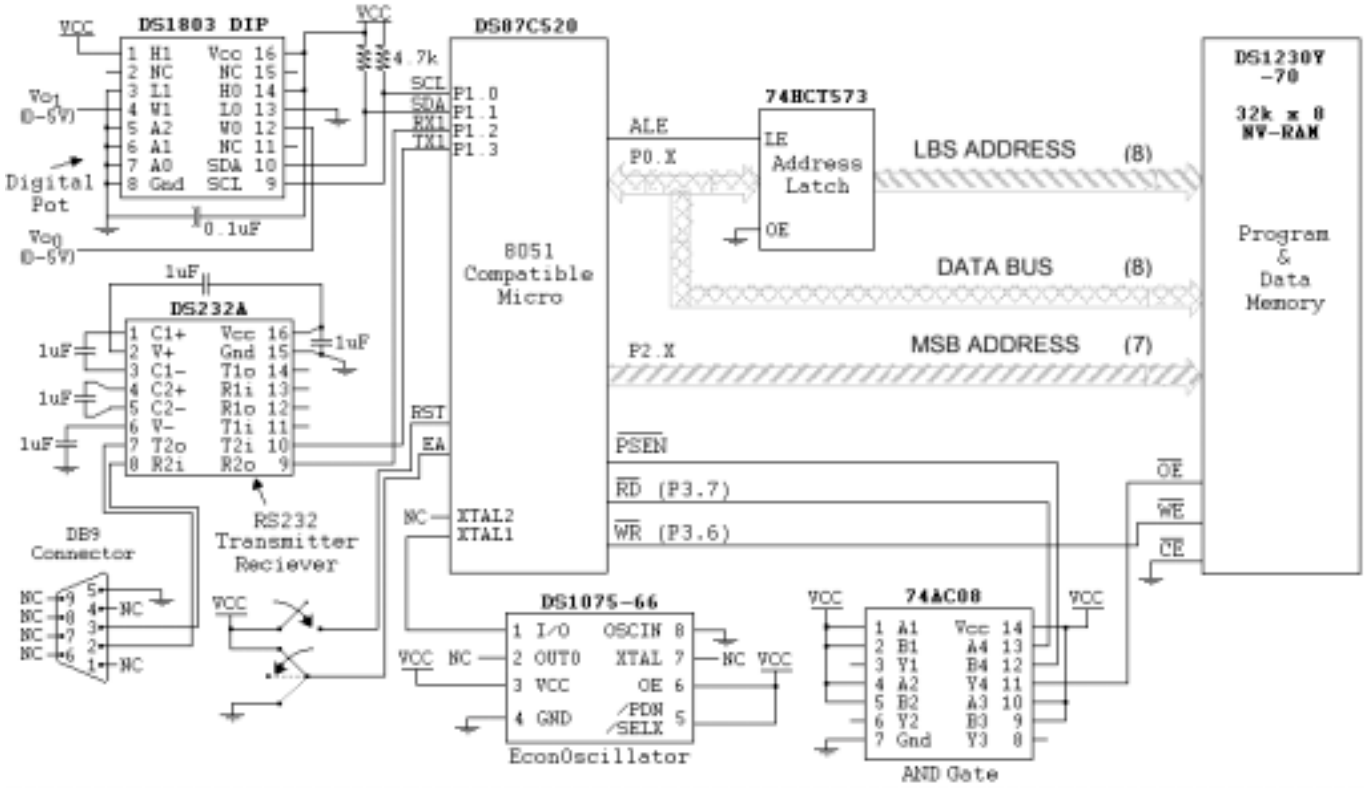

Figure 6 – Schematic of the Hardware Used to Communicate with the DS1803

The hardware setup shown in Fig. 6 was used to develop the code to communicate with the DS1803. This system is a bootstrap loader board. The board was built by Dallas Semiconductor because it promotes quick code development without sacrificing too much of the microcontroller's resources. All of ports 1 and 3 are available to the user with the exceptions of P3.6, P3.7, P1.2 and P1.3, which are used for NV-SRAM and serial port 1 access.

The DS87C520 has a bootstrap loader program loaded into its internal memory. When the microcontroller is reset, it will do one of two things. If the EA pin is connected to Vcc, the boot loader program in the internal EEPROM memory of the controller will transfer the data passed to it via serial port 1 to the NV RAM. The NV RAM will then save the data for execution at a later time. If the EA pin is grounded at the time of reset, the microcontroller will execute the code stored in the NV-RAM. Because the NV-RAM is being used as both the program memory and data memory, its OE pin has to be asserted when either the RD pin or the PSEN pin is active. This is accomplished by the AND gate (74AC08) because both signals are active low.

The DS232A chip is an RS232 transmitter/receiver chip. It accepts 5V power and ground, and generates its own ±12V supply. Once it has the ±12V supplies, it can accept and send ±12V signals to and from the RS232 terminal, and it translates them into standard 0-5V CMOS signals for the microprocessor. This allows a single 5V-power supply for the entire microprocessor board. The capacitors shown connected to this chip are required for the part to generate its own ±12V supply.

The DS1075-66 is an all-silicon oscillator chip. This chip has an internal oscillator that operates at 66.667 Mhz. It also contains a pre-scalar and a divider chain that can be used to slow the oscillator down by up to a factor of 2052. The oscillator chip provides a 22.22 MHz clock signal. This frequency was chosen because the original design incorporated a 22.118 MHz crystal to allow serial communication at 19200 baud for the bootstrap loader. An alternative to using this chip would be to use a 22.118 MHz crystal and two capacitors as shown in the Dallas Semiconductor High Speed Microprocessor User's Guide (available online at www.dalsemi.com).

The SCL and SDA lines of the DS1803 are connected to P1.0 and P1.1 respectively. Both potentiometers are connected in a voltage divider configuration; therefore, their output is a 0-5V signal. When the board is first powered up, the output will be 0V because the DS1803 contains volatile memory.

Appendix C – Output of DS1803 Program if the DS1803 Was Powered Down Before Operation

```
 Jason's Proto-board, Rev. 0.1
 Now uses DS1075 for a clock, DS1803
 added for 2-Wire Communication Demo.

 This program talks via a 2-wire interface
 to a DS1803, and uses serial port 1 to
 communicate with the user


00
00

01
00

01
02

13
13

24
25
```