Application Note 516
Using Multiplexed Bus Clocks with a Microcontroller

www.maxim-ic.com

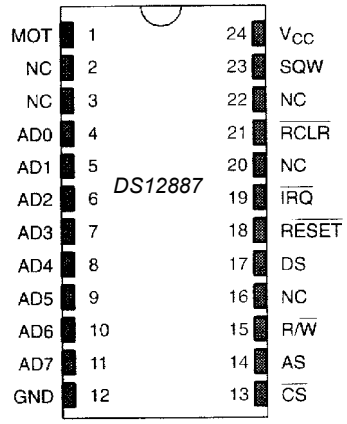
INTRODUCTION

The multiplexed bus family of real-time clock (RTC) products provide time and date functions, plus other features such as battery-backed NV SRAM and system power control functions. The multiplexed-bus RTCs use eight I/O pins to pass address information and data between the RTC and the bus, thus reducing the pin count. Multiplexed bus RTCs are best suited for use with microcontrollers or microprocessors that support the multiplexed bus interface. An example of this is the 8051 microcontroller.

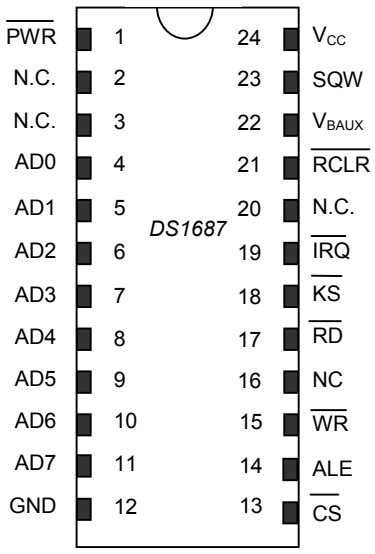
In this example, a DS12887, DS1687, or DS17287 is connected to a DS87C320. The DS87C320 high-speed microcontroller is compatible with the industry-standard 8051 architecture. For related application notes, please refer to the DS87C320 and the DS1687 Quick View data sheets.

PIN CONFIGURATIONS

TOP VIEW



PDIP



PDIP

```
/*********************************************************************/
/* DSMux.c This program uses a 8051-type microcontroller to interface */
/* with a multiplexed-bus RTC such as a DS12887, DS1687 or DS17887.   */
/* This is an example program only and is not supported by Dallas     */
/* Semiconductor Maxim                                                */
/* Revision 1.0 2/19/03                                               */
/*********************************************************************/
#pragma code symbols debug
#include <stdio.h>            /* Prototypes for I/O functions         */
#include <DS5000.h>           /* Register declarations for DS5000     */
#include <absacc.h>           /* needed to define xdata addresses     */
#define RDADDR    XBYTE[0x0004]
#define WR1ADDR   XBYTE[0x0001]
#define WR0ADDR   XBYTE[0x0000]
/************** definitions unique to high-speed micros  **************/
sfr CKCON = 0x8e;
/***************************** Defines ********************************/
#define     CLK_SECS    XBYTE[0x0000]     /* time, date & alarm regs */
#define     CLK_SECS_ALM      XBYTE[0x0001]
#define     CLK_MINS    XBYTE[0x0002]
#define     CLK_MINS_ALM      XBYTE[0x0003]
#define     CLK_HRS     XBYTE[0x0004]
#define     CLK_HRS_ALM XBYTE[0x0005]
#define     CLK_DOW     XBYTE[0x0006]
#define     CLK_DOM     XBYTE[0x0007]
#define     CLK_MON     XBYTE[0x0008]
#define     CLK_YR      XBYTE[0x0009]
#define     REGA        XBYTE[0x000a]     /* control registers */
#define     REGB        XBYTE[0x000b]
#define     REGC        XBYTE[0x000c]
#define     REGD        XBYTE[0x000d]
#define     REG4A       XBYTE[0x004a]
#define     REG4B       XBYTE[0x004b]
#define     RS2         0x0f  /* bits for sqw/periodic int rate select */
#define     RS4         0x0e
#define     RS8         0x0d
#define     RS16        0x0c
#define     RS32        0x0b
#define     RS64        0x0a
#define     RS128       0x09
#define     RS256       0x08
#define     RS512       0x07
#define     RS1024      0x06
#define     RS2048      0x05
#define     RS4096      0x04
#define     RS8192      0x03
#define     VRT         0x80  /* battery low bit */
#define     E32K        0x40  /* reg 4B */
#define     DAYS        "Sun\0Mon\0Tue\0Wed\0Thu\0Fri\0Sat\0"
#define     DVX         0x01  /* bits for error message */
#define     C887        0x02
#define     VRT_SET     0x04
#define     VRT2_SET    0x08
#define     VLDMDL      0x10
#define     CRCSEROK    0x20
/************************ Global Variables ***************************/
uchar model = -99, int_flg = 0;
/********************** Function Prototypes **************************/
void  ext0_int(void);
void  writereg();
```

```
uchar updcrc(uchar, uchar);
void  model_num();
void  init_rtc();
void kickstart();
void init_alrm();
void disp_ctrl_regs();
void UIP_clk_rd();
void PIE_clk_rd();
void UIE_clk_rd();
void set_time();
void BurstRamRead();
void BurstRamWrite();
/*********************************************************************/
void writereg()   /* -------- write data to specified register --------- */
{
uchar Add, Data;

   printf("\nEnter Address (hex): ");        /* Get Address & Data */
   scanf("%bx", &Add);
   printf("DATA (hex): ");
   scanf("%bx", &Data);

   XBYTE[Add] = Data;
}
uchar updcrc(uchar crc, uchar val)  /* ---------- calculate 8-bit CRC ---------- */
{
uchar inc, tmp;

      for(inc = 0; inc < 8; inc++)
      {
            tmp = crc << 7;         /* save X7 bit value */
            crc >>= 1;              /* shift crc */
            if( (tmp >> 7) ^  (val & 0x01) )    /* if X7 xor X8 (input data) */
            {
                  crc ^= 0x8c;            /* xor crc with X4 and X5, X1 = X7^X8 */
                  crc |= 0x80;            /* carry */
            }
            val >>= 1;
      }
      return      crc;
}
void  model_num() /* ------ determine the model number of the RTC ------ */
{
      REGA |= 0x10;           /* select bank 1 */
      XBYTE[0x40] ^= 0xff;    /* try to write to compliment data */
      if( (XBYTE[0x40] ^ 0xff) != XBYTE[0x40] ) /* if bits cannot be written, it's
bank 1 */
      {
            model = XBYTE[0x40];
      }
      else
      {
            XBYTE[0x40] ^= 0xff;    /* restore original data */
            model = 0;        /* default if device does not support model number */
      }
      REGA &= 0xef;           /* select bank 0 */

      printf("\nModel: ");
      switch(model)
      {
```

```
            case 0x71:  printf("DS1687");       break;
            case 0x72:  printf("DS17287");      break;
            case 0x73:  printf("DS1691/93");    break;
            case 0x74:  printf("DS17487");      break;
            case 0x78:  printf("DS17887");      break;
            default:    printf("DS12887");
      }
}
void  init_rtc()  /* ----- id the model of the RTC and set the registers ----- */
{
uchar addr, crc = 0, initstat = 0;

      if( (REGA & 0x60) != 0x20)    /* DV2=0 & DV1=1 for the RTC to keep time */
            initstat = DVX;         /*  if they're not, assume we need to initialize
the RTC */

      if( !(REGD & 0x80) ) initstat += VRT_SET; /* if VRT bit is set, warn that RTC
is N/G */

      model_num();      /* get the model number, if it exists */

      if(model > 0)     /* will be 0 if part does not support bank 1 */
      {
            REGA |= 0x10;     /* switch to bank 1 */

            /* model number must be between 71h & 78h */
            if( XBYTE[0x40] > 0x70 & XBYTE[0x40] < 0x79 )   initstat += VLDMDL;

            crc = updcrc(0, XBYTE[0x40]);       /* calculate CRC */
            crc = updcrc(crc, XBYTE[0x41]);     /* 48 bit unique # */
            crc = updcrc(crc, XBYTE[0x42]);
            crc = updcrc(crc, XBYTE[0x43]);
            crc = updcrc(crc, XBYTE[0x44]);
            crc = updcrc(crc, XBYTE[0x45]);
            crc = updcrc(crc, XBYTE[0x46]);
            /* if CRCs do not match, then it cannot be an RTC with a bank 1 (serial
number) */
            /* printf("\ncalc CRC:%2.bx readcrc:%2.bx", crc, XBYTE[0x47] ); */
            if (crc == XBYTE[0x47] && crc)      initstat += CRCSEROK;   /*  crc  must
match and be non-zero */

            if( !(REGD & 0x80) ) initstat += VRT2_SET;      /* check VRT bit */

            REGA &= 0xef;     /* switch to bank 0 */
      }

      if( initstat & VRT_SET )
      {
            printf("\nWarning: RTC battery low - invalid RTC data");
            /* specific routines to terminate or work-around non-functional RTC goes
here */
      }
      if( initstat & VRT2_SET )
      {
            printf("\nWarning: RTC Aux battery low");
            /* specific routines to terminate or work-around non-functional RTC goes
here */
      }
      if(initstat & DVX)      /* if osc is not running */
      {
```

```
            printf("\nInitializing RTC...");

            if(initstat & DVX)      /* if osc was disabled, start it */
            {
                  printf("\nStarting osc...");
                  REGA |= 0x20;     /* enable osc */
                  REGA &= 0xA0;     /*  enable  countdown  chain,  clear  rate  select
bits */
                  REGA |= RS2;      /* set for 2Hz */

                  /* Assume we need to initialize the clock, pick a mode...
                     Clear SET bit, disable PIE, AIE, UIE & SQWE */
                  REGB = 0x03;      /* 24 hr BCD mode, Daylight Savings Enabled */

                  if(model)
                  {
                        /*  and  assume  we  need  to  initialize  the  other  control
registers */
                        REGA |= 0x10;     /* select bank 1 */
                        REG4A = 0x08;     /* PAB: power is active, clear KF bit */
                        REG4B = 0x00;     /* disable everything */
                        REGA &= 0xef;     /* select bank 0 */
                  }
            }

            REGA |= 0x10;           /* back to bank 1 */

            printf("\ninitstat:%2.bx", initstat);
            if( (initstat & 0x30) == (VLDMDL | CRCSEROK) )  /*  if  RTC  has  extended
features */
            {
                  printf("\nInitializing bank 1...");
                  REGA = (0x20 | 0x10);   /* select bank 1 */
                  if(!model)  model = XBYTE[0x40];    /* save model number */
                  REG4A = 0x08;                       /* PAB: power is active, clear
KF bit */
                  REG4B = 0;                          /* disable everything */

                  /************ use model byte to clear extended RAM here if needed
**********/

            }
            REGA &= 0xef;     /* go to bank 0 */
      }
}
void kickstart()        /* --------- test the kickstart function --------- */
{
int   inc;

      if(!model)  return;     /* if device doesn't support kickstart, why bother? */

      REGA |= 0x10;     /* select bank 1 */
      REG4A = 0;        /* PAB: power is active, clear KF bit */
      REG4B = 0x81;     /* aux battery enable, Kickstart interrupt enable */

      REG4A = 0x08;     /***** power down the system by setting PAB to a 1 *****/
      /* system will power down as code continues to execute -- not necessarily a
good practice */
      REGA &= 0xef;     /* select bank 0 */
}
```

```
void init_alrm()  /* --------- set the alarm time & masks -------------- */
/* Note: NO error checking is done on the user entries! Assumes 24hr BCD mode */
{
uchar hr, min, sec, val;

      REGC; /* clear flags */
      printf("\nAlarm hour (1-23, C0-FF=Don't care): ");
      scanf("%bx", &hr);
      printf("Alarm minute (0-59, C0-FF=Don't care): ");
      scanf("%bx", &min);
      printf("Alarm second (0-59, C0-FF=Don't care): ");
      scanf("%bx", &sec);

      CLK_SECS_ALM = sec;
      CLK_MINS_ALM = min;
      CLK_HRS_ALM = hr;
      val = REGA;
      val = val | 0x20; /* set AIE */
      REGA = val;
}
void disp_ctrl_regs()         /* ---------------------------------------- */
{
uchar oldval;

      oldval = REGA;
      printf("\nCRA:  %02.bx  CRB: %02.bx", REGA, REGB );
      printf("\nCRC:  %02.bx  CRD: %02.bx", REGC, REGD );
      REGA = (oldval | 0x10); /* select bank 1 */
      printf("\nCR4A: %02.bx CR4B: %02.bx", REG4A, REG4B );
      REGA = oldval;    /* select bank 0 */
}
void disp_clk_regs()          /* ---------------------------------------- */
{
      if(REGB & 0x04)   /* Binary data */
      {
            printf("\nDUT %02.bd  %02.bd  %02.bd  %02.bd  %02.bd:%02.bd:%02.bd ",
                  CLK_YR, CLK_MON, CLK_DOM, CLK_DOW, (CLK_HRS & 0x7f), CLK_MINS,
                  CLK_SECS);
            if(!(REGB & 2) )
            {
                  if(CLK_HRS & 0x80)
                        printf("PM ");
                  else
                        printf("AM ");
            }
            printf("%3s", DAYS + (CLK_DOW - 1)*4 );
      }
      else              /* BCD data mode */
      {
            printf("\nDUT %02.bX  %02.bX  %02.bX  %02.bX  %02.bX:%02.bX:%02.bX ",
                  CLK_YR, CLK_MON, CLK_DOM, CLK_DOW, (CLK_HRS & 0x7f), CLK_MINS,
                  CLK_SECS);
            if(!(REGB & 2) )
            {
                  if(CLK_HRS & 0x80)
                        printf("PM ");
                  else
                        printf("AM ");
            }
            printf("%3s", DAYS + (CLK_DOW - 1)*4 );
```

```
      }

}
void UIP_clk_rd() /* ------ display time using UIP ------ */
{
/* This routine shows how reading the clock might be implemented using the
   UIP bit.  In this case, the uC could perform other tasks, and periodically
   execute the code to read the clock.  The code would check the state of
   the UIP bit, and wait if UIP is set.  if UIP is low, you have at least
   244us to read the time and date registers */

uchar prv_sec = 99;

      EX0 = 1;    /* enable interrupt 0 */
      IT0 = 1;    /* edge activated */
      PX0 = 0;    /* low priority */
      EX1 = 0;    /* disable interrupt 1 */
      EA = 1;     /* enable all interrupts */

      printf("\n    Yr  Mon Dte Day Hr:Mn:Sec");
      while(!RI)  /* Read & Display Clock Registers */
      {
            while(REGA & 0x80);     /* if UIP is set, wait until it's not */

            if(CLK_SECS != prv_sec) /* display every time seconds change */
            {
                  disp_clk_regs();
            }
            prv_sec = CLK_SECS;
      }
      RI = 0;  /* Swallow keypress to exit loop */
}
void PIE_clk_rd() /* ------ display time using interrupt ------ */
{
/* This routine shows how reading the clock might be implemented using
   the update-ended interrupt.  In this case, the uC may perform other
   tasks.  When an interrupt occurs, the clock is read and displayed.
   You have about 999ms to complete the task */

uchar stat;

      printf("\n    Yr  Mon Dte Day Hr:Mn:Sec");

      EX0 = 1;    /* enable interrupt 0 */
      IT0 = 1;    /* edge activated */
      PX0 = 0;    /* low priority */
      EX1 = 0;    /* disable interrupt 1 */
      EA = 1;     /* enable all interrupts */

      REGA |= 0x0f;     /* set RS3-0 to 2Hz */
      REGB |= 0x40;     /* set PIE (Periodic Interrupt Enable) bit */
      while(!RI)        /* Read & Display Clock Registers */
      {
            /* a real application would execute other code instead of waiting for an
interrupt */
            while(!( (stat = REGC) & 0x80) );   /* wait for an interrupt */

            if(stat & 0x40)   /* only display if interrupt occured because of PF */
            {
```

```
                  EX0 = EX1 = 0;    /* disable interrupts while updating the display
*/
                  disp_clk_regs();
                  EX0 = EX1 = 1;    /* re-enable interrupts */
            }
      }
      EX0 = 0;    /* disable interrupt */
      REGB &= 0xbf;     /* clear PIE bit */
      RI = 0;  /* Swallow keypress to exit loop */
}
void UIE_clk_rd() /* ------ display time using interrupt ------ */
{
/* This routine shows how reading the clock might be implemented using
   the periodic interrupt.  In this case, the uC may perform other
   tasks.  When an interrupt occurs, the clock is read and displayed.
   The periodic interrupt rate should be longer than 244us to insure
   that an interrupt does not occur while UIP is high.  The data should
   be read within (tpi/2 + tbuc) to insure that the data is not read
   during an update.  This example sets the interrupt rate to 2Hz, so
   the time and date are displayed twice per second. */

uchar stat;

      printf("\n    Yr  Mon Dte Day Hr:Mn:Sec");

      EX0 = 1;    /* enable interrupt 0 */
      IT0 = 1;    /* edge activated */
      PX0 = 0;    /* low priority */
      EX1 = 0;    /* disable interrupt 1 */
      EA = 1;     /* enable all interrupts */

      REGB |= 0x10;     /* set UIE (Update Ended Interrupt Enable) bit */
      while(!RI)        /* Read & Display Clock Registers */
      {
            /* a real application would execute code instead of waiting for an
interrupt */
            while(!int_flg);  /* wait for interrupt */

            while(!( (stat = int_flg) & 0x80) );

            if(stat & 0x10)   /* only display if interrupt occured because of UF */
            {
                  EX0 = EX1 = 0;    /* disable interrupts while updating the display
*/
                  disp_clk_regs();
                  int_flg = 0;      /* reset interrupt flag */
                  EX0 = EX1 = 1;    /* re-enable interrupts */
            }
      }
      EX0 = 0;    /* disable interrupt */
      REGB &= 0xef;     /* clear UIE bit */
      RI = 0;  /* Swallow keypress to exit loop */
}
void set_time()   /* ------------ set the time and date ----------------- */
/* Note: NO error checking is done on the user entries! */
{
uchar yr, mn, dt, dy, hr, min, sec;

      if(REGB & 0x04)   /* Binary data */
      {
```

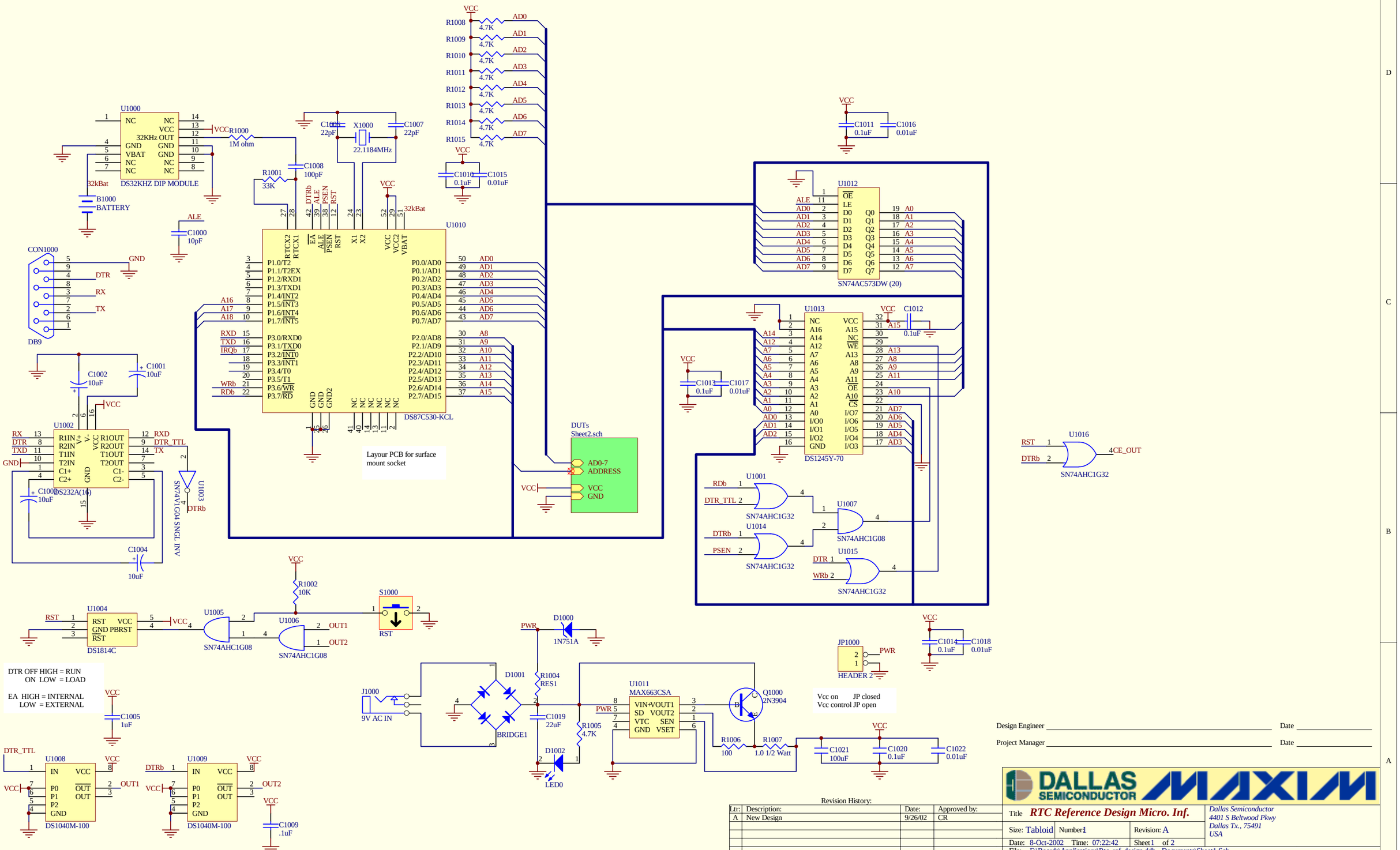
```
        printf("\nEnter the year (0-99): ");
        scanf("%bd", &yr);
        printf("Enter the month (1-12): ");
        scanf("%bd", &mn);
        printf("Enter the date (1-31): ");
        scanf("%bd", &dt);
        printf("Enter the day (1-7): ");
        scanf("%bd", &dy);

        if(REGB & 2)      /* if 24 hour mode */
        {
              printf("Enter the hour (1-23): ");
              scanf("%bd", &hr);
        }
        else
        {
              printf("Enter the hour (1-11): ");
              scanf("%bd", &hr);

              printf("A)M or P)M (A/P) " );
              scanf("%1bs", &min);

              if(min == 'P' || min == 'p')
                    hr |= 0x80; /* add PM indicator */
        }

        printf("Enter the minute (0-59): ");
        scanf("%bd", &min);
        printf("Enter the second (0-59): ");
        scanf("%bd", &sec);
  }
  else              /* BCD data mode */
  {
        printf("\nEnter the year (0-99): ");
        scanf("%bx", &yr);
        printf("Enter the month (1-12): ");
        scanf("%bx", &mn);
        printf("Enter the date (1-31): ");
        scanf("%bx", &dt);
        printf("Enter the day (1-7): ");
        scanf("%bx", &dy);

        if(REGB & 2)      /* if 24 hour mode */
        {
              printf("Enter the hour (1-23: ");
              scanf("%bx", &hr);
        }
        else
        {
              printf("Enter the hour (1-11:) ");
              scanf("%bx", &hr);
              printf("A)M or P)M (A/P) ");
              scanf("%1bs", &min);

              if(min == 'P' || min == 'p')
                    hr |= 0x80; /* add PM indicator */
        }

        printf("\nEnter the minute (0-59): ");
        scanf("%bx", &min);
```

```
            printf("Enter the second (0-59): ");
            scanf("%bx", &sec);
      }

      REGB |= 0x80;     /* inhibit update while writing to clock */
      CLK_SECS = sec;
      CLK_MINS = min;
      CLK_HRS = hr;
      CLK_DOW = dy;
      CLK_DOM = dt;
      CLK_MON = mn;
      CLK_YR = yr;
      REGB &= 0x7f;     /* allow update of buffered set of time & date registers */
}
void BurstRamRead()           /* --------- display contents of RAM -------- */
{
unsigned int j, maxadd;
uchar tmp;

      printf("\nBank 0");
      for (j = 0; j <= 0x7f; ++j)
      {
            if(!(j % 16)) printf("\n%02.x ", j);
            printf("%02.bx ", XBYTE[j]);
      }
      printf("Press any key ");
      _getkey();

      REGA |= 0x10;           /* select bank 1 */
      printf("\nBank 1");
      for (j = 0; j <= 0x7f; ++j)
      {
            if(!(j % 16)) printf("\n%02.x ", j);
            printf("%02.bx ", XBYTE[j]);
      }

      tmp = updcrc(0, XBYTE[0x40] );      /* calculate and display CRC */
      tmp = updcrc(tmp, XBYTE[0x41] );
      tmp = updcrc(tmp, XBYTE[0x42] );
      tmp = updcrc(tmp, XBYTE[0x43] );
      tmp = updcrc(tmp, XBYTE[0x44] );
      tmp = updcrc(tmp, XBYTE[0x45] );
      tmp = updcrc(tmp, XBYTE[0x46] );
      printf("\ncrc byte: %2.bx calc crc: %02.bx ", XBYTE[0x47], tmp);

      printf("Press any key ");
      _getkey();

      switch(model)
      {
            case 0x71:  maxadd = 127;     break;
            case 0x72:  maxadd = 2047;    break;
            case 0x74:  maxadd = 4095;    break;
            case 0x78:  maxadd = 8191;    break;
            default:    maxadd = 0;
      }

      if(maxadd == 0)   return;     /* if RTC has no extended RAM, exit */

      printf("\nExtended RAM");
```

```
      if(model < 0)     model_num();      /* must know model to determine EX RAM
size */

      for (j = 0; j <= maxadd; ++j)
      {
            if(!(j % 0x80) && (j != 0) )
            {
                  printf("\nPress a key or Esc to exit ");
                  if( _getkey() == 0x1b)  break;
            }
            if(!(j % 16)) printf("\n%04.x ", j);
            XBYTE[0x50] = (uchar) j;            /* LSB of extended RAM */
            XBYTE[0x51] = (uchar) (j >> 8);     /* MSB of extended RAM */
            printf("%02.bx ", XBYTE[0x53]);
      }
      REGA &= 0xef;           /* select bank 0 */
}
void BurstRamWrite()          /* --- fill each bank with unique data --- */
{
uchar offset = 0;
unsigned int j, maxadd;

      for (j = 0x0e; j <= 0x7f; ++j)
      {
            XBYTE[j] = 0xaa;
      }

      REGA |= 0x10;           /* select bank 1 */

      for (j = 0x0e; j <= 0x3f; ++j)
      {
            XBYTE[j] = 0x55;
      }

      if(model < 0)     model_num();      /* must know model to determine EX RAM
size */

      switch(model)
      {
            case 0x71:  maxadd = 127;     break;
            case 0x72:  maxadd = 2047;    break;
            case 0x74:  maxadd = 4095;    break;
            case 0x78:  maxadd = 8191;    break;
            default:    maxadd = 0;
      }
      if(maxadd == 0)   return;     /* if no extended RAM, exit */

      for (j = 0; j <= maxadd; ++j)
      {
            XBYTE[0x50] = (uchar) j;      /* set address of extended ram to read
from */
            XBYTE[0x51] = (uchar) (j >> 8);
            XBYTE[0x53] = j + offset;
            if( (j & 0xff) == 0xff) offset += 1;
      }

      REGA &= 0xef;           /* select bank 0 */

}
```

```
void  external0_int(void) interrupt 0     /* --- display time/date on interrupt from
RTC --- */
{
      EX0 = EX1 = 0;    /* disable interrupt */
      int_flg = REGC;   /* clear the interrupt source, save the info */
      EX0 = EX1 = 1;    /* re-enable interrupt */
}
main (void)       /* ------------------------------------------------------ */
{
uchar M, M1;

      CKCON &= 0xf8;    /* set data memory cycle stretch value to 0 (2 clk strobe
width) */

      if(model < 0)     model_num();

      while (1)
      {
            printf("\nDSmux--DS12887/DS1687/DS17887 Ver 1.0\n");
            printf("CI Init RTC   CP PIE clk rd\n");
            printf("CU UIP clk rd CS Clock set\n");
            printf("CE uiE clk rd K  KS test\n");
            printf("RW Write RAM  RR Read RAM\n");
            printf("ER  Reg Read  EW Reg Write\n");
            printf("A  Set alarm \n");
            printf("Enter Menu Selection:");

            M = _getkey();

            switch(M)
            {
                  case 'A':
                  case 'a':   init_alrm();      break;

                  case 'C':
                  case 'c':
                  printf("\rEnter Clock Routine to run: C");
                  M1 = _getkey();
                  switch(M1)
                  {
                        case 'I':
                        case 'i':   init_rtc(); break;

                        case 'E':
                        case 'e':   UIE_clk_rd();     break;

                        case 'P':
                        case 'p':   PIE_clk_rd();     break;

                        case 'S':
                        case 's':   set_time(); break;

                        case 'U':
                        case 'u':   UIP_clk_rd();     break;
                  }
                  break;

                  case 'E':
                  case 'e':
                  printf("\rEnter Register routine to run: E");
```

```
                M1 = _getkey();
                switch(M1)
                {
                      case 'R':
                      case 'r':   disp_ctrl_regs(); break;

                      case 'W':
                      case 'w':   writereg(); break;
                }
                break;

                case 'K':
                case 'k':   kickstart();      break;

                case 'R':
                case 'r':
                printf("\rEnter Ram Routine to run: R");
                M1 = _getkey();

                switch(M1)
                {
                      case 'C':
                      case 'c':   break;

                      case 'R':
                      case 'r':   BurstRamRead();   break;

                      case 'W':
                      case 'w':   BurstRamWrite();  break;
                }
                break;
          }     /* end switch */
      }     /* end while */
}
```



Design Engineer _____ Date _____

Project Manager _____ Date _____

Revision History:

Ltr:	Description:	Date:	Approved by:
A	New Design	9/26/02	CR

DALLAS SEMICONDUCTOR MAXIM

Title **RTC Reference Design Micro. Inf.**

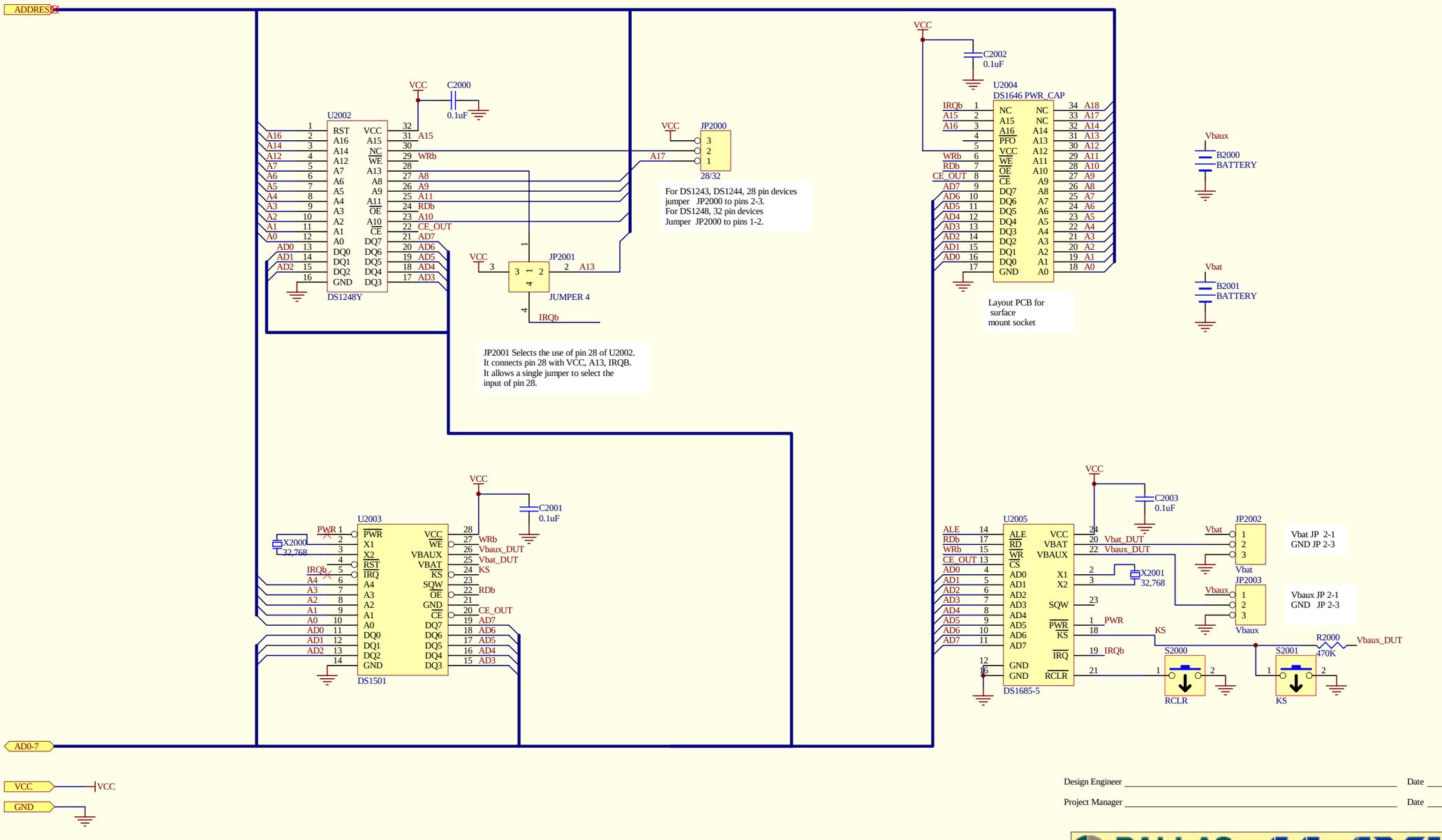
Size: Tabloid | Number 1 | Revision: A

Date: 8-Oct-2002 Time: 07:22:42 | Sheet 1 of 2

File: F:\Boards\Applications\Rtc_ref_design.ddb - Documents\Sheet1.Sch

Dallas Semiconductor
4401 S Beltwood Pkwy
Dallas Tx., 75491
USA

Note: Populate U2002, U2003, U2004 or U2005 as needed.



Design Engineer _____ Date _____

Project Manager _____ Date _____

Revision History:

Ltr:	Description:	Date:	Approved by:
A	New Design	9/26/02	CR

DALLAS SEMICONDUCTOR MAXIM

Title *RTC Reference Design - R.T.C. Inf.*

Size: Tabloid | Number1 | Revision: A

Date: 8-Oct-2002 Time: 07:22:44 | Sheet 2 of 2

File: F:\Boards\Applications\Rtc_ref_design.ddb - Documents\Sheet2.sch

Dallas Semiconductor
4401 S Beltwood Pkwy
Dallas Tx., 75491
USA