```
/*****************************************************************************/
/* DEMO1678.C - This file is provided to show an example of communication  */
/* routines for interfacing to the DS1678.  These routines are provided    */
/* for example only and are not supported by Maxim                         */
/*****************************************************************************/
#include <stdio.h>            /* Prototypes for I/O functions        */
#include <DS5000.h>           /* Register declarations for DS5000     */

#define     ACK   0
#define     NACK  1
#define     ADDRTC      0x94  /* slave addresses */
sbit scl = P0^0;              /* I2C pin definitions */
sbit sda = P0^1;
sbit INTb = P0^2;
/* function definitions */
void  start();
void  stop();
void  I2Cwrite(uchar);
uchar I2Cread(uchar);
void  readbyte();
void  writebyte();
void  int_test();
void  initialize();
void  disp_clk_regs(uchar);
void  burstramread();
void  burstramwrite();
void  mission();

/* global variables */

void start()            /* -------------- I2C start ------------- */
{
  sda = 1;  scl = 1;    /* Initiate start condition */
  sda = 0;
}
void stop()       /* -------------- I2C stop -------------- */
{
  sda = 0;  sda = 0;  sda = 0;  sda = 0;  /* Initiate stop condition */
  scl = 1;  scl = 1;  sda = 1;
}
void I2Cwrite(uchar d)        /* ------ write one byte -------- */
{
char i;

      scl = 0;
      for (i = 1; i <= 8; i++)
      {
            sda = (d >> 7);
            scl = 1;
            d = d << 1;
            scl = 0;
      }
      sda = 1;    /* Release the sda line */
      scl = 0;
      scl = 1;
      if (sda) printf("Ack bit missing  %02X\n",(unsigned int)d);
      scl = 0;
```

```
}
uchar I2Cread(uchar b)  /* ------ read one byte -------- */
{
char i;
uchar d;

      sda = 1;             /* Let go of sda line */
      scl = 0;
      for (i = 1; i <= 8; i++)      /* read the msb first */
      {
            scl = 1;
            d = d << 1;
            d = d | (unsigned char)sda;
            scl = 0;
      }
      sda = b;          /* Hold sda low for acknowledge */

      scl = 0;
      scl = 1;
      if(b == NACK)     sda = 1;    /* sda = 1 if next cycle is reset */
      scl = 0;

      sda = 1;          /* Release the sda line */
      return d;
}
void writebyte()  /* --- write one byte, user defined address & data --- */
{
uchar Add, Data;

      printf("\nAddress (hex): ");  /* Get Address */
      scanf("%bx", &Add);
      printf("DATA (hex): ");
      scanf("%bx", &Data);    /* and data */
      start();
      I2Cwrite(ADDRTC);
      I2Cwrite(Add);
      I2Cwrite(Data);
      stop();
}
void readbyte()   /* ---------------------------------------------- */
{
uchar Add;
      printf("\nADDRESS: ");  /* Get Address */
      scanf("%bx", &Add);
      start();
      I2Cwrite(ADDRTC);
      I2Cwrite(Add);
      start();
      I2Cwrite(ADDRTC | 1);
      printf("%02bx", I2Cread(NACK) );
      stop();
}
void initialize() /* ---------------------------------------- */
/* Note: NO error checking is done on the user entries! */
{
uchar yr, mn, dt, dy, hr, min, sec, day;
```

```
   start();       /* The following Enables the Oscillator */
   I2Cwrite(ADDRTC);    /* address the part to write */
   I2Cwrite(0x0e);      /* control register address */
   I2Cwrite(0x01);      /* disable mission, duration interval select bits,
enable osc */
   stop();
   start();       /* The following Enables the Oscillator */
   I2Cwrite(ADDRTC);    /* address the part to write */
   I2Cwrite(0x08);      /* seconds alarm register address */
   I2Cwrite(0x80);      /* mask for once per second interrupt */
   I2Cwrite(0x80);
   I2Cwrite(0x80);
   I2Cwrite(0x80);
   stop();

   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);
   printf("Enter the hour (1-23): ");
   scanf("%bx", &hr);
   hr = hr & 0x3f;      /* force clock to 24 hour mode */
   printf("Enter the minute (0-59): ");
   scanf("%bx", &min);
   printf("Enter the second (0-59): ");
   scanf("%bx", &sec);

   start();
   I2Cwrite(ADDRTC);    /* write slave address, write 1678 */
   I2Cwrite(0x00);      /* write register address, 1st clock register */
   I2Cwrite(sec);
   I2Cwrite(min);
   I2Cwrite(hr);
   I2Cwrite(dy);
   I2Cwrite(dt);
   I2Cwrite(mn);
   I2Cwrite(yr);
   I2Cwrite(0x20);      /* write century */
   I2Cwrite(0);   /* clear alarm */
   I2Cwrite(0);
   I2Cwrite(0);
   I2Cwrite(0);   /* register 0x0b */
   I2Cwrite(0);   /* dummy */
   I2Cwrite(0);
   I2Cwrite(0x40);      /* control byte, enable memory clear */
   I2Cwrite(0x10);      /* status byte, clear memory */
   stop();
}
void disp_clk_regs(uchar lp)  /* ----display time & date, once or loop -----
*/
{
uchar Sec, prv_sec = 99, Min, Hrs, Dte, Mon, Day, Yr, Cn;
```

```
      printf("\nYear/Mn/Dt Dy Hr:Mn:Sec");
      while(!RI)  /* Read & Display Clock Registers */
      {
            start();
            I2Cwrite(ADDRTC); /* write slave address, write 1678 */
            I2Cwrite(0x00);   /* write register address, 1st clock register
*/
            start();
            I2Cwrite(ADDRTC | 1);   /* write slave address, read 1678 */
            Sec = I2Cread(ACK);           /* starts w/current register
pointer address*/
            Min = I2Cread(ACK);
            Hrs = I2Cread(ACK);
            Day = I2Cread(ACK);
            Dte = I2Cread(ACK);
            Mon = I2Cread(ACK);
            Yr  = I2Cread(ACK);
            Cn  = I2Cread(NACK);
            stop();
            if(Sec != prv_sec)      /* display every time seconds change */
            {
                  printf("\n%02bX%02bX/%02bX/%02bX %02bX", Cn, Yr, Mon, Dte,
Day);
                  printf(" %02bX:%02bX:%02bX", Hrs, Min, Sec);
            }
            prv_sec = Sec;
            if(!lp)     return;
      }
      RI = 0;  /* Swallow keypress to exit loop */
}
void  disp_ts_etc()     /* ---- display time & date stamp, event 0 elapsed
time ---- */
{
uchar Sec, Min, Hrs, Dte, Mon, Day, Yr, Cn, etclsb, etcmsb;

      /* ------- display control register settings ------- */
      start();
      I2Cwrite(ADDRTC); /* write slave address + write bit */
      I2Cwrite(0x0e);   /* write register address, control byte */
      start();
      I2Cwrite(ADDRTC | 1);   /* write slave address, read 1678 */
      Sec = I2Cread(ACK);     /* starts w/last address stored in register
pointer */
      Min = I2Cread(NACK);    /* starts w/last address stored in register
pointer */
      stop();

      printf("\n\nMission Enable (ME): ");
      if(Sec & 0x80)
            printf("ON");
      else
            printf("OFF");

      printf("\nClear Enable (CLR): ");
      if(Sec & 0x40)
            printf("ON ");
      else
```

```
      printf("OFF ");

Hrs = (Sec & 0x30);     /* mask DISx bits */
printf("\nDuration interval (DISx): ");
switch(Hrs)
{
      case 0:     printf("Alarm Enabled");      break;
      case 0x10:  printf("Counter=1/sec (Max=18.2Hrs)");    break;
      case 0x20:  printf("Counter=1/min (Max=45.5Days)");   break;
      case 0x30:  printf("Counter=1/hour (Max=7.5Yrs)");    break;
}

printf("\nRoll-Over (RO): ");
if(Sec & 8)
      printf("Enabled ");
else
      printf("Disabled ");

Hrs = (Sec & 0x06);     /* mast TRx bits */
printf("\nTrigger select (TRx): ");
switch(Hrs)
{
      case 0:     printf("N/A");          break;
      case 0x2:   printf("Falling Edge"); break;
      case 0x4:   printf("Rising Edge");  break;
      case 0x6:   printf("Rising & Falling Edge");    break;
}

printf("\nEnable Oscillator (EOSC): ");
if(Sec & 1)
      printf("Enabled  ");
else
      printf("Disabled ");

/* ------- display status register ------- */
printf("\nMemory  (MEM CLR): ");
if(Min & 0x40)
      printf("Clear");
else
      printf("No   ");

printf("\nMission   (MIP) :");
if(Min & 0x20)
      printf("In Progress");
else
      printf("Not running");

printf("\nClear Memory (CM): ");
if(Min & 0x10)
      printf("True ");
else
      printf("False");

printf("\nLow Bat (LOWBAT): ");
if(Min & 8)
      printf("True ");
else
```

```
            printf("False");

      printf("\nRoll Over  (ROF) : ");
      if(Min & 4)
            printf("True ");
      else
            printf("False");

      printf("\nAlarm Flag (ALMF): ");
      if(Min & 1)
            printf("True ");
      else
            printf("False");

      if(!(Min & 0x20) || (Sec & 0x80) )  /* will read back as zeros if ME or
MIP = 1, so don't bother */
      {
            start();
            I2Cwrite(ADDRTC); /* write slave address, write 1678 */
            I2Cwrite(0x30);   /* write register address, 1st time stamp
register */
            start();
            I2Cwrite(ADDRTC | 1);   /* write slave address, read 1678 */
            Sec = I2Cread(ACK);     /* starts w/last address stored in
register pointer */
            Min = I2Cread(ACK);
            Hrs = I2Cread(ACK);
            Day = I2Cread(ACK);
            Dte = I2Cread(ACK);
            Mon = I2Cread(ACK);
            Yr  = I2Cread(ACK);
            Cn  = I2Cread(ACK);
            etclsb  = I2Cread(ACK);
            etcmsb  = I2Cread(NACK);
            stop();

            printf("\nYear/Mn/Dt Dy Hr:Mn:Sec last event");
            printf("\n%02bX%02bX/%02bX/%02bX %02bX", Cn, Yr, Mon, Dte, Day);
            printf(" %02bX:%02bX:%02bX %02bX%02bX ", Hrs, Min, Sec, etcmsb,
etclsb);
      }
}
void  disp_evnt_count() /* ---- display event counter contents ---- */
{
uchar low, mid, high;

      start();
      I2Cwrite(ADDRTC); /* write slave address, write 1678 */
      I2Cwrite(0x3a);   /* write register address, 1st time stamp register */
      start();
      I2Cwrite(ADDRTC | 1);   /* write slave address, read 1678 */
      low = I2Cread(ACK);     /* starts w/last address stored in register
pointer */
      mid = I2Cread(ACK);
      high = I2Cread(NACK);
      stop();
      printf("\nEvent count: %02bX%02bX%02bX", high, mid, low);
```

```
}
void  disp_etc_addptr() /* ---- display etc and address pointer ---- */
{
uchar low, high, alow, ahigh;

      start();
      I2Cwrite(ADDRTC); /* write slave address, write 1678 */
      I2Cwrite(0x3d);   /* write register address, ETC start */
      start();
      I2Cwrite(ADDRTC | 1);   /* write slave address, read 1678 */
      low = I2Cread(ACK);     /* starts w/last address stored in register
pointer */
      high = I2Cread(ACK);
      alow = I2Cread(ACK);
      ahigh = I2Cread(NACK);
      stop();
      printf("\nETC: %02bX%02bX Addr ptr: %02bX%02bX ", high, low, ahigh,
alow);
}
void  disp_dlog() /* ---- display data log contents ---- */
{
uint  inc, num_evnts;
uchar dat;

      start();
      I2Cwrite(ADDRTC);       /* write slave address, write 1678 */
      I2Cwrite(0x3a);         /* write register address, event counter */
      start();
      I2Cwrite(ADDRTC | 1);   /* write slave address, read 1678 */
      num_evnts = I2Cread(ACK);     /* read lower byte of event count */
      num_evnts += ( ((int) I2Cread(NACK) << 8) );    /* read middle byte of
event count */
      stop();
      if(num_evnts > 0) num_evnts = num_evnts - 1;    /* subtract the 1st
event (timestamp) */
      if(num_evnts > 1024)    num_evnts = 1024;       /* max of 1024 events
in log mem */
      printf("Events in log: %u Dlog:", num_evnts);   /*  (+timestamp=1025)
*/

      for(inc = 0; inc < (num_evnts + num_evnts); inc+=2)
      {
            start();
            I2Cwrite(ADDRTC); /* write slave address, write 1678 */
            I2Cwrite(0x41);   /* write register address, 1st time stamp
register */
            I2Cwrite( (uchar) inc );      /* set lower 8 bits of data log RAM
address */
            I2Cwrite( (uchar) (inc >> 8) ); /* set upper 8 bits of address */
            start();
            I2Cwrite(ADDRTC | 1);   /* write slave address, read 1678 */
            dat = I2Cread(NACK);
            start();
            I2Cwrite(ADDRTC); /* write slave address, write 1678 */
            I2Cwrite(0x41);   /* write register address, 1st time stamp
register */
```

```
            I2Cwrite( (uchar) (inc + 1) );      /* set lower 8 bits of data
log RAM address */
            I2Cwrite( (uchar) (inc >> 8) ); /* set upper 8 bits of address */
            start();
            I2Cwrite(ADDRTC | 1);   /* write slave address, read 1678 */
            if(!(inc % 32) )  /* format output 16 events wide */
                  printf("\n");
            else
                  printf(" ");
            printf("%02bx%02bX", I2Cread(NACK), dat);
            stop();
      }
}
void  disp_dlog2()      /* ---- display data log contents, addr auto-
increment ---- */
{
uint  inc, num_evnts;
uchar dat;

      start();
      I2Cwrite(ADDRTC); /* write slave address, write 1678 */
      I2Cwrite(0x3a);         /* write register address, event counter */
      start();
      I2Cwrite(ADDRTC | 1);   /* write slave address, read 1678 */
      num_evnts = I2Cread(ACK);     /* read lower byte of event count */
      num_evnts += ( (uint) (I2Cread(NACK) << 8) );   /* read middle byte of
event count */
      stop();

      start();
      I2Cwrite(ADDRTC); /* write slave address, write 1678 */
      I2Cwrite(0x41);   /* write register address, 1st time stamp register */
      I2Cwrite(0);      /* set lower 8 bits of data log RAM address */
      I2Cwrite(0);      /* set upper 8 bits of address */
      stop();

      printf("Dlog: ");
      start();
      I2Cwrite(ADDRTC | 1);   /* write slave address, read 1678 */
      for(inc = 0; inc < num_evnts; inc++)
      {
            dat = I2Cread(ACK);
            printf("%02bx%02bX ", I2Cread(ACK), dat);
      }
      I2Cread(NACK);    /* dummy read to send NACK */
      stop();
}
void burstramread()           /* ---------------------------------------- */
{
uchar inc;

      printf("\nUser RAM");

      start();
      I2Cwrite(ADDRTC); /* write slave address, write 1678 */
      I2Cwrite(0x10);   /* write register address, 1st event location LSB*/
      start();
```

```
      I2Cwrite(ADDRTC | 1);   /* write slave address + read bit */

      for (inc = 0; inc < 32; inc++)
      {
            if(!(inc % 16) ) printf("\n%02bx ", inc);
            printf("%02bX ", I2Cread(ACK) );
      }
      I2Cread(NACK);    /* extra read to terminate read */
      stop();
}
void burstramwrite()          /* ---------------------------------------- */
{
uchar inc, Data;

      printf("\nEnter the data to write: ");
      scanf("%bx", &Data);

      start();
      I2Cwrite(ADDRTC); /* write slave address, write 1678 */
      I2Cwrite(0x10);   /* write register address, 1st RAM location */
      for (inc = 0; inc < 32; inc++)
      {
            I2Cwrite(Data);
      }
      stop();
}
void ramclear()   /* ---- clear the data log RAM ---- */
{
      start();
      I2Cwrite(ADDRTC); /* write slave address, write 1678 */
      I2Cwrite(0x0e);   /* pointer to control register */
      I2Cwrite(0x41);   /* set CLR bit to enable memory clear, enable
oscillator */
      I2Cwrite(0x10);   /* set CM bit in status register to clear memory */
      start();
      I2Cwrite(ADDRTC); /* write slave address, write 1678 */
      I2Cwrite(0x0e);   /* pointer to control register */
      I2Cwrite(0x97);   /* once per second resolution, trigger on both edges,
enable osc */
      I2Cwrite(0x20);   /* set MIP bit  to start mission */
      start();
      I2Cwrite(ADDRTC); /* write slave address, write 1678 */
      I2Cwrite(0x0e);   /* pointer to control register */
      start();
      I2Cwrite(ADDRTC + 1);
      printf("\n%02bx %02bx", I2Cread(ACK), I2Cread(NACK) );
      stop();
}
void  mission()   /* ---- get user info and start a mission ---- */
{
uchar inc, val, mstart;

      start();
      I2Cwrite(ADDRTC); /* address the part to write */
      I2Cwrite(0x0e);   /* control register address */
      I2Cwrite(1);            /* eosc = 1: let the oscillator get started */
      stop();
```

```
	do
	{
		printf("\nEnter a duration\n1) 1/sec\n2) 1/min\n3) 1/Hour: ");
		scanf("%1bx", &inc);
	}	while(inc > 3 || inc == 0);
	val = inc << 4;

	do
	{
		printf("\nEnter the trigger condition\n1) Falling Edge\n2) Rising
Edge"
			"\n3) Rising & Falling Edges : ");
		scanf("%1bx", &inc);
	}	while(inc > 3 || inc == 0);
	val += inc << 1;

	do
	{
		printf("\nRoll-Over: \n0) Disabled\n1) Enabled: ");
		scanf("%1bx", &inc);
	}	while(inc > 2);
	val += inc << 3;
	do
	{
		printf("\nEnter start method:\n1) Immediately or 2) Event: ");
		scanf("%1bx", &mstart);
	}	while(mstart > 2 || mstart == 0);

	disp_clk_regs(0); /* display starting time */

	start();
	I2Cwrite(ADDRTC); /* address the part to write */
	I2Cwrite(0x0e);   /* control register address */
	I2Cwrite(val | 0x41);   /* load user entries plus enable clear & eosc
*/
	I2Cwrite(0x10);   /* clear the memory */
	stop();

	for(inc = 0; inc < 254; inc++);     /* wait for memory to clear */

	start();
	I2Cwrite(ADDRTC); /* address the part to write */
	I2Cwrite(0x0e);   /* control register address */

	if(mstart == 1)
	{
		I2Cwrite(val + 1);      /* start mission via software (ME = 0) */
		I2Cwrite(0x20);   /* this will start the mission immediately (MIP
= 1) */
	}
	else
		I2Cwrite(val + 0x81);   /* start mission via external event (ME =
1) */
	stop();

}
```

```
void  endmission()      /* ----- clear MIP to stop the mission ----- */
{
      start();
      I2Cwrite(ADDRTC); /* address the part to write */
      I2Cwrite(0x0e);   /* control register address */
      I2Cwrite(0);            /* stop mission */
      stop();
}
void  show_log()  /* ----- show log registers ----- */
{
uchar stat;

      start();
      I2Cwrite(ADDRTC); /* address the part to write */
      I2Cwrite(0x0f);   /* status register address */
      start();
      I2Cwrite(ADDRTC | 1);   /* address the part to write */
      stat = I2Cread(NACK);
      stop();

      disp_ts_etc();
      if(!(stat & 0x20) )     /* will be zeros if mission is running, so
don't bother */
      {
            disp_evnt_count();
            disp_etc_addptr();
            disp_dlog();
      }
}
void  int_test()  /* ----- set up alarm for 5 seconds ------ */
{
uchar inc;

      start();          /* The following Enables the Oscillator */
      I2Cwrite(ADDRTC); /* address the part to write */
      I2Cwrite(0x0e);   /* control register address */
      I2Cwrite(0x01);   /* disable mission, duration interval select bits,
enable osc */
      stop();
      start();          /* The following Enables the Oscillator */
      I2Cwrite(ADDRTC); /* slave address + write bit */
      I2Cwrite(0);            /* write register address, 1st clock register
*/
      I2Cwrite(0);
      I2Cwrite(0);
      I2Cwrite(0x12);
      I2Cwrite(1);
      I2Cwrite(0x12);
      I2Cwrite(2);
      I2Cwrite(2);
      I2Cwrite(0x20);   /* write century */
      I2Cwrite(0x05);   /* alarm when seconds match*/
      I2Cwrite(0x80);
      I2Cwrite(0x80);
      I2Cwrite(0x80);   /* register 0x0b */
      I2Cwrite(0);            /* dummy */
      I2Cwrite(0);
```

```
      I2Cwrite(1);      /* EOSC=1, DISx must be 0 for ALMF to drive /INT
output */
      I2Cwrite(0);      /* status byte, clear memory */
      stop();
}
main (void)       /* ------------------------------------------------- */
{
uchar i, M, M1;

      INTb = 1;

      while (1)
      {
      printf("\nDS1678 demo build: %s\n", __DATE__);
      printf("CI Init    CR Clk rd\n");
      printf("BR Byte rd BW Byte wr E End mission\n");
      printf("RR RAM rd  RW RAM Wr  D dlog\n");
      printf("RC Ram Clr M  Mission N iNt test\n");
      printf("I  Int tog L show log\n");
      printf("Enter Menu Selection: ");

      M = _getkey();

      switch(M)
      {
            case 'B':
            case 'b':
            printf("\nRead or Write: ");
            M1 = _getkey();

            switch(M1)
            {
                  case 'R':
                  case 'r':   readbyte();
                              break;
                  case 'W':
                  case 'w':   writebyte();
                              break;
            }
            break;

            case 'C':
            case 'c':
            printf("\rEnter Clock Routine to run:C");
            M1 = _getkey();

            switch(M1)
            {
                  case 'I':
                  case 'i':   initialize();
                              break;

                  case 'R':
                  case 'r':   disp_clk_regs(1);
                              break;
            }
            break;
```

```
            case 'E':
            case 'e':   endmission();     break;

            case 'I':
            case 'i':   INTb ^= 1;  break;

            case 'L':
            case 'l':   show_log(); break;

            case 'N':
            case 'n':   int_test(); break;

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

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

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

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

            case 'M':
            case 'm':   mission();  break;

            case '2':   disp_dlog2();     break;
      }
   }
}
```