Application Note 506

Recording Total On-Time and Power Cycling Information Using the DS1682

www.maxim-ic.com

OVERVIEW

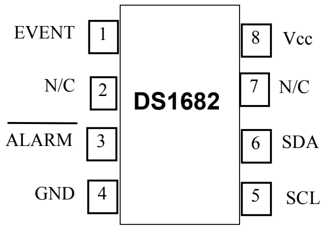
The DS1682 provides a method of counting power cycles, total on-time, and alarm-on match functions that reduce the amount of processor overhead. Total on-time information is useful in applications from Hobbs meters to equipment warranty, repair, and maintenance.

The DS1682 contains a 32-bit elapsed time counter (ETC), a 17-bit event counter, and a 32-bit alarm register. In addition, the DS1682 has 10-bytes of user-programmable memory. The counter increments in º seconds. The ETC and event counter data are stored in EEPROM at the end of each event.

In this example, the EVENT input is driven by a DS1233-15 EconoReset. A Shottky diode and capacitor provide the energy necessary to allow the DS1682 to store the data to the internal EEPROM when V_{CC} goes away. An 8051-type processor is used to read the registers and transmit the information to a display via a serial port. An LED is included in case the alarm function is required.

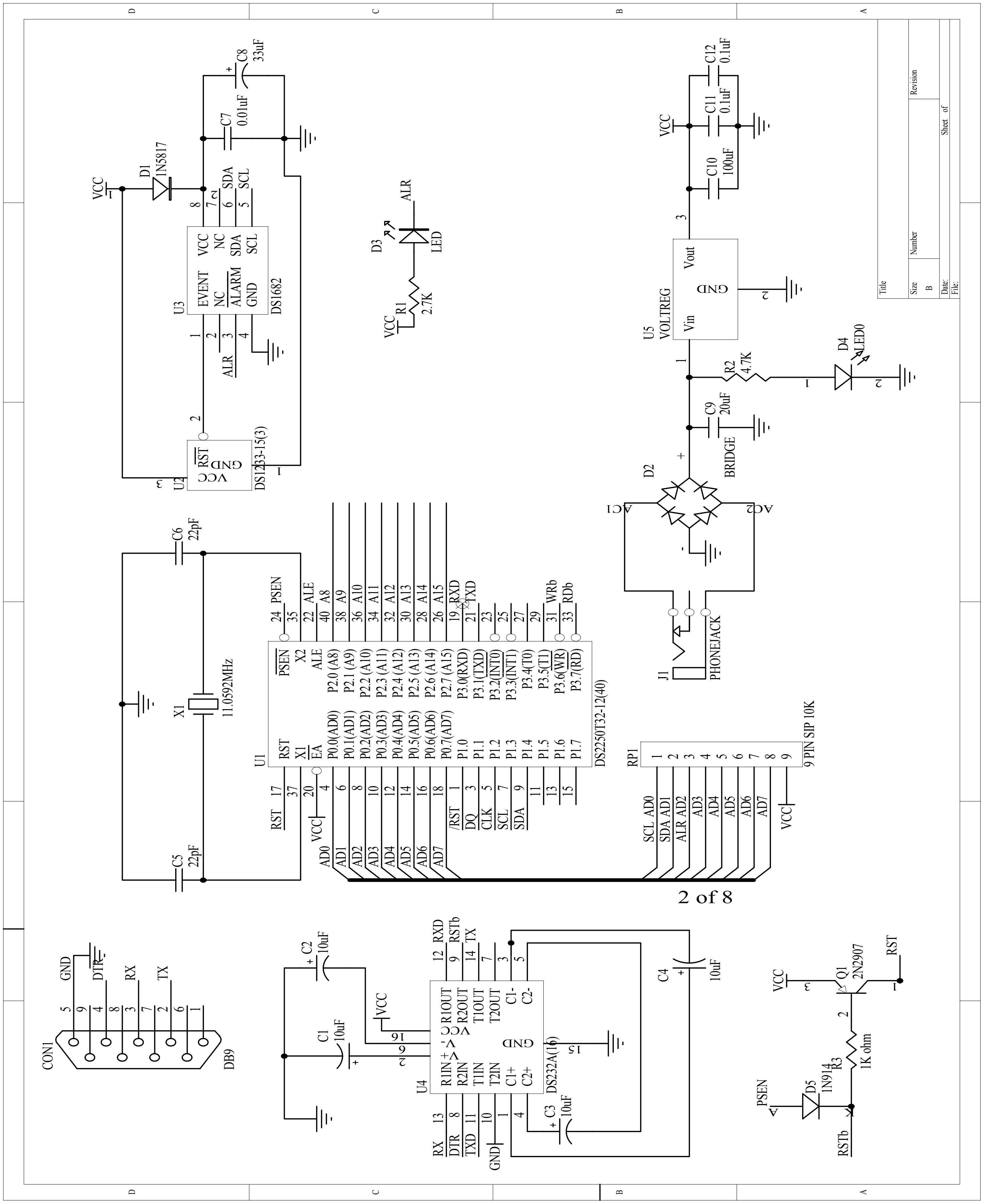
A schematic of the circuit is shown in Figure 1. Software for initializing the DS1682 and reading the registers is show in Figure 2.

PIN ASSIGNMENT



Top View
DS1682S 8-Pin SO (150mil)

Figure 1. **CIRCUIT SCHEMATIC**



2 of 8

Figure 2. **DS1682 SOFTWARE**

```
/****************************************************************************/
/* DS1682AN.C - Software for application note                               */
/****************************************************************************/
#pragma code symbols debug
#include <stdio.h>        /* Prototypes for I/O functions        */
#include <DS5000.h>       /* Register declarations for DS5000    */
/************************* Global Variables *****************************/
/***************************** Defines ********************************/
#define      AOS    0x08
#define      AP     0x02
#define      RE     0x04
#define      ACK    0
#define      NACK   1
#define      ADD1682      0xd6   /* 2-wire base address (read) */
sbit scl = P0^0;                 /* 2-wire pin definitions */
sbit sda = P0^1;
sbit ALR = P0^2;
/*********************** Function Prototypes ****************************/
uchar rbyte();
void  wbyte(uchar);

void   alarm();
void   disp_regs();
void   start2w();
void   stop2w();
void   writebyte2w(uchar);
uchar  read_byte(uchar);
uchar  readbyte2w(uchar);
void   readreg();
void   writereg();

void start2w()      /* ---------------------------------------------- */
{
       sda = 1; sda = 1;  scl = 1;      /* Initiate start condition */
       sda = 0;
}
void stop2w()       /* ---------------------------------------------- */
{
       sda = 0;  sda = 0;  sda = 0;     /* Initiate stop condition */
       scl = 1; scl = 1;  scl = 1;  sda = 1;
}
void writebyte2w(uchar d)  /* ----- write 2-wire, current register ----- */
{
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 = 1;
       if (sda) printf("Ack bit missing  %02X\n",(unsigned int)d);
       scl = 0;
}
uchar readbyte2w(uchar b)  /* ------- read 2-wire, current register ------- */
{
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 = 1;
       if(b == NACK) sda = 1;     /* sda = 1 if next cycle is reset */
       scl = 0;

       sda = 1;          /* Release the sda line */
       return d;
}
void writereg()     /* ------------ single byte user write ------------ */
{
uchar add, dat;

       printf("\naddress (hex): ");     /* Get Address */
       scanf("%bx", &add);
       printf("\nDATA (hex): ");
       scanf("%bx", &dat); /* and data */
       start2w();
       writebyte2w(ADD1682);
       writebyte2w(add);
       writebyte2w(dat);
       stop2w();
}
void readreg()      /* -------------- single byte user read -------------- */
{
uchar add;
       printf("\nEnter address (hex): ");      /* Get Address */
       scanf("%bx", &add);
       start2w();
       writebyte2w(ADD1682);
       writebyte2w(add);
       start2w();
       writebyte2w(ADD1682 | 1);
       printf("%2.bx", readbyte2w(NACK) );
       stop2w();
}
void loop_read()           /* ------------ loop reading alarm and tta registers -------------
*/
{
long int tmp;
uchar  cnfg, altpl, altplm, altphm, altph, ttal_last, ttal, ttalm, ttahm, ttah, evntl, evnth;

       while(!RI)
       {
              cnfg = read_byte(0);       /* starts w/last address stored in register pointer */
              altpl = read_byte(1);
              altplm = read_byte(2);
              altphm = read_byte(3);
              altph = read_byte(4);
              ttal = read_byte(5);
              ttalm = read_byte(6);
              ttahm = read_byte(7);
              ttah = read_byte(8);
              evntl  = read_byte(9);
              evnth  = read_byte(0xa);
              if( (ttal & 0xfc) != (ttal_last & 0xfc) )      /* update display once per second
while event high */
              {
                     printf("\rCnfg: %2.bx", cnfg);
```

```
                    tmp = ((long int) altph << 24) + ((long int) altphm << 16) + ((long int)
altplm << 8) + (long int) altpl;
                    tmp >>= 2;
                    printf(" Alarm reg: %10ld sec ", tmp);
                    tmp = ((long int) ttah << 24) + ((long int) ttahm << 16) + ((long int)
ttalm << 8) + (long int) ttal;
                    tmp >>= 2;
                    printf("Event Time: %10ld sec ", tmp);
                    printf("Event Count: %d", ((int) evnth << 8) + evntl);
                    ttal_last = ttal;
             }
      }
      _getkey();
}
void  init() /* ----------- get setup and configure DUT ---------- */
{
/* Note: NO error checking is done on the user entries! */

unsigned long int   alrm;  /* assumes 32 bit long int */
uchar config, dat;

      printf("\nThis routine will initialize and reset the DS1682, after which");
      printf("\nno changes can be made.  Continue? Y/N: ");
      scanf("%c", &dat);
      dat &= 0xdf; /* make upper case if not already */
      if(dat != 'Y')
             return;      /* do not pass go, do not collect $200 */

      printf("\nEnter alarm time in whole seconds 0 to 4294967292 ");
      scanf("%ld", &alrm);
      alrm <<= 2;

      start2w();
      writebyte2w(ADD1682);      /* write slave address + write */
      writebyte2w(0x00);         /* write register address */
      writebyte2w(config);
      writebyte2w( (uchar) (alrm & 0xff) );
      writebyte2w( (uchar) ((alrm & 0xff00) >> 8)  );
      writebyte2w( (uchar) ((alrm & 0xff0000) >> 16)  );
      writebyte2w( (uchar) ((alrm & 0xff000000) >> 24) );
      stop2w();

      start2w();
      writebyte2w(ADD1682);      /* write slave address + write */
      writebyte2w(0x0b);         /* write register address */
      for(config = 1; config < 11; config++)
      {
             printf("Enter User RAM data byte HEX %bd ", config);
             scanf("%bx", &dat);
             writebyte2w(dat);
      }
      stop2w();

      printf("\nLock User RAM? (Y/N): ");     /* select memory lock options */
      scanf("%*c %c", &dat);
      if(dat == 'Y' || dat == 'y')
      {
             for(dat = 0; dat < 2; dat++)      /* write reg 0x1f twice with 0xf0 data */
             {
                    start2w();
                    writebyte2w(ADD1682);      /* write slave address + write */
                    writebyte2w(0x1f);         /* write register address */
                    writebyte2w(0xf0);         /* write data */
                    stop2w();
             }
      }      /* WMDF should now read back as a '1' */
```

```
        printf("\nLock Alarm, ETC & Event registers? (Y/N): ");

        scanf("%c", &dat);
        if(dat == 'Y' || dat == 'y')
        {
                for(dat = 0; dat < 2; dat++)      /* write reg 0x1f twice with 0xf0 data */
                {
                        start2w();
                        writebyte2w(ADD1682);      /* write slave address + write */
                        writebyte2w(0x1e);         /* write register address */
                        writebyte2w(0xaa);         /* write data */
                        stop2w();
                }
        }       /* WDF should now read back as a '1' */

        do      /* now set up the alarm output options */
        {
                printf("\nAlarm output Pulse or Constant? (P/C): ");
                scanf("%c", &dat);
                switch(dat)
                {
                        case 'P':       /* set AOS bit in configuration register to a '0' */
                        case 'p':       config = 0;     break;
                        case 'C':       /* set AOS bit in configuration register to a '1' */
                        case 'c':       config = AOS;   break;
                        default:        dat = 0;
                }
        }       while(!dat);

        if(dat == 'C' || dat == 'c')
        {
                printf("\nAlarm Polarity Active High or Low? (H/L): ");
                scanf("%c", &dat);
                switch(dat)
                {
                        case 'H':       /* set AP bit in configuration register to a '1' */
                        case 'h':       config |= AP;   break;
                        case 'L':       /* set AP bit in configuration register to a '0' */
                        case 'l':       config &= (AP ^ 0xff);     break;
                        default:        dat = 0;
                }
        }       while(!dat);

        start2w();
        writebyte2w(ADD1682);           /* write slave address + write */
        writebyte2w(0);                 /* write register address */
        writebyte2w(config + RE);       /* write configuration data, enable reset */
        stop2w();

        /* note that it doesn't matter if there are data in the ETC and event counters.
                We don't care if EVENT is high or low, or if we've stored the alarm data
                in EEPROM.  The reset command will transfer that data. */

        for(dat = 0; dat < 2; dat++)      /* write reg 0x1f twice with 0xf0 data */
        {
                start2w();
                writebyte2w(ADD1682);      /* write slave address + write */
                writebyte2w(0x1d);         /* write register address */
                writebyte2w(0x55);         /* write data */
                stop2w();
        }
        /* the eeprom will take tEW to finish, so we're depending upon the menu rountine
                and the user to take enough time for the write to finish, before we talk
                to the part again.  a delay routine should be added here if we add any
                code that could access the part before the eeprom write terminates */
}
```

```
void disp_regs()          /* --------------displays all of the user registers ---------------
- */
{
uchar  cnfg, altpl, altplm, altphm, altph, ttal, ttalm, ttahm, ttah, evntl, evnth;

       cnfg = read_byte(0);       /* starts w/last address stored in register pointer */
       altpl = read_byte(1);
       altplm = read_byte(2);
       altphm = read_byte(3);
       altph = read_byte(4);
       ttal = read_byte(5);
       ttalm = read_byte(6);
       ttahm = read_byte(7);
       ttah = read_byte(8);
       evntl  = read_byte(9);
       evnth  = read_byte(0xa);
       printf("\nCnfg: %2.bx", cnfg);
       printf("\nAlrm Trip point: %02.bX%02.bX%02.bX%02.bX", altph, altphm, altplm, altpl);
       printf("\nTtl Time Acc:    %02.bX%02.bX%02.bX%02.bX", ttah, ttahm, ttalm, ttal);
       printf("\nEvent Counter:   %02.bX%02.bX", evnth, evntl);
       printf("\nUser RAM: ");
       for(cnfg = 0xb; cnfg < 0x15; cnfg++)    printf("%02.bX ", read_byte(cnfg) );
}
uchar  read_byte(uchar reg_add)   /* ---- read byte from address as entered ---- */
{
uchar  dat;

       start2w();
       writebyte2w(ADD1682);
       writebyte2w(reg_add);
       start2w();
       writebyte2w(ADD1682 | 1);
       dat = readbyte2w(NACK);
       stop2w();
       return(dat);
}
void   alarm()      /* ---- monitor alarm output, read regs on change ---- */
{
/* routine assumes part is powered & ready for test mode.  Assumes alarm is inactive on entry
*/

uchar  inc;

       printf("\nAlarm Test:");
       printf("\nStart:");
       for(inc = 0; inc < 0x15; inc++)
       {
             printf("%2.bx ", read_byte(inc) );
       }
       puts("");

       inc = ALR;
       while(inc == ALR)
             inc = ALR;

       printf("\nMatch  :");
       for(inc = 0; inc < 0x15; inc++)
       {
             printf("%2.bx ", read_byte(inc) );
       }
       puts("");
}
main (void)         /* ----------------------------------------------------- */
{
uchar i, M, M1;
```

```
       disp_regs(); /* read device every time code starts */
       while (1)
       {
              printf("\nDS1682 \n");
              printf("R Read regs   L  Loop Read\n");

              printf("BR Byte Read  BW Write Byte\n");
              printf("I initialize  A  Alarm monitor\n");
              printf("\nEnter Menu Selection:");

              M = _getkey();
              printf("%c", M);
              switch(M)
              {
                     case 'A':
                     case 'a':    alarm();     break;

                     case 'B':
                     case 'b':
                     printf("\nRead or Write: ");
                     M1 = _getkey();
                     printf("%c", M1);
                     switch(M1)
                     {
                            case 'R':
                            case 'r':    readreg();   break;
                            case 'W':
                            case 'w':    writereg();  break;
                     }
                     break;

                     case 'I':
                     case 'i':    init();      break;

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

                     case 'R':
                     case 'r':    disp_regs(); break;
              }
       }
}
```