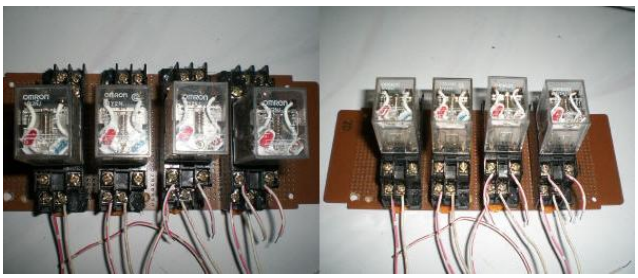
LAMPIRAN A

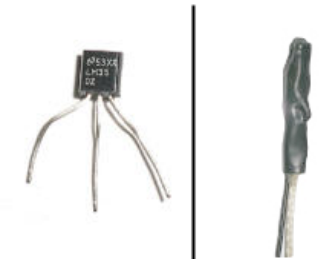
Lampiran A.1 RTC

Lampiran A.2 Pemanas

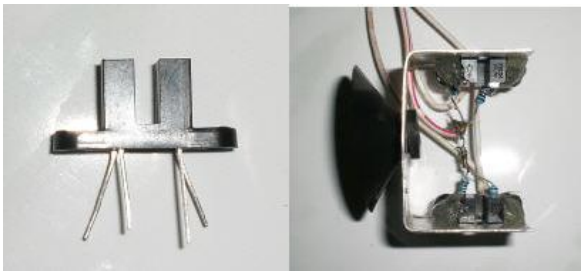


Lampiran A.3 Relai

Lampiran A.4 Katup Selenoid



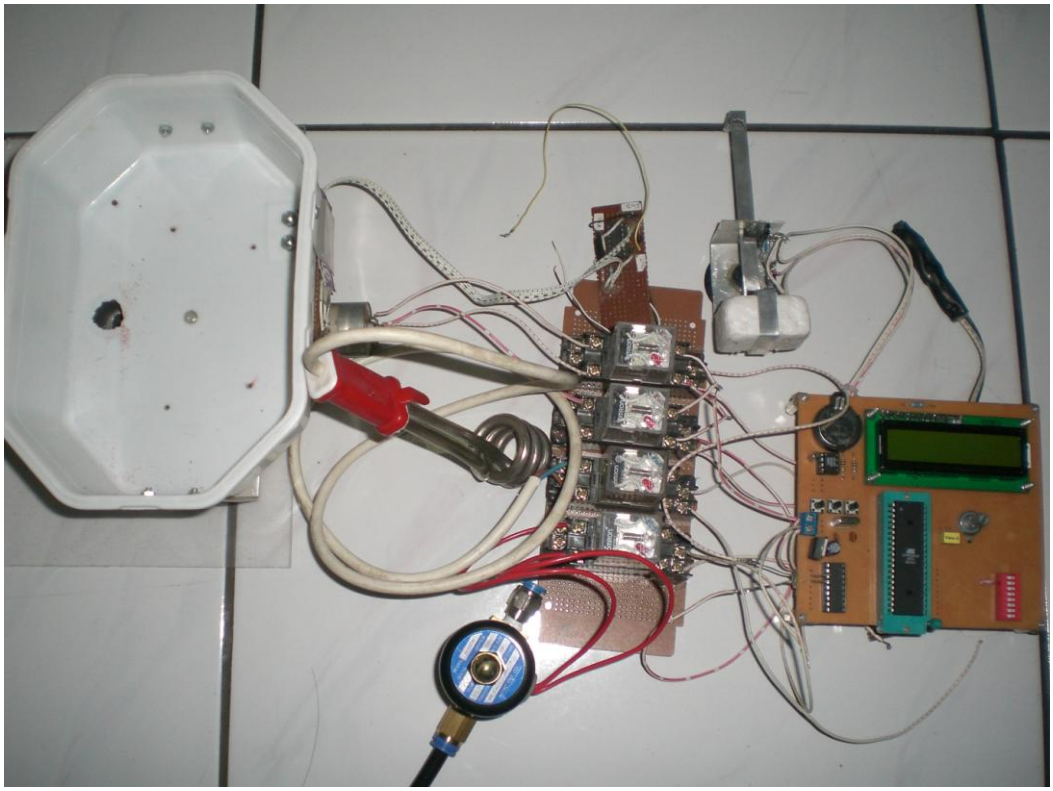
Lampiran A.5 Sensor Temperatur LM35



Lampiran A.6 Sensor *Optocoupler*

Lampiran A.7 Sensor Pelampung

Lampiran A.8 Motor DC Dengan Sensor *Optocoupler*



Lampiran A.9 Alat Keseluruhan.

LAMPIRAN B

```
#include <mega16.h>
#include <delay.h>
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include <math.h>

#define out1 PORTD.0   //motor  arah kanan
#define out2 PORTD.1   //motor arah kiri
#define out3 PORTD.2   //selenoid
#define out4 PORTD.3   //pemanas
#define lama1  1000
#define lama2  2000
#define lama3  3500


int suhu,adc_byte,hours1,v,lama_makanan,status_opto ;
char h,m,s,c;
char tampil1[4] ;
void get_time(void);
int gerak_motor1(void);
int gerak_motor2(void);
float get_adc(int d);
void menu_awal(int x , int y);
void menu_alarm(int x1 , int y1) ;
int jl_makanan(void);
void set_time(char sjam, char smenit,char sdetik);
int keypad(void);
void cek_opto_pelampung(void);
int jam1[5];
int menit1[5];
int suhu1[6];
int count=0;
int mark=0;
int tandax=0;
void cek_waktu(int jam3,int menit3);
void cek_suhu(int suhu2);

#include <rtc.h>
#include <lcd.h>
#define ADC_VREF_TYPE 0x00
interrupt [ADC_INT] void adc_isr(void)
{
adc_data=ADCH;
}
```

```
unsigned char read_adc(unsigned char adc_input)
{
ADMUX=adc_input|ADC_VREF_TYPE;
return adc_data;
}

void main(void)
{     char tampilc[4] ;
      int mode1;

PORTA=0x00;
DDRA=0x00;
PORTB=0xE0;
DDRB=0x03;
PORTC=0x00;
DDRC=0x00;
PORTD=0x00;
DDRD=0xFF;

TCCR0=0x00;
TCNT0=0x00;
OCR0=0x00;

TCCR1A=0x00;
TCCR1B=0x00;
TCNT1H=0x00;
TCNT1L=0x00;
ICR1H=0x00;
ICR1L=0x00;
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;

ASSR=0x00;
TCCR2=0x00;
TCNT2=0x00;
OCR2=0x00;

MCUCR=0x00;
MCUCSR=0x00;

TIMSK=0x00;

ACSR=0x80;
SFIOR=0x00;

ADMUX=ADC_VREF_TYPE;
```

```
ADCSRA=0x8F;

rtcInit();
lcdInit();
#asm("sei")

    set_time(07,00,00);
    menu_awal(0,0) ;
    menu_alarm(0,0);
    mode1=jl_makanan() ;
    lcdInit();
    lcdGotoXY(0,0);
    sprintf(tampilc,"%d",mode1);
    lcd_puts(tampilc);

while (1)
     {
      get_time();
      cek_waktu((int)h,(int)m);
      cek_suhu(suhu1[5]);
      cek_opto_pelampung();
      delay_ms(990);
     };
}

void get_time(void)
{
 rtcGetTime(&h,&m,&s);
 lcdGotoXY(2,0); lcd_putchar(':');
 lcdGotoXY(5,0); lcd_putchar(':');

 if( (int)h < 10 )
  { lcdGotoXY(0,0); lcd_putchar('0');
    lcdGotoXY(1,0);
    sprintf(tampil1,"%d",h); lcd_puts(tampil1);
  }

 if( (int)h >= 10 )
  { lcdGotoXY(0,0);
    sprintf(tampil1,"%d",h); lcd_puts(tampil1);
  }

 if( (int)m < 10 )
  {lcdGotoXY(3,0); lcd_putchar('0');
    lcdGotoXY(4,0);
    sprintf(tampil1,"%d",m); lcd_puts(tampil1);
  }

 if( (int)m >= 10 )
```

```
  {lcdGotoXY(3,0);
   sprintf(tampil1,"%d",m); lcd_puts(tampil1);
  }

 if( (int)s < 10 )
  {lcdGotoXY(6,0); lcd_putchar('0');
   lcdGotoXY(7,0);
   sprintf(tampil1,"%d",s); lcd_puts(tampil1);
  }

 if( (int)s >= 10 )
  {lcdGotoXY(6,0);
   sprintf(tampil1,"%d",s); lcd_puts(tampil1);
  }
}

float get_adc(int d)
{
 float volt;
 read_adc(d);
 (float) volt = ((float)5*(float)(ADCH)/255) ;
 return volt;
}

void menu_awal(int x , int y)
{
 lcdInit();
 lcdGotoXY(x,y);
 lcd_putsf("Tugas Akhir");
 lcdGotoXY(0,1);
 lcd_putsf("Albert  0122162");
 delay_ms(1000);
 lcdInit();
}

void menu_alarm(int x1 , int y1)
{
 int jam=0,menit=0,tombol=0,suhu=25; int counter=0;
 int batas=0,batas_min;
 char tampil2[4] ;
 jam=0;menit=0;suhu=25;counter=1;
 lcdInit(); lcdGotoXY(x1,y1);
 lcd_putsf("Set Alarm  "); delay_ms(500);
 lcdGotoXY(x1,y1+1); lcd_putchar('0');
 lcdGotoXY(x1+1,y1+1); lcd_putchar('0');

 cari: {
       if(counter==1)
        { batas=24,batas_min=0; lcdGotoXY(x1,y1); lcd_putsf("Input Jam 1  "); }
```

```
        if(counter==2)
         { batas=60,batas_min=0; lcdGotoXY(x1,y1); lcd_putsf("Input Menit 1"); }

        if(counter==3)
         { batas=24,batas_min=0; lcdGotoXY(x1,y1); lcd_putsf("Input Jam 2  "); }

        if(counter==4)
         { batas=60,batas_min=0; lcdGotoXY(x1,y1); lcd_putsf("Input Menit 2"); }

tombol = keypad();
if(tombol==1)
 { if(jam < 10)
    { if(jam<=batas_min)
       { jam=batas;
         lcdGotoXY(x1,y1+1); lcd_putchar('0');
         lcdGotoXY(x1+1,y1+1); lcd_putchar('0');
         delay_ms(200); goto cari;
       }

      lcdGotoXY(x1,y1+1); lcd_putchar('0');
      lcdGotoXY(x1+1,y1+1);
      sprintf(tampil2,"%d",jam); lcd_puts(tampil2);
      jam=jam-1; delay_ms(200); goto cari ;
    }

   if(jam >= 10)
    { lcdGotoXY(x1,y1+1);
      sprintf(tampil2,"%d",jam); lcd_puts(tampil2);
      jam=jam-1; delay_ms(200); goto cari;
    }
 }

if(tombol==2)
 { if(jam < 10)
    { lcdGotoXY(x1,y1+1); lcd_putchar('0');
       lcdGotoXY(x1+1,y1+1);
      sprintf(tampil2,"%d",jam); lcd_puts(tampil2);
      jam=jam+1; delay_ms(200); goto cari;
    }

   if(jam >= 10)
    { if(jam==batas)
       { jam=batas_min;
         lcdGotoXY(x1,y1+1); lcd_putchar('0');
         lcdGotoXY(x1+1,y1+1); lcd_putchar('0');
       }

      lcdGotoXY(x1,y1+1);
```

```
     sprintf(tampil2,"%d",jam); lcd_puts(tampil2);
     jam=jam+1; delay_ms(200); goto cari;
    }
 }

if(tombol==3)
 { if(counter==1){ jam1[counter] = jam ; counter=counter+1 ; lcdInit();delay_ms(500); jam = 0;
   lcdGotoXY(x1,y1+1); lcd_putchar('0'); lcdGotoXY(x1+1,y1+1); lcd_putchar('0');goto cari;}
   if(counter==2){ menit1[counter] = jam ; counter=counter+1 ; lcdInit();delay_ms(500); jam = 0;
   lcdGotoXY(x1,y1+1); lcd_putchar('0'); lcdGotoXY(x1+1,y1+1); lcd_putchar('0');goto cari;}
   if(counter==3){ jam1[counter] = jam ; counter=counter+1 ; lcdInit();delay_ms(500); jam = 0;
   lcdGotoXY(x1,y1+1); lcd_putchar('0'); lcdGotoXY(x1+1,y1+1); lcd_putchar('0');goto cari ;}
   if(counter==4){ menit1[counter] = jam ; counter=counter+1 ; lcdInit();delay_ms(500); suhu= 25;
   lcdGotoXY(x1,y1+1); lcd_putchar('2'); lcdGotoXY(x1+1,y1+1); lcd_putchar('5');goto cari2;}
 }
     }

cari2: {
        if(counter==5)
         { batas=34,batas_min=25; lcdGotoXY(x1,y1); lcd_putsf("Set Suhu   "); }

tombol = keypad();
if(tombol==1)
 { if(suhu < batas_min)
    { suhu=batas;
      lcdGotoXY(x1,y1+1);
      sprintf(tampil2,"%d",suhu); lcd_puts(tampil2);
      suhu=suhu-1; delay_ms(200); goto cari2 ;
    }

   if(suhu >= batas_min)
    { lcdGotoXY(x1,y1+1); sprintf(tampil2,"%d",suhu);
      lcd_puts(tampil2); suhu=suhu-1;
      delay_ms(200); goto cari2;
    }
 }

if(tombol==2)
 { if(suhu <= batas)
    { lcdGotoXY(x1,y1+1);
      sprintf(tampil2,"%d",suhu); lcd_puts(tampil2);
      suhu=suhu+1; delay_ms(200); goto cari2;
    }

   if(suhu > batas)
    { suhu=batas_min; lcdGotoXY(x1,y1+1);
      sprintf(tampil2,"%d",suhu); lcd_puts(tampil2);
      suhu=suhu+1; delay_ms(200); goto cari2;
    }
```

```
   }

  if(tombol==3)
   { if(counter==5){ suhu1[counter] = suhu ; counter=counter+1 ;
     lcdInit(); delay_ms(500); goto run ;
   }
        }
       }
  run:{delay_ms(200);}
}

int jl_makanan(void)
{
  int tombol = 0 ;
  lcdInit(); lcdGotoXY(0,0);
  lcd_putsf("Mode Makanan"); delay_ms(500);
  lcdGotoXY(0,0); lcd_putsf("1 Dikit ");
  lcdGotoXY(8,0); lcd_putsf("2 Sedang");
  lcdGotoXY(0,1); lcd_putsf("3 Banyak");

  tombol = keypad();
  if (tombol==1){lama_makanan = lama1 ;}
  if (tombol==2){lama_makanan = lama2 ;}
  if (tombol==3){lama_makanan = lama3 ;}
  return lama_makanan;
}

void cek_waktu(int jam3,int menit3)
{
  if(count==0)
   { if(mark==0)
      { if((jam3==jam1[1]) && (menit3==menit1[2]))
         { gerak_motor1(); count=1;}

        if((jam3==jam1[3]) && (menit3==menit1[4]))
         { gerak_motor1(); count=1;}
      }

     if(mark==1)
      { if((jam3==jam1[1]) && (menit3==menit1[2]+1))
         { mark=0; goto run;}

        if((jam3==jam1[3]) && (menit3==menit1[4]+1))
         { mark=0; goto run;}
      }
   }

  if(count==1)
   { if((jam3==jam1[1]) && (menit3==menit1[2]))
```

```
     { gerak_motor2(); count=0; mark=1; goto run;}

    if((jam3==jam1[3]) && (menit3==menit1[4]))
     { gerak_motor2(); count=0; mark=1; goto run;}
   }

 run:{}
}

void cek_suhu(int suhu2)
{
 float volt2,suhu;
 char tampil3[4] ;
 volt2=get_adc(0);
 suhu=volt2*100;

 if((int)suhu<(int)(suhu2)) {out4=1;}
 if((int)suhu>(int)(suhu2)) {out4=0;}

 lcdGotoXY(0,1); lcd_putsf("Suhu = ");
 lcdGotoXY(7,1);
 sprintf(tampil3,"%f",suhu); lcd_puts(tampil3);
 lcdGotoXY(9,1); lcd_putsf("      ");
}

int keypad (void)
{
 int a=0,b=0,c=0 ;
 a=0;b=0;c=0;
 cari: {
        if ((PINB.5 == 0) || (PINB.6 == 0) || ( PINB.7 == 0))
         { if(PINB.6==0){b=b+2; return b;}
           if(PINB.7==0){c=c+3; return c;}
           if(PINB.5==0){a=a+1; return a;}
         }
        if ((PINB.5 == 1) || (PINB.6 == 1) || (PINB.7 == 1)) {goto cari;}
       }
}

void cek_opto_pelampung(void)
{
 if (PINB.3==0){out3=1;}
 if (PINB.4==0){out3=0;}
 return;
}

void set_time(char sjam,char smenit,char sdetik)
{ rtc_set_time( sjam,smenit, sdetik); }
```

```
int gerak_motor1(void)
{
 out1=1; delay_ms(300);
 cek:{
      if(PINB.2==1){out1=1; delay_ms(30); goto cek;}
      if(PINB.2==0){out1=0; delay_ms(lama_makanan);}
     }
 return;
}

int gerak_motor2(void)
{
 out2=1; delay_ms(300);
 cek:{
      if(PINB.2==1){out2=1; delay_ms(30); goto cek;}
      if(PINB.2==0){out2=0; delay_ms(200);}
     }
 return;
}
```

LCD.h

```
#ifndef LCD_H
#define LCD_H
#include "global.h"
#include "lcdconf.h"
#ifndef LCD_DELAY
#if F_CPU >= 11059200
#define LCD_DELAY      asm volatile ("nop\n nop\n nop\n nop\n nop\n nop\n nop\n nop\n nop\n nop\n nop\n nop\n nop\n nop\n nop\n");
#else
#if F_CPU >= 12000000
#define LCD_DELAY      asm volatile ("nop\n nop\n nop\n nop\n nop\n nop\n nop\n nop\n nop\n nop\n nop\n");
#else
#if F_CPU >= 8000000
#define LCD_DELAY      asm volatile ("nop\n nop\n nop\n nop\n nop\n nop\n nop\n nop\n");
#else
#if F_CPU >= 4000000
#define LCD_DELAY      asm volatile ("nop\n nop\n nop\n nop\n nop\n");
#else
#define LCD_DELAY      asm volatile ("nop\n nop\n nop\n");
#endif
#endif
#endif
#endif
#endif

#define LCD_CLR              0     // DB0: clear display
#define LCD_HOME             1     // DB1: return to home position
#define LCD_ENTRY_MODE       2     // DB2: set entry mode
#define LCD_ENTRY_INC        1     // DB1: increment
#define LCD_ENTRY_SHIFT      0     //   DB2: shift
#define LCD_ON_CTRL          3     // DB3: turn lcd/cursor on
#define LCD_ON_DISPLAY       2     //   DB2: turn display on
#define LCD_ON_CURSOR        1     //   DB1: turn cursor on
#define LCD_ON_BLINK         0     //   DB0: blinking cursor
#define LCD_MOVE             4     // DB4: move cursor/display
#define LCD_MOVE_DISP        3     //   DB3: move display (0-> move cursor)
#define LCD_MOVE_RIGHT       2     //   DB2: move right (0-> left)
#define LCD_FUNCTION         5     // DB5: function set
#define LCD_FUNCTION_8BIT    4     //   DB4: set 8BIT mode (0->4BIT mode)
#define LCD_FUNCTION_2LINES  3     //   DB3: two lines (0->one line)
#define LCD_FUNCTION_10DOTS  2     //   DB2: 5x10 font (0->5x7 font)
#define LCD_CGRAM            6     // DB6: set CG RAM address
#define LCD_DDRAM            7     // DB7: set DD RAM address
#define LCD_BUSY             7     // DB7: LCD is busy

#ifdef LCD_DATA_4BIT
#define LCD_FDEF_1                   (0<<LCD_FUNCTION_8BIT)
```

```
#else
#define LCD_FDEF_1                  (1<<LCD_FUNCTION_8BIT)
#endif
#define LCD_FDEF_2                      (1<<LCD_FUNCTION_2LINES)
#define LCD_FUNCTION_DEFAULT    ((1<<LCD_FUNCTION)  |  LCD_FDEF_1  |
LCD_FDEF_2)
#define LCD_MODE_DEFAULT        ((1<<LCD_ENTRY_MODE)                |
(1<<LCD_ENTRY_INC))

extern unsigned char __attribute__ ((progmem)) LcdCustomChar[];
#define LCDCHAR_PROGRESS05              0
#define LCDCHAR_PROGRESS15              1
#define LCDCHAR_PROGRESS25              2
#define LCDCHAR_PROGRESS35              3
#define LCDCHAR_PROGRESS45              4
#define LCDCHAR_PROGRESS55              5
#define LCDCHAR_REWINDARROW             6
#define LCDCHAR_STOPBLOCK       7
#define LCDCHAR_PAUSEBARS       8
#define LCDCHAR_FORWARDARROW    9
#define LCDCHAR_SCROLLUPARROW   10
#define LCDCHAR_SCROLLDNARROW   11
#define LCDCHAR_BLANK           12
#define LCDCHAR_ANIPLAYICON0    13
#define LCDCHAR_ANIPLAYICON1    14
#define LCDCHAR_ANIPLAYICON2    15
#define LCDCHAR_ANIPLAYICON3    16
#define PROGRESSPIXELS_PER_CHAR 6

void lcdInitHW(void);
void lcdBusyWait(void);
void lcdControlWrite(u08 data);
u08 lcdControlRead(void);
void lcdDataWrite(u08 data);
u08 lcdDataRead(void);

void lcdInit(void);
void lcdHome(void);
void lcdClear(void);
void lcdGotoXY(u08 row, u08 col);
void lcdLoadCustomChar(u08* lcdCustomCharArray, u08 romCharNum, u08 lcdCharNum);
void lcdPrintData(char* data, u08 nBytes);
void lcdProgressBar(u16 progress, u16 maxprogress, u08 length);

#endif
```

LCD.c

```
#include <avr/io.h>
#include <avr/pgmspace.h>
#include "global.h"
#include "timer.h"
#include "lcd.h"

unsigned char __attribute__ ((progmem)) LcdCustomChar[] =
{
        0x00, 0x1F, 0x00, 0x00, 0x00, 0x00, 0x1F, 0x00, // 0. 0/5 full progress block
        0x00, 0x1F, 0x10, 0x10, 0x10, 0x10, 0x1F, 0x00, // 1. 1/5 full progress block
        0x00, 0x1F, 0x18, 0x18, 0x18, 0x18, 0x1F, 0x00, // 2. 2/5 full progress block
        0x00, 0x1F, 0x1C, 0x1C, 0x1C, 0x1C, 0x1F, 0x00, // 3. 3/5 full progress block
        0x00, 0x1F, 0x1E, 0x1E, 0x1E, 0x1E, 0x1F, 0x00, // 4. 4/5 full progress block
        0x00, 0x1F, 0x1F, 0x1F, 0x1F, 0x1F, 0x1F, 0x00, // 5. 5/5 full progress block
        0x03, 0x07, 0x0F, 0x1F, 0x0F, 0x07, 0x03, 0x00, // 6. rewind arrow
        0x00, 0x1F, 0x1F, 0x1F, 0x1F, 0x1F, 0x1F, 0x00, // 7. stop block
        0x1B, 0x1B, 0x1B, 0x1B, 0x1B, 0x1B, 0x1B, 0x00, // 8. pause bars
        0x18, 0x1C, 0x1E, 0x1F, 0x1E, 0x1C, 0x18, 0x00, // 9. fast-forward arrow
        0x00, 0x04, 0x04, 0x0E, 0x0E, 0x1F, 0x1F, 0x00, // 10. scroll up arrow
        0x00, 0x1F, 0x1F, 0x0E, 0x0E, 0x04, 0x04, 0x00, // 11. scroll down arrow
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // 12. blank character
        0x00, 0x0E, 0x19, 0x15, 0x13, 0x0E, 0x00, 0x00,     // 13. animated play icon frame 0
        0x00, 0x0E, 0x15, 0x15, 0x15, 0x0E, 0x00, 0x00,     // 14. animated play icon frame 1
        0x00, 0x0E, 0x13, 0x15, 0x19, 0x0E, 0x00, 0x00,     // 15. animated play icon frame 2
        0x00, 0x0E, 0x11, 0x1F, 0x11, 0x0E, 0x00, 0x00,     // 16. animated play icon frame 3
};

void lcdInitHW(void)
{

#ifdef LCD_PORT_INTERFACE

        cbi(LCD_CTRL_PORT, LCD_CTRL_RS);
        cbi(LCD_CTRL_PORT, LCD_CTRL_RW);
        cbi(LCD_CTRL_PORT, LCD_CTRL_E);

        sbi(LCD_CTRL_DDR, LCD_CTRL_RS);
        sbi(LCD_CTRL_DDR, LCD_CTRL_RW);
        sbi(LCD_CTRL_DDR, LCD_CTRL_E);

        #ifdef LCD_DATA_4BIT
                outb(LCD_DATA_DDR, inb(LCD_DATA_DDR)&0x0F);
                outb(LCD_DATA_POUT, inb(LCD_DATA_POUT)|0xF0);
        #else
                outb(LCD_DATA_DDR, 0x00);
                outb(LCD_DATA_POUT, 0xFF);
        #endif
#else
```

```
        sbi(MCUCR, SRE);
#endif
}

void lcdBusyWait(void)
{

#ifdef LCD_PORT_INTERFACE
        cbi(LCD_CTRL_PORT, LCD_CTRL_RS);
        #ifdef LCD_DATA_4BIT
                outb(LCD_DATA_DDR, inb(LCD_DATA_DDR)&0x0F);
                outb(LCD_DATA_POUT, inb(LCD_DATA_POUT)|0xF0);
        #else
                outb(LCD_DATA_DDR, 0x00);
                outb(LCD_DATA_POUT, 0xFF);
        #endif
        sbi(LCD_CTRL_PORT, LCD_CTRL_RW);
        sbi(LCD_CTRL_PORT, LCD_CTRL_E);
        LCD_DELAY;
        while(inb(LCD_DATA_PIN) & 1<<LCD_BUSY)
        {
                cbi(LCD_CTRL_PORT, LCD_CTRL_E);
                LCD_DELAY;

                LCD_DELAY;

                sbi(LCD_CTRL_PORT, LCD_CTRL_E);
                LCD_DELAY;

                LCD_DELAY;

                #ifdef LCD_DATA_4BIT
                        cbi(LCD_CTRL_PORT, LCD_CTRL_E);
                        LCD_DELAY;

                        LCD_DELAY;

                        sbi(LCD_CTRL_PORT, LCD_CTRL_E);
                        LCD_DELAY;

                        LCD_DELAY;

                #endif
        }
        cbi(LCD_CTRL_PORT, LCD_CTRL_E);

#else
        while( (*((volatile unsigned char *) (LCD_CTRL_ADDR))) & (1<<LCD_BUSY) );
```

```
        #endif
}

void lcdControlWrite(u08 data)
{
#ifdef LCD_PORT_INTERFACE
        lcdBusyWait();
        cbi(LCD_CTRL_PORT, LCD_CTRL_RS);
        cbi(LCD_CTRL_PORT, LCD_CTRL_RW);
        #ifdef LCD_DATA_4BIT

                sbi(LCD_CTRL_PORT, LCD_CTRL_E);
                outb(LCD_DATA_DDR, inb(LCD_DATA_DDR)|0xF0);
                outb(LCD_DATA_POUT, (inb(LCD_DATA_POUT)&0x0F) | (data&0xF0) );
                LCD_DELAY;
                LCD_DELAY;
                cbi(LCD_CTRL_PORT, LCD_CTRL_E);
                LCD_DELAY;
                LCD_DELAY;
                sbi(LCD_CTRL_PORT, LCD_CTRL_E);
                outb(LCD_DATA_POUT, (inb(LCD_DATA_POUT)&0x0F) | (data<<4) );
                LCD_DELAY;
                LCD_DELAY;
                cbi(LCD_CTRL_PORT, LCD_CTRL_E);
        #else

                sbi(LCD_CTRL_PORT, LCD_CTRL_E);
                outb(LCD_DATA_DDR, 0xFF);
                outb(LCD_DATA_POUT, data);
                LCD_DELAY;
                LCD_DELAY;
                cbi(LCD_CTRL_PORT, LCD_CTRL_E);
        #endif

        #ifdef LCD_DATA_4BIT
                outb(LCD_DATA_DDR, inb(LCD_DATA_DDR)&0x0F);
                outb(LCD_DATA_POUT, inb(LCD_DATA_POUT)|0xF0);
        #else
                outb(LCD_DATA_DDR, 0x00);
                outb(LCD_DATA_POUT, 0xFF);
        #endif
#else

        lcdBusyWait();
        *((volatile unsigned char *) (LCD_CTRL_ADDR)) = data;

#endif
}
```

```
u08 lcdControlRead(void)
{

	register u08 data;
#ifdef LCD_PORT_INTERFACE
	lcdBusyWait();
	#ifdef LCD_DATA_4BIT
		outb(LCD_DATA_DDR, inb(LCD_DATA_DDR)&0x0F);
		outb(LCD_DATA_POUT, inb(LCD_DATA_POUT)|0xF0);
	#else
		outb(LCD_DATA_DDR, 0x00);
		outb(LCD_DATA_POUT, 0xFF);
	#endif
	cbi(LCD_CTRL_PORT, LCD_CTRL_RS);
	sbi(LCD_CTRL_PORT, LCD_CTRL_RW);
	#ifdef LCD_DATA_4BIT

		sbi(LCD_CTRL_PORT, LCD_CTRL_E);
		LCD_DELAY;
		LCD_DELAY;
		data = inb(LCD_DATA_PIN)&0xF0;
		cbi(LCD_CTRL_PORT, LCD_CTRL_E);
		LCD_DELAY;
		LCD_DELAY;
		sbi(LCD_CTRL_PORT, LCD_CTRL_E);
		LCD_DELAY;
		LCD_DELAY;
		data |= inb(LCD_DATA_PIN)>>4;
		cbi(LCD_CTRL_PORT, LCD_CTRL_E);
	#else

		sbi(LCD_CTRL_PORT, LCD_CTRL_E);
		LCD_DELAY;
		LCD_DELAY;
		data = inb(LCD_DATA_PIN);
		cbi(LCD_CTRL_PORT, LCD_CTRL_E);
	#endif

#else

	lcdBusyWait();
	data = *((volatile unsigned char *) (LCD_CTRL_ADDR));

#endif
	return data;
}

void lcdDataWrite(u08 data)
{
```

```
#ifdef LCD_PORT_INTERFACE
        lcdBusyWait();
        sbi(LCD_CTRL_PORT, LCD_CTRL_RS);
        cbi(LCD_CTRL_PORT, LCD_CTRL_RW);
        #ifdef LCD_DATA_4BIT

                sbi(LCD_CTRL_PORT, LCD_CTRL_E);
                outb(LCD_DATA_DDR, inb(LCD_DATA_DDR)|0xF0);
                outb(LCD_DATA_POUT, (inb(LCD_DATA_POUT)&0x0F) | (data&0xF0) );
                LCD_DELAY;
                LCD_DELAY;
                cbi(LCD_CTRL_PORT, LCD_CTRL_E);
                LCD_DELAY;
                LCD_DELAY;
                sbi(LCD_CTRL_PORT, LCD_CTRL_E);
                outb(LCD_DATA_POUT, (inb(LCD_DATA_POUT)&0x0F) | (data<<4) );
                LCD_DELAY;
                LCD_DELAY;
                cbi(LCD_CTRL_PORT, LCD_CTRL_E);
        #else

                sbi(LCD_CTRL_PORT, LCD_CTRL_E);
                outb(LCD_DATA_DDR, 0xFF);
                outb(LCD_DATA_POUT, data);
                LCD_DELAY;
                LCD_DELAY;
                cbi(LCD_CTRL_PORT, LCD_CTRL_E);
        #endif

        #ifdef LCD_DATA_4BIT
                outb(LCD_DATA_DDR, inb(LCD_DATA_DDR)&0x0F);
                outb(LCD_DATA_POUT, inb(LCD_DATA_POUT)|0xF0);
        #else
                outb(LCD_DATA_DDR, 0x00);
                outb(LCD_DATA_POUT, 0xFF);
        #endif
#else
                lcdBusyWait();
        *((volatile unsigned char *) (LCD_DATA_ADDR)) = data;
#endif
}

u08 lcdDataRead(void)
{

        register u08 data;
#ifdef LCD_PORT_INTERFACE
        lcdBusyWait();
```

```
        #ifdef LCD_DATA_4BIT
                outb(LCD_DATA_DDR, inb(LCD_DATA_DDR)&0x0F);
                outb(LCD_DATA_POUT, inb(LCD_DATA_POUT)|0xF0);
        #else
                outb(LCD_DATA_DDR, 0x00);
                outb(LCD_DATA_POUT, 0xFF);
        #endif
        sbi(LCD_CTRL_PORT, LCD_CTRL_RS);
        sbi(LCD_CTRL_PORT, LCD_CTRL_RW);
        #ifdef LCD_DATA_4BIT

                sbi(LCD_CTRL_PORT, LCD_CTRL_E);
                LCD_DELAY;
                LCD_DELAY;
                data = inb(LCD_DATA_PIN)&0xF0;
                cbi(LCD_CTRL_PORT, LCD_CTRL_E);
                LCD_DELAY;
                LCD_DELAY;
                sbi(LCD_CTRL_PORT, LCD_CTRL_E);
                LCD_DELAY;
                LCD_DELAY;
                data |= inb(LCD_DATA_PIN)>>4;
                cbi(LCD_CTRL_PORT, LCD_CTRL_E);
        #else

                sbi(LCD_CTRL_PORT, LCD_CTRL_E);
                LCD_DELAY;
                LCD_DELAY;
                data = inb(LCD_DATA_PIN);
                cbi(LCD_CTRL_PORT, LCD_CTRL_E);
        #endif

#else
        lcdBusyWait();
        data = *((volatile unsigned char *) (LCD_DATA_ADDR));

#endif
        return data;
}

void lcdInit()
{
        lcdInitHW();
        lcdControlWrite(LCD_FUNCTION_DEFAULT);
        lcdControlWrite(1<<LCD_CLR);
        delay(60000);      // wait 60ms
        lcdControlWrite(1<<LCD_ENTRY_MODE | 1<<LCD_ENTRY_INC);
        1<<LCD_ON_BLINK);
        lcdControlWrite(1<<LCD_ON_CTRL | 1<<LCD_ON_DISPLAY );
```

```
	lcdControlWrite(1<<LCD_HOME);
	lcdControlWrite(1<<LCD_DDRAM | 0x00);

	lcdLoadCustomChar((u08*)LcdCustomChar,0,0);
	lcdLoadCustomChar((u08*)LcdCustomChar,1,1);
	lcdLoadCustomChar((u08*)LcdCustomChar,2,2);
	lcdLoadCustomChar((u08*)LcdCustomChar,3,3);
	lcdLoadCustomChar((u08*)LcdCustomChar,4,4);
	lcdLoadCustomChar((u08*)LcdCustomChar,5,5);
	lcdLoadCustomChar((u08*)LcdCustomChar,6,6);
	lcdLoadCustomChar((u08*)LcdCustomChar,7,7);
}

void lcdHome(void)
{
	lcdControlWrite(1<<LCD_HOME);
}

void lcdClear(void)
{
	lcdControlWrite(1<<LCD_CLR);
}

void lcdGotoXY(u08 x, u08 y)
{
	register u08 DDRAMAddr;

	switch(y)
	{
	case 0: DDRAMAddr = LCD_LINE0_DDRAMADDR+x; break;
	case 1: DDRAMAddr = LCD_LINE1_DDRAMADDR+x; break;
	case 2: DDRAMAddr = LCD_LINE2_DDRAMADDR+x; break;
	case 3: DDRAMAddr = LCD_LINE3_DDRAMADDR+x; break;
	default: DDRAMAddr = LCD_LINE0_DDRAMADDR+x;
	}

	lcdControlWrite(1<<LCD_DDRAM | DDRAMAddr);
}

void lcdLoadCustomChar(u08* lcdCustomCharArray, u08 romCharNum, u08 lcdCharNum)
{
	register u08 i;
	u08 saveDDRAMAddr;

	saveDDRAMAddr = lcdControlRead() & 0x7F;

	lcdCharNum = (lcdCharNum<<3);
	romCharNum = (romCharNum<<3);
```

```
        for(i=0; i<8; i++)
        {
                lcdControlWrite((1<<LCD_CGRAM) | (lcdCharNum+i));
                lcdDataWrite( pgm_read_byte(lcdCustomCharArray+romCharNum+i) );
        }

        lcdControlWrite(1<<LCD_DDRAM | saveDDRAMAddr);

}

void lcdPrintData(char* data, u08 nBytes)
{
        register u08 i;

        if (!data) return;

        for(i=0; i<nBytes; i++)
        {
                lcdDataWrite(data[i]);
        }
}

void lcdProgressBar(u16 progress, u16 maxprogress, u08 length)
{
        u08 i;
        u32 pixelprogress;
        u08 c;

        pixelprogress = ((progress*(length*PROGRESSPIXELS_PER_CHAR))/maxprogress);
        for(i=0; i<length; i++)
        {
                if( ((i*(u16)PROGRESSPIXELS_PER_CHAR)+5) > pixelprogress )
                {
                        if( ((i*(u16)PROGRESSPIXELS_PER_CHAR)) > pixelprogress )
                        {
                                c = 0;
                        }
                        else
                        {
                                c = pixelprogress % PROGRESSPIXELS_PER_CHAR;
                        }
                }
                else
                {
                        c = 5;
                }
                lcdDataWrite(c);
        }
}
```

RTC.h

```
#ifndef RTC_H
#define RTC_H
#include "global.h"

typedef struct struct_RtcTime
{
        u08 tics;
        u16 totaltics;
        u08 hours;
        u08 minutes;
        u08 seconds;
        u08 day;
        u08 month;
        u16 year;
} RtcTimeType;

void rtcInit(void);
void rtcService(void);
RtcTimeType* rtcGetTime(void);

#endif
```

RTC.c

```
#ifndef WIN32
	#include <avr/io.h>
	#include <avr/interrupt.h>
	#include <avr/pgmspace.h>
#endif

#include "global.h"
#ifdef __AVR_ATmega16__
	#include "timer16.h"
#else
	#include "timer.h"
#endif
#include "rtc.h"

static char __attribute__ ((progmem)) MonthDayTable[] = {31,28,31,30,31,30,31,31,30,31,30,31};

RtcTimeType RtcTime;

void rtcInit(void)
{
	RtcTime.totaltics = 0;
	RtcTime.tics = 0;
	RtcTime.seconds = 0;
	RtcTime.minutes = 0;
	RtcTime.hours = 0;
	RtcTime.day = 1;
	RtcTime.month = 1;
	RtcTime.year = 2000;

	#ifdef AS2
		timer2Init();
		timer2SetPrescaler(TIMER_CLK_DIV8);
		sbi(ASSR, AS2);
		timerAttach(TIMER2OVERFLOW_INT, rtcService);
	#else
	#ifdef AS0
		timer0Init();
		timer0SetPrescaler(TIMER_CLK_DIV8);
		sbi(ASSR, AS0);
		timerAttach(TIMER0OVERFLOW_INT, rtcService);
	#endif
	#endif
}

void rtcService(void)
{
	RtcTime.totaltics++;
	RtcTime.tics++;
```

```
	if(RtcTime.tics == 16)
	{
		RtcTime.tics = 0;
		RtcTime.seconds++;
		if(RtcTime.seconds > 59)
		{
			RtcTime.seconds -= 60;
			RtcTime.minutes++;
			if(RtcTime.minutes > 59)
			{
				RtcTime.minutes -= 60;
				RtcTime.hours++;
				if(RtcTime.hours > 23)
				{
					RtcTime.hours -= 24;
					RtcTime.day++;

					if(RtcTime.day                                    ==
pgm_read_byte(&MonthDayTable[RtcTime.month-1]))
					{
						RtcTime.day = 1;
						RtcTime.month++;
						if(RtcTime.month == 13)
						{
							RtcTime.month = 1;
							RtcTime.year++;
						}
					}
				}
			}
		}
	}
}

RtcTimeType* rtcGetTime(void)
{

	return &RtcTime;
}
```

LAMPIRAN C

DALLAS SEMICONDUCTOR

DS1307
64 X 8 Serial Real Time Clock

www.dalsemi.com

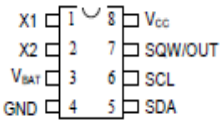
FEATURES

- Real time clock counts seconds, minutes, hours, date of the month, month, day of the week, and year with leap year compensation valid up to 2100
- 56 byte nonvolatile RAM for data storage
- 2-wire serial interface
- Programmable squarewave output signal
- Automatic power-fail detect and switch circuitry
- Consumes less than 500 nA in battery backup mode with oscillator running
- Optional industrial temperature range -40°C to +85°C
- Available in 8-pin DIP or SOIC
- Recognized by Underwriters Laboratory

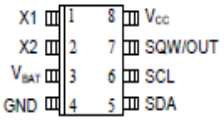
ORDERING INFORMATION

DS1307	8-Pin DIP
DS1307Z	8-Pin SOIC (150 mil)
DS1307N	8-Pin DIP (Industrial)
DS1307ZN	8-Pin SOIC (Industrial)

PIN ASSIGNMENT



DS1307 8-Pin DIP (300 mil)



DS1307Z 8-Pin SOIC (150 mil)

PIN DESCRIPTION

V_{CC}	- Primary Power Supply
X1, X2	- 32.768 kHz Crystal Connection
V_{BAT}	- +3V Battery Input
GND	- Ground
SDA	- Serial Data
SCL	- Serial Clock
SQW/OUT	- Square wave/Output Driver

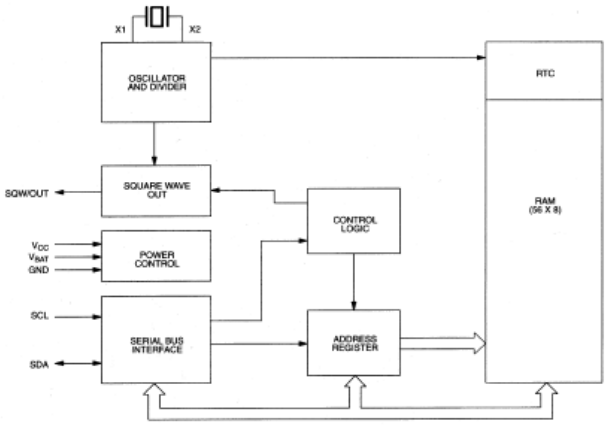
DESCRIPTION

The DS1307 Serial Real Time Clock is a low power, full BCD clock/calendar plus 56 bytes of nonvolatile SRAM. Address and data are transferred serially via a 2-wire bi-directional bus. The clock/calendar provides seconds, minutes, hours, day, date, month, and year information. The end of the month date is automatically adjusted for months with less than 31 days, including corrections for leap year. The clock operates in either the 24-hour or 12-hour format with AM/PM indicator. The DS1307 has a built-in power sense circuit which detects power failures and automatically switches to the battery supply.

OPERATION

The DS1307 operates as a slave device on the serial bus. Access is obtained by implementing a START condition and providing a device identification code followed by a register address. Subsequent registers can be accessed sequentially until a STOP condition is executed. When V_{CC} falls below 1.25 x V_{BAT} the device terminates an access in progress and resets the device address counter. Inputs to the device will not be recognized at this time to prevent erroneous data from being written to the device from an out of tolerance system. When V_{CC} falls below V_{BAT} the device switches into a low current battery backup mode. Upon power up, the device switches from battery to V_{CC} when V_{CC} is greater than V_{BAT}+0.2V and recognizes inputs when V_{CC} is greater than 1.25 x V_{BAT}. The block diagram in Figure 1 shows the main elements of the Serial Real Time Clock.

DS1307 BLOCK DIAGRAM Figure 1



SIGNAL DESCRIPTIONS

V_{CC}, GND - DC power is provided to the device on these pins. V_{CC} is the +5 volt input. When 5 volts is applied within normal limits, the device is fully accessible and data can be written and read. When a 3-volt battery is connected to the device and V_{CC} is below 1.25 x V_{BAT}, reads and writes are inhibited. However, the Timekeeping function continues unaffected by the lower input voltage. As V_{CC} falls below V_{BAT} the RAM and timekeeper are switched over to the external power supply (nominal 3.0V DC) at V_{BAT}.

V_{BAT} - Battery input for any standard 3-volt lithium cell or other energy source. Battery voltage must be held between 2.0 and 3.5 volts for proper operation. The nominal write protect trip point voltage at which access to the real time clock and user RAM is denied is set by the internal circuitry as 1.25 x V_{BAT} nominal. A lithium battery with 48 mAhr or greater will back up the DS1307 for more than 10 years in the absence of power at 25 degrees C.

SCL (Serial Clock Input) - SCL is used to synchronize data movement on the serial interface.

SDA (Serial Data Input/Output) - SDA is the input/output pin for the 2-wire serial interface. The SDA pin is open drain which requires an external pullup resistor.

SQW/OUT (Square Wave/ Output Driver) - When enabled, the SQWE bit set to 1, the SQW/OUT pin outputs one of four square wave frequencies (1 Hz, 4 kHz, 8 kHz, 32 kHz). The SQW/OUT pin is open drain which requires an external pullup resistor. SQW/OUT will operate with either Vcc or Vbat applied.

X1, X2 - Connections for a standard 32.768 kHz quartz crystal. The internal oscillator circuitry is designed for operation with a crystal having a specified load capacitance (CL) of 12.5 pF.

For more information on crystal selection and crystal layout considerations, please consult Application Note 58, "Crystal Considerations with Dallas Real Time Clocks." The DS1307 can also be driven by an external 32.768 kHz oscillator. In this configuration, the X1 pin is connected to the external oscillator signal and the X2 pin is floated.

Please review Application Note 95, "Interfacing the DS1307 with a 8051-Compatible Microcontroller" for additional information.

RTC AND RAM ADDRESS MAP

The address map for the RTC and RAM registers of the DS1307 is shown in Figure 2. The real time clock registers are located in address locations 00h to 07h. The RAM registers are located in address locations 08h to 3Fh. During a multi-byte access, when the address pointer reaches 3Fh, the end of RAM space, it wraps around to location 00h, the beginning of the clock space.

DS1307 ADDRESS MAP Figure 2

00H	SECONDS
	MINUTES
	HOURS
	DAY
	DATE
	MONTH
	YEAR
07H	CONTROL
08H	RAM 56 x 8
3FH	

CLOCK AND CALENDAR

The time and calendar information is obtained by reading the appropriate register bytes. The real time clock registers are illustrated in Figure 3. The time and calendar are set or initialized by writing the appropriate register bytes. The contents of the time and calendar registers are in the Binary-Coded Decimal (BCD) format. Bit 7 of Register 0 is the Clock Halt (CH) bit. When this bit is set to a 1, the oscillator is disabled. When cleared to a 0, the oscillator is enabled.

Please note that the initial power on state of all registers is not defined. Therefore it is important to enable the oscillator (CH bit=0) during initial configuration.

The DS1307 can be run in either 12-hour or 24-hour mode. Bit 6 of the hours register is defined as the 12- or 24-hour mode select bit. When high, the 12-hour mode is selected. In the 12-hour mode, bit 5 is the AM/PM bit with logic high being PM. In the 24-hour mode, bit 5 is the second 10 hour bit (20-23 hours).

On a 2-wire START, the current time is transferred to a second set of registers. The time information is read from these secondary registers, while the clock may continue to run. This eliminates the need to re-read the registers in case of an update of the main registers during a read.

DS1307 TIMEKEEPER REGISTERS Figure 3

	BIT7							BIT0		
00H	CH	10 SECONDS			SECONDS					00–59
	X	10 MINUTES			MINUTES					00–59
	X	12/24	10 HR A/P	10 HR	HOURS					01–12 / 00–23
	X	X	X	X	X	DAY				1–7
	X	X	10 DATE		DATE					01–28/29 / 01–30 / 01–31
	X	X	X	10 MONTH	MONTH					01–12
	10 YEAR				YEAR					00–99
07H	OUT	X	X	SQWE	X	X	RS1	RS0		

CONTROL REGISTER

The DS1307 Control Register is used to control the operation of the SQW/OUT pin.

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
OUT	X	X	SQWE	X	X	RS1	RS0

OUT (Output control): This bit controls the output level of the SQW/OUT pin when the square wave output is disabled. If SQWE=0, the logic level on the SQW/OUT pin is 1 if OUT=1 and is 0 if OUT=0.

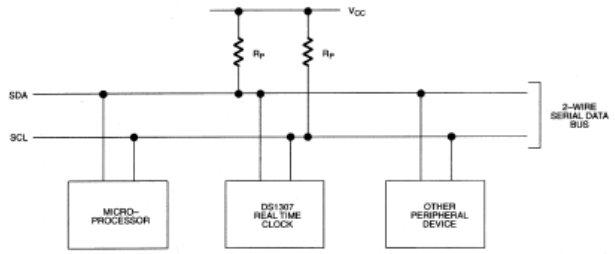
SQWE (Square Wave Enable): This bit, when set to a logic 1, will enable the oscillator output. The frequency of the square wave output depends upon the value of the RS0 and RS1 bits.

RS (Rate Select): These bits control the frequency of the square wave output when the square wave output has been enabled. Table 1 lists the square wave frequencies that can be selected with the RS bits.

2-WIRE SERIAL DATA BUS

The DS1307 supports a bi-directional 2-wire bus and data transmission protocol. A device that sends data onto the bus is defined as a transmitter and a device receiving data as a receiver. The device that controls the message is called a master. The devices that are controlled by the master are referred to as slaves. The bus must be controlled by a master device which generates the serial clock (SCL), controls the bus access, and generates the START and STOP conditions. The DS1307 operates as a slave on the 2-wire bus. A typical bus configuration using this 2-wire protocol is show in Figure 4.

TYPICAL 2-WIRE BUS CONFIGURATION Figure 4



Figures 5, 6, and 7 detail how data is transferred on the 2-wire bus.

- Data transfer may be initiated only when the bus is not busy.
- During data transfer, the data line must remain stable whenever the clock line is HIGH. Changes in the data line while the clock line is high will be interpreted as control signals.

Accordingly, the following bus conditions have been defined:

Bus not busy: Both data and clock lines remain HIGH.

Start data transfer: A change in the state of the data line, from HIGH to LOW, while the clock is HIGH, defines a START condition.

Stop data transfer: A change in the state of the data line, from LOW to HIGH, while the clock line is HIGH, defines the STOP condition.

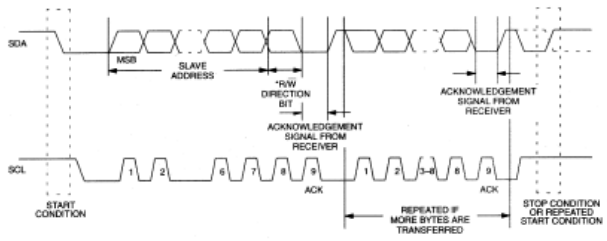
Data valid: The state of the data line represents valid data when, after a START condition, the data line is stable for the duration of the HIGH period of the clock signal. The data on the line must be changed during the LOW period of the clock signal. There is one clock pulse per bit of data.

Each data transfer is initiated with a START condition and terminated with a STOP condition. The number of data bytes transferred between START and STOP conditions is not limited, and is determined by the master device. The information is transferred byte-wise and each receiver acknowledges with a ninth bit. Within the 2-wire bus specifications a regular mode (100 kHz clock rate) and a fast mode (400 kHz clock rate) are defined. The DS1307 operates in the regular mode (100 kHz) only.

Acknowledge: Each receiving device, when addressed, is obliged to generate an acknowledge after the reception of each byte. The master device must generate an extra clock pulse which is associated with this acknowledge bit.

A device that acknowledges must pull down the SDA line during the acknowledge clock pulse in such a way that the SDA line is stable LOW during the HIGH period of the acknowledge related clock pulse. Of course, setup and hold times must be taken into account. A master must signal an end of data to the slave by not generating an acknowledge bit on the last byte that has been clocked out of the slave. In this case, the slave must leave the data line HIGH to enable the master to generate the STOP condition.

DATA TRANSFER ON 2-WIRE SERIAL BUS Figure 5



Depending upon the state of the R/$\overline{W}$ bit, two types of data transfer are possible:

1. **Data transfer from a master transmitter to a slave receiver.** The first byte transmitted by the master is the slave address. Next follows a number of data bytes. The slave returns an acknowledge bit after each received byte. Data is transferred with the most significant bit (MSB) first.

2. **Data transfer from a slave transmitter to a master receiver.** The first byte (the slave address) is transmitted by the master. The slave then returns an acknowledge bit. This is followed by the slave transmitting a number of data bytes. The master returns an acknowledge bit after all received bytes other than the last byte. At the end of the last received byte, a 'not acknowledge' is returned.

The master device generates all of the serial clock pulses and the START and STOP conditions. A transfer is ended with a STOP condition or with a repeated START condition. Since a repeated START condition is also the beginning of the next serial transfer, the bus will not be released. Data is transferred with the most significant bit (MSB) first.

The DS1307 may operate in the following two modes:

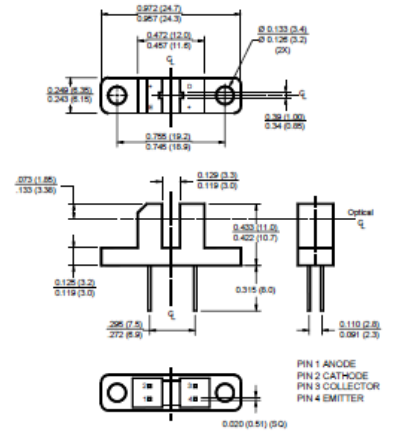
1. **Slave receiver mode (DS1307 write mode):** Serial data and clock are received through SDA and SCL. After each byte is received an acknowledge bit is transmitted. START and STOP conditions are recognized as the beginning and end of a serial transfer. Address recognition is performed by hardware after reception of the slave address and *direction bit (See Figure 6). The address byte is the first byte received after the start condition is generated by the master. The address byte contains the 7 bit DS1307 address, which is 1101000, followed by the *direction bit (R/$\overline{W}$) which, for a write, is a 0. After receiving and decoding the address byte the device outputs an acknowledge on the SDA line. After the DS1307 acknowledges the slave address + write bit, the master transmits a register address to the DS1307 This will set the register pointer on the DS1307. The master will then begin transmitting each byte of data with the DS1307 acknowledging each byte received. The master will generate a stop condition to terminate the data write.

2. **Slave transmitter mode (DS1307 read mode):** The first byte is received and handled as in the slave receiver mode. However, in this mode, the *direction bit will indicate that the transfer direction is reversed. Serial data is transmitted on SDA by the DS1307 while the serial clock is input on SCL. START and STOP conditions are recognized as the beginning and end of a serial transfer (See Figure 7). The address byte is the first byte received after the start condition is generated by the master. The address byte contains the 7-bit DS1307 address, which is 1101000, followed by the *direction bit (R/$\overline{W}$) which, for a read, is a 1. After receiving and decoding the address byte the device inputs an acknowledge on the SDA line. The DS1307 then begins to transmit data starting with the register address pointed to by the register pointer. If the register pointer is not written to before the initiation of a read mode the first address that is read is the last one stored in the register pointer. The DS1307 must receive a Not Acknowledge to end a read.

FAIRCHILD
SEMICONDUCTOR®

H21A1 / H21A2 / H21A3
PHOTOTRANSISTOR
OPTICAL INTERRUPTER SWITCH

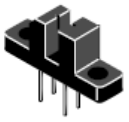
PACKAGE DIMENSIONS



NOTES:

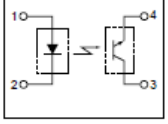
1. Dimensions for all drawings are in inches (mm).
2. Tolerance of ± .010 (.25) on all non-nominal dimensions unless otherwise specified.

DESCRIPTION



The H21A1, H21A2 and H21A3 consist of a gallium arsenide infrared emitting diode coupled with a silicon phototransistor in a plastic housing. The packaging system is designed to optimize the mechanical resolution, coupling efficiency, ambient light rejection, cost and reliability. The gap in the housing provides a means of interrupting the signal with an opaque material, switching the output from an "ON" to an "OFF" state.

SCHEMATIC



FEATURES

- Opaque housing
- Low cost
- .035" apertures
- High $I_{C(ON)}$

1. Derate power dissipation linearly 1.33 mW/°C above 25°C.
2. RMA flux is recommended.
3. Methanol or isopropyl alcohols are recommended as cleaning agents.
4. Soldering iron tip 1/16" (1.6mm) minimum from housing.

ABSOLUTE MAXIMUM RATINGS (T_A = 25°C unless otherwise specified)

Parameter	Symbol	Rating	Unit
Operating Temperature	T_{OPR}	-55 to +100	°C
Storage Temperature	T_{STG}	-55 to +100	°C
Soldering Temperature (Iron)(2,3 and 4)	T_{SOL-I}	240 for 5 sec	°C
Soldering Temperature (Flow)(2 and 3)	T_{SOL-F}	260 for 10 sec	°C
INPUT (EMITTER) Continuous Forward Current	I_F	50	mA
Reverse Voltage	V_R	6	V
Power Dissipation (1)	P_D	100	mW
OUTPUT (SENSOR) Collector to Emitter Voltage	V_{CEO}	30	V
Emitter to Collector Voltage	V_{ECO}	4.5	V
Collector Current	I_C	20	mA
Power Dissipation (T_C = 25°C)(1)	P_D	150	mW

ELECTRICAL / OPTICAL CHARACTERISTICS (T_A =25°C)(All measurements made under pulse condition)

PARAMETER	TEST CONDITIONS	SYMBOL	DEVICES	MIN	TYP	MAX	UNITS
INPUT (EMITTER) Forward Voltage	I_F = 60 mA	V_F	All	—	—	1.7	V
Reverse Breakdown Voltage	I_R = 10 µA	V_R	All	6.0	—	—	V
Reverse Leakage Current	V_R = 3 V	I_R	All	—	—	1.0	µA
OUTPUT (SENSOR) Emitter to Collector Breakdown	I_F = 100 µA, Ee = 0	BV_{ECO}	All	6.0	—	—	V
Collector to Emitter Breakdown	I_C = 1 mA, Ee = 0	BV_{CEO}	All	30	—	—	V
Collector to Emitter Leakage	V_{CE} = 25 V, Ee = 0	I_{CEO}	All	—	—	100	nA
COUPLED On-State Collector Current	I_F = 5 mA, V_{CE} = 5 V	$I_{C(ON)}$	H21A1	0.15	—	—	mA
			H21A2	0.30	—	—	
			H21A3	0.60	—	—	
	I_F = 20 mA, V_{CE} = 5 V		H21A1	1.0	—	—	
			H21A2	2.0	—	—	
			H21A3	4.0	—	—	
	I_F = 30 mA, V_{CE} = 5 V		H21A1	1.9	—	—	
			H21A2	3.0	—	—	
			H21A3	5.5	—	—	
Saturation Voltage	I_F = 20 mA, I_C = 1.8 mA	$V_{CE(SAT)}$	H21A2/3	—	—	0.40	V
	I_F = 30 mA, I_C = 1.8 mA		H21A1	—	—	0.40	V
Turn-On Time	I_F = 30 mA, V_{CC} = 5 V, R_L = 2.5 KΩ	t_{on}	All	—	8	—	µs
Turn-Off Time	I_F = 30 mA, V_{CC} = 5 V, R_L = 2.5 KΩ	t_{off}	All	—	50	—	µs

National Semiconductor

November 2000

LM35
Precision Centigrade Temperature Sensors

General Description

The LM35 series are precision integrated-circuit temperature sensors, whose output voltage is linearly proportional to the Celsius (Centigrade) temperature. The LM35 thus has an advantage over linear temperature sensors calibrated in ° Kelvin, as the user is not required to subtract a large constant voltage from its output to obtain convenient Centigrade scaling. The LM35 does not require any external calibration or trimming to provide typical accuracies of ±¼°C at room temperature and ±¾°C over a full −55 to +150°C temperature range. Low cost is assured by trimming and calibration at the wafer level. The LM35's low output impedance, linear output, and precise inherent calibration make interfacing to readout or control circuitry especially easy. It can be used with single power supplies, or with plus and minus supplies. As it draws only 60 µA from its supply, it has very low self-heating, less than 0.1°C in still air. The LM35 is rated to operate over a −55° to +150°C temperature range, while the LM35C is rated for a −40° to +110°C range (−10° with improved accuracy). The LM35 series is available packaged in hermetic TO-46 transistor packages, while the LM35C, LM35CA, and LM35D are also available in the plastic TO-92 transistor package. The LM35D is also available in an 8-lead surface mount small outline package and a plastic TO-220 package.

Features

- Calibrated directly in ° Celsius (Centigrade)
- Linear + 10.0 mV/°C scale factor
- 0.5°C accuracy guaranteeable (at +25°C)
- Rated for full −55° to +150°C range
- Suitable for remote applications
- Low cost due to wafer-level trimming
- Operates from 4 to 30 volts
- Less than 60 µA current drain
- Low self-heating, 0.08°C in still air
- Nonlinearity only ±¼°C typical
- Low impedance output, 0.1 Ω for 1 mA load

Typical Applications

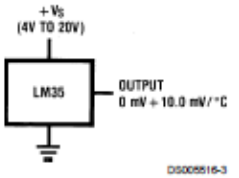
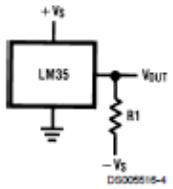


FIGURE 1. Basic Centigrade Temperature Sensor (+2°C to +150°C)

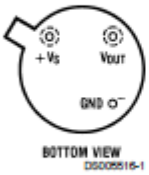


Choose $R_1 = -V_S/50\ \mu A$
V_{OUT} = +1,500 mV at +150°C
= +250 mV at +25°C
= −550 mV at −55°C

FIGURE 2. Full-Range Centigrade Temperature Sensor

Connection Diagrams

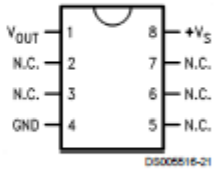
TO-46
Metal Can Package*



*Case is connected to negative pin (GND)

Order Number LM35H, LM35AH, LM35CH, LM35CAH or LM35DH
See NS Package Number H03H

SO-8
Small Outline Molded Package



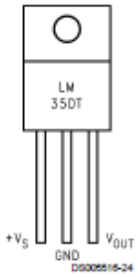
N.C. = No Connection

Top View
Order Number LM35DM
See NS Package Number M08A

TO-92
Plastic Package

Order Number LM35CZ,
LM35CAZ or LM35DZ
See NS Package Number Z03A

TO-220
Plastic Package*

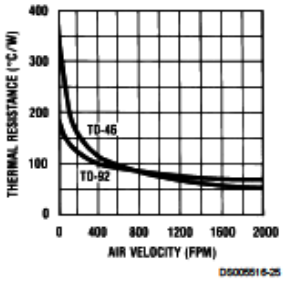


*Tab is connected to the negative pin (GND).
Note: The LM35DT pinout is different than the discontinued LM35DP.

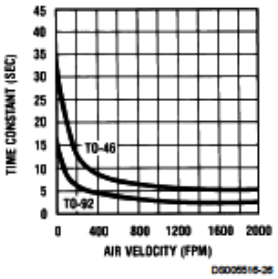
Order Number LM35DT
See NS Package Number TA03F

Typical Performance Characteristics

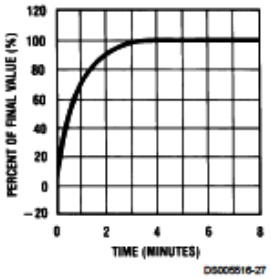
Thermal Resistance Junction to Air



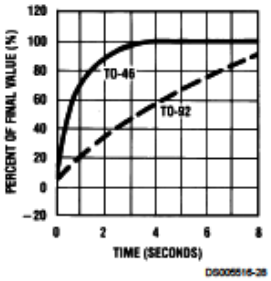
Thermal Time Constant



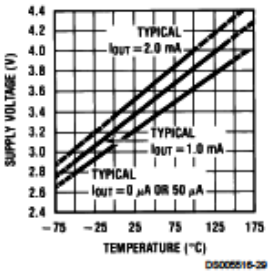
Thermal Response in Still Air



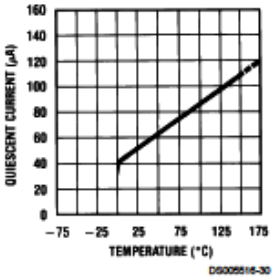
Thermal Response in Stirred Oil Bath



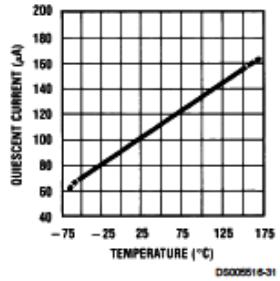
Minimum Supply Voltage vs. Temperature



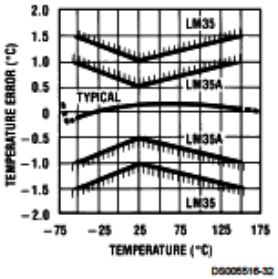
Quiescent Current vs. Temperature (In Circuit of *Figure 1*.)



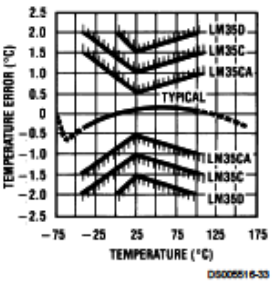
Quiescent Current vs. Temperature (In Circuit of *Figure 2*.)



Accuracy vs. Temperature (Guaranteed)

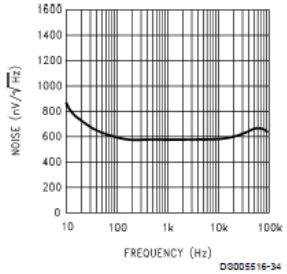


Accuracy vs. Temperature (Guaranteed)

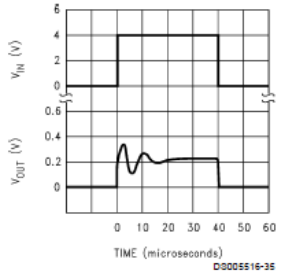


Typical Performance Characteristics (Continued)

Noise Voltage



Start-Up Response



Applications

The LM35 can be applied easily in the same way as other integrated-circuit temperature sensors. It can be glued or cemented to a surface and its temperature will be within about 0.01°C of the surface temperature.

This presumes that the ambient air temperature is almost the same as the surface temperature; if the air temperature were much higher or lower than the surface temperature, the actual temperature of the LM35 die would be at an intermediate temperature between the surface temperature and the air temperature. This is expecially true for the TO-92 plastic package, where the copper leads are the principal thermal path to carry heat into the device, so its temperature might be closer to the air temperature than to the surface temperature.

To minimize this problem, be sure that the wiring to the LM35, as it leaves the device, is held at the same temperature as the surface of interest. The easiest way to do this is to cover up these wires with a bead of epoxy which will insure that the leads and wires are all at the same temperature as the surface, and that the LM35 die's temperature will not be affected by the air temperature.

The TO-46 metal package can also be soldered to a metal surface or pipe without damage. Of course, in that case the V− terminal of the circuit will be grounded to that metal. Alternatively, the LM35 can be mounted inside a sealed-end metal tube, and can then be dipped into a bath or screwed into a threaded hole in a tank. As with any IC, the LM35 and accompanying wiring and circuits must be kept insulated and dry, to avoid leakage and corrosion. This is especially true if the circuit may operate at cold temperatures where condensation can occur. Printed-circuit coatings and varnishes such as Humiseal and epoxy paints or dips are often used to insure that moisture cannot corrode the LM35 or its connections.

These devices are sometimes soldered to a small light-weight heat fin, to decrease the thermal time constant and speed up the response in slowly-moving air. On the other hand, a small thermal mass may be added to the sensor, to give the steadiest reading despite small deviations in the air temperature.

Temperature Rise of LM35 Due To Self-heating (Thermal Resistance, θ_{JA})

	TO-46, no heat sink	TO-46*, small heat fin	TO-92, no heat sink	TO-92**, small heat fin	SO-8 no heat sink	SO-8** small heat fin	TO-220 no heat sink
Still air	400°C/W	100°C/W	180°C/W	140°C/W	220°C/W	110°C/W	90°C/W
Moving air	100°C/W	40°C/W	90°C/W	70°C/W	105°C/W	90°C/W	26°C/W
Still oil	100°C/W	40°C/W	90°C/W	70°C/W			
Stirred oil	50°C/W	30°C/W	45°C/W	40°C/W			
(Clamped to metal, Infinite heat sink)	(24°C/W)				(55°C/W)		

*Wakefield type 201, or 1" disc of 0.020" sheet brass, soldered to case, or similar.
**TO-92 and SO-8 packages glued and leads soldered to 1" square of 1/16" printed circuit board with 2 oz. foil or similar.