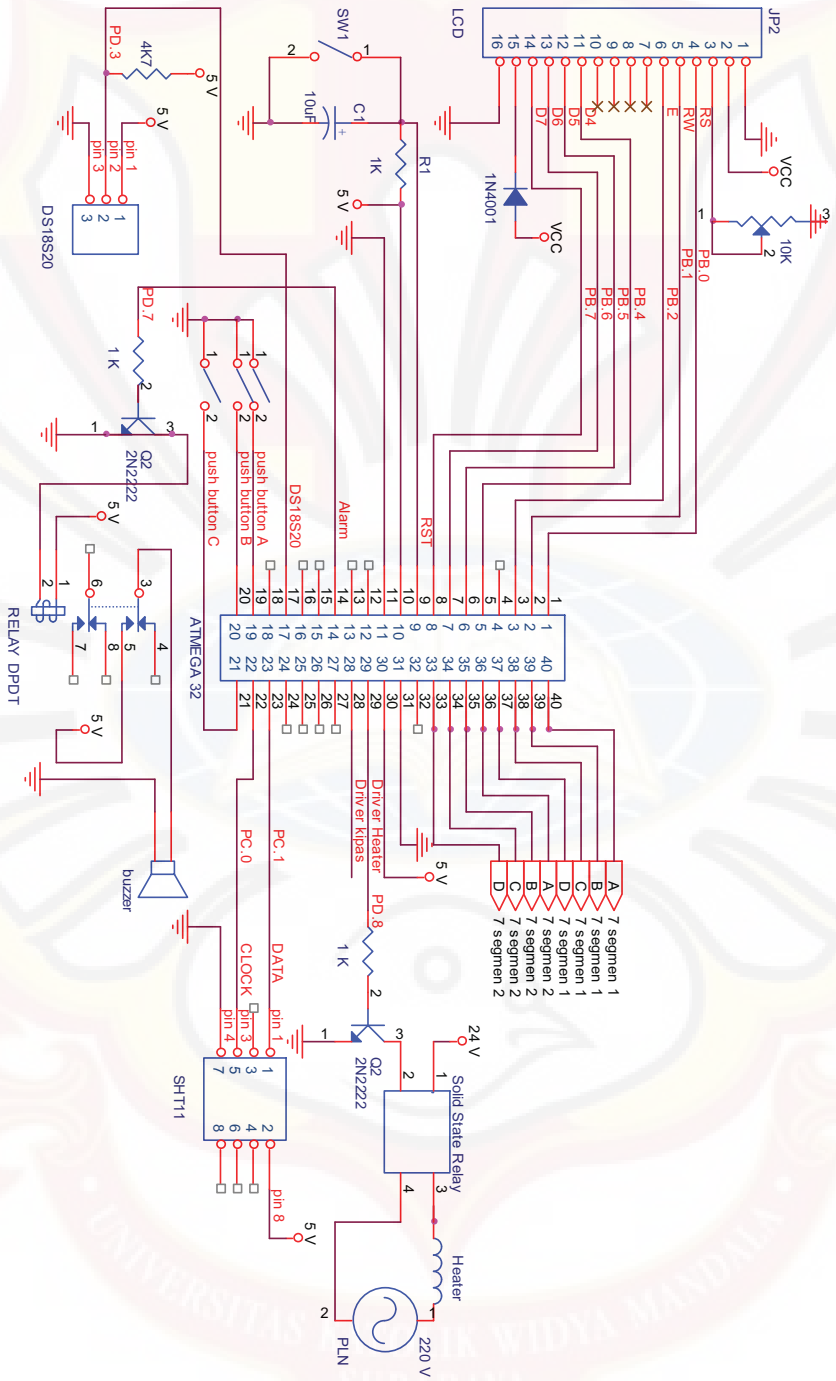
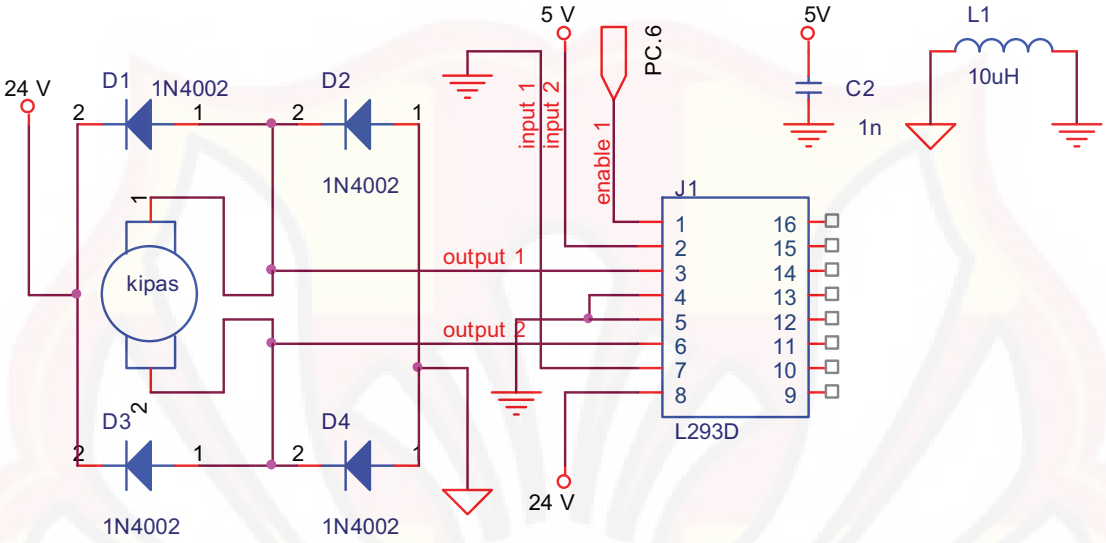
LAMPIRAN 1

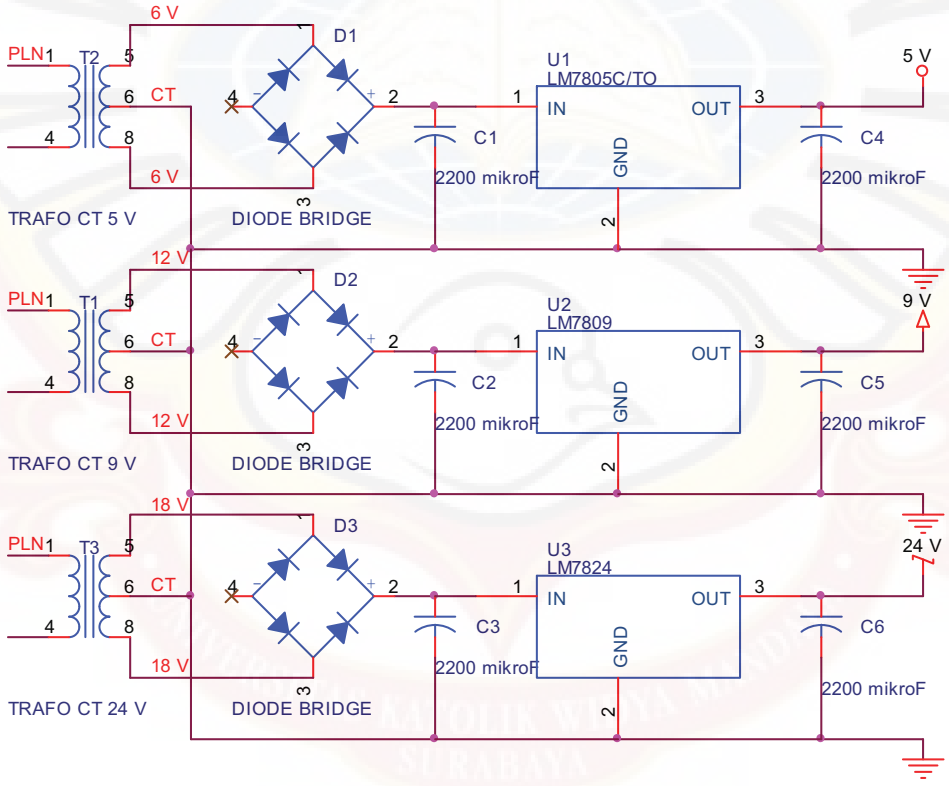
GAMBAR SKEMATIK LENGKAP



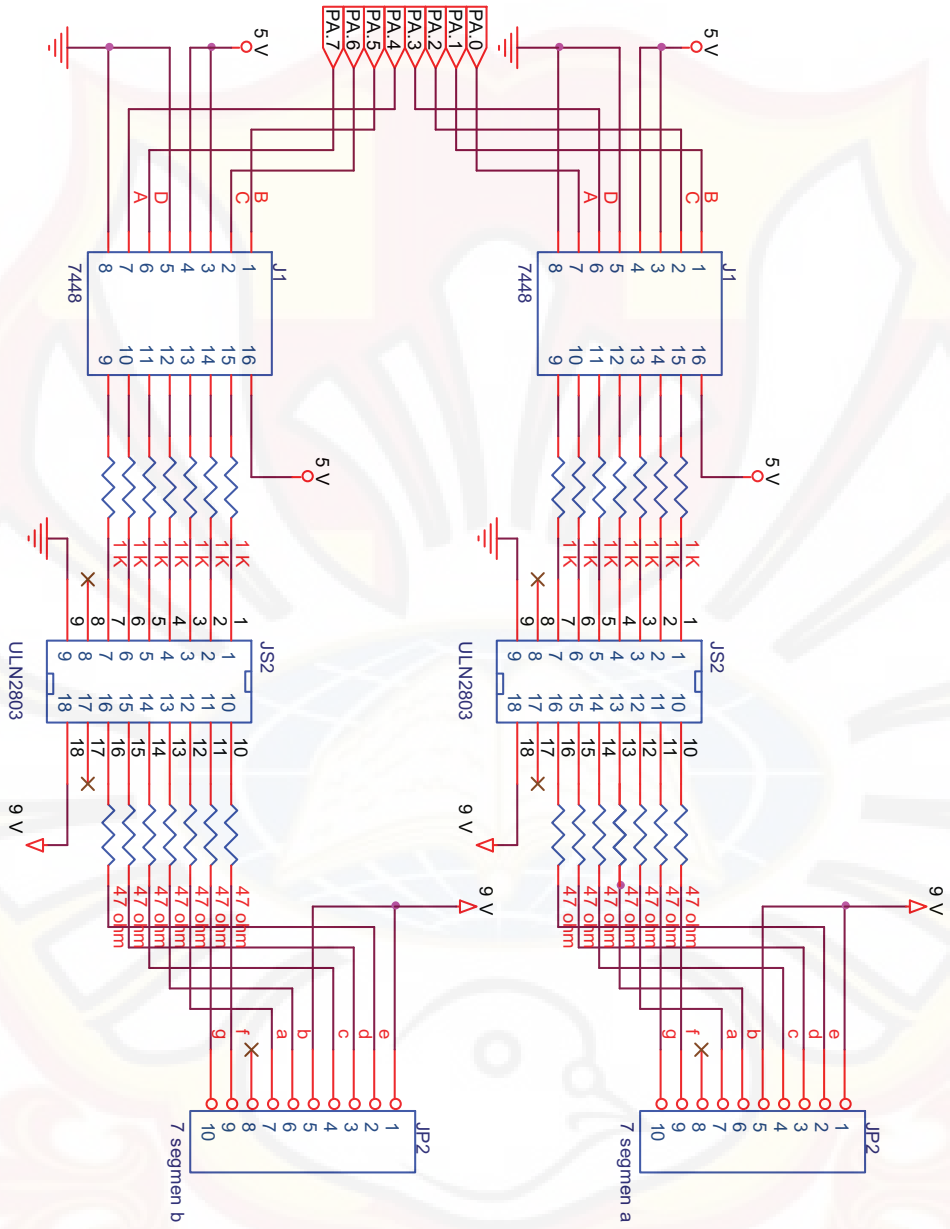
Gambar L-1 Rangkaian Mikrokontroler, Alarm, *Driver Heater*, dan Sensor



Gambar L-2 Rangkaian *Driver* Kipas L293D



Gambar L-3 Rangkaian Catu Daya



Gambar L-4 Rangkaian *Driver* LED 7 segmen

LAMPIRAN 2

LISTING PROGRAM

```
////////////////
//MAIN PROGRAM//
////////////////
#define F_CPU 8000000UL
#include <avr/io.h>
#include <avr/pgmspace.h>
#include <avr/interrupt.h>
#include <inttypes.h>
#include <util/delay.h>
#include <string.h>
#include "lcd_lib.c"
#include <sensor.c>
#include "avr/eeprom.h"
#include <onewire.c>
#include <ds18x20.c>
#define MAXSENSORS 5
#define C   (PIND & 0b10000000)
#define B   (PIND & 0b01000000)
#define A   (PIND & 0b00100000)

//Temporary Variable
unsigned char gSensorIDs[MAXSENSORS][OW_ROMCODE_SIZE];
unsigned char nSensors, i;
unsigned char subzero, cel, cel_frac_bits;
char buffer[sizeof(int)*8+1];
uint16_t decicelsius;
uint8_t i, j;
unsigned int temp,counterPWM = 0, nilaiPWM, nilaiPWMKipas,
nilaiPWMKipasx, counterdisplay;
unsigned char TimeOut,Ackbit;
unsigned char puluhan, satuan, tampil, bunyi;
unsigned char i,j,k,l,m,alarm;
unsigned int suhuSekarang,kelembaban,counter,counterx;
int error,errorx,sumError,deltaError,KP,KD,KI,hasilPID;
unsigned int setpoint,TMAX, TMIN, HMAX, HMIN;
unsigned int EEMEM setpointEE,TMAXEE, TMINEE, HMAXEE, HMINEE;

const uint8_t judul[] PROGMEM="PENGENDALI SUHU";
const uint8_t judul1[] PROGMEM="INKUBATOR BAYI DGN";
const uint8_t judul2[] PROGMEM="SISTEM KONTROL PID";
const uint8_t judul3[] PROGMEM="ANTONY/5103006036";
const uint8_t main1[] PROGMEM="    MAIN DISPLAY";
const uint8_t main2[] PROGMEM=" TEMP  SKTEMP  HUMI";
const uint8_t main3[] PROGMEM="RUN           SETUP";
const uint8_t conf1[] PROGMEM="    CONFIRMATION";
const uint8_t conf2[] PROGMEM="ARE YOU SURE TO RUN";
const uint8_t conf3[] PROGMEM="THE SYSTEM? YES/NO";
const uint8_t conf4[] PROGMEM="YES             NO";
```

```
const uint8_t st11[] PROGMEM="  (SETTEMP:     )";
const uint8_t st1[] PROGMEM="RUN  (SETTEMP:     )";
const uint8_t st2[] PROGMEM=" TEMP  SKTEMP  HUMI";
const uint8_t st3[] PROGMEM="        STOP        ";
const uint8_t stup1[] PROGMEM="SETUP (SETTEMP:    )";
const uint8_t stup2[] PROGMEM="TMAX=   C HMAX=  %RH";
const uint8_t stup3[] PROGMEM="TMIN=   C HMIN=  %RH";
const uint8_t stup4[] PROGMEM="TEMP     OK    ALARM";
const uint8_t tset1[] PROGMEM="    TEMP SETTING";
const uint8_t tset2[] PROGMEM="CURRENT SETTING:   C";
const uint8_t tset3[] PROGMEM="NEW CURRENT    :   C";
const uint8_t tset4[] PROGMEM=" +      SET       -";
const uint8_t aset1[] PROGMEM="   ALARM SETTING";
const uint8_t aset2[] PROGMEM="TMAX=   C HMAX=  %RH";
const uint8_t aset3[] PROGMEM="TMIN=   C HMIN=  %RH";
const uint8_t aset4[] PROGMEM=" +      SET       -";
const uint8_t al11[] PROGMEM="                    ";
const uint8_t al1[] PROGMEM=" TEMP IS TOO HIGH!!!";
const uint8_t al2[] PROGMEM=" TEMP IS TOO LOW!!! ";
const uint8_t al3[] PROGMEM=" HUMI IS TOO HIGH!!!";
const uint8_t al4[] PROGMEM=" HUMI IS TOO LOW!!! ";
const uint8_t press[] PROGMEM="PRESS RESET BUTTON";
const uint8_t pressx[] PROGMEM="                  ";

//====================INITIALIZE====================
void initialize(void)
{
//==============
//     PORT
//==============
//PORTA- 7 segment
//PORTB-LCD (7:1)
//PORTC-
//PORTD-
PORTA = 0x00;
DDRA  = 0xFF;
PORTB = 0xFF;
DDRB  = 0x00;
PORTC = 0x00;
DDRC  = 0xE3; //1110 0011
PORTD = 0xF0;
DDRD  = 0xFB;

//==============
//Turn off Analog comparator power
ACSR = 0x80;
//==============
//    Timer
//==============
//Timer 0 = 8 MHz div by 64
```

```
//For button sample delay
TCCR0 = 0x03;
TCNT0 = 0x00;
TIMSK = 0x01;
_delay_ms(500);
//==============
//     LCD
//==============
// LCD 20x4 Char
LCDinit();
LCDcursorOFF();
LCDclr();
_delay_ms(500);
CopyStringtoLCD(judul,  2, 0);
_delay_ms(1200);
CopyStringtoLCD(judul1, 1, 1);
_delay_ms(1200);
CopyStringtoLCD(judul2, 1, 2);
_delay_ms(1200);
CopyStringtoLCD(judul3, 1, 3);
_delay_ms(1700);
//LCDcursorOFF();
LCDclr();
_delay_ms(1000);
CopyStringtoLCD(main1,  0, 0);
CopyStringtoLCD(main2, 0, 1);
CopyStringtoLCD(main3, 0, 3);
}

ISR(TIMER0_OVF_vect)
{
        counterPWM++;
        if (counterPWM>50) {
                        if      (nilaiPWM>0) {
                                PORTC |= 0x80;
                        }
                        if      (nilaiPWMKipas>0) {
                                PORTC |= 0x40;
                        }
                        counterPWM = 0;
        }
        if (counterPWM>nilaiPWM) {
                PORTC &= 0x7F;
                }
        if (counterPWM>nilaiPWMKipas) {
                PORTC &= 0xBF;
        }
}

uint8_t search_sensors(void)
{
```

```
        uint8_t i;
        uint8_t id[OW_ROMCODE_SIZE];
        uint8_t diff, nSensors;
        nSensors = 0;
        for( diff = OW_SEARCH_FIRST;
        diff != OW_LAST_DEVICE && nSensors < MAXSENSORS ; )
        {
                DS18X20_find_sensor( &diff, &id[0] );
                if( diff == OW_PRESENCE_ERR ) {
                break;
                }
                if( diff == OW_DATA_ERR ) {
                break;
                }
                for (i=0;i<OW_ROMCODE_SIZE;i++)
                gSensorIDs[nSensors][i]=id[i];
                nSensors++;
        }
        return nSensors;
}

void updateDisplay(void)
{
_delay_ms(11);
if (alarm==1) {
                temp = smeasure() -200;
                temp /= 100;
                //==============
                //   7 segment
                //==============
                satuan = temp % 10;
                puluhan = temp / 10;
                puluhan = puluhan << 4;
                tampil = puluhan + satuan;
                PORTA = tampil;
                }
                if ((A!=0) && (B!=0) && (C!=0))       {
                                temp = smeasure() -200;
                                temp /= 10;
                                suhuSekarang = temp;
                }
                if ((A!=0) && (B!=0) && (C!=0))       {
                                temp = smeasureX();
                                kelembaban = temp;
                                if (suhuSekarang > (TMIN+5)) {
                                alarm = 0 ;
                                }
                }
                if ((A!=0) && (B!=0) && (C!=0))       {
                                nSensors = search_sensors();
                                if ( nSensors == 1 ) {
```

```
        i = gSensorIDs[0][0];
        DS18X20_start_meas( DS18X20_POWER_PARASITE, NULL );
        _delay_ms(DS18B20_TCONV_12BIT);
        DS18X20_read_meas_single(i, &subzero, &cel,
        &cel_frac_bits);
        LCDGotoXY(7,2);
        decicelsius = DS18X20_temp_to_decicel(subzero, cel,
        cel_frac_bits);
        LCDsendChar(decicelsius/100 %10 + 0x30);
        LCDsendChar(decicelsius/10 %10 + 0x30);
        LCDsendChar('.');
        LCDsendChar(decicelsius %10 + 0x30);
        LCDGotoXY(12,2);
        LCDsendChar('C');
        LCDGotoXY(11,2);
        LCDsendChar(0);
        }
                                temp = suhuSekarang;
                                LCDGotoXY(5,2);
                                LCDsendChar('C');
                                LCDGotoXY(4,2);
                                LCDsendChar(0);
                                LCDGotoXY(3,2);
                                LCDsendChar((temp % 10 ) | '0');
                                temp = temp / 10;
                                LCDGotoXY(2,2);
                                LCDsendChar('.');
                                LCDGotoXY(1,2);
                                LCDsendChar((temp % 10 ) | '0');
                                temp = temp / 10;
                                LCDGotoXY(0,2);
                                LCDsendChar((temp % 10 ) | '0');
                                temp = temp / 10;

                                temp=kelembaban;
                                LCDGotoXY(19,2);
                                LCDsendChar('H');
                        `       LCDGotoXY(18,2);
                                LCDsendChar('R');
                                LCDGotoXY(17,2);
                                LCDsendChar('%');
                                LCDGotoXY(16,2);
                                LCDsendChar((temp % 10 ) | '0');
                                temp = temp / 10;
                                LCDGotoXY(15,2);
                                LCDsendChar((temp % 10 ) | '0');
                                temp = temp / 10;
                        }
}

void run(void)
```

```
{
        CopyStringtoLCD(st1,  0, 0);
        CopyStringtoLCD(st2,  0, 1);
        CopyStringtoLCD(st3,  0, 3);
        counterx++;
        if (counterx > 2){
        counterx = 1;
        }
        if (counterx==1){
              LCDGotoXY(0,0);
              LCDsendChar(' ');
              LCDGotoXY(1,0);
              LCDsendChar(' ');
              LCDGotoXY(2,0);
              LCDsendChar(' ');
              }
        else if (counterx==2){
              }
                           LCDGotoXY(18,0);
                           LCDsendChar('C');
                           LCDGotoXY(17,0);
                           LCDsendChar(0);
                           temp = setpoint;
                           temp = temp / 10;
                           LCDGotoXY(16,0);
                           LCDsendChar((temp % 10 ) | '0');
                           temp = temp / 10;
                           LCDGotoXY(15,0);
                           LCDsendChar((temp % 10 ) | '0');
              if (alarm == 0) {                    //alarm
                           if (suhuSekarang>TMAX) {
                           PORTD |= 0x01;
                           counter++;
                           if (counter > 2){
                           counter = 1;
                           }
                           if (counter==1){
                           CopyStringtoLCD(al1,  0, 0);
                           CopyStringtoLCD(press,  0, 3);
                           LCDGotoXY(18,3);
                           LCDsendChar(2);
                           LCDGotoXY(19,3);
                           LCDsendChar(2);
                           }
                           if (counter==2){
                           CopyStringtoLCD(al11,  0, 0);
                           CopyStringtoLCD(pressx,  0, 3);
                           }
                           bunyi = 1;
                           }
                      else if (suhuSekarang<TMIN) {
```

```
PORTD |= 0x01;
counter++;
if (counter > 2){
counter = 1;
}
if (counter==1){
CopyStringtoLCD(al2,  0, 0);
CopyStringtoLCD(press,  0, 3);
LCDGotoXY(18,3);
LCDsendChar(2);
LCDGotoXY(19,3);
LCDsendChar(2);
}
if (counter==2){
CopyStringtoLCD(al11,  0, 0);
CopyStringtoLCD(pressx,  0, 3);
}
bunyi = 1;
}
else if (kelembaban>HMAX) {
PORTD |= 0x01;
counter++;
if (counter > 2){
counter = 1;
}
if (counter==1){
CopyStringtoLCD(al3,  0, 0);
CopyStringtoLCD(press,  0, 3);
LCDGotoXY(18,3);
LCDsendChar(2);
LCDGotoXY(19,3);
LCDsendChar(2);
}
if (counter==2){
CopyStringtoLCD(al11,  0, 0);
CopyStringtoLCD(pressx,  0, 3);
}
bunyi = 1;
}
else if (kelembaban<HMIN) {
PORTD |= 0x01;
counter++;
if (counter > 2){
counter = 1;
}
if (counter==1){
CopyStringtoLCD(al4,  0, 0);
CopyStringtoLCD(press,  0, 3);
LCDGotoXY(18,3);
LCDsendChar(2);
LCDGotoXY(19,3);
```

```
                    LCDsendChar(2);
                    }
                    if (counter==2){
                    CopyStringtoLCD(al11,  0, 0);
                    CopyStringtoLCD(pressx,  0, 3);
                    }
                    bunyi = 1;
                    }
                    else {
                    bunyi = 0;
                    PORTD &= 0xFE;
                    CopyStringtoLCD(st11,  3, 0);
                    LCDGotoXY(18,0);
                    LCDsendChar('C');
                    LCDGotoXY(17,0);
                    LCDsendChar(0);
                    temp = setpoint;
                    temp = temp / 10;
                    LCDGotoXY(16,0);
                    LCDsendChar((temp % 10 ) | '0');
                    temp = temp / 10;
                    LCDGotoXY(15,0);
                    LCDsendChar((temp % 10 ) | '0');
                    }
            }
            if (bunyi==1) {
                        counterdisplay++;
                        if (counterdisplay >= 2){
                              counterdisplay = 0;
                        }
if ((alarm == 0) && (counterdisplay < 1))
                        {
                        temp = smeasure()-200;
                        temp /= 100;
                        //==============
                        //   7 segment
                        //==============
                        satuan = temp % 10;
                        puluhan = temp / 10;
                        puluhan = puluhan << 4;
                        tampil = puluhan + satuan;
                        PORTA = tampil;
                        }
else if ((alarm == 0) && (counterdisplay < 2))
                        {
                              PORTA = 0xFF;
                        }
else {
                        temp = smeasure() -200;
                        temp /= 100;
                        //==============
```

```
                        //   7 segment
                        //==============
                        satuan = temp % 10;
                        puluhan = temp / 10;
                        puluhan = puluhan << 4;
                        tampil = puluhan + satuan;
                        PORTA = tampil;
                        }
                }
                else {
                        temp = smeasure()-200;
                        temp /= 100;
                        //==============
                        //   7 segment
                        //==============
                        satuan = temp % 10;
                        puluhan = temp / 10;
                        puluhan = puluhan << 4;
                        tampil = puluhan + satuan;
                        PORTA = tampil;
                }
                updateDisplay();
                //=====================
                //=====================
                //        PID
                //=====================
                //=====================
                error = setpoint - suhuSekarang;
                sumError = sumError + error ;
                if (sumError>100) {
                        sumError = 100 ;
                        }
                if (sumError<0) {
                        sumError = 0 ;
                        }
        hasilPID = KP*error + sumError/4 ;
        if (error>=0) {
        //suhu masih dibawah atau sama setpoint
                        if (hasilPID>50) {
                        hasilPID = 50;
                        }
                        nilaiPWM = hasilPID;
                        }
        else {
        //suhu sudah diatas setpoint
                        nilaiPWM = 0;
                        }
        if (error>=0) {
        //suhu masih dibawah atau sama setpoint

                if (hasilPID>50) {
```

```
                                hasilPID = 50;
                                }
                                nilaiPWMKipas = hasilPID;
                                }
                        else {
                        //suhu sudah diatas setpoint
                                nilaiPWMKipas = 0;
                                }
                }

void conf(void)
{
        while(A!=0&&C!=0)
        {
                CopyStringtoLCD(conf1,  0, 0);
                CopyStringtoLCD(conf2,  0, 1);
                CopyStringtoLCD(conf3,  0, 2);
                CopyStringtoLCD(conf4,  0, 3);
        }
        if(A==0)
        {
                while(A==0){}
                LCDclr();
                _delay_ms(50);
                while(B!=0)
                {
                        run();
                }
                PORTD &= 0xFE; // matikan alarm
                alarm = 1;
                sumError = 0;
        }
        if(C==0)
        while(C==0){}
}

void settm(void)
{
        CopyStringtoLCD(tset1,  0, 0);
        CopyStringtoLCD(tset2,  0, 1);
        CopyStringtoLCD(tset3,  0, 2);
        CopyStringtoLCD(tset4,  0, 3);
        i=setpoint/10;
        if(i>38)i=38;
        if(i<30)i=30;
        k=1;
                                temp = i;
                                LCDGotoXY(17,1);
                                LCDsendChar((temp % 10 ) | '0');
                                temp = temp / 10;
```

```
                                LCDGotoXY(16,1);
                                LCDsendChar((temp % 10 ) | '0');
                                LCDGotoXY(18,1);
                                LCDsendChar(0);
                                temp = i;
                                LCDGotoXY(17,2);
                                LCDsendChar((temp % 10 ) | '0');
                                temp = temp / 10;
                                LCDGotoXY(16,2);
                                LCDsendChar((temp % 10 ) | '0');
                                LCDGotoXY(18,2);
                                LCDsendChar(0);

        while(B!=0)
        {
                LCDGotoXY(18,2);LCDcursorOnBlink();
                if(A==0)
                {
                        _delay_ms(200);
                        i++;
                        if(i>38)i=38;
                                temp = i;
                                LCDGotoXY(17,2);
                                LCDsendChar((temp % 10 ) | '0');
                                temp = temp / 10;
                                LCDGotoXY(16,2);
                                LCDsendChar((temp % 10 ) | '0');
                }
                if(C==0)
                {
                        _delay_ms(200);
                        i--;
                        if(i<30)i=30;
                                temp = i;
                                LCDGotoXY(17,2);
                                LCDsendChar((temp % 10 ) | '0');
                                temp = temp / 10;
                                LCDGotoXY(16,2);
                                LCDsendChar((temp % 10 ) | '0');
                }
        }
        while(B==0){}
        setpoint = i*10;
        eeprom_write_word(&setpointEE, setpoint);
        _delay_ms(10);
        LCDclr();
        _delay_ms(50);
}

void setal(void)
{
```

```
CopyStringtoLCD(aset1,  0, 0);
CopyStringtoLCD(aset2,  0, 1);
CopyStringtoLCD(aset3,  0, 2);
CopyStringtoLCD(aset4,  0, 3);
i=TMAX/10;
if(i>39)i=39;
if(i<29)i=29;
j=TMIN/10;
if(j>39)j=39;
if(j<29)j=29;
k=1;
l=HMAX;
if(l>90)l=90;
if(l<20)l=20;
m=HMIN;
if(m>90)m=90;
if(m<20)m=20;
                        temp = i;
                        LCDGotoXY(6,1);
                        LCDsendChar((temp % 10 ) | '0');
                        temp = temp / 10;
                        LCDGotoXY(5,1);
                        LCDsendChar((temp % 10 ) | '0');
                        LCDGotoXY(7,1);
                        LCDsendChar(0);

                        temp = j;
                        LCDGotoXY(6,2);
                        LCDsendChar((temp % 10 ) | '0');
                        temp = temp / 10;
                        LCDGotoXY(5,2);
                        LCDsendChar((temp % 10 ) | '0');
                        LCDGotoXY(7,2);
                        LCDsendChar(0);

                        temp = l;
                        LCDGotoXY(16,1);
                        LCDsendChar((temp % 10 ) | '0');
                        temp = temp / 10;
                        LCDGotoXY(15,1);
                        LCDsendChar((temp % 10 ) | '0');

                        temp = m;
                        LCDGotoXY(16,2);
                        LCDsendChar((temp % 10 ) | '0');
                        temp = temp / 10;
                        LCDGotoXY(15,2);
                        LCDsendChar((temp % 10 ) | '0');

while(k!=5)
{
```

```
if(B==0)
{
        while(B==0){}
        k++;
}
        if(k==1)
        {
        LCDGotoXY(7,1);LCDcursorOnBlink();
        if(A==0)
                {
                        _delay_ms(200);
                        i++;
                        if(i>39)i=39;
                       temp = i;
      LCDGotoXY(6,1);
      LCDsendChar((temp % 10 ) | '0');
      temp = temp / 10;
      LCDGotoXY(5,1);
      LCDsendChar((temp % 10 ) | '0');
                }
                if(C==0)
                {
                        _delay_ms(200);
                        i--;
                        if(i<29)i=29;
                         temp = i;
      LCDGotoXY(6,1);
      LCDsendChar((temp % 10 ) | '0');
      temp = temp / 10;
      LCDGotoXY(5,1);
      LCDsendChar((temp % 10 ) | '0');
        }
        }
        if(k==2)
        {
        LCDGotoXY(7,2);LCDcursorOnBlink();
        if(A==0)
                {
                        _delay_ms(200);
                        j++;
                        if(j>39)j=39;
                         temp = j;
      LCDGotoXY(6,2);
      LCDsendChar((temp % 10 ) | '0');
      temp = temp / 10;
      LCDGotoXY(5,2);
      LCDsendChar((temp % 10 ) | '0');
                }
                if(C==0)
                {
                        _delay_ms(200);
```

```
                        j--;
                        if(j<29)j=29;
                         temp = j;
        LCDGotoXY(6,2);
        LCDsendChar((temp % 10 ) | '0');
        temp = temp / 10;
        LCDGotoXY(5,2);
        LCDsendChar((temp % 10 ) | '0');
}
           }
           if(k==3)
           {
           LCDGotoXY(17,1);LCDcursorOnBlink();
           if(A==0)
                  {
                        _delay_ms(200);
                        l++;
                        if(l>90)l=90;
                         temp = l;
        LCDGotoXY(16,1);
        LCDsendChar((temp % 10 ) | '0');
        temp = temp / 10;
        LCDGotoXY(15,1);
        LCDsendChar((temp % 10 ) | '0');
                  }
                  if(C==0)
                  {
                        _delay_ms(200);
                        l--;
                        if(l<20)l=20;
                       temp = l;
        LCDGotoXY(16,1);
        LCDsendChar((temp % 10 ) | '0');
        temp = temp / 10;
        LCDGotoXY(15,1);
        LCDsendChar((temp % 10 ) | '0');
                  }
           }
           if(k==4)
           {
           LCDGotoXY(17,2);LCDcursorOnBlink();
           if(A==0)
                  {
                        _delay_ms(200);
                        m++;
                        if(m>90)m=90;
                         temp = m;
        LCDGotoXY(16,2);
        LCDsendChar((temp % 10 ) | '0');
        temp = temp / 10;
        LCDGotoXY(15,2);
```

```
                        LCDsendChar((temp % 10 ) | '0');
                                }
                                if(C==0)
                                {
                                _delay_ms(200);
                                m--;
                                if(m<20)m=20;
                                temp = m;
                        LCDGotoXY(16,2);
                        LCDsendChar((temp % 10 ) | '0');
                        temp = temp / 10;
                        LCDGotoXY(15,2);
                        LCDsendChar((temp % 10 ) | '0');
                                }
                           }
        }
        while(B==0){}
        TMAX = i*10;
        eeprom_write_word(&TMAXEE, TMAX);
        _delay_ms(10);
        TMIN = j*10;
        eeprom_write_word(&TMINEE, TMIN);
        _delay_ms(10);
        HMAX = l;
        eeprom_write_word(&HMAXEE, HMAX);
        _delay_ms(10);
        HMIN = m;
        eeprom_write_word(&HMINEE, HMIN);
        _delay_ms(10);
        LCDclr();
        _delay_ms(50);
}

void setup(void)
{
                CopyStringtoLCD(stup1,  0, 0);
                CopyStringtoLCD(stup2,  0, 1);
                CopyStringtoLCD(stup3,  0, 2);
                CopyStringtoLCD(stup4,  0, 3);
                                LCDGotoXY(18,0);
                                LCDsendChar('C');
                                LCDGotoXY(17,0);
                                LCDsendChar(0);
                                temp = setpoint;
                                temp = temp / 10;
                                LCDGotoXY(17,0);
                                LCDsendChar(0);
                                LCDGotoXY(16,0);
                                LCDsendChar((temp % 10 ) | '0');
                                temp = temp / 10;
                                LCDGotoXY(15,0);
```

```
                        LCDsendChar((temp % 10 ) | '0');

                        temp = TMAX;
                        temp = temp / 10;
                        LCDGotoXY(7,1);
                        LCDsendChar(0);
                        LCDGotoXY(6,1);
                        LCDsendChar((temp % 10 ) | '0');
                        temp = temp / 10;
                        LCDGotoXY(5,1);
                        LCDsendChar((temp % 10 ) | '0');

                        temp = TMIN;
                        temp = temp / 10;
                        LCDGotoXY(7,2);
                        LCDsendChar(0);
                        LCDGotoXY(6,2);
                        LCDsendChar((temp % 10 ) | '0');
                        temp = temp / 10;
                        LCDGotoXY(5,2);
                        LCDsendChar((temp % 10 ) | '0');

                        temp = HMAX;
                        LCDGotoXY(16,1);
                        LCDsendChar((temp % 10 ) | '0');
                        temp = temp / 10;
                        LCDGotoXY(15,1);
                        LCDsendChar((temp % 10 ) | '0');

                        temp = HMIN;
                        LCDGotoXY(16,2);
                        LCDsendChar((temp % 10 ) | '0');
                        temp = temp / 10;
                        LCDGotoXY(15,2);
                        LCDsendChar((temp % 10 ) | '0');


while(B!=0)             //dilakukan slm 'ok' g d tkn
{
        if(A==0)        //set temp
        {
                while(A==0){}
                LCDclr();
                _delay_ms(50);
                settm();
                LCDcursorOFF();
                CopyStringtoLCD(stup1,  0, 0);
                CopyStringtoLCD(stup2,  0, 1);
                CopyStringtoLCD(stup3,  0, 2);
                CopyStringtoLCD(stup4,  0, 3);
                        LCDGotoXY(18,0);
```

```
                    LCDsendChar('C');
                    LCDGotoXY(17,0);
                    LCDsendChar(0);
                    temp = setpoint;
                    temp = temp / 10;
                    LCDGotoXY(17,1);
                    LCDsendChar(0);
                    LCDGotoXY(16,0);
                    LCDsendChar((temp % 10 ) | '0');
                    temp = temp / 10;
                    LCDGotoXY(15,0);
                    LCDsendChar((temp % 10 ) | '0');

                    temp = TMAX;
                    temp = temp / 10;
                    LCDGotoXY(7,1);
                    LCDsendChar(0);
                    LCDGotoXY(6,1);
                    LCDsendChar((temp % 10 ) | '0');
                    temp = temp / 10;
                    LCDGotoXY(5,1);
                    LCDsendChar((temp % 10 ) | '0');

                    temp = TMIN;
                    temp = temp / 10;
                    LCDGotoXY(7,2);
                    LCDsendChar(0);
                    LCDGotoXY(6,2);
                    LCDsendChar((temp % 10 ) | '0');
                    temp = temp / 10;
                    LCDGotoXY(5,2);
                    LCDsendChar((temp % 10 ) | '0');

                    temp = HMAX;
                    LCDGotoXY(16,1);
                    LCDsendChar((temp % 10 ) | '0');
                    temp = temp / 10;
                    LCDGotoXY(15,1);
                    LCDsendChar((temp % 10 ) | '0');

                    temp = HMIN;
                    LCDGotoXY(16,2);
                    LCDsendChar((temp % 10 ) | '0');
                    temp = temp / 10;
                    LCDGotoXY(15,2);
                    LCDsendChar((temp % 10 ) | '0');
}

if(C==0)        //set alarm
{
        while(C==0){}
```

```
LCDclr();
_delay_ms(50);
setal();
LCDcursorOFF();
CopyStringtoLCD(stup1,  0, 0);
CopyStringtoLCD(stup2,  0, 1);
CopyStringtoLCD(stup3,  0, 2);
CopyStringtoLCD(stup4,  0, 3);

        LCDGotoXY(18,0);
        LCDsendChar('C');
        LCDGotoXY(17,0);
        LCDsendChar(0);
        temp = setpoint;
        temp = temp / 10;
        LCDGotoXY(17,1);
        LCDsendChar(0);
        LCDGotoXY(16,0);
        LCDsendChar((temp % 10 ) | '0');
        temp = temp / 10;
        LCDGotoXY(15,0);
        LCDsendChar((temp % 10 ) | '0');

        temp = TMAX;
        temp = temp / 10;
        LCDGotoXY(7,1);
        LCDsendChar(0);
        LCDGotoXY(6,1);
        LCDsendChar((temp % 10 ) | '0');
        temp = temp / 10;
        LCDGotoXY(5,1);
        LCDsendChar((temp % 10 ) | '0');

        temp = TMIN;
        temp = temp / 10;
        LCDGotoXY(7,2);
        LCDsendChar(0);
        LCDGotoXY(6,2);
        LCDsendChar((temp % 10 ) | '0');
        temp = temp / 10;
        LCDGotoXY(5,2);
        LCDsendChar((temp % 10 ) | '0');

        temp = HMAX;
        LCDGotoXY(16,1);
        LCDsendChar((temp % 10 ) | '0');
        temp = temp / 10;
        LCDGotoXY(15,1);
        LCDsendChar((temp % 10 ) | '0');

        temp = HMIN;
```

```
                                LCDGotoXY(16,2);
                                LCDsendChar((temp % 10 ) | '0');
                                temp = temp / 10;
                                LCDGotoXY(15,2);
                                LCDsendChar((temp % 10 ) | '0');
                }
        }

}

//==============================
//==============================
//        MAIN PROGRAM
//==============================
//==============================
int main(void)
{
initialize();
sreset();
sstart();
swrite(0x06);swrite(0x00);
nSensors = ow_reset();
LCDdefinechar(degree,0);
LCDdefinechar(backslash,1);
LCDdefinechar(panah,2);
LCDdefinechar(r,3);
LCDdefinechar(u,4);
LCDdefinechar(n,5);
alarm = 1;
_delay_ms (1000);
setpoint = eeprom_read_word(&setpointEE);
TMAX = eeprom_read_word(&TMAXEE);
TMIN = eeprom_read_word(&TMINEE);
HMAX = eeprom_read_word(&HMAXEE);
HMIN = eeprom_read_word(&HMINEE);
KP = 10;
sei();
//updateDisplay();
//-----------------------------------------
while(1) //Invinite Loop, Interrupt only
//-----------------------------------------
        {
        CopyStringtoLCD(main1,  0, 0);        //MAIN DISPLAY
        CopyStringtoLCD(main2, 0, 1);
        CopyStringtoLCD(main3, 0, 3);
        updateDisplay();                      //READ SENSOR

        if(A==0)        //RUN
                {
                while(A==0){}
                LCDclr();
```

```
                _delay_ms(50);
                conf();
                LCDclr();
                _delay_ms(50);
                LCDcursorOFF();
                nilaiPWM = 0;
                nilaiPWMKipas = 0;
                }
        if(C==0)        //SETUP
                {
                while(C==0){}
                LCDclr();
                _delay_ms(50);
                setup();
                LCDclr();
                _delay_ms(50);
                LCDcursorOFF();
                }
        }
  }

////////////////
/// lcd_lib.h///
////////////////
#ifndef LCD_LIB
#define LCD_LIB
#include <inttypes.h>
//*****************************************
#define LCD_4bit
//*****************************************************
#define LCD_RS 0
#define LCD_RW 1
#define LCD_E  2
#define LCD_D0 0
#define LCD_D1 1
#define LCD_D2 2
#define LCD_D3 3
#define LCD_D4 4
#define LCD_D5 5
#define LCD_D6 6
#define LCD_D7 7
#define LDP PORTB      //define MCU port connected to
                        LCD data pins
#define LCP PORTB      //define MCU port connected to
                       LCD control pins
#define LDDR DDRB      //define MCU direction register
                for port connected to LCD data pins
#define LCDR DDRB      //define MCU direction register
                 for port connected to LCD control pins

#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
#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
// reading:
#define LCD_BUSY            7 //DB7: LCD is busy
#define LCD_LINES             4   //visible lines
#define LCD_LINE_LENGTH      20   //line length
                                (in characters)
// cursor position to DDRAM mapping
#define LCD_LINE0_DDRAMADDR         0x00
#define LCD_LINE1_DDRAMADDR         0x40
#define LCD_LINE2_DDRAMADDR         0x14
#define LCD_LINE3_DDRAMADDR         0x54
// progress bar defines
#define PROGRESSPIXELS_PER_CHAR     5
void LCDsendChar(uint8_t);    //forms data ready to
                                 send to 74HC164
void LCDsendCommand(uint8_t); //forms data ready to
                                 send to 74HC164
void LCDinit(void);           //Initializes LCD
void LCDclr(void);            //Clears LCD
void LCDhome(void);           //LCD cursor home
void LCDstring(uint8_t*, uint8_t); //Outputs string
                                      to LCD
void LCDGotoXY(uint8_t, uint8_t);  //Cursor to X Y
                                      position
void CopyStringtoLCD(const uint8_t*, uint8_t, uint8_t);//copies
flash string to LCD at x,y
```

```
void LCDdefinechar(const uint8_t *,uint8_t);//write
                                        char to LCD CGRAM
void LCDshiftRight(uint8_t);  //shift by n
                                         characters Right
void LCDshiftLeft(uint8_t);   //shift by n
                                         characters Left
void LCDcursorOn(void);              //Underline cursor ON
void LCDcursorOnBlink(void);  //Underline blinking
                                         cursor ON
void LCDcursorOFF(void);      //Cursor OFF
void LCDblank(void);   //LCD blank but not cleared
void LCDvisible(void);        //LCD visible
void LCDcursorLeft(uint8_t);  //Shift cursor left
                                         by n
void LCDcursorRight(uint8_t); //shif cursor right
                                         by n
void  LCDprogressBar(uint8_t  progress,  uint8_t  maxprogress,
uint8_t length);
#endif

////////////////
/// lcd_lib.c///
////////////////
#include "lcd_lib.h"
#include <inttypes.h>
#include <avr/io.h>
#include <avr/pgmspace.h>
#include <util/delay.h>
const uint8_t LcdCustomChar[] PROGMEM=//define 8 custom LCD
chars
{
       0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // 0.
0/5 full progress block
       0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10, // 1.
1/5 full progress block
       0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, // 2.
2/5 full progress block
       0x1C, 0x1C, 0x1C, 0x1C, 0x1C, 0x1C, 0x1C, 0x1C, // 3.
3/5 full progress block
       0x1E, 0x1E, 0x1E, 0x1E, 0x1E, 0x1E, 0x1E, 0x1E, // 4.
4/5 full progress block
       0x1F, 0x1F, 0x1F, 0x1F, 0x1F, 0x1F, 0x1F, 0x1F, // 5.
5/5 full progress block
       0x03, 0x07, 0x0F, 0x1F, 0x0F, 0x07, 0x03, 0x00, // 6.
rewind arrow
       0x18, 0x1C, 0x1E, 0x1F, 0x1E, 0x1C, 0x18, 0x00  // 7.
fast-forward arrow
};

void LCDsendChar(uint8_t ch)          //Sends Char to LCD
{
```

```
#ifdef LCD_4bit
        //4 bit part
        LDP=(ch&0b11110000);
        LCP|=1<<LCD_RS;
        LCP|=1<<LCD_E;
        _delay_us(40);
        LCP&=~(1<<LCD_E);
        LCP&=~(1<<LCD_RS);
        _delay_us(40);
        LDP=((ch&0b00001111)<<4);
        LCP|=1<<LCD_RS;
        LCP|=1<<LCD_E;
        _delay_us(40);
        LCP&=~(1<<LCD_E);
        LCP&=~(1<<LCD_RS);
        _delay_us(40);
#else
        //8 bit part
        LDP=ch;
        LCP|=1<<LCD_RS;
        LCP|=1<<LCD_E;
        _delay_ms(1);
        LCP&=~(1<<LCD_E);
        LCP&=~(1<<LCD_RS);
        _delay_ms(1);
#endif
}
void LCDsendCommand(uint8_t cmd)      //Sends Command to LCD
{
#ifdef LCD_4bit
        //4 bit part
        LDP=(cmd&0b11110000);
        LCP|=1<<LCD_E;
        _delay_us(40);
        LCP&=~(1<<LCD_E);
        _delay_us(40);
        LDP=((cmd&0b00001111)<<4);
        LCP|=1<<LCD_E;
        _delay_us(40);
        LCP&=~(1<<LCD_E);
        _delay_us(40);
#else
        //8 bit part
        LDP=cmd;
        LCP|=1<<LCD_E;
        _delay_ms(1);
        LCP&=~(1<<LCD_E);
        _delay_ms(1);
#endif
}
void LCDinit(void)//Initializes LCD
```

```
{
#ifdef LCD_4bit
        //4 bit part
        _delay_ms(15);
        LDP=0x00;
        LCP=0x00;
        LDDR|=1<<LCD_D7|1<<LCD_D6|1<<LCD_D5|1<<LCD_D4;
        LCDR|=1<<LCD_E|1<<LCD_RW|1<<LCD_RS;
        //---------one------
        LDP=0<<LCD_D7|0<<LCD_D6|1<<LCD_D5|1<<LCD_D4;
        LCP|=1<<LCD_E|0<<LCD_RW|0<<LCD_RS;
        _delay_ms(1);
        LCP&=~(1<<LCD_E);
        _delay_ms(1);
        //-----------two-----------
        LDP=0<<LCD_D7|0<<LCD_D6|1<<LCD_D5|1<<LCD_D4;
        LCP|=1<<LCD_E|0<<LCD_RW|0<<LCD_RS;
        _delay_ms(1);
        LCP&=~(1<<LCD_E);
        _delay_ms(1);
        //-------three-------------
        LDP=0<<LCD_D7|0<<LCD_D6|1<<LCD_D5|0<<LCD_D4;
        LCP|=1<<LCD_E|0<<LCD_RW|0<<LCD_RS;
        _delay_ms(1);
        LCP&=~(1<<LCD_E);
        _delay_ms(1);
        //--------4 bit--dual line---------------
        LCDsendCommand(0b00101000);
        LCDsendCommand(0b00001100);
        uint8_t ch=0, chn=0;
        while(ch<64)
        {
                LCDdefinechar((LcdCustomChar+ch),chn++);
                ch=ch+8;
        }
#else
        //8 bit part
        _delay_ms(15);
        LDP=0x00;
        LCP=0x00;
        LDDR|=1<<LCD_D7|1<<LCD_D6|1<<LCD_D5|1<<LCD_D4|1<<LCD_D3
                        |1<<LCD_D2|1<<LCD_D1|1<<LCD_D0;
        LCDR|=1<<LCD_E|1<<LCD_RW|1<<LCD_RS;
   //---------one------
        LDP=0<<LCD_D7|0<<LCD_D6|1<<LCD_D5|1<<LCD_D4|0<<LCD_D3
                        |0<<LCD_D2|0<<LCD_D1|0<<LCD_D0;
        LCP|=1<<LCD_E|0<<LCD_RW|0<<LCD_RS;
        _delay_ms(1);
        LCP&=~(1<<LCD_E);
        _delay_ms(1);
        //-----------two-----------
```

```
        LDP=0<<LCD_D7|0<<LCD_D6|1<<LCD_D5|1<<LCD_D4|0<<LCD_D3
        |0<<LCD_D2|0<<LCD_D1|0<<LCD_D0;
        LCP|=1<<LCD_E|0<<LCD_RW|0<<LCD_RS;
        _delay_ms(1);
        LCP&=~(1<<LCD_E);
        _delay_ms(1);
        //-------three-------------
        LDP=0<<LCD_D7|0<<LCD_D6|1<<LCD_D5|1<<LCD_D4|0<<LCD_D3
        |0<<LCD_D2|0<<LCD_D1|0<<LCD_D0;
        LCP|=1<<LCD_E|0<<LCD_RW|0<<LCD_RS;
        _delay_ms(1);
        LCP&=~(1<<LCD_E);
        _delay_ms(1);
        //--------8 bit dual line----------
        LDP=0<<LCD_D7|0<<LCD_D6|1<<LCD_D5|1<<LCD_D4|1<<LCD_D3
        |0<<LCD_D2|0<<LCD_D1|0<<LCD_D0;
        LCP|=1<<LCD_E|0<<LCD_RW|0<<LCD_RS;
        _delay_ms(1);
        LCP&=~(1<<LCD_E);
        _delay_ms(1);
    //-----increment address, invisible cursor shift------
        LDP=0<<LCD_D7|0<<LCD_D6|0<<LCD_D5|0<<LCD_D4|1<<LCD_D3
        |1<<LCD_D2|0<<LCD_D1|0<<LCD_D0;
        LCP|=1<<LCD_E|0<<LCD_RW|0<<LCD_RS;
        _delay_ms(1);
        LCP&=~(1<<LCD_E);
        _delay_ms(5);
        uint8_t ch=0, chn=0;
        while(ch<64)
        {
                LCDdefinechar((LcdCustomChar+ch),chn++);
                ch=ch+8;
        }
#endif
}
void LCDclr(void)                                   //Clears LCD
{
        LCDsendCommand(1<<LCD_CLR);
}
void LCDhome(void)              //LCD cursor home
{
        LCDsendCommand(1<<LCD_HOME);
}
void LCDstring(uint8_t* data, uint8_t nBytes)        //Outputs
string to LCD
{
register uint8_t i;
        if (!data) return;
        for(i=0; i<nBytes; i++)
        {
                LCDsendChar(data[i]);
```

```
        }
}
void LCDGotoXY(uint8_t x, uint8_t y)  //Cursor to X Y position
{
        register uint8_t 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;
        }
        LCDsendCommand(1<<LCD_DDRAM | DDRAMAddr);
}
void CopyStringtoLCD(const uint8_t *FlashLoc, uint8_t x,
uint8_t y)
{
        uint8_t i;
        LCDGotoXY(x,y);
        for(i=0;(uint8_t)pgm_read_byte(&FlashLoc[i]);i++)
        {
        LCDsendChar((uint8_t)pgm_read_byte(&FlashLoc[i]));
        }
}

const uint8_t degree[] PROGMEM=
{
0b00000110,//degree
0b00001001,
0b00001001,
0b00000110,
0b00000000,
0b00000000,
0b00000000,
0b00000000
};
void LCDdefinechar(const uint8_t *pc,uint8_t char_code){
        uint8_t a, pcc;
        uint16_t i;
        a=(char_code<<3)|0x40;
        for (i=0; i<8; i++){
                pcc=pgm_read_byte(&pc[i]);
                LCDsendCommand(a++);
                LCDsendChar(pcc);
                }
}
void LCDshiftLeft(uint8_t n)  //Scrol n of characters Right
{
        for (uint8_t i=0;i<n;i++)
        {
```

```
                LCDsendCommand(0x1E);
        }
}
void LCDshiftRight(uint8_t n) //Scrol n of characters Left
{
        for (uint8_t i=0;i<n;i++)
        {
                LCDsendCommand(0x18);
        }
}
void LCDcursorOn(void) //displays LCD cursor
{
        LCDsendCommand(0x0E);
}
void LCDcursorOnBlink(void)   //displays LCD blinking cursor
{
        LCDsendCommand(0x0F);
}
void LCDcursorOFF(void)       //turns OFF cursor
{
        LCDsendCommand(0x0C);
}
void LCDblank(void)           //blanks LCD
{
        LCDsendCommand(0x08);
}
void LCDvisible(void)         //Shows LCD
{
        LCDsendCommand(0x0C);
}
void LCDcursorLeft(uint8_t n) //Moves cursor by n poisitions
left
{
        for (uint8_t i=0;i<n;i++)
        {
                LCDsendCommand(0x10);
        }
}
void LCDcursorRight(uint8_t n)        //Moves cursor by n
poisitions left
{
        for (uint8_t i=0;i<n;i++)
        {
                LCDsendCommand(0x14);
        }
}
void LCDprogressBar(uint8_t progress, uint8_t maxprogress,
uint8_t length)
{
        uint8_t i;
        uint16_t pixelprogress;
```

```
        uint8_t c;
        pixelprogress =
((progress*(length*PROGRESSPIXELS_PER_CHAR))/maxprogress);
        for(i=0; i<length; i++)
        {
        if( ((i*(uint16_t)PROGRESSPIXELS_PER_CHAR)+5) >
pixelprogress )
                {
        if( ((i*(uint16_t)PROGRESSPIXELS_PER_CHAR)) >
pixelprogress )
                        {
                        c = 0;
                        }
                        else
                        {
                        c = pixelprogress %
PROGRESSPIXELS_PER_CHAR;
                        }
                }
                else
                {
                c = 5;
                }
        LCDsendChar(c);
        }
}

/////////////////
//// sht11.h ///
/////////////////
#ifndef _sensor_
#define _sensor_
#define cclr()   PORTC &= ~0x01
#define cset()   PORTC |= 0x01
#define dclr()   PORTC &= ~0x02
#define dset()   PORTC |= 0x02
extern unsigned char TimeOut,Ackbit;
unsigned char read_bit(void);
void sstart(void);
void sreset(void);
void swait(void);
void swrite(unsigned char data);
unsigned char sread(void);
unsigned int smeasure(void);
unsigned int smeasureX(void);
void inits(void);
#endif


/////////////////
//// sht11.c ///
```

```
///////////////
#include "sensor.h"
unsigned char read_bit(){
unsigned char result;
        dset();
        cset();_delay_us(2);
        DDRC = 0x01;
        cclr();
        if(PINC&(1<<1))result = 1;
        else result = 0;
        DDRC = 0x03;
        return result;
}
void sstart(){
        dset();
        cclr();_delay_us(2);
        cset();_delay_us(2);
        dclr();_delay_us(2);
        cclr();_delay_us(2);
        _delay_us(2);_delay_us(2);
        cset();_delay_us(2);
        dset();_delay_us(2);
        cclr();_delay_us(2);
}
void sreset()
{
    unsigned char i;
    dset();
    cclr();_delay_us(2);
    for (i=0; i<9; i++)
      {
        cset();_delay_us(2);
        cclr();_delay_us(2);
      }
    sstart();
}
void swait()
{
    dset();_delay_us(2);
       DDRC = 0x01;
    do{
        TimeOut=(PINC&(1<<1));
        _delay_us(2);
      }while(TimeOut!=0);
       DDRC = 0x03;
}
void swrite(unsigned char data)
{
    unsigned char i;
       dset();
       cclr();
```

```
      for (i=0; i<8; i++)
      {
        if ((data>>7)==1) dset();
        else dclr();
              cset();_delay_us(2);
        cclr();_delay_us(2);
        data<<=1;
      }

    read_bit();
}
unsigned char sread()
{
    unsigned char i,data;
    dset();
       data = 0;
       DDRC = 0x01;

       for (i=0; i<8; i++)
      {
              data<<=1;
       cset();_delay_us(2);
              data|=(PINC&(1<<1))>>1;
        cclr();
      }
       DDRC = 0x03;
    if (Ackbit==1) dset();
    else dclr();
    cset();_delay_us(2);
    cclr();
    return data;
}
unsigned int smeasure()
{
    unsigned int ADC_Temp;
    ADC_Temp = 0;
    sstart();
    swrite(0x03);
    swait();
           Ackbit=0;
           ADC_Temp=sread();
           ADC_Temp<<=8;
           Ackbit=1;
           ADC_Temp|=sread();
              ADC_Temp-=4000;
    return ADC_Temp;
}
unsigned int smeasureX()
{
    unsigned int ADC_Temp;
    ADC_Temp = 0;
```

```
    sstart();
    swrite(0x05);
    swait();
              Ackbit=0;
            ADC_Temp=sread();
            ADC_Temp<<=8;
            Ackbit=1;
            ADC_Temp|=sread();
               ADC_Temp = ((float)ADC_Temp * 0.0405) -
((float)ADC_Temp * 0.0000028 * (float)ADC_Temp)- 4;
               return ADC_Temp;
}

////////////////
/// onewire.h///
////////////////
#ifndef ONEWIRE_H_
#define ONEWIRE_H_
#include <stdint.h>
#define OW_ONE_BUS
#ifdef OW_ONE_BUS
#define OW_PIN  PD3
#define OW_IN   PIND
#define OW_OUT  PORTD
#define OW_DDR  DDRD
#define OW_CONF_DELAYOFFSET 0
#else
#if F_CPU<1843200
 #warning | experimental multi-bus-mode is not tested for
 #warning | frequencies below 1,84MHz - use OW_ONE_WIRE or
 #warning | faster clock-source (i.e. internal 2MHz R/C-Osc.)
#endif
#define OW_CONF_CYCLESPERACCESS 13
#define OW_CONF_DELAYOFFSET
#endif
#define        OW_MATCH_ROM   0x55
#define        OW_SKIP_ROM           0xCC
#define OW_SEARCH_ROM  0xF0
#define OW_SEARCH_FIRST        0xFF
#define OW_PRESENCE_ERR        0xFF
#define OW_DATA_ERR            0xFE
#define        OW_LAST_DEVICE 0x00
#define OW_ROMCODE_SIZE 8
extern uint8_t ow_reset(void);
extern uint8_t ow_bit_io( uint8_t b );
extern uint8_t ow_byte_wr( uint8_t b );
extern uint8_t ow_byte_rd( void );
extern uint8_t ow_rom_search( uint8_t diff, uint8_t *id );
extern void ow_command( uint8_t command, uint8_t *id );
extern void ow_parasite_enable(void);
extern void ow_parasite_disable(void);
```

```
extern uint8_t ow_input_pin_state(void);
#ifndef OW_ONE_BUS
extern void ow_set_bus(volatile uint8_t* in,
volatile uint8_t* out, volatile uint8_t* ddr,
uint8_t pin);
#endif
#endif

////////////////
/// onewire.c///
////////////////
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include "onewire.h"
#ifdef OW_ONE_BUS
#define OW_GET_IN()   ( OW_IN & (1<<OW_PIN))
#define OW_OUT_LOW()  ( OW_OUT &= (~(1 << OW_PIN)) )
#define OW_OUT_HIGH() ( OW_OUT |= (1 << OW_PIN) )
#define OW_DIR_IN()   ( OW_DDR &= (~(1 << OW_PIN )) )
#define OW_DIR_OUT()  ( OW_DDR |= (1 << OW_PIN) )
#else
/* set bus-config with ow_set_bus() */
uint8_t OW_PIN_MASK;
volatile uint8_t* OW_IN;
volatile uint8_t* OW_OUT;
volatile uint8_t* OW_DDR;
#define OW_GET_IN()   ( *OW_IN & OW_PIN_MASK )
#define OW_OUT_LOW()  ( *OW_OUT &= (uint8_t) ~OW_PIN_MASK )
#define OW_OUT_HIGH() ( *OW_OUT |= (uint8_t)  OW_PIN_MASK )
#define OW_DIR_IN()   ( *OW_DDR &= (uint8_t) ~OW_PIN_MASK )
#define OW_DIR_OUT()  ( *OW_DDR |= (uint8_t)  OW_PIN_MASK )

void ow_set_bus(volatile uint8_t* in,
       volatile uint8_t* out,
       volatile uint8_t* ddr,
       uint8_t pin)
{
       OW_DDR=ddr;
       OW_OUT=out;
       OW_IN=in;
       OW_PIN_MASK=(1<<pin);
       ow_reset();
}
#endif

uint8_t ow_input_pin_state()
{
       return OW_GET_IN();
}
void ow_parasite_enable(void)
```

```
{
    OW_OUT_HIGH();
       OW_DIR_OUT();
}
void ow_parasite_disable(void)
{
    OW_OUT_LOW();
       OW_DIR_IN();
}
uint8_t ow_reset(void)
{
       uint8_t err;
       uint8_t sreg;
       OW_OUT_LOW(); // disable internal pull-up
                           (maybe on from parasite)
       OW_DIR_OUT(); // pull OW-Pin low for 480us
       _delay_us(480);
       sreg=SREG;
       cli();
       OW_DIR_IN(); // input
       _delay_us(66);
       err = OW_GET_IN();     // no presence detect
       SREG=sreg; // sei()
       _delay_us(480-66);
       if( OW_GET_IN() == 0 ) // short circuit
       err = 1;
       return err;
}
uint8_t ow_bit_io( uint8_t b )
{
       uint8_t sreg;
       sreg=SREG;
       cli();
       OW_DIR_OUT();
       _delay_us(1);
       if ( b ) OW_DIR_IN();
       _delay_us(15-1-OW_CONF_DELAYOFFSET);
       if( OW_GET_IN() == 0 ) b = 0;         _delay_us(60-15);
       OW_DIR_IN();
       SREG=sreg;
       _delay_us(30);
       return b;
}
uint8_t ow_byte_wr( uint8_t b )
{
       uint8_t i = 8, j;
       do     {
              j = ow_bit_io( b & 1 );
              b >>= 1;
              if( j ) b |= 0x80;
              }
```

```
        while( --i );
        return b;
}
uint8_t ow_byte_rd( void )
{
  return ow_byte_wr( 0xFF );
}
uint8_t ow_rom_search( uint8_t diff, uint8_t *id )
{
        uint8_t i, j, next_diff;
        uint8_t b;
        if( ow_reset() ) return OW_PRESENCE_ERR;
        ow_byte_wr( OW_SEARCH_ROM );
        next_diff = OW_LAST_DEVICE;
        i = OW_ROMCODE_SIZE * 8;                                    do
{
        j = 8;                                                      do
{
        b = ow_bit_io( 1 );
        if( ow_bit_io( 1 ) ) {
        if( b )
        return OW_DATA_ERR;                                         }
        else {
        if( !b ) {
        if( diff > i || ((*id & 1) && diff != i) ) {
                        b = 1;
                next_diff = i;
        }
                        }
                        }
        ow_bit_io( b );
        *id >>= 1;
        if( b ) *id |= 0x80;
        i--;
        } while( --j );
        id++;
        } while( i );
        return next_diff;
        }
void ow_command( uint8_t command, uint8_t *id )
{
        uint8_t i;
        ow_reset();
        if( id ) {
                ow_byte_wr( OW_MATCH_ROM );
        i = OW_ROMCODE_SIZE;
                do      {
                        ow_byte_wr( *id );
                        id++;
                        }
                        while( --i );
```

```
        }
        else {
                ow_byte_wr( OW_SKIP_ROM );
        }
                ow_byte_wr( command );
        }

////////////////
///// crc.h ////
////////////////
#ifndef CRC8_H_
#define CRC8_H_
#include <inttypes.h>
uint8_t crc8 (uint8_t* data_in, uint16_t
number_of_bytes_to_read);
#endif

////////////////
///// crc.c ////
////////////////
#include <stdint.h>
#define CRC8INIT       0x00
#define CRC8POLY       0x18     //0X18 = X^8+X^5+X^4+X^0
uint8_t crc8 ( uint8_t *data_in, uint16_t
number_of_bytes_to_read )
{
        uint8_t  crc;
        uint16_t loop_count;
        uint8_t  bit_counter;
        uint8_t  data;
        uint8_t  feedback_bit;
        crc = CRC8INIT;
        for (loop_count = 0; loop_count !=
        number_of_bytes_to_read; loop_count++)
        {
                data = data_in[loop_count];
                bit_counter = 8;
                do {
                feedback_bit = (crc ^ data) & 0x01;
                        if ( feedback_bit == 0x01 ) {
                                crc = crc ^ CRC8POLY;
                        }
                        crc = (crc >> 1) & 0x7F;
                        if ( feedback_bit == 0x01 ) {
                                crc = crc | 0x80;
                        }
                        data = data >> 1;
                        bit_counter--;
                } while (bit_counter > 0);
        }
        return crc;
```

```
}

////////////////
/// ds18s20.h //
////////////////
#ifndef DS18X20_H_
#define DS18X20_H_
#include <stdlib.h>
#include <stdint.h>
#define DS18X20_EEPROMSUPPORT
#define DS18X20_VERBOSE
#define DS18X20_OK              0x00
#define DS18X20_ERROR           0x01
#define DS18X20_START_FAIL      0x02
#define DS18X20_ERROR_CRC       0x03
#define DS18X20_POWER_PARASITE 0x00
#define DS18X20_POWER_EXTERN   0x01
#define DS18S20_ID 0x10
#define DS18B20_ID 0x28
#define DS18X20_CONVERT_T             0x44
#define DS18X20_READ                  0xBE
#define DS18X20_WRITE                 0x4E
#define DS18X20_EE_WRITE              0x48
#define DS18X20_EE_RECALL             0xB8
#define DS18X20_READ_POWER_SUPPLY     0xB4
#define DS18B20_CONF_REG    4
#define DS18B20_9_BIT       0
#define DS18B20_10_BIT      (1<<5)
#define DS18B20_11_BIT      (1<<6)
#define DS18B20_12_BIT      ((1<<6)|(1<<5))
#define DS18B20_9_BIT_UNDF       ((1<<0)|(1<<1)|(1<<2))
#define DS18B20_10_BIT_UNDF      ((1<<0)|(1<<1))
#define DS18B20_11_BIT_UNDF      ((1<<0))
#define DS18B20_12_BIT_UNDF      0
#define DS18B20_TCONV_12BIT      750
#define DS18B20_TCONV_11BIT      DS18B20_TCONV_12_BIT/2
#define DS18B20_TCONV_10BIT   DS18B20_TCONV_12_BIT/4
#define DS18B20_TCONV_9BIT       DS18B20_TCONV_12_BIT/8
#define DS18S20_TCONV            DS18B20_TCONV_12_BIT
#define DS18X20_FRACCONV          625
#define DS18X20_SP_SIZE  9
#define DS18X20_WRITE_SCRATCHPAD  0x4E
#define DS18X20_COPY_SCRATCHPAD   0x48
#define DS18X20_RECALL_E2         0xB8
#define DS18X20_COPYSP_DELAY      10 /* ms */
#define DS18X20_TH_REG      2
#define DS18X20_TL_REG      3

extern void DS18X20_find_sensor(uint8_t *diff, uint8_t id[]);
extern uint8_t DS18X20_get_power_status(uint8_t id[]);
extern uint8_t DS18X20_start_meas( uint8_t with_external,
```

```
uint8_t id[]);
extern uint8_t DS18X20_read_meas(uint8_t id[],
uint8_t *subzero, uint8_t *cel, uint8_t *cel_frac_bits);
extern uint8_t DS18X20_read_meas_single(uint8_t familycode,
uint8_t *subzero, uint8_t *cel, uint8_t *cel_frac_bits);
extern uint8_t DS18X20_meas_to_cel( uint8_t fc, uint8_t *sp,
uint8_t* subzero, uint8_t* cel, uint8_t* cel_frac_bits);
extern uint16_t DS18X20_temp_to_decicel(uint8_t subzero,
uint8_t cel, uint8_t cel_frac_bits);
extern int8_t DS18X20_temp_cmp(uint8_t subzero1, uint16_t
cel1,uint8_t subzero2, uint16_t cel2);

#ifdef DS18X20_EEPROMSUPPORT
// write th, tl and config-register to scratchpad (config
ignored on S20)
uint8_t DS18X20_write_scratchpad( uint8_t id[],
uint8_t th, uint8_t tl, uint8_t conf);
// read scratchpad into array SP
uint8_t DS18X20_read_scratchpad( uint8_t id[], uint8_t sp[]);
// copy values th,tl (config) from scratchpad to DS18x20 eeprom
uint8_t DS18X20_copy_scratchpad( uint8_t with_power_extern,
uint8_t id[] );
// copy values from DS18x20 eeprom to scratchpad
uint8_t DS18X20_recall_E2( uint8_t id[] );
#endif
#ifdef DS18X20_VERBOSE
extern void DS18X20_show_id_uart( uint8_t *id, size_t n );
extern uint8_t DS18X20_read_meas_all_verbose( void );
#endif
#endif

////////////////
/// ds18s20.c //
////////////////
#include <avr/io.h>
#include "ds18x20.h"
#include "onewire.h"
#include "crc8.c"
#ifdef DS18X20_EEPROMSUPPORT
#include <util/delay.h>
#endif
#ifdef DS18X20_VERBOSE
#include <string.h>

void show_sp_uart( uint8_t *sp, size_t n )
{
        size_t i;
        for( i = 0; i < n; i++ ) {
        }
}
uint8_t DS18X20_read_meas_all_verbose( void )
```

```
{
        uint8_t id[OW_ROMCODE_SIZE], sp[DS18X20_SP_SIZE], diff;
        uint8_t i;
        uint16_t meas;
        uint8_t subzero, cel, cel_frac_bits;
        for( diff = OW_SEARCH_FIRST; diff != OW_LAST_DEVICE; )
        {
                diff = ow_rom_search( diff, &id[0] );
                if( diff == OW_PRESENCE_ERR ) {
                    return OW_PRESENCE_ERR;
                }
                if( diff == OW_DATA_ERR ) {
                    return OW_DATA_ERR;
                }
        DS18X20_show_id_uart( id,OW_ROMCODE_SIZE );
        if( id[0] == DS18B20_ID || id[0] == DS18S20_ID ) {    //
        temperature sensor
        ow_byte_wr( DS18X20_READ );   // read command
                for ( i=0 ; i< DS18X20_SP_SIZE; i++ )
                                sp[i]=ow_byte_rd();
                if ( crc8( &sp[0], DS18X20_SP_SIZE ) )
                meas = sp[0];
                meas |= (uint16_t) (sp[1] << 8);
        if( id[0] == DS18S20_ID ) {}
                else if ( id[0] == DS18B20_ID ) {
        i=sp[DS18B20_CONF_REG];
                if ( (i & DS18B20_12_BIT) ==
                        DS18B20_12_BIT ) {}
                else if ( (i & DS18B20_11_BIT) ==
                                DS18B20_11_BIT ) {}
                else if ( (i & DS18B20_10_BIT) ==
                                DS18B20_10_BIT ) {}
                else { }
       DS18X20_meas_to_cel(id[0], sp, &subzero, &cel,
       &cel_frac_bits);
         }
         }
         return DS18X20_OK;
}
#endif

uint8_t DS18X20_meas_to_cel( uint8_t fc, uint8_t *sp,uint8_t*
subzero, uint8_t* cel, uint8_t* cel_frac_bits)
{
        uint16_t meas;
        uint8_t  i;
        meas = sp[0];
        meas |= ((uint16_t)sp[1])<<8;
        if( fc == DS18S20_ID ) {
        meas &= (uint16_t) 0xfffe;
        meas <<= 3;
```

```
        meas += (16 - sp[6]) - 4;
        }
        if ( meas & 0x8000 )  {
                *subzero=1;
                meas ^= 0xffff;
                meas++;
        }
        else *subzero=0;

        if ( fc == DS18B20_ID )
                i = sp[DS18B20_CONF_REG];
                if ( (i & DS18B20_12_BIT) ==
                        DS18B20_12_BIT ) ;
                else if ( (i & DS18B20_11_BIT) ==
                                DS18B20_11_BIT )
                        meas &= ~(DS18B20_11_BIT_UNDF);
                else if ( (i & DS18B20_10_BIT) ==
                                DS18B20_10_BIT )
                        meas &= ~(DS18B20_10_BIT_UNDF);
                else {
                        meas &= ~(DS18B20_9_BIT_UNDF);
                }
        }
        *cel  = (uint8_t)(meas >> 4);
        *cel_frac_bits = (uint8_t)(meas & 0x000F);
        return DS18X20_OK;
}

uint16_t DS18X20_temp_to_decicel(uint8_t subzero, uint8_t cel,
        uint8_t cel_frac_bits)
{
        uint16_t h;
        uint8_t  i;
        uint8_t need_rounding[] = { 1, 3, 4, 6, 9, 11, 12, 14 };
        h = cel_frac_bits*DS18X20_FRACCONV/1000;
        h += cel*10;
        if (!subzero) {
        for (i=0; i<sizeof(need_rounding); i++) {
              if ( cel_frac_bits == need_rounding[i] ) {
                h++;
                break;
                }
                }
        }
        return h;}

int8_t DS18X20_temp_cmp(uint8_t subzero1, uint16_t cel1,
uint8_t subzero2, uint16_t cel2)
{
int16_t t1 = (subzero1) ? (cel1*(-1)) : (cel1);
int16_t t2 = (subzero2) ? (cel2*(-1)) : (cel2);
```

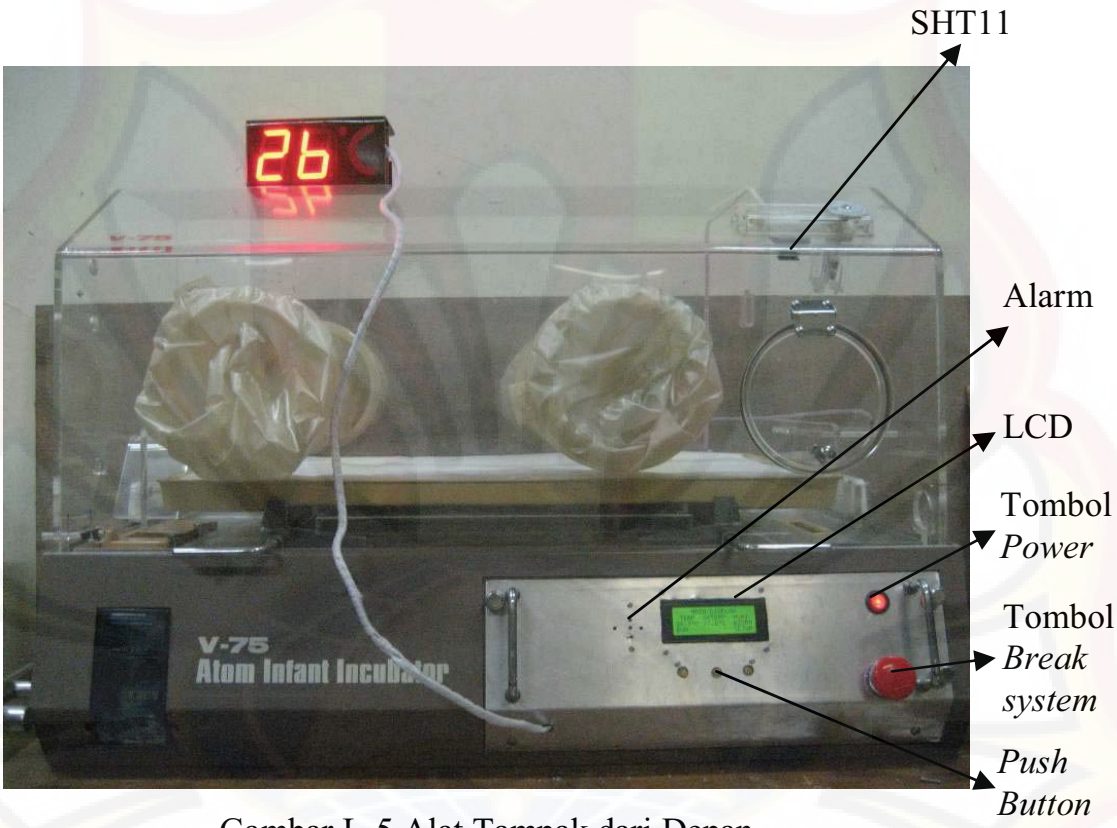
```
        if (t1<t2) return -1;
        if (t1>t2) return 1;
        return 0;
}

void DS18X20_find_sensor(uint8_t *diff, uint8_t id[])
{
        for (;;) {
        *diff = ow_rom_search( *diff, &id[0] );
        if ( *diff==OW_PRESENCE_ERR || *diff==OW_DATA_ERR ||
        *diff == OW_LAST_DEVICE ) return;
        if ( id[0] == DS18B20_ID || id[0] ==
        DS18S20_ID ) return;
        }
}

uint8_t DS18X20_get_power_status(uint8_t id[])
{
    uint8_t pstat;
    ow_reset();
    ow_command(DS18X20_READ_POWER_SUPPLY, id);
     pstat=ow_bit_io(1);
     ow_reset();
     return (pstat) ?
     DS18X20_POWER_EXTERN:DS18X20_POWER_PARASITE;
}

uint8_t DS18X20_start_meas( uint8_t with_power_extern, uint8_t
id[])
{
 ow_reset();
 if( ow_input_pin_state() ){
     ow_command( DS18X20_CONVERT_T, id );
        if (with_power_extern != DS18X20_POWER_EXTERN)
                ow_parasite_enable();
                return DS18X20_OK;
        }
        else {
                #ifdef DS18X20_VERBOSE
                #endif
                return DS18X20_START_FAIL;
        }
}

uint8_t DS18X20_read_meas(uint8_t id[], uint8_t
*subzero,uint8_t *cel, uint8_t *cel_frac_bits)
{
        uint8_t i;
        uint8_t sp[DS18X20_SP_SIZE];
        ow_reset();
        ow_command(DS18X20_READ, id);
```

```
        for ( i=0 ; i< DS18X20_SP_SIZE; i++ ) sp[i]=ow_byte_rd();
        if ( crc8( &sp[0], DS18X20_SP_SIZE ) )
        return DS18X20_ERROR_CRC;
        DS18X20_meas_to_cel(id[0], sp, subzero, cel,
cel_frac_bits);
        return DS18X20_OK;
}

uint8_t DS18X20_read_meas_single(uint8_t familycode, uint8_t
*subzero,
        uint8_t *cel, uint8_t *cel_frac_bits)
{
        uint8_t i;
        uint8_t sp[DS18X20_SP_SIZE];

        ow_command(DS18X20_READ, NULL);
        for ( i=0 ; i< DS18X20_SP_SIZE; i++ ) sp[i]=ow_byte_rd();
        if ( crc8( &sp[0], DS18X20_SP_SIZE ) )
        return DS18X20_ERROR_CRC;
        DS18X20_meas_to_cel(familycode, sp, subzero, cel,
cel_frac_bits);
        return DS18X20_OK;
}

#ifdef DS18X20_EEPROMSUPPORT

uint8_t DS18X20_write_scratchpad( uint8_t id[],
        uint8_t th, uint8_t tl, uint8_t conf)
{
        ow_reset();
        if( ow_input_pin_state() ) {
        ow_command( DS18X20_WRITE_SCRATCHPAD, id );
        ow_byte_wr(th);
        ow_byte_wr(tl);
        if (id[0] == DS18B20_ID) ow_byte_wr(conf);   return
DS18X20_OK;
        }
        else {
                #ifdef DS18X20_VERBOSE
                #endif
                return DS18X20_ERROR;
        }
}

uint8_t DS18X20_read_scratchpad( uint8_t id[], uint8_t sp[] )
{
        uint8_t i;
        ow_reset();
        if( ow_input_pin_state() ) {
        ow_command( DS18X20_READ, id );
        for ( i=0 ; i< DS18X20_SP_SIZE; i++ ) sp[i]=ow_byte_rd();
```

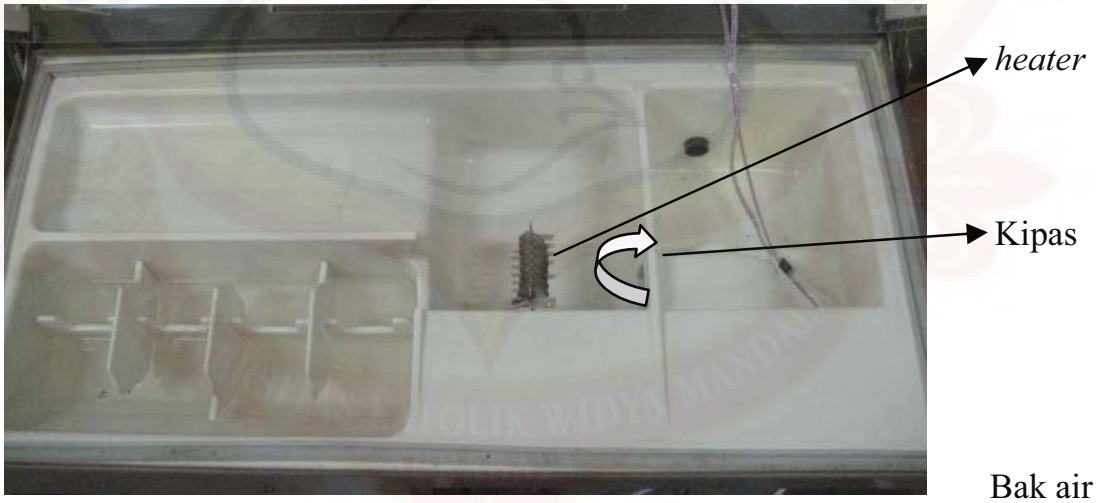
```
        return DS18X20_OK;
        }
        else {
                #ifdef DS18X20_VERBOSE
                #endif
                return DS18X20_ERROR;
        }
}

uint8_t DS18X20_copy_scratchpad( uint8_t with_power_extern,
uint8_t id[] )
{
     ow_reset();
     if( ow_input_pin_state() ) {
     ow_command( DS18X20_COPY_SCRATCHPAD, id );
        if (with_power_extern != DS18X20_POWER_EXTERN)
           ow_parasite_enable();
     _delay_ms(DS18X20_COPYSP_DELAY);
        if (with_power_extern != DS18X20_POWER_EXTERN)
           ow_parasite_disable();
     return DS18X20_OK;
        }
     else {
                #ifdef DS18X20_VERBOSE
                #endif
                return DS18X20_START_FAIL;
        }
}

uint8_t DS18X20_recall_E2( uint8_t id[] )
{
        ow_reset();
        if( ow_input_pin_state() ) {
ow_command( DS18X20_RECALL_E2, id );
            _delay_ms(DS18X20_COPYSP_DELAY);
            return DS18X20_OK;
        }
        else {
                #ifdef DS18X20_VERBOSE
                #endif
                return DS18X20_ERROR;
        }
}
#endif
```

LAMPIRAN 3

FOTO ALAT



Gambar L-5 Alat Tampak dari Depan



Gambar L-6 Letak Posisi *Heater* dan Kipas pada Inkubator

LAMPIRAN 4

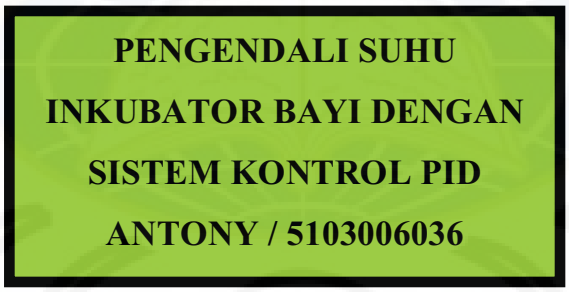
PETUNJUK PENGGUNAAN ALAT

Pada lampiran ini akan dijelaskan beberapa petunjuk dalam penggunaan alat sebagai berikut :

- Deskripsi alat
- Pengaturan suhu (*setpoint*)
- Pengaturan alarm
- Sistem keselamatan dan tombol *reset*

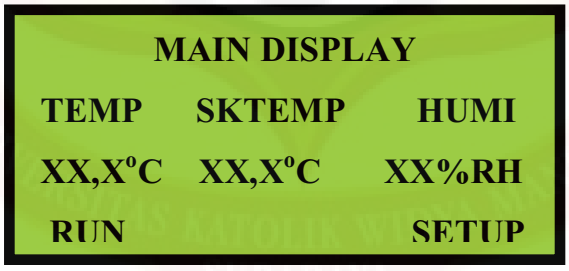
Deskripsi Alat

- Pada saat *power* dari alat dinyalakan, maka alat akan menampilkan tampilan awal berupa nama alat seperti pada gambar L-7.



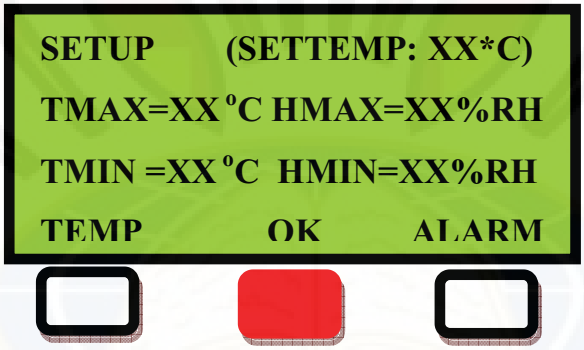
Gambar L-7 Tampilan awal ketika *power* dinyalakan

- Setelah beberapa detik menampilkan nama alat dan mahasiswa maka alat akan masuk ke tampilan MAIN DISPLAY seperti pada gambar L-8.



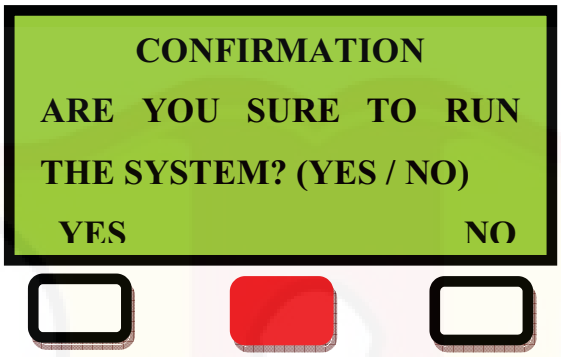
Gambar L-8 Tampilan utama alat

- Pada gambar L-8, TEMP menunjukkan suhu ruangan inkubator, SKTEMP menunjukkan suhu kulit bayi, HUMI menunjukkan kelembaban ruangan inkubator.
- Pada MAIN DISPLAY ada 2 tombol, yaitu tombol "RUN" dan "SETUP".
- Untuk mengatur suhu dan alarm, maka tombol yang ditekan adalah yang berada di bawah "SETUP" pada gambar L-8. Dan akan masuk ke tampilan menu pengaturan suhu dan alarm seperti pada gambar L-9.



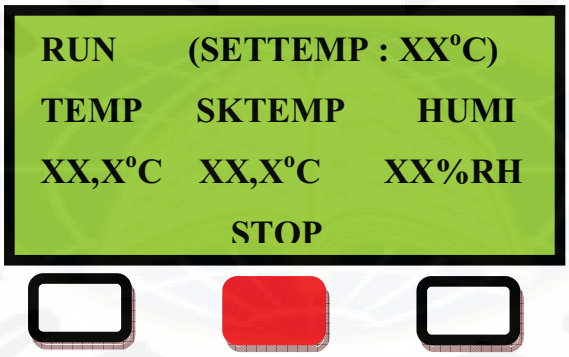
Gambar L-9 Tampilan menu pengaturan suhu dan alarm

- Pada gambar L-9, SETTEMP menunjukkan nilai suhu (*setpoint*) yang ditentukan, TMAX, TMIN, HMAX, dan HMIN menunjukkan batas suhu maksimal, suhu minimal, kelembaban maksimal, dan kelembaban minimal alarm akan berbunyi dan LED 7 segmen akan berkedip-kedip sebagai peringatan bahaya.
- Tekan tombol "TEMP" untuk mengatur suhu, tekan tombol "ALARM" untuk mengatur alarm, dan langsung tekan tombol "OK" jika sudah selesai mengatur suhu dan alarm / tetap menggunakan pengaturan suhu dan alarm yang sebelumnya.
- Untuk menjalankan sistem, maka tombol yang ditekan adalah yang berada di bawah "RUN" pada gambar L-8. Setelah ditekan, maka akan muncul konfirmasi sistem seperti gambar L-10.



Gambar L-10 Tampilan konfirmasi sistem

- Tekan tombol "YES" untuk menjalankan sistem dan akan masuk ke tampilan ketika sistem berjalan seperti pada gambar L-11. Tekan tombol "NO" jika tidak ingin menjalankan sistem dan akan kembali ke tampilan "MAIN DISPLAY" seperti pada gambar L-8.



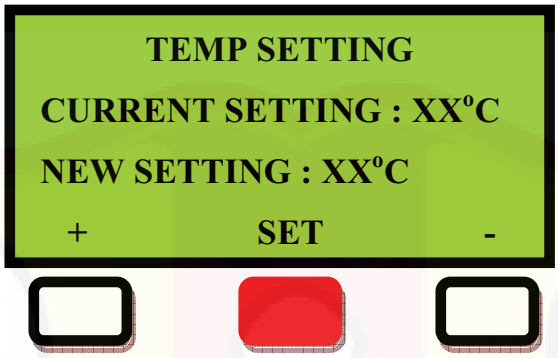
Gambar L-11 Tampilan ketika sistem dijalankan

- Tekan tombol "STOP" pada gambar L-11 untuk menghentikan sistem dan sistem akan kembali ke MAIN DISPLAY seperti gambar L-8.

Pengaturan Suhu (*setpoint*)

Pengaturan suhu untuk menentukan suhu yang akan dikontrol pada inkubator dengan rentang suhu 30 – 38 °C. Langkah-langkahnya adalah sebagai berikut :

- Tekan tombol "TEMP" yang ada pada menu pengaturan suhu dan alarm pada gambar L-9 untuk mengatur suhu (*setpoint*).
- Setelah ditekan, maka tampilan akan berubah seperti gambar L-12.



Gambar L-12 Tampilan untuk pengaturan suhu

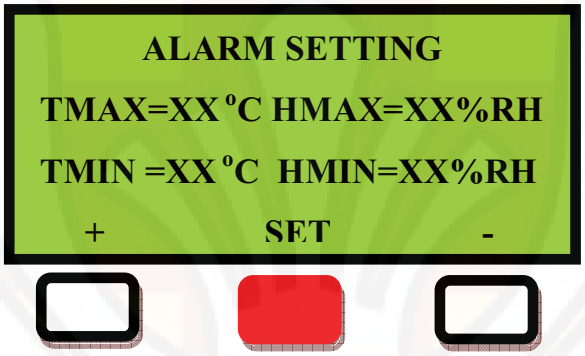
- Pada gambar L-12, CURRENT SETTING menunjukkan pengaturan suhu sebelumnya dan NEW SETTING menunjukkan pengaturan suhu yang baru.
- Kursor akan berkedip-kedip di bagian NEW SETTING untuk menandakan bahwa alat dalam kondisi pengaturan suhu.
- Untuk menambah suhu, maka tombol yang ditekan adalah yang berada di bawah "+". Sedangkan untuk mengurangi suhu, maka tombol yang ditekan adalah yang berada di bawah "-".
- Setelah selesai mengatur, maka tekan tombol yang ada di bawah "SET". Hasil pengaturan suhu akan disimpan dan kembali ke menu pengaturan suhu dan alarm pada gambar L-9 dan menampilkan pengaturan suhu pada "SETTEMP".

Pengaturan Alarm

Pengaturan alarm untuk menentukan batas alarm suhu maksimal (TMAX), suhu minimal (TMIN), kelembaban maksimal (HMAX), dan kelembaban minimal (HMIN). Jika suhu atau kelembaban inkubator melebihi batas maksimal atau di bawah batas minimal dari pengaturan, maka alarm akan berbunyi dan LED 7 segmen akan berkedip-kedip sebagai peringatan bahaya.

Berikut adalah langkah-langkah pengaturan alarm :

- Tekan tombol "ALARM" yang ada pada menu pengaturan suhu dan alarm pada gambar L-9 untuk mengatur alarm.
- Setelah ditekan, maka akan muncul tampilan seperti gambar L-13.



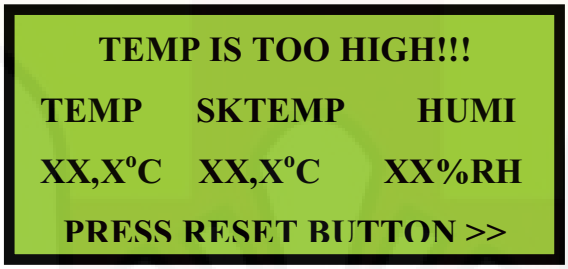
Gambar L-13 Tampilan untuk pengaturan alarm

- Pada gambar L-13, tombol "+" untuk menambah batas alarm suhu dan kelembaban dan tombol "-" untuk mengurangi.
- Kursor akan berkedip-kedip di bagian TMAX terlebih dahulu. Setelah mengatur TMAX, maka tekan tombol "SET" untuk mengatur TMIN dan kursor juga akan berpindah ke bagian TMIN. Begitu seterusnya hingga mengatur HMIN.
- Tekan tombol "SET" setelah selesai mengatur HMIN, maka pengaturan alarm akan disimpan dan kembali ke menu pengaturan suhu dan alarm dan menampilkan pengaturan alarm seperti pada gambar L-9.

Sistem Keselamatan dan Tombol *reset*

Sistem keselamatan pada alat ini berupa alarm audio (*buzzer*) dan alarm visual (LED 7 segmen). Sistem keselamatan akan bekerja jika suhu atau kelembaban ruangan inkubator melewati batas maksimal atau di bawah batas minimal pengaturan alarm dengan membunyikan *buzzer* dan LED 7 segmen akan berkedip-kedip sebagai peringatan bahaya. Selain itu, LCD

akan menampilkan kondisi apa yang menimbulkan adanya peringatarn alarm yang ditunjukkan seperti gambar L-14.



Gambar L-14 Tampilan LCD jika ada peringatan bahaya

Pada gambar L-14 menunjukkan adanya peringatan bahaya karena suhu ruangan inkubator terlalu tinggi hingga melebihi TMAX (batas suhu maksimal alarm). Untuk peringatan bahaya jika di bawah TMIN, LCD akan menampilkan "TEMP IS TOO LOW!!!". Untuk peringatan bahaya jika di bawah HMAX, LCD akan menampilkan "HUMI IS TOO HIGH!!!". Untuk peringatan bahaya jika di bawah HMIN, LCD akan menampilkan "HUMI IS TOO LOW!!!".

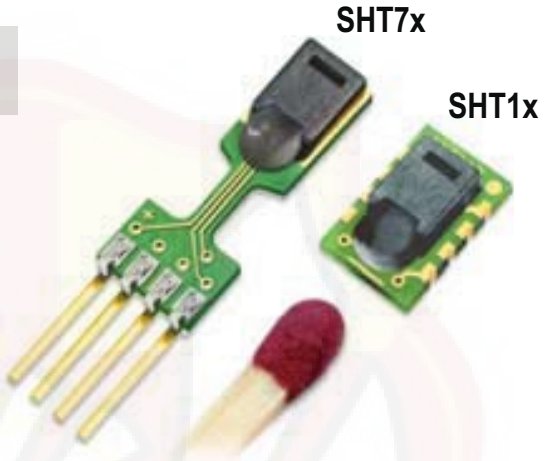
Tombol *reset* berfungsi untuk me-*reset* sistem jika adanya peringatan bahaya. Setelah tombol tersebut ditekan, maka sistem akan di-*reset* dan tampilan akan kembali seperti pada awalnya ketika *power* dinyalakan yang ditunjukkan pada gambar L-7.

SENSIRION
THE SENSOR COMPANY

SHT1x / SHT7x

Humidity & Temperature Sensor

Evaluation Kit Available



_ Relative humidity and temperature sensors
_ Dew point
_ Fully calibrated, digital output
_ Excellent long-term stability
_ No external components required
_ Ultra low power consumption
_ Surface mountable or 4-pin fully interchangeable
_ Small size
_ Automatic power down

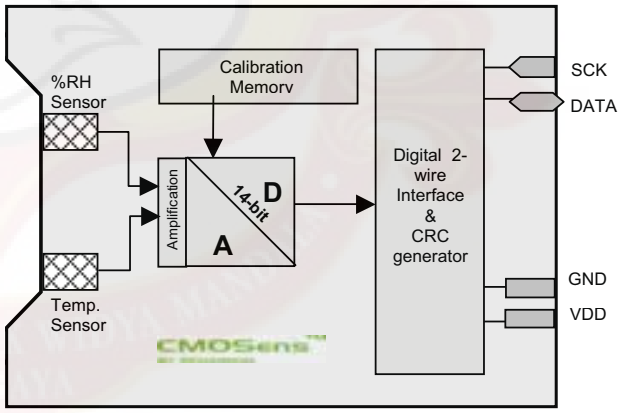
SHT1x / SHT7x Product Summary

The SHTxx is a single chip relative humidity and temperature multi sensor module comprising a calibrated digital output. Application of industrial CMOS processes with patented micro-machining (CMOSens® technology) ensures highest reliability and excellent long term stability. The device includes a capacitive polymer sensing element for relative humidity and a bandgap temperature sensor. Both are seamlessly coupled to a 14bit analog to digital converter and a serial interface circuit on the same chip. This results in superior signal quality, a fast response time and insensitivity to external disturbances (EMC) at a very competitive price.
Each SHTxx is individually calibrated in a precision humidity chamber with a chilled mirror hygrometer as reference. The calibration coefficients are programmed into the OTP memory. These coefficients are used internally during measurements to calibrate the signals from the sensors.
The 2-wire serial interface and internal voltage regulation allows easy and fast system integration. Its tiny size and low power consumption makes it the ultimate choice for even the most demanding applications.
The device is supplied in either a surface-mountable LCC (Leadless Chip Carrier) or as a pluggable 4-pin single-in-line type package. Customer specific packaging options may be available on request.

Applications

_ HVAC
_ Automotive
_ Consumer Goods
_ Weather Stations
_ (De-) Humidifiers
_ Test & Measurement
_ Data Logging
_ Automation
_ White Goods
_ Medical

Block Diagram



Ordering Information

Part Number	Humidity accuracy	Temperature accuracy	Package
SHT11	±3.5%RH	±0.5°C @25°C	SMT (LCC)
SHT15	±2.0%RH	±0.4°C	SMT (LCC)
SHT71	±3.5%RH	±0.5°C @25°C	4-pin single-in-line
SHT75	±2.0%RH	±0.4°C	4-pin single-in-line

1 Sensor Performance Specifications

Parameter	Conditions	Min.	Typ.	Max.	Units
Humidity					
Resolution (2)		0.5	0.03	0.03	% RH
		8	12	12	bit
Repeatability			±0.1		% RH
Accuracy (1) Uncertainty	linearized	see figure 1			
Interchangeability		Fully interchangeable			
Nonlinearity	raw data		±3		% RH
	linearized		<<1		% RH
Range		0		100	% RH
Response time	1/e (63%) slowly moving air		4		s
Hysteresis			±1		% RH
Long term stability	Typical		< 1		% RH/yr
Temperature					
Resolution (2)		0.04	0.01	0.01	°C
		0.07	0.02	0.02	°F
		12	14	14	bit
Repeatability			±0.1		°C
			±0.2		°F
Accuracy		see figure 1			
Range		-40		123.8	°C
		-40		254.9	°F
Response Time	1/e (63%)	5		30	s

Table 1 Sensor Performance Specifications

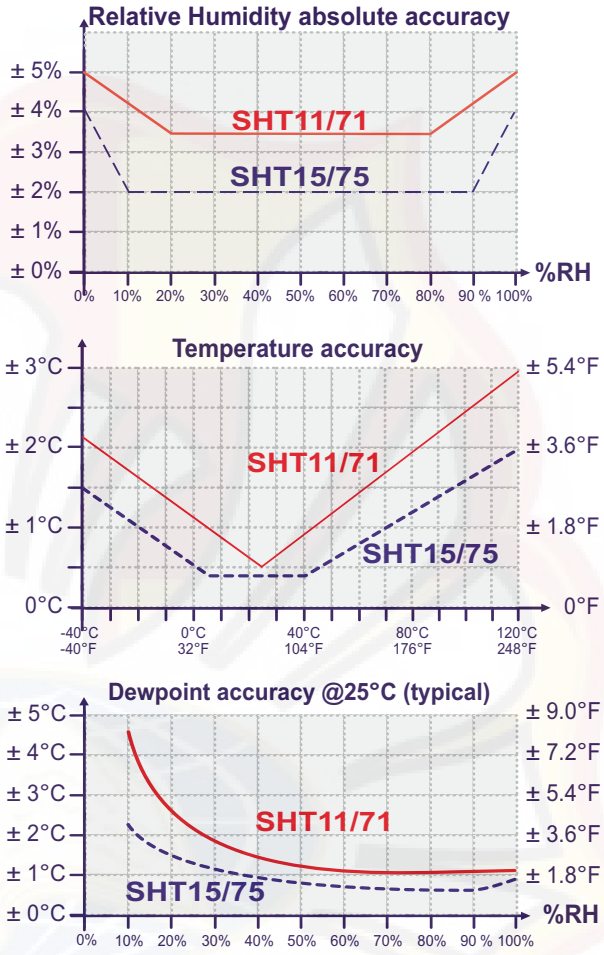


Figure 1 Rel. Humidity, Temperature and Dewpoint accuracies

2 Interface Specifications

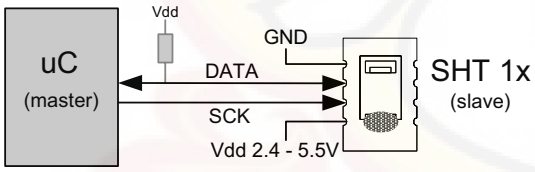


Figure 2 Typical application circuit

2.1 Power Pins

The SHTxx requires a voltage supply between 2.4V and 5.5V. After powerup the device needs 11ms to reach its "sleep" state. No commands should be sent before that time. Power supply pins (VDD, GND) may be decoupled with a 100 nF capacitor.

2.2 Serial Interface (Bidirectional 2-wire)

The serial interface of the SHTxx is optimized for sensor readout and power consumption and is not compatible with I²C interfaces, see FAQ for details.

2.2.1 Serial clock input (SCK)

The SCK is used to synchronize the communication between a microcontroller and the SHT1x / SHT7x. Since the interface consists of fully static logic there is no minimum SCK frequency.

2.2.2 Serial data (DATA)

The DATA tristate pin is used to transfer data in and out of the device. DATA **changes after the falling edge** and is **valid on the rising edge** of the serial clock SCK. During communication the DATA line must remain stable while SCK is high. To avoid signal contention the microcontroller should only drive DATA low. An external pull-up resistor (e.g 10kΩ) is required to pull the signal high. (See Figure 2) Pull-up resistors are often included in I/O circuits of microcontrollers. See Table 5 for detailed IO characteristics.

(1) Each SHTxx is tested to be within RH accuracy specifications at 25°C (77°F) and 48°C (118.4°F)
(2) The default measurement resolution of 14bit (temperature) and 12bit (humidity) can be reduced to 12 and 8 bit through the status register.

2.2.3 Sending a command

To initiate a transmission, a "Transmission Start" sequence has to be issued. It consists of a lowering of the DATA line while SCK is high, followed by a low pulse on SCK and raising DATA again while SCK is still high.

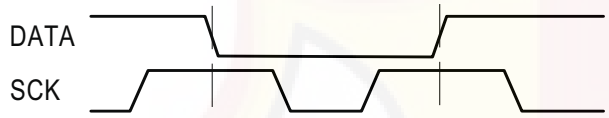


Figure 3 "Transmission Start" sequence

The subsequent command consists of three address bits (only "000" is currently supported) and five command bits. The SHT1x/SHT7x indicates the proper reception of a command by pulling the DATA pin low (ACK bit) after the falling edge of the 8th SCK clock. The DATA line is released (and goes high) after the falling edge of the 9th SCK clock.

Command	Code
Reserved	0000x
Measure Temperature	**00011**
Measure Humidity	**00101**
Read Status Register	00111
Write Status Register	00110
Reserved	0101x-1110x
Soft reset, resets the interface, clears the status register to default values wait minimum 11ms before next command	**11110**

Table 2 SHTxx list of commands

2.2.4 Measurement sequence (RH and T)

After issuing a measurement command ('00000101' for RH, '00000011' for Temperature) the controller has to wait for the measurement to complete. This takes approximately 11/55/210ms for a 8/12/14bit measurement. The exact time varies by up to ±15% with the speed of the internal oscillator. To signal the completion of a measurement, the SHT1x pulls down the data line. The controller **must** wait for this "data ready" signal before starting to toggle SCK again.

Two bytes of measurement data and one byte of CRC checksum will then be transmitted. The uC must acknowledge each byte by pulling the DATA line low. All values are MSB first, right justified. (e.g. the 5th SCK is MSB for a 12bit value, for a 8bit result the first byte is not used).
Communication terminates after the acknowledge bit of the CRC data. If CRC-8 checksum is not used the controller may terminate the communication after the measurement data LSB by keeping ack high.
The device automatically returns to sleep mode after the measurement and communication have ended.

Warning: To keep self heating below 0.1°C the SHTxx should not be active for more than 15% of the time (e.g. max. 3 measurements / second for 12bit accuracy).

2.2.5 Connection reset sequence

If communication with the device is lost the following signal sequence will reset its serial interface:
While leaving DATA high, toggle SCK 9 or more times. This must be followed by a "Transmission Start" sequence preceding the next command. This sequence resets the interface only. The status register preserves its content.

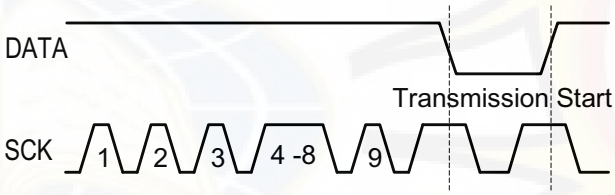


Figure 4 Connection reset sequence

2.2.6 CRC-8 Checksum calculation

The whole digital transmission is secured by a 8 bit checksum. It ensures that any wrong data can be detected and eliminated.
Please consult application note "CRC-8 Checksum Calculation" for information on how to calculate the CRC.

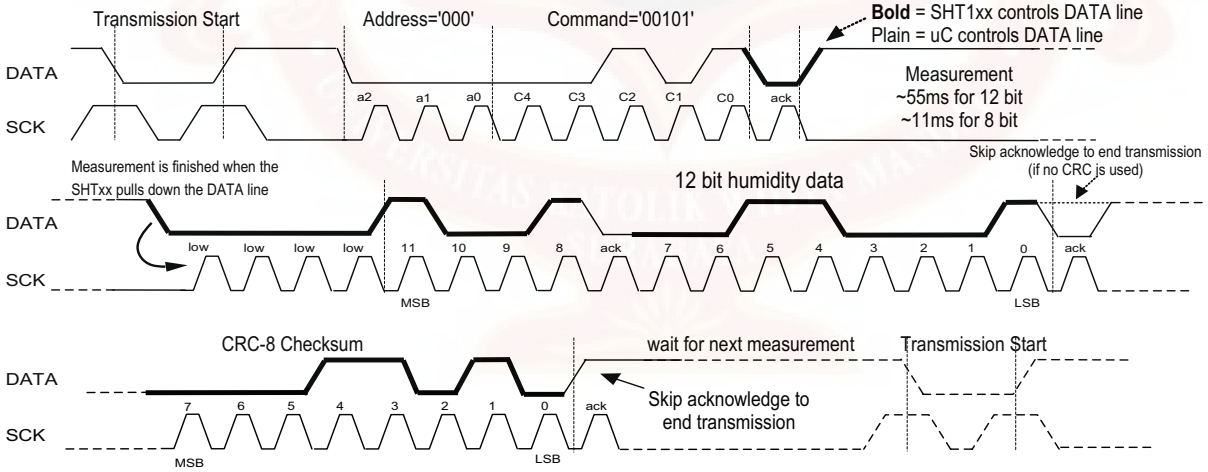


Figure 5 Example RH measurement sequence for value "0000'1001 ' 0011'0001"= 2353 = 75.79%RH @ 25°C

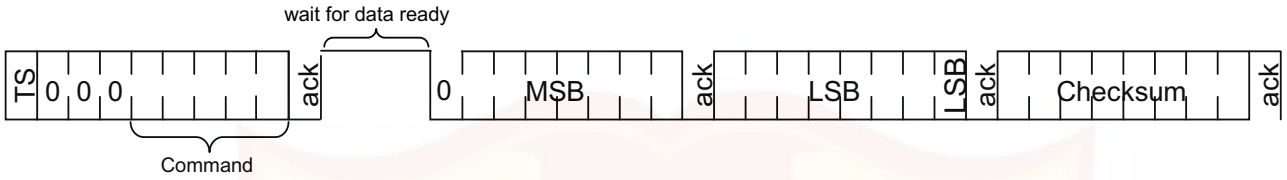


Figure 6 Overview of Measurement Sequence (TS =Transmission Start)

2.3 Status Register

Some of the advanced functions of the SHTxx are available through the status register. The following section gives a brief overview of these features. A more detailed description is available in the application note "Status Register"

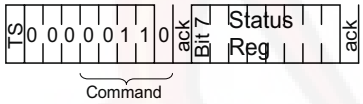


Figure 7 Status Register Write

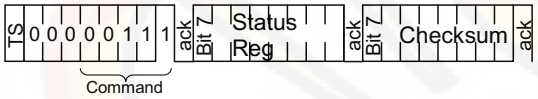


Figure 8 Status Register Read

Bit	Type	Description	Default	
7		reserved	0	
6	R	End of Battery (low voltage detection) '0' for Vdd > 2.47 '1' for Vdd < 2.47	X	No default value, bit is only updated after a measurement
5		reserved	0	
4		reserved	0	
3		For Testing only, do not use	0	
2	R/W	Heater	0	off
1	R/W	no reload from OTP	0	reload
0	R/W	'1' = 8bit RH / 12bit Temperature resolution '0' = 12bit RH / 14bit Temperature resolution	0	12bit RH 14bit Temp.

Table 3 Status Register Bits

2.3.1 Measurement resolution

The default measurement resolution of 14bit (temperature) and 12bit (humidity) can be reduced to 12 and 8 bit. This is especially useful in high speed or extreme low power applications.

2.3.2 End of Battery

The "End of Battery" function detects VDD voltages below 2.47V. Accuracy is ±0.05V

2.3.3 Heater

An on chip heating element can be switched on. It will increase the temperature of the sensor by approximately 5°C (9°F). Power consumption will increase by ~8mA @ 5V.
Applications:
By comparing temperature and humidity values before and after switching on the heater, proper functionality of both sensors can be verified.

- In high (>95%) RH environments heating the sensor element will prevent condensation, improve response time and accuracy

Warning: While heated the SHTxx will show higher temperatures and a lower relative humidity than with no heating.

2.4 Electrical Characteristics[1]

VDD=5V, Temperature= 25°C unless otherwise noted

Parameter	Conditions	Min.	Typ.	Max.	Units
Power supply DC		2.4	5	5.5	V
Supply current	measuring		550		µA
	average	2[2]	28[3]		µA
	sleep		0.3	1	µA
Low level output voltage		0		20%	Vdd
High level output voltage		75%		100%	Vdd
Low level input voltage	Negative going	0		20%	Vdd
High level input voltage	Positive going	80%		100%	Vdd
Input current on pads				1	µA
Output peak current	on			4	mA
	Tristated (off)		10		µA

Table 4 SHTxx DC Characteristics

	Parameter	Conditions	Min	Typ.	Max.	Unit
F_{SCK}	SCK frequency	VDD > 4.5 V			10	MHz
		VDD < 4.5 V			1	MHz
T_{RFO}	DATA fall time	Output load 5 pF	3.5	10	20	ns
		Output load 100 pF	30	40	200	ns
T_{CLx}	SCK hi/low time		100			ns
T_V	DATA valid time			250		ns
T_{SU}	DATA set up time		100			ns
T_{HO}	DATA hold time		0	10		ns
T_R/T_F	SCK rise/fall time			200		ns

Table 5 SHTxx I/O Signals Characteristics

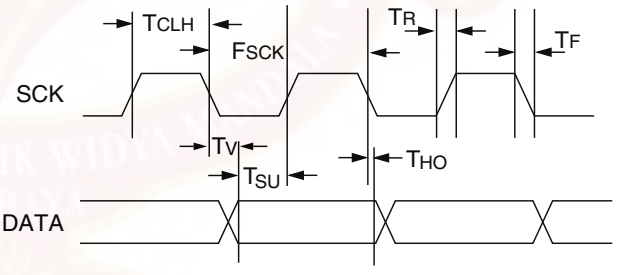


Figure 9 Timing Diagram

1) Parameters are periodically sampled and not 100% tested
(2) With one measurement of 8 bit accuracy without OTP reload per second
(3) With one measurement of 12bit accuracy per second

3 Converting Output to Physical Values

3.1 Relative Humidity

To compensate for the non-linearity of the humidity sensor and to obtain the full accuracy it is recommended to convert the readout with the following formula[1]:

$$RH_{linear} = c_1 + c_2 \bullet SO_{RH} + c_3 \bullet SO_{RH}^2$$

SO_{RH}	c_1	c_2	c_3
12 bit	-4	0.0405	$-2.8 * 10^{-6}$
8 bit	-4	0.648	$-7.2 * 10^{-4}$

Table 6 Humidity conversion coefficients

For simplified, less computation intense conversion formulas see application note “RH and Temperature Non-Linearity Compensation”.

The humidity sensor has no significant voltage dependency.

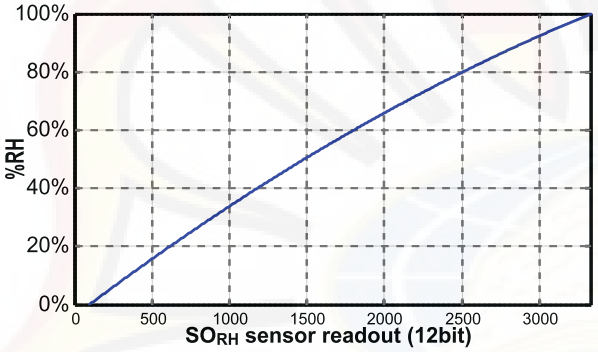


Figure 10 Conversion from SO_{RH} to relative humidity

3.1.1 Compensation of RH/Temperature dependency

For temperatures significantly different from 25°C (~77°F) the temperature coefficient of the RH sensor should be considered:

$$RH_{true} = (T_{°C} - 25) \bullet (t_1 + t_2 \bullet SO_{RH}) + RH_{linear}$$

SO_{RH}	t_1	t_2
12 bit	0.01	0.00008
8 bit	0.01	0.00128

Table 7 Temperature compensation coefficients

This equals ~0.12%RH / °C @ 50%RH

3.2 Temperature

The bandgap PTAT (Proportional To Absolute Temperature) temperature sensor is very linear by design. Use the following formula to convert from digital readout to temperature:

$$Temperature = d_1 + d_2 \bullet SO_T$$

VDD	d_1 [°C]	d_1 [°f]
5V	-40.00	-40.00
4V	-39.75	-39.50
3.5V	-39.66	-39.35
3V	-39.60	-39.28
2.5V	-39.55	-39.23

SO_T	d_2 [°C]	d_2 [°f]
14bit	0.01	0.018
12bit	0.04	0.072

Table 8 Temperature conversion coefficients

For improved accuracies in extreme temperatures with more computation intense conversion formulas see application note “RH and Temperature Non-Linearity Compensation”.

3.3 Dewpoint

Since humidity and temperature are both measured on the same monolithic chip, the SHTxx allows superb dewpoint measurements. See application note “Dewpoint calculation” for more.

[1] Where SO_{RH} is the sensor output for relative humidity

4 Applications Information

4.1 Operating and Storage Conditions

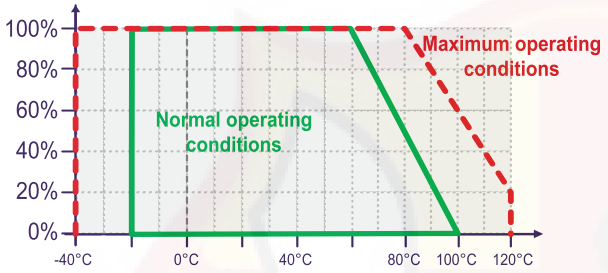


Figure 11 Recommended operating conditions

Conditions outside the recommended range may temporarily offset the RH signal up to ±3%RH. After return to normal conditions it will slowly return towards calibration state by itself. See 4.3 "Reconditioning Procedure" to accelerate this process. Prolonged exposure to extreme conditions may accelerate ageing.

4.2 Exposure to Chemicals

Vapors may interfere with the polymer layers used for capacitive humidity sensors. The diffusion of chemicals into the polymer may cause a shift in both offset and sensitivity. In a clean environment the contaminants will slowly outgas. The reconditioning procedure described below will accelerate this process.
High levels of pollutants may cause permanent damage to the sensing polymer.

4.3 Reconditioning Procedure

The following reconditioning procedure will bring the sensor back to calibration state after exposure to extreme conditions or chemical vapors.
80-90°C (176-194°F) at < 5%RH for 24h (baking) followed by
20-30°C (70-90°F) at > 74%RH for 48h (re-hydration)

4.4 Qualifications

Extensive tests were performed in various environments. Please contact SENSIRION for additional information.

Environment	Norm	Results[1]
Temperature Cycles	JESD22-A104-B -40°C / 125°C, 1000cy	Within Specifications
HAST Pressure Cooker	JESD22-A110-B 2.3bar 125°C 85%RH	Reversible shift by +2% RH
Salt Atmosphere	DIN-50021ss	Within Specifications
Freezing cycles fully submerged	-20 / +90°C, 100cy 30min dwell time	Reversible shift by +2% RH
Various Automotive Chemicals	DIN 72300-5	Within Specifications
Cigarette smoke	Equivalent to 15years in a mid-size car	Within Specifications

Table 9 Qualification tests (excerpt)

[1] The temperature sensor passed all tests without any detectable drift. Package and electronics also passed 100%

4.5 ESD (Electrostatic Discharge)

ESD immunity is qualified according to MIL STD 883E, method 3015 (Human Body Model at ±2kV)).
Latch-up immunity is provided at a force current of ±100 mA with T_{amb}=80°C according to JEDEC 17.
See application note "ESD, Latchup and EMC" for more information.

4.6 Temperature Effects

The relative humidity of a gas strongly depends on its temperature. It is therefore essential to keep humidity sensors at the same temperature as the air of which the relative humidity is to be measured.
If the SHTxx shares a PCB with electronic components that give off heat it should be mounted far away and below the heat source and the housing must remain well ventilated.
To reduce heat conduction copper layers between the SHT1x and the rest of the PCB should be minimized and a slit may be milled in between. (See figure 14)

4.7 Materials Used for Sealing / Mounting

Many materials absorb humidity and will act as a buffer, increasing response times and hysteresis. Materials in the vicinity of the sensor must therefore be carefully chosen.
Recommended materials are:
All Metals, LCP, POM (Delrin), PTFE (Teflon), PE, PEEK, PP, PB, PPS, PSU, PVDF, PVF
For sealing and gluing (use sparingly):
high filled epoxy for electronic packaging (e.g. glob top, underfill), and Silicone are recommended.

4.8 Membranes

A membrane can be used to prevent dirt from entering the housing and to protect the sensor. It will also reduce peak concentrations of chemical vapors. For optimal response times air volume behind the membrane must be kept to a minimum.

4.9 Light

The SHTxx is not light sensitive. However prolonged direct exposure to sunshine or strong UV radiation may age the housing.

4.10 Wiring Considerations and Signal Integrity

Carrying the SCK and DATA signal parallel and in close proximity (e.g. in wires) for more than 10cm may result in cross talk and loss of communication. This may be resolved by routing VDD and/or GND between the two data signals.
Please see the application note "ESD, Latchup and EMC" for more information.

5 Package Information

5.1 SHT1x (surface mountable)

Pin	Name	Comment
1	GND	Ground
2	DATA	Serial data, bidirectional
3	SCK	Serial clock, input
4	VDD	Supply 2.4 – 5.5V
	NC	Remaining pins must be left unconnected

Table 10 SHT1x Pin Description

5.1.1 Package type

The SHT1x is supplied in a surface-mountable LCC (Leadless Chip Carrier) type package. The sensors housing consists of a Liquid Crystal Polymer (LCP) cap with epoxy glob top on a standard 0.8mm FR4 substrate. The device is free of lead, Cd and Hg.
Device size is 7.42 x 4.88 x 2.5 mm (0.29 x 0.19 x 0.1 inch)
Weight 100mg

The production date is printed onto the cap in white numbers in the form wwy. e.g. "351" = week 35, 2001.

5.1.2 Delivery Conditions

The SHT1x are shipped in standard IC tubes by 80 units per tube or in 12mm tape. Reels are individually labelled with barcode and human readable labels.

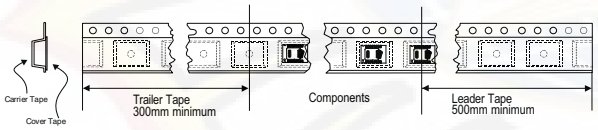


Figure 12 Tape configuration and unit orientation

5.1.3 Mounting Examples

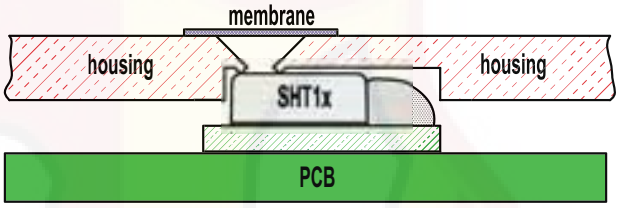


Figure 13 SHT1x housing mounting example

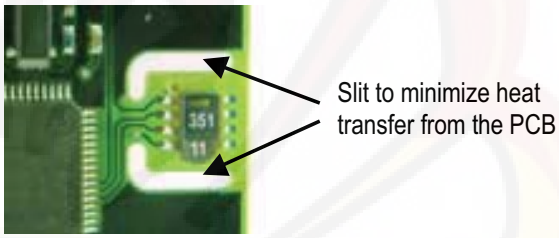


Figure 14 SHT1x PCB Mounting example

5.1.4 Soldering Information

Standard reflow soldering ovens may be used at maximum 235°C for 20 seconds.
For manual soldering contact time must be limited to 5 seconds at up to 350°C.
After soldering the devices should be stored at >74%RH for at least 24h to allow the polymer to rehydrate.
Please consult the application note "Soldering procedure" for more information.

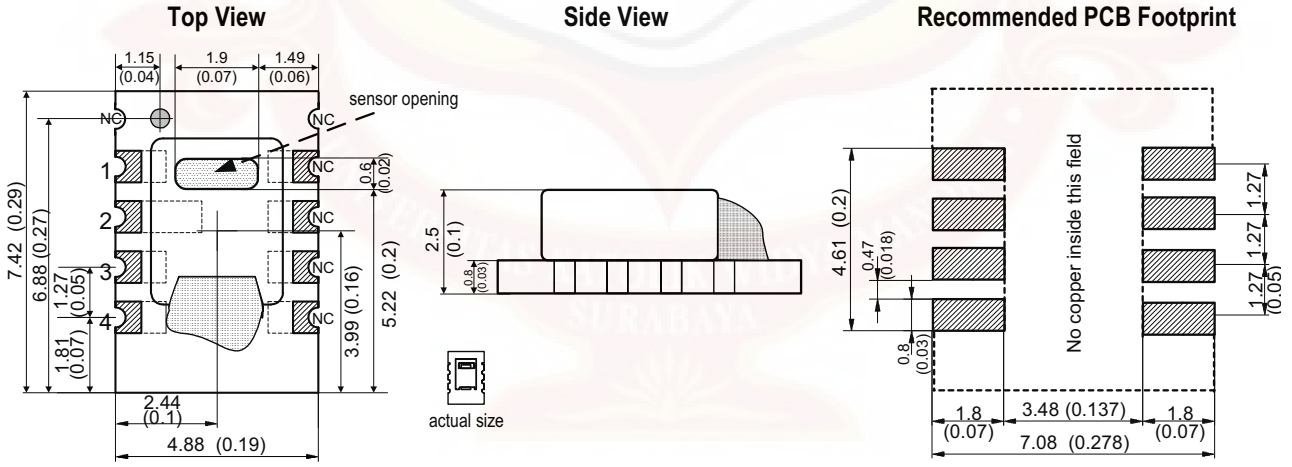


Figure 15 SHT1x drawing and footprint dimensions in mm (inch)

5.2 SHT7x (4-pin single-in-line)

Pin	Name	Comment
1	*SCK*	*Serial clock input*
2	*VDD*	*Supply 2.4 – 5.5V*
3	*GND*	*Ground*
4	*DATA*	*Serial data bidirectional*

Table 11 SHT7x Pin Description

5.2.1 Package type[1]

The device is supplied in a single-in-line pin type package. The sensor housing consists of a Liquid Crystal Polymer (LCP) cap with epoxy glob top on a standard 0.6mm FR4 substrate. The device is Cd and Hg free.
The sensor head is connected to the pins by a small bridge to minimize heat conduction and response times.
A 100nF capacitor is mounted on the back side between VDD and GND.
All pins are gold plated to avoid corrosion. They can be soldered or mate with most 1.27mm (0.05") sockets
e.g.: Preci-dip / Mill-Max 851-93-004-20-001 or similar
Total weight: 168mg, weight of sensor head: 73mg

The production date is printed onto the cap in white numbers in the form wwy. e.g. "351" = week 35, 2001.

5.2.2 Delivery Conditions

The SHT7x are shipped in 32mm tape. These reeled parts in standard option are shipped with 500 units per 13inch diameter reel. Reels are individually labelled with barcode and human readable labels.

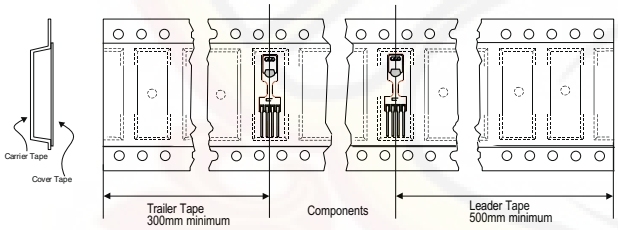


Figure 16 Tape configuration and unit orientation

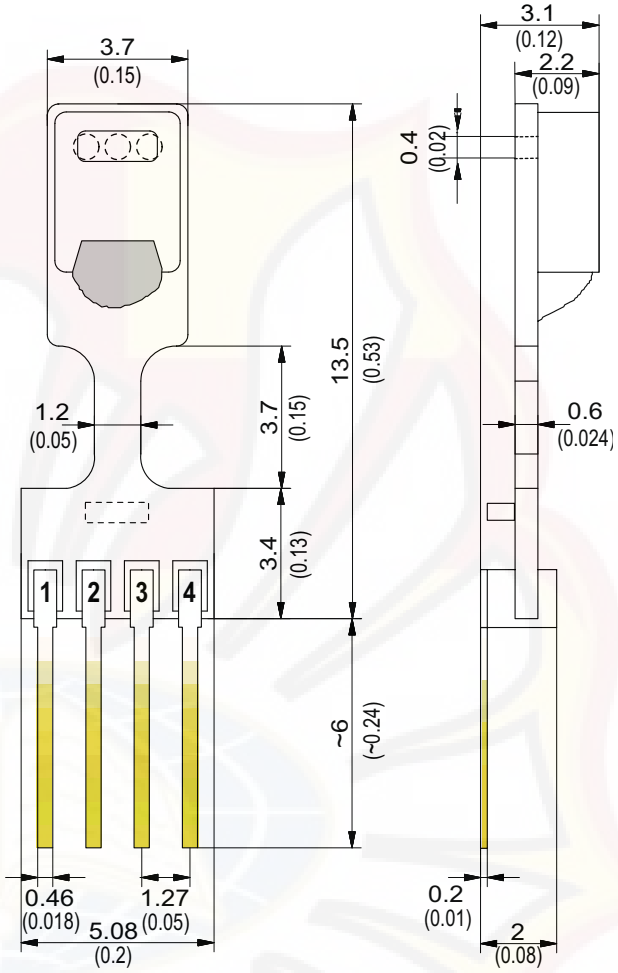


Figure 17 SHT7x dimensions in mm (inch)

5.2.3 Soldering Information

Standard wave SHT7x soldering ovens may be used at maximum 235°C for 20 seconds.
For manual soldering contact time must be limited to 5 seconds at up to 350°C.
After wave soldering the devices should be stored at >74%RH for at least 24h to allow the polymer to rehydrate.
Please consult the application note "Soldering procedure" for more information.

[1] Other packaging options may be available on request.

6 Revision history

Date	Version	Page(s)	Changes
February 2002	Preliminary	1-9	First public release
June 2002	Preliminary		Added SHT7x information
March 2003	Final v2.0	1-9	Major remake, added application information etc. Various small modifications

The latest version of this document and all application notes can be found at:
www.sensirion.com/en/download/humiditysensor/SHT11.htm

7 Important Notices

7.1 Warning, personal injury

Do not use this product as safety or emergency stop devices or in any other application where failure of the product could result in personal injury. Failure to comply with these instructions could result in death or serious injury.

Should buyer purchase or use SENSIRION AG products for any such unintended or unauthorized application, Buyer shall indemnify and hold SENSIRION AG and its officers, employees, subsidiaries, affiliates and distributors harmless against all claims, costs, damages and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that SENSIRION AG was negligent regarding the design or manufacture of the part.

7.2 ESD Precautions

The inherent design of this component causes it to be sensitive to electrostatic discharge (ESD). To prevent ESD-induced damage and/or degradation, take normal ESD precautions when handling this product.

See application note “ESD, Latchup and EMC” for more information.

7.3 Warranty

SENSIRION AG makes no warranty, representation or guarantee regarding the suitability of its product for any particular purpose, nor does SENSIRION AG assume any liability arising out of the application or use of any product or circuit and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters can and do vary in different applications. All operating parameters, including “Typical” must be validated for each customer applications by customer’s technical experts.

SENSIRION AG reserves the right, without further notice, to change the product specifications and/or information in this document and to improve reliability, functions and design.

Headquarters and Sales Office

SENSIRION AG
Eggbühlstr. 14
P.O. Box
CH-8052 Zürich
Switzerland

Phone: + 41 (0)1 306 40 00
Fax: + 41 (0)1 306 40 30
e-mail: info@sensirion.com
http://www.sensirion.com/

Sensirion humidity sensors are available from:

find your local representative at:
www.sensirion.com/reps

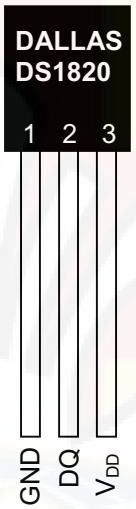
DS18S20
High Precision
1-Wire® Digital Thermometer

www.dalsemi.com

FEATURES

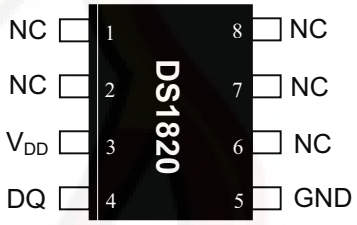
- Unique 1-wire interface requires only one port pin for communication
- Each device has a unique 64-bit serial code stored in an on-board ROM
- Multi-drop capability simplifies distributed temperature sensing applications
- Requires no external components
- Can be powered from data line. Power supply range is 3.0V to 5.5V
- Measures temperatures from –55°C to +125°C (–67°F to +257°F)
- ±0.5°C accuracy from –10°C to +85°C
- 9-bit thermometer resolution
- Converts temperature in 750 ms (max.)
- User-definable nonvolatile alarm settings
- Alarm search command identifies and addresses devices whose temperature is outside of programmed limits (temperature alarm condition)
- Applications include thermostatic controls, industrial systems, consumer products, thermometers, or any thermally sensitive system

PIN ASSIGNMENT



(BOTTOM VIEW)

TO-92
(DS18S20)



8-pin 150-mil SOIC
(DS18S20Z)

PIN DESCRIPTION

GND - Ground
DQ - Data In/Out
V_{DD} - Power Supply Voltage
NC - No Connect

DESCRIPTION

The DS18S20 Digital Thermometer provides 9–bit centigrade temperature measurements and has an alarm function with nonvolatile user-programmable upper and lower trigger points. The DS18S20 communicates over a 1-wire bus that by definition requires only one data line (and ground) for communication with a central microprocessor. It has an operating temperature range of –55°C to +125°C and is accurate to ±0.5°C over the range of –10°C to +85°C. In addition, the DS18S20 can derive power directly from the data line ("parasite power"), eliminating the need for an external power supply.

Each DS18S20 has a unique 64-bit serial code, which allows multiple DS18S20s to function on the same 1–wire bus; thus, it is simple to use one microprocessor to control many DS18S20s distributed over a large area. Applications that can benefit from this feature include HVAC environmental controls, temperature monitoring systems inside buildings, equipment or machinery, and process monitoring and control systems.

DETAILED PIN DESCRIPTIONS Table 1

8-PIN SOIC*	TO-92	SYMBOL	DESCRIPTION
5	1	GND	**Ground.**
4	2	DQ	**Data Input/Output pin.** Open-drain 1-wire interface pin. Also provides power to the device when used in parasite power mode (see "Parasite Power" section.)
3	3	V_{DD}	**Optional V_{DD} pin.** V_{DD} must be grounded for operation in parasite power mode.

*All pins not specified in this table are "No Connect" pins.

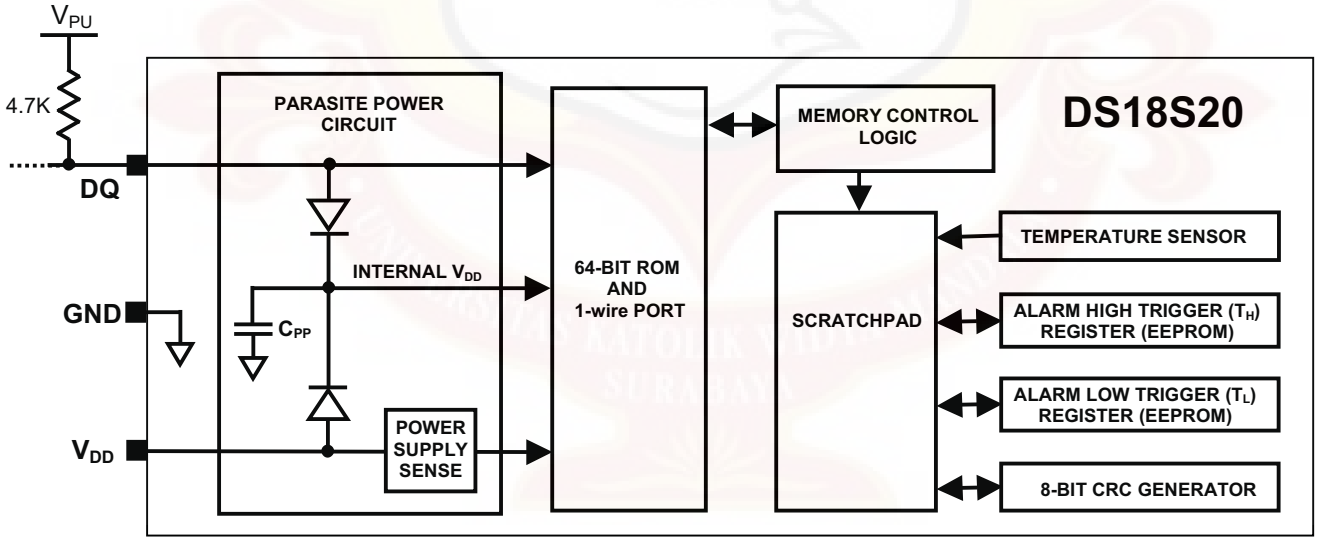
OVERVIEW

Figure 1 shows a block diagram of the DS18S20, and pin descriptions are given in Table 1. The 64-bit ROM stores the device's unique serial code. The scratchpad memory contains the 2-byte temperature register that stores the digital output from the temperature sensor. In addition, the scratchpad provides access to the 1-byte upper and lower alarm trigger registers (T_H and T_L). The T_H and T_L registers are nonvolatile (EEPROM), so they will retain data when the device is powered down.

The DS18S20 uses Dallas' exclusive 1-wire bus protocol that implements bus communication using one control signal. The control line requires a weak pullup resistor since all devices are linked to the bus via a 3-state or open-drain port (the DQ pin in the case of the DS18S20). In this bus system, the microprocessor (the master device) identifies and addresses devices on the bus using each device's unique 64-bit code. Because each device has a unique code, the number of devices that can be addressed on one bus is virtually unlimited. The 1-wire bus protocol, including detailed explanations of the commands and "time slots," is covered in the 1-WIRE BUS SYSTEM section of this datasheet.

Another feature of the DS18S20 is the ability to operate without an external power supply. Power is instead supplied through the 1-wire pullup resistor via the DQ pin when the bus is high. The high bus signal also charges an internal capacitor (C_{PP}), which then supplies power to the device when the bus is low. This method of deriving power from the 1-wire bus is referred to as "parasite power." As an alternative, the DS18S20 may also be powered by an external supply on V_{DD}.

DS18S20 BLOCK DIAGRAM Figure 1



OPERATION – MEASURING TEMPERATURE

The core functionality of the DS18S20 is its direct-to-digital temperature sensor. The temperature sensor output has 9-bit resolution, which corresponds to 0.5°C steps. The DS18S20 powers-up in a low-power idle state; to initiate a temperature measurement and A-to-D conversion, the master must issue a Convert T [44h] command. Following the conversion, the resulting thermal data is stored in the 2-byte temperature register in the scratchpad memory and the DS18S20 returns to its idle state. If the DS18S20 is powered by an external supply, the master can issue "read time slots" (see the 1-WIRE BUS SYSTEM section) after the Convert T command and the DS18S20 will respond by transmitting 0 while the temperature conversion is in progress and 1 when the conversion is done. If the DS18S20 is powered with parasite power, this notification technique cannot be used since the bus must be pulled high by a strong pullup during the entire temperature conversion. The bus requirements for parasite power are explained in detail in the POWERING THE DS18S20 section of this datasheet.

The DS18S20 output data is calibrated in degrees centigrade; for Fahrenheit applications, a lookup table or conversion routine must be used. The temperature data is stored as a 16-bit sign-extended two's complement number in the temperature register (see Figure 2). The sign bits (S) indicate if the temperature is positive or negative: for positive numbers S = 0 and for negative numbers S = 1. Table 2 gives examples of digital output data and the corresponding temperature reading.

Resolutions greater than 9 bits can be calculated using the data from the temperature, COUNT REMAIN and COUNT PER °C registers in the scratchpad. Note that the COUNT PER °C register is hard-wired to 16 (10h). After reading the scratchpad, the TEMP_READ value is obtained by truncating the 0.5°C bit (bit 0) from the temperature data (see Figure 2). The extended resolution temperature can then be calculated using the following equation:

$$TEMPERATURE = TEMP\_READ - 0.25 + \frac{COUNT\_PER\_C - COUNT\_REMAIN}{COUNT\_PER\_C}$$

Additional information about high-resolution temperature calculations can be found in Application Note 105: "High Resolution Temperature Measurement with Dallas Direct-to-Digital Temperature Sensors".

TEMPERATURE REGISTER FORMAT Figure 2

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
LS Byte	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}
	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
MS Byte	S	S	S	S	S	S	S	S

TEMPERATURE/DATA RELATIONSHIP Table 2

TEMPERATURE	DIGITAL OUTPUT (Binary)	DIGITAL OUTPUT (Hex)
+85.0°C*	0000 0000 1010 1010	00AAh
+25.0°C	0000 0000 0011 0010	0032h
+0.5°C	0000 0000 0000 0001	0001h
0°C	0000 0000 0000 0000	0000h
-0.5°C	1111 1111 1111 1111	FFFFh
-25.0°C	1111 1111 1100 1110	FFCEh
-55.0°C	1111 1111 1001 0010	FF92h

*The power-on reset value of the temperature register is +85°C

OPERATION – ALARM SIGNALING

After the DS18S20 performs a temperature conversion, the temperature value is compared to the user-defined two's complement alarm trigger values stored in the 1-byte T_H and T_L registers (see Figure 3). The sign bit (S) indicates if the value is positive or negative: for positive numbers S = 0 and for negative numbers S = 1. The T_H and T_L registers are nonvolatile (EEPROM) so they will retain data when the device is powered down. T_H and T_L can be accessed through bytes 2 and 3 of the scratchpad as explained in the MEMORY section of this datasheet.

T_H AND T_L REGISTER FORMAT Figure 3

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
S	2^6	2^5	2^5	2^5	2^2	2^1	2^0

Only bits 8 through 1 of the temperature register are used in the T_H and T_L comparison since T_H and T_L are 8-bit registers. If the result of a temperature measurement is higher than T_H or lower than T_L, an alarm condition exists and an alarm flag is set inside the DS18S20. This flag is updated after every temperature measurement; therefore, if the alarm condition goes away, the flag will be turned off after the next temperature conversion.

The master device can check the alarm flag status of all DS18S20s on the bus by issuing an Alarm Search [ECh] command. Any DS18S20s with a set alarm flag will respond to the command, so the master can determine exactly which DS18S20s have experienced an alarm condition. If an alarm condition exists and the T_H or T_L settings have changed, another temperature conversion should be done to validate the alarm condition.

POWERING THE DS18S20

The DS18S20 can be powered by an external supply on the V_{DD} pin, or it can operate in "parasite power" mode, which allows the DS18S20 to function without a local external supply. Parasite power is very useful for applications that require remote temperature sensing or that are very space constrained. Figure 1 shows the DS18S20's parasite-power control circuitry, which "steals" power from the 1-wire bus via the DQ pin when the bus is high. The stolen charge powers the DS18S20 while the bus is high, and some of the charge is stored on the parasite power capacitor (C_{PP}) to provide power when the bus is low. When the DS18S20 is used in parasite power mode, the V_{DD} pin must be connected to ground.

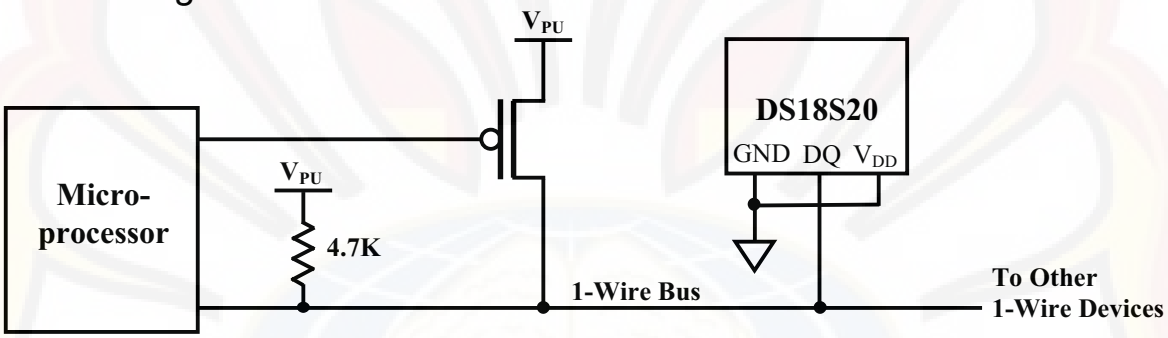
In parasite power mode, the 1-wire bus and C_{PP} can provide sufficient current to the DS18S20 for most operations as long as the specified timing and voltage requirements are met (refer to the DC ELECTRICAL CHARACTERISTICS and the AC ELECTRICAL CHARACTERISTICS sections of this data sheet). However, when the DS18S20 is performing temperature conversions or copying data from the scratchpad memory to EEPROM, the operating current can be as high as 1.5 mA. This current can cause an unacceptable voltage drop across the weak 1-wire pullup resistor and is more current than can be supplied by C_{PP}. To assure that the DS18S20 has sufficient supply current, it is necessary to provide a strong pullup on the 1-wire bus whenever temperature conversions are taking place or data is being copied from the scratchpad to EEPROM. This can be accomplished by using a MOSFET to pull the bus directly to the rail as shown in Figure 4. The 1-wire bus must be switched to the strong pullup within 10 µs (max) after a Convert T [44h] or Copy Scratchpad [48h] command is issued, and the bus must be held high by the pullup for the duration of the conversion (t_{conv}) or data transfer (t_{wr} = 10 ms). No other activity can take place on the 1-wire bus while the pullup is enabled.

The DS18S20 can also be powered by the conventional method of connecting an external power supply to the V_{DD} pin, as shown in Figure 5. The advantage of this method is that the MOSFET pullup is not required, and the 1–wire bus is free to carry other traffic during the temperature conversion time.

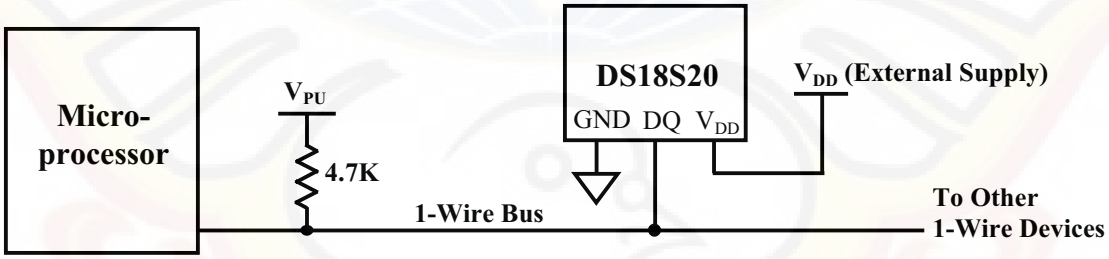
The use of parasite power is not recommended for temperatures above 100°C since the DS18S20 may not be able to sustain communications due to the higher leakage currents that can exist at these temperatures. For applications in which such temperatures are likely, it is strongly recommended that the DS18S20 be powered by an external power supply.

In some situations the bus master may not know whether the DS18S20s on the bus are parasite powered or powered by external supplies. The master needs this information to determine if the strong bus pullup should be used during temperature conversions. To get this information, the master can issue a Skip ROM [CCh] command followed by a Read Power Supply [B4h] command followed by a "read time slot". During the read time slot, parasite powered DS18S20s will pull the bus low, and externally powered DS18S20s will let the bus remain high. If the bus is pulled low, the master knows that it must supply the strong pullup on the 1-wire bus during temperature conversions.

SUPPLYING THE PARASITE-POWERED DS18S20 DURING TEMPERATURE CONVERSIONS Figure 4



POWERING THE DS18S20 WITH AN EXTERNAL SUPPLY Figure 5



64-BIT LASERED ROM CODE

Each DS18S20 contains a unique 64–bit code (see Figure 6) stored in ROM. The least significant 8 bits of the ROM code contain the DS18S20's 1–wire family code: 10h. The next 48 bits contain a unique serial number. The most significant 8 bits contain a cyclic redundancy check (CRC) byte that is calculated from the first 56 bits of the ROM code. A detailed explanation of the CRC bits is provided in the CRC GENERATION section. The 64–bit ROM code and associated ROM function control logic allow the DS18S20 to operate as a 1–wire device using the protocol detailed in the 1-WIRE BUS SYSTEM section of this datasheet.

64-BIT LASERED ROM CODE Figure 6

8-BIT CRC		48-BIT SERIAL NUMBER		8-BIT FAMILY CODE (10h)	
MSB	LSB	MSB	LSB	MSB	LSB

MEMORY

The DS18S20's memory is organized as shown in Figure 7. The memory consists of an SRAM scratchpad with nonvolatile EEPROM storage for the high and low alarm trigger registers (T_H and T_L). Note that if the DS18S20 alarm function is not used, the T_H and T_L registers can serve as general-purpose memory. All memory commands are described in detail in the DS18S20 FUNCTION COMMANDS section.

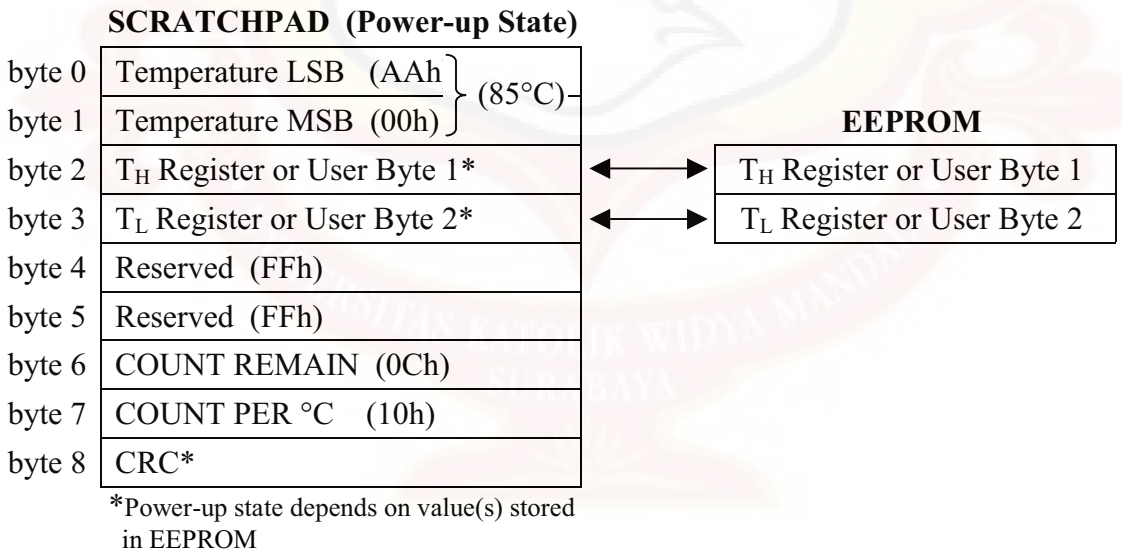
Byte 0 and byte 1 of the scratchpad contain the LSB and the MSB of the temperature register, respectively. These bytes are read-only. Bytes 2 and 3 provide access to T_H and T_L registers. Bytes 4 and 5 are reserved for internal use by the device and cannot be overwritten; these bytes will return all 1s when read. Bytes 6 and 7 contain the COUNT REMAIN and COUNT PER ºC registers, which can be used to calculate extended resolution results as explained in the OPERATION – MEASURING TEMPERATURE section.

Byte 8 of the scratchpad is read-only and contains the cyclic redundancy check (CRC) code for bytes 0 through 7 of the scratchpad. The DS18S20 generates this CRC using the method described in the CRC GENERATION section.

Data is written to bytes 2 and 3 of the scratchpad using the Write Scratchpad [4Eh] command; the data must be transmitted to the DS18S20 starting with the least significant bit of byte 2. To verify data integrity, the scratchpad can be read (using the Read Scratchpad [BEh] command) after the data is written. When reading the scratchpad, data is transferred over the 1-wire bus starting with the least significant bit of byte 0. To transfer the T_H and T_L data from the scratchpad to EEPROM, the master must issue the Copy Scratchpad [48h] command.

Data in the EEPROM registers is retained when the device is powered down; at power-up the EEPROM data is reloaded into the corresponding scratchpad locations. Data can also be reloaded from EEPROM to the scratchpad at any time using the Recall E^2 [B8h] command. The master can issue "read time slots" (see the 1-WIRE BUS SYSTEM section) following the Recall E^2 command and the DS18S20 will indicate the status of the recall by transmitting 0 while the recall is in progress and 1 when the recall is done.

DS18S20 MEMORY MAP cáÖ éÉ=T



*Power-up state depends on value(s) stored in EEPROM

CRC GENERATION

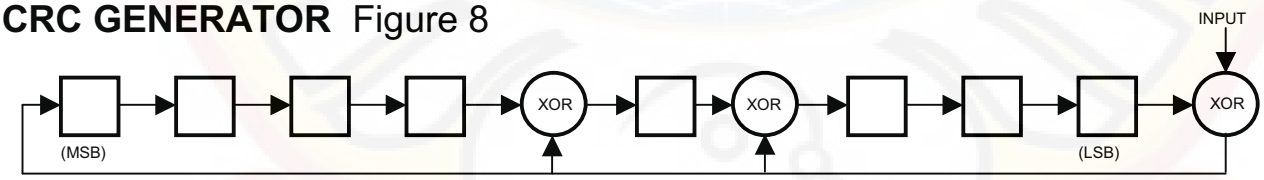
CRC bytes are provided as part of the DS18S20's 64-bit ROM code and in the 9th byte of the scratchpad memory. The ROM code CRC is calculated from the first 56 bits of the ROM code and is contained in the most significant byte of the ROM. The scratchpad CRC is calculated from the data stored in the scratchpad, and therefore it changes when the data in the scratchpad changes. The CRCs provide the bus master with a method of data validation when data is read from the DS18S20. To verify that data has been read correctly, the bus master must re-calculate the CRC from the received data and then compare this value to either the ROM code CRC (for ROM reads) or to the scratchpad CRC (for scratchpad reads). If the calculated CRC matches the read CRC, the data has been received error free. The comparison of CRC values and the decision to continue with an operation are determined entirely by the bus master. There is no circuitry inside the DS18S20 that prevents a command sequence from proceeding if the DS18S20 CRC (ROM or scratchpad) does not match the value generated by the bus master.

The equivalent polynomial function of the CRC (ROM or scratchpad) is:

$$CRC = X^8 + X^5 + X^4 + 1$$

The bus master can re-calculate the CRC and compare it to the CRC values from the DS18S20 using the polynomial generator shown in Figure 8. This circuit consists of a shift register and XOR gates, and the shift register bits are initialized to 0. Starting with the least significant bit of the ROM code or the least significant bit of byte 0 in the scratchpad, one bit at a time should shifted into the shift register. After shifting in the 56th bit from the ROM or the most significant bit of byte 7 from the scratchpad, the polynomial generator will contain the re-calculated CRC. Next, the 8-bit ROM code or scratchpad CRC from the DS18S20 must be shifted into the circuit. At this point, if the re-calculated CRC was correct, the shift register will contain all 0s. Additional information about the Dallas 1-wire cyclic redundancy check is available in Application Note 27 entitled "Understanding and Using Cyclic Redundancy Checks with Dallas Semiconductor Touch Memory Products."

CRC GENERATOR Figure 8



1-WIRE BUS SYSTEM

The 1-wire bus system uses a single bus master to control one or more slave devices. The DS18S20 is always a slave. When there is only one slave on the bus, the system is referred to as a "single-drop" system; the system is "multi-drop" if there are multiple slaves on the bus.

All data and commands are transmitted least significant bit first over the 1-wire bus.

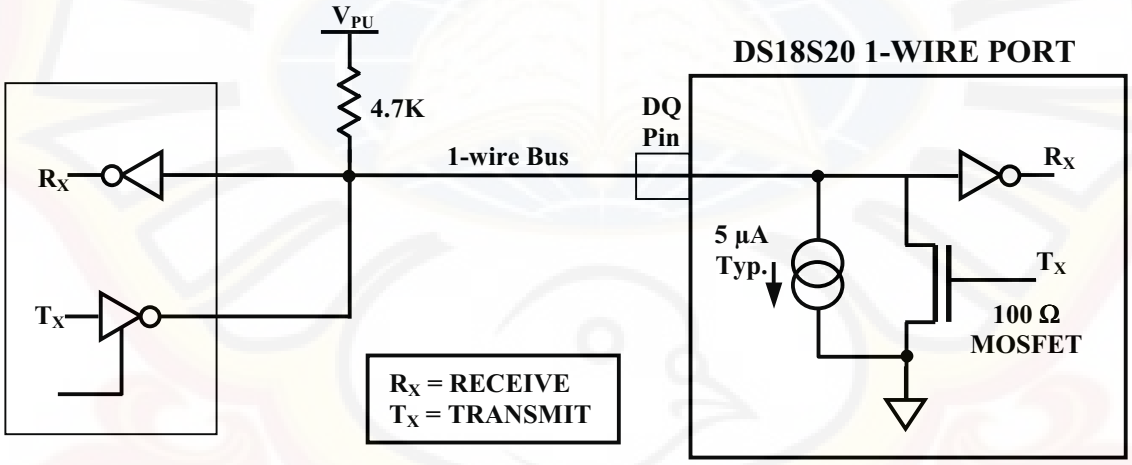
The following discussion of the 1-wire bus system is broken down into three topics: hardware configuration, transaction sequence, and 1-wire signaling (signal types and timing).

HARDWARE CONFIGURATION

The 1-wire bus has by definition only a single data line. Each device (master or slave) interfaces to the data line via an open drain or 3–state port. This allows each device to "release" the data line when the device is not transmitting data so the bus is available for use by another device. The 1-wire port of the DS18S20 (the DQ pin) is open drain with an internal circuit equivalent to that shown in Figure 9.

The 1-wire bus requires an external pullup resistor of approximately 5 kΩ; thus, the idle state for the 1-wire bus is high. If for any reason a transaction needs to be suspended, the bus MUST be left in the idle state if the transaction is to resume. Infinite recovery time can occur between bits so long as the 1-wire bus is in the inactive (high) state during the recovery period. If the bus is held low for more than 480 µs, all components on the bus will be reset.

HARDWARE CONFIGURATION Figure=V=



TRANSACTION SEQUENCE

The transaction sequence for accessing the DS18S20 is as follows:

Step 1. Initialization

Step 2. ROM Command (followed by any required data exchange)

Step 3. DS18S20 Function Command (followed by any required data exchange)

It is very important to follow this sequence every time the DS18S20 is accessed, as the DS18S20 will not respond if any steps in the sequence are missing or out of order. Exceptions to this rule are the Search ROM [F0h] and Alarm Search [ECh] commands. After issuing either of these ROM commands, the master must return to Step 1 in the sequence.

INITIALIZATION

All transactions on the 1-wire bus begin with an initialization sequence. The initialization sequence consists of a reset pulse transmitted by the bus master followed by presence pulse(s) transmitted by the slave(s). The presence pulse lets the bus master know that slave devices (such as the DS18S20) are on the bus and are ready to operate. Timing for the reset and presence pulses is detailed in the 1-WIRE SIGNALING section.

ROM COMMANDS

After the bus master has detected a presence pulse, it can issue a ROM command. These commands operate on the unique 64–bit ROM codes of each slave device and allow the master to single out a specific device if many are present on the 1-wire bus. These commands also allow the master to determine how many and what types of devices are present on the bus or if any device has experienced an alarm condition. There are five ROM commands, and each command is 8 bits long. The master device must issue an appropriate ROM command before issuing a DS18S20 function command. A flowchart for operation of the ROM commands is shown in Figure 14.

SEARCH ROM [F0h]

When a system is initially powered up, the master must identify the ROM codes of all slave devices on the bus, which allows the master to determine the number of slaves and their device types. The master learns the ROM codes through a process of elimination that requires the master to perform a Search ROM cycle (i.e., Search ROM command followed by data exchange) as many times as necessary to identify all of the slave devices. If there is only one slave on the bus, the simpler Read ROM command (see below) can be used in place of the Search ROM process. For a detailed explanation of the Search ROM procedure, refer to the iButton Book of Standards at www.ibutton.com/ibuttons/standard.pdf. After every Search ROM cycle, the bus master must return to Step 1 (Initialization) in the transaction sequence.

READ ROM [33h]

This command can only be used when there is one slave on the bus. It allows the bus master to read the slave's 64-bit ROM code without using the Search ROM procedure. If this command is used when there is more than one slave present on the bus, a data collision will occur when all the slaves attempt to respond at the same time.

MATCH ROM [55h]

The match ROM command followed by a 64–bit ROM code sequence allows the bus master to address a specific slave device on a multi-drop or single-drop bus. Only the slave that exactly matches the 64–bit ROM code sequence will respond to the function command issued by the master; all other slaves on the bus will wait for a reset pulse.

SKIP ROM [CCh]

The master can use this command to address all devices on the bus simultaneously without sending out any ROM code information. For example, the master can make all DS18S20s on the bus perform simultaneous temperature conversions by issuing a Skip ROM command followed by a Convert T [44h] command. Note, however, that the Skip ROM command can only be followed by the Read Scratchpad [BEh] command when there is one slave on the bus. This sequence saves time by allowing the master to read from the device without sending its 64–bit ROM code. This sequence will cause a data collision on the bus if there is more than one slave since multiple devices will attempt to transmit data simultaneously.

ALARM SEARCH [ECh]

The operation of this command is identical to the operation of the Search ROM command except that only slaves with a set alarm flag will respond. This command allows the master device to determine if any DS18S20s experienced an alarm condition during the most recent temperature conversion. After

every Alarm Search cycle (i.e., Alarm Search command followed by data exchange), the bus master must return to Step 1 (Initialization) in the transaction sequence. Refer to the OPERATION – ALARM SIGNALING section for an explanation of alarm flag operation.

DS18S20 FUNCTION COMMANDS

After the bus master has used a ROM command to address the DS18S20 with which it wishes to communicate, the master can issue one of the DS18S20 function commands. These commands allow the master to write to and read from the DS18S20's scratchpad memory, initiate temperature conversions and determine the power supply mode. The DS18S20 function commands, which are described below, are summarized in Table 4 and illustrated by the flowchart in Figure 15.

CONVERT T [44h]

This command initiates a single temperature conversion. Following the conversion, the resulting thermal data is stored in the 2-byte temperature register in the scratchpad memory and the DS18S20 returns to its low-power idle state. If the device is being used in parasite power mode, within 10 µs (max) after this command is issued the master must enable a strong pullup on the 1-wire bus for the duration of the conversion (t_{conv}) as described in the POWERING THE DS18S20 section. If the DS18S20 is powered by an external supply, the master can issue read time slots after the Convert T command and the DS18S20 will respond by transmitting 0 while the temperature conversion is in progress and 1 when the conversion is done. In parasite power mode this notification technique cannot be used since the bus is pulled high by the strong pullup during the conversion.

WRITE SCRATCHPAD [4Eh]

This command allows the master to write 2 bytes of data to the DS18S20's scratchpad. The first byte is written into the T_H register (byte 2 of the scratchpad), and the second byte is written into the T_L register (byte 3 of the scratchpad). Data must be transmitted least significant bit first. Both bytes MUST be written before the master issues a reset, or the data may be corrupted.

READ SCRATCHPAD [BEh]

This command allows the master to read the contents of the scratchpad. The data transfer starts with the least significant bit of byte 0 and continues through the scratchpad until the 9th byte (byte 8 – CRC) is read. The master may issue a reset to terminate reading at any time if only part of the scratchpad data is needed.

COPY SCRATCHPAD [48h]

This command copies the contents of the scratchpad T_H and T_L registers (bytes 2 and 3) to EEPROM. If the device is being used in parasite power mode, within 10 µs (max) after this command is issued the master must enable a strong pullup on the 1-wire bus for at least 10 ms as described in the POWERING THE DS18S20 section.

RECALL E^2 [B8h]

This command recalls the alarm trigger values (T_H and T_L) from EEPROM and places the data in bytes 2 and 3, respectively, in the scratchpad memory. The master device can issue read time slots following the Recall E^2 command and the DS18S20 will indicate the status of the recall by transmitting 0 while the recall is in progress and 1 when the recall is done. The recall operation happens automatically at power-up, so valid data is available in the scratchpad as soon as power is applied to the device.

READ POWER SUPPLY [B4h]

The master device issues this command followed by a read time slot to determine if any DS18S20s on the bus are using parasite power. During the read time slot, parasite powered DS18S20s will pull the bus low, and externally powered DS18S20s will let the bus remain high. Refer to the POWERING THE DS18S20 section for usage information for this command.

DS18S20 FUNCTION COMMAND SET Table 4

Command	Description	Protocol	1-Wire Bus Activity After Command is Issued	Notes
TEMPERATURE CONVERSION COMMANDS				
Convert T	Initiates temperature conversion.	44h	DS18S20 transmits conversion status to master (not applicable for parasite-powered DS18S20s).	1
MEMORY COMMANDS				
Read Scratchpad	Reads the entire scratchpad including the CRC byte.	BEh	DS18S20 transmits up to 9 data bytes to master.	2
Write Scratchpad	Writes data into scratchpad bytes 2 and 3 (T_H and T_L).	4Eh	Master transmits 2 data bytes to DS18S20.	3
Copy Scratchpad	Copies T_H and T_L data from the scratchpad to EEPROM.	48h	None	1
Recall E^2	Recalls T_H and T_L data from EEPROM to the scratchpad.	B8h	DS18S20 transmits recall status to master.	
Read Power Supply	Signals DS18S20 power supply mode to the master.	B4h	DS18S20 transmits supply status to master.	

NOTES:

1. For parasite-powered DS18S20s, the master must enable a strong pullup on the 1-wire bus during temperature conversions and copies from the scratchpad to EEPROM. No other bus activity may take place during this time.
2. The master can interrupt the transmission of data at any time by issuing a reset.
3. Both bytes must be written before a reset is issued.

1-WIRE SIGNALING

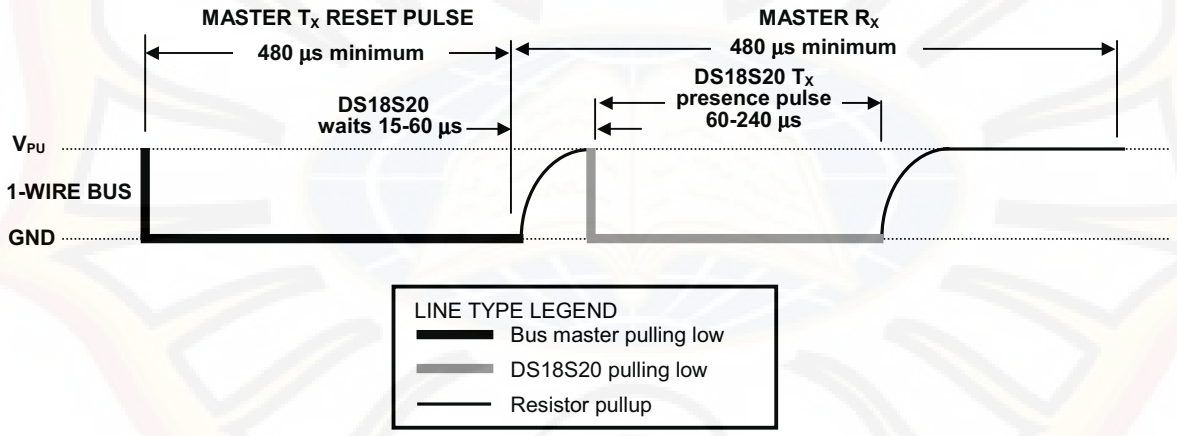
The DS18S20 uses a strict 1-wire communication protocol to insure data integrity. Several signal types are defined by this protocol: reset pulse, presence pulse, write 0, write 1, read 0, and read 1. All of these signals, with the exception of the presence pulse, are initiated by the bus master.

INITIALIZATION PROCEDURE: RESET AND PRESENCE PULSES

All communication with the DS18S20 begins with an initialization sequence that consists of a reset pulse from the master followed by a presence pulse from the DS18S20. This is illustrated in Figure 10. When the DS18S20 sends the presence pulse in response to the reset, it is indicating to the master that it is on the bus and ready to operate.

During the initialization sequence the bus master transmits (T_X) the reset pulse by pulling the 1-wire bus low for a minimum of 480 µs. The bus master then releases the bus and goes into receive mode (R_X). When the bus is released, the 5k pullup resistor pulls the 1-wire bus high. When the DS18S20 detects this rising edge, it waits 15–60 µs and then transmits a presence pulse by pulling the 1-wire bus low for 60–240 µs.

INITIALIZATION TIMING Figure 10



READ/WRITE TIME SLOTS

The bus master writes data to the DS18S20 during write time slots and reads data from the DS18S20 during read time slots. One bit of data is transmitted over the 1-wire bus per time slot.

WRITE TIME SLOTS

There are two types of write time slots: “Write 1” time slots and “Write 0” time slots. The bus master uses a Write 1 time slot to write a logic 1 to the DS18S20 and a Write 0 time slot to write a logic 0 to the DS18S20. All write time slots must be a minimum of 60 µs in duration with a minimum of a 1 µs recovery time between individual write slots. Both types of write time slots are initiated by the master pulling the 1-wire bus low (see Figure 11).

To generate a Write 1 time slot, after pulling the 1-wire bus low, the bus master must release the 1-wire bus within 15 µs. When the bus is released, the 5k pullup resistor will pull the bus high. To generate a Write 0 time slot, after pulling the 1-wire bus low, the bus master must continue to hold the bus low for the duration of the time slot (at least 60 µs). The DS18S20 samples the 1-wire bus during a window that lasts from 15 µs to 60 µs after the master initiates the write time slot. If the bus is high during the sampling window, a 1 is written to the DS18S20. If the line is low, a 0 is written to the DS18S20.

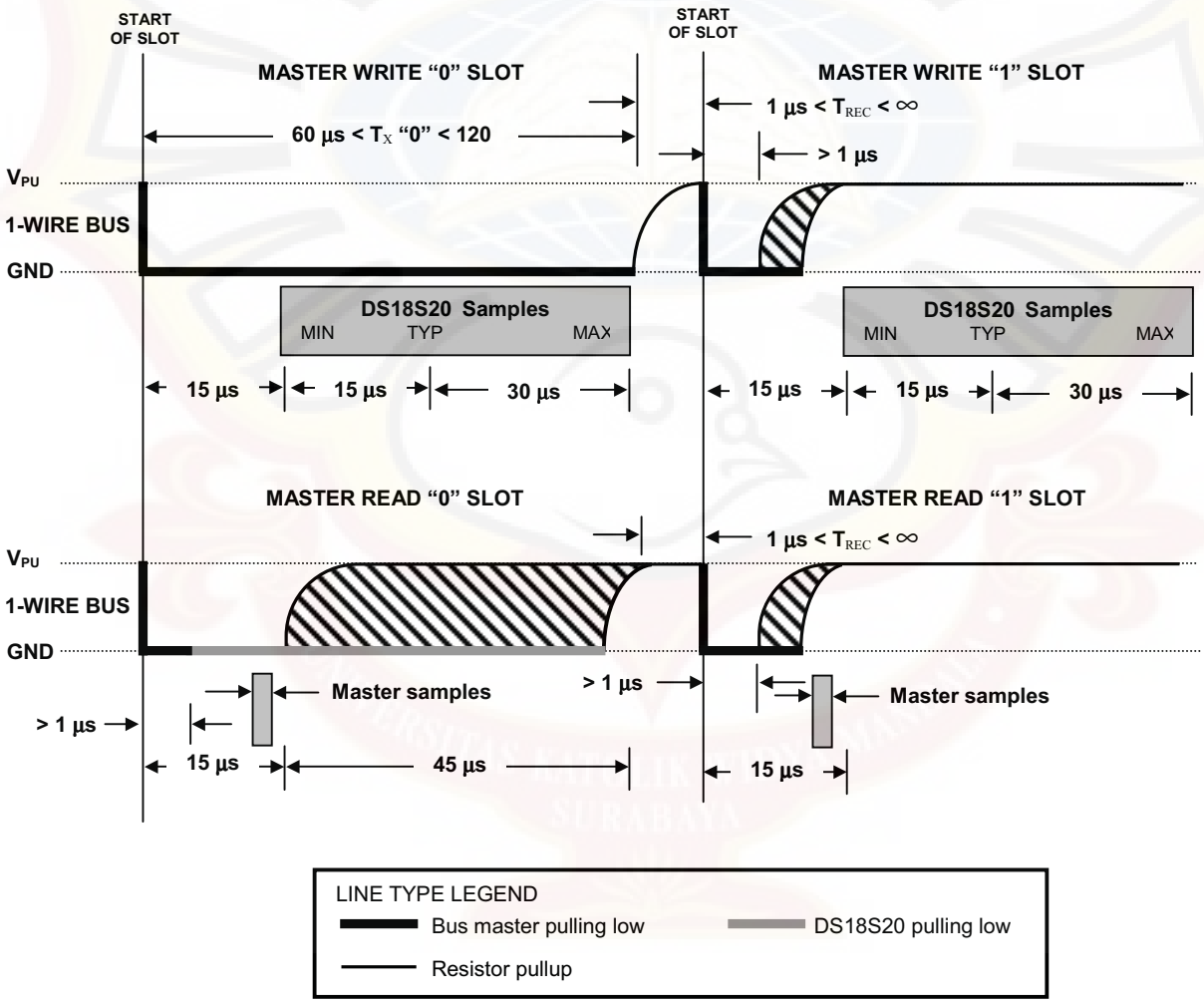
READ TIME SLOTS

The DS18S20 can only transmit data to the master when the master issues read time slots. Therefore, the master must generate read time slots immediately after issuing a Read Scratchpad [BEh] or Read Power Supply [B4h] command, so that the DS18S20 can provide the requested data. In addition, the master can generate read time slots after issuing Convert T [44h] or Recall E^2 [B8h] commands to find out the status of the operation as explained in the DS18S20 FUNCTION COMMAND section.

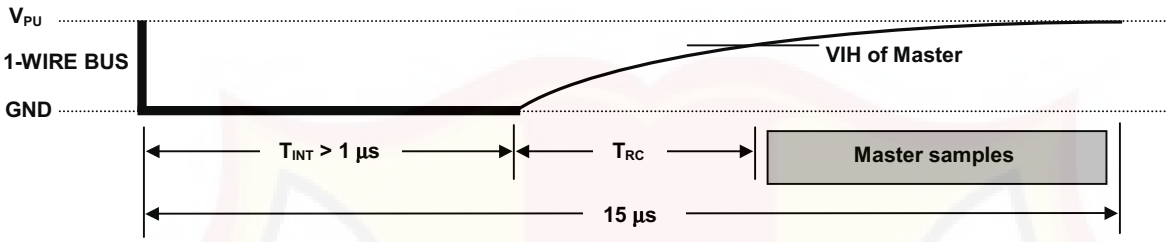
All read time slots must be a minimum of 60 µs in duration with a minimum of a 1 µs recovery time between slots. A read time slot is initiated by the master device pulling the 1-wire bus low for a minimum of 1 µs and then releasing the bus (see Figure 11). After the master initiates the read time slot, the DS18S20 will begin transmitting a 1 or 0 on bus. The DS18S20 transmits a 1 by leaving the bus high and transmits a 0 by pulling the bus low. When transmitting a 0, the DS18S20 will release the bus by the end of the time slot, and the bus will be pulled back to its high idle state by the pullup resister. Output data from the DS18S20 is valid for 15 µs after the falling edge that initiated the read time slot. Therefore, the master must release the bus and then sample the bus state within 15 µs from the start of the slot.

Figure 12 illustrates that the sum of T_{INIT}, T_{RC}, and T_{SAMPLE} must be less than 15 µs for a read time slot. Figure 13 shows that system timing margin is maximized by keeping T_{INIT} and T_{RC} as short as possible and by locating the master sample time during read time slots towards the end of the 15 µs period.

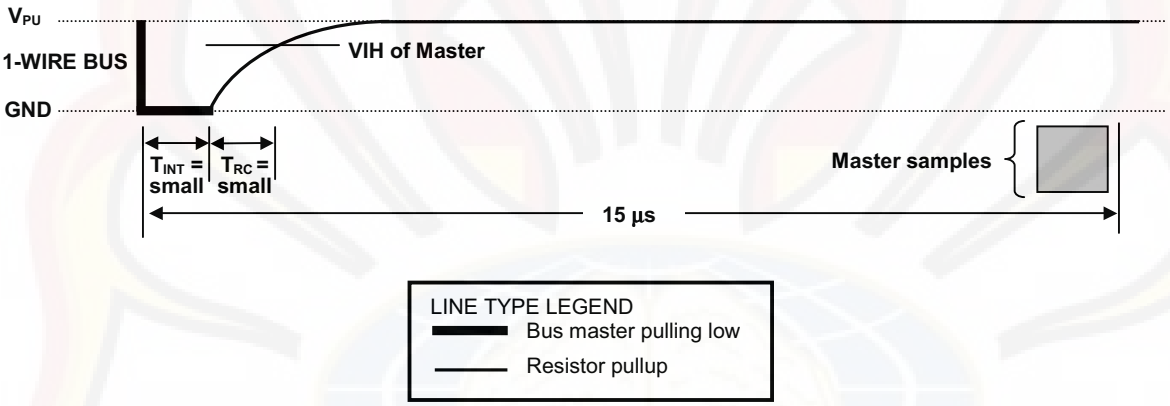
READ/WRITE TIME SLOT TIMING DIAGRAM Figure 11



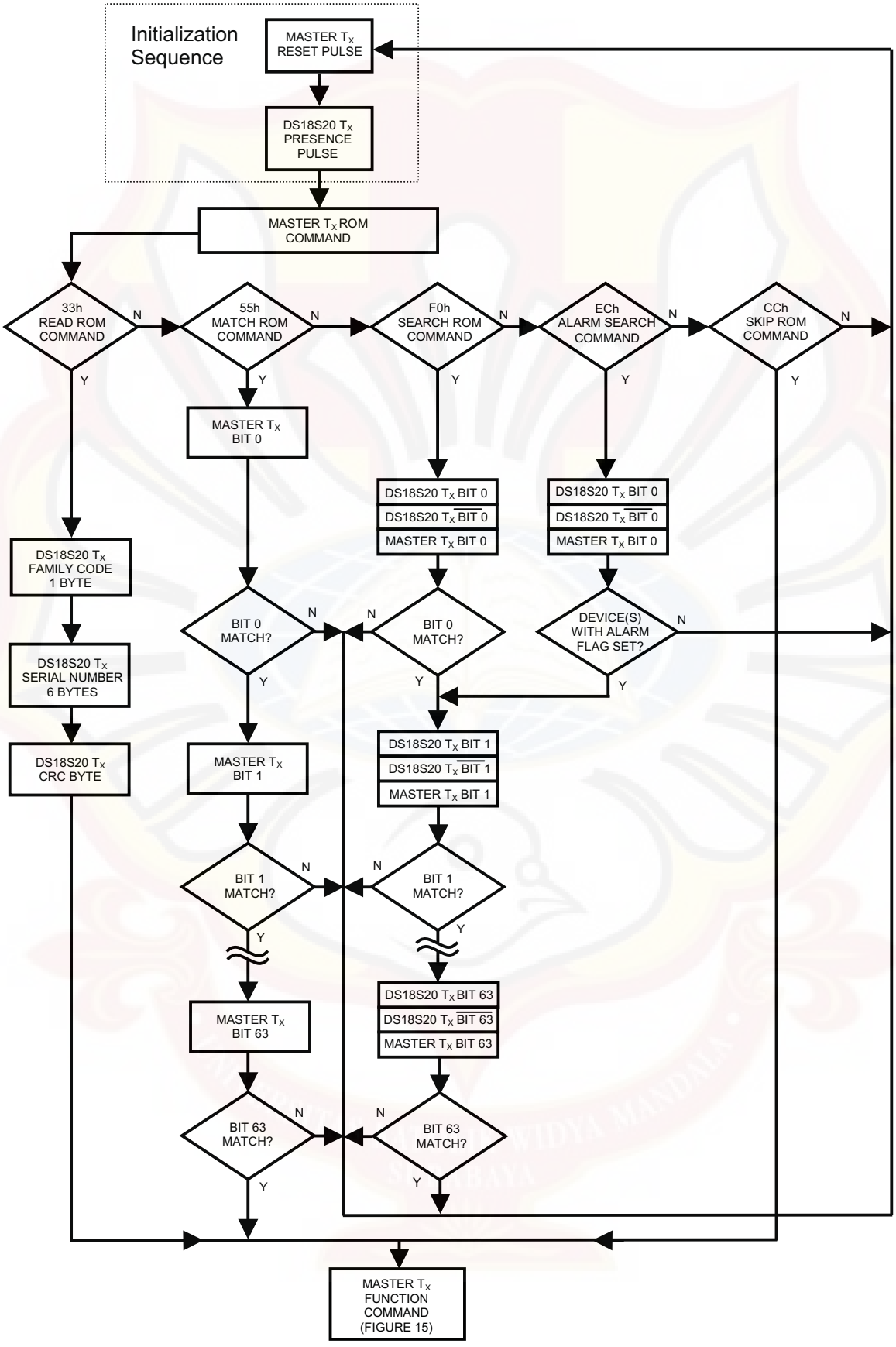
DETAILED MASTER READ 1 TIMING Figure 12



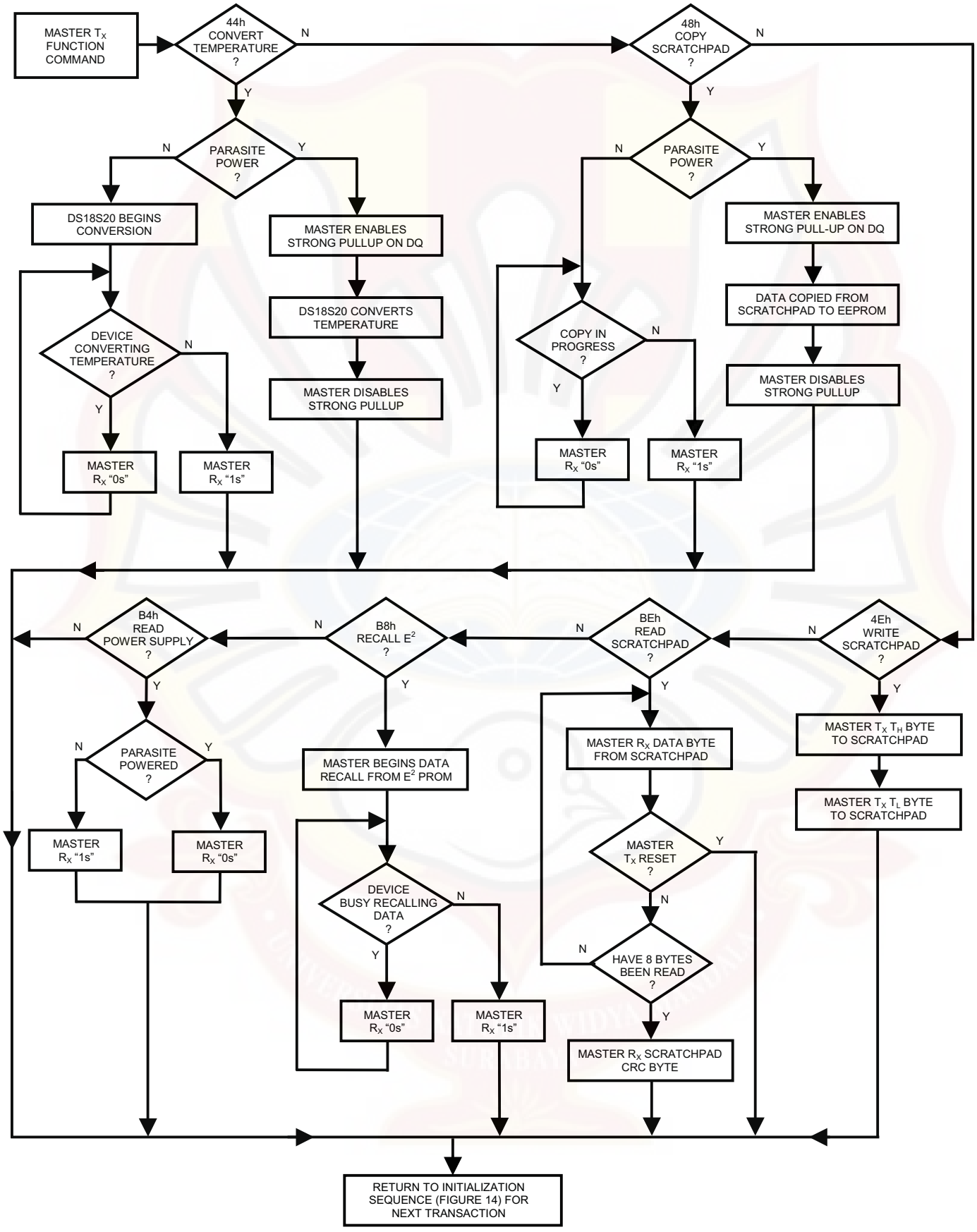
RECOMMENDED MASTER READ 1 TIMING Figure 13



ROM COMMANDS FLOW CHART Figure 14



DS18S20 FUNCTION COMMANDS FLOW CHART Figure 15



DS18S20 OPERATION EXAMPLE 1

In this example there are multiple DS18S20s on the bus and they are using parasite power. The bus master initiates a temperature conversion in a specific DS18S20 and then reads its scratchpad and recalculates the CRC to verify the data.

MASTER MODE	DATA (LSB FIRST)	COMMENTS
TX	Reset	Master issues reset pulse.
RX	Presence	DS18S20s respond with presence pulse.
TX	55h	Master issues Match ROM command.
TX	64-bit ROM code	Master sends DS18S20 ROM code.
TX	44h	Master issues Convert T command.
TX	DQ line held high by strong pullup	Master applies strong pullup to DQ for the duration of the conversion (t_{conv}).
TX	Reset	Master issues reset pulse.
RX	Presence	DS18S20s respond with presence pulse.
TX	55h	Master issues Match ROM command.
TX	64-bit ROM code	Master sends DS18S20 ROM code.
TX	BEh	Master issues Read Scratchpad command.
RX	9 data bytes	Master reads entire scratchpad including CRC. The master then recalculates the CRC of the first eight data bytes from the scratchpad and compares the calculated CRC with the read CRC (byte 9). If they match, the master continues; if not, the read operation is repeated.

DS18S20 OPERATION EXAMPLE 2

In this example there is only one DS18S20 on the bus and it is using parasite power. The master writes to the T_H and T_L registers in the DS18S20 scratchpad and then reads the scratchpad and recalculates the CRC to verify the data. The master then copies the scratchpad contents to EEPROM.

MASTER MODE	DATA (LSB FIRST)	COMMENTS
TX	Reset	Master issues reset pulse.
RX	Presence	DS18S20 responds with presence pulse.
TX	CCh	Master issues Skip ROM command.
TX	4Eh	Master issues Write Scratchpad command.
TX	2 data bytes	Master sends two data bytes to scratchpad (T_H and T_L)
TX	Reset	Master issues reset pulse.
RX	Presence	DS18S20 responds with presence pulse.
TX	CCh	Master issues Skip ROM command.
TX	BEh	Master issues Read Scratchpad command.
RX	9 data bytes	Master reads entire scratchpad including CRC. The master then recalculates the CRC of the first eight data bytes from the scratchpad and compares the calculated CRC with the read CRC (byte 9). If they match, the master continues; if not, the read operation is repeated.
TX	Reset	Master issues reset pulse.
RX	Presence	DS18S20 responds with presence pulse.
TX	CCh	Master issues Skip ROM command.
TX	48h	Master issues Copy Scratchpad command.
TX	DQ line held high by strong pullup	Master applies strong pullup to DQ for at least 10 ms while copy operation is in progress.

DS18S20 OPERATION EXAMPLE 3

In this example there is only one DS18S20 on the bus and it is using parasite power. The bus master initiates a temperature conversion then reads the DS18S20 scratchpad and calculates a higher resolution result using the data from the temperature, COUNT REMAIN and COUNT PER °C registers.

MASTER MODE	DATA (LSB FIRST)	COMMENTS
TX	Reset	Master issues reset pulse.
TR	Presence	DS18S20 responds with presence pulse.
TX	CCh	Master issues Skip ROM command.
TX	44h	Master issues Convert T command.
TX	DQ line held high by strong pullup	Master applies strong pullup to DQ for the duration of the conversion (t_{conv}).
TX	Reset	Master issues reset pulse.
RX	Presence	DS18S20 responds with presence pulse.
TX	CCh	Master issues Skip ROM command.
TX	BEh	Master issues Read Scratchpad command.
RX	9 data bytes	Master reads entire scratchpad including CRC. The master then recalculates the CRC of the first eight data bytes from the scratchpad and compares the calculated CRC with the read CRC (byte 9). If they match, the master continues; if not, the read operation is repeated. The master also calculates the TEMP_READ value and stores the contents of the COUNT REMAIN and COUNT PER °C registers.
TX	Reset	Master issues reset pulse.
RX	Presence	DS18S20 responds with presence pulse.
-	-	CPU calculates extended resolution temperature using the equation in the OPERATION - MEASURING TEMPERATURE section of this datasheet.

RELATED APPLICATION NOTES

The following Application Notes can be applied to the DS18S20. These notes can be obtained from the Dallas Semiconductor "Application Note Book," via the Dallas website at http://www.dalsemi.com/, or through our faxback service at (214) 450–0441.

Application Note 27: "Understanding and Using Cyclic Redundancy Checks with Dallas Semiconductor Touch Memory Product"
Application Note 55: "Extending the Contact Range of Touch Memories"
Application Note 74: "Reading and Writing Touch Memories via Serial Interfaces"
Application Note 104: "Minimalist Temperature Control Demo"
Application Note 105: "High Resolution Temperature Measurement with Dallas Direct-to-Digital Temperature Sensors"
Application Note 106: "Complex MicroLANs"
Application Note 108: "MicroLAN – In the Long Run"

Sample 1-wire subroutines that can be used in conjunction with AN74 can be downloaded from the Dallas website or anonymous FTP Site.

ABSOLUTE MAXIMUM RATINGS*

Voltage on any pin relative to ground	–0.5V to +6.0V
Operating temperature	–55°C to +125°C
Storage temperature	–55°C to +125°C
Soldering temperature	See J-STD-020A Specification

*These are stress ratings only and functional operation of the device at these or any other conditions above those indicated in the operation sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods of time may affect reliability.

DC ELECTRICAL CHARACTERISTICS (-55°C to +125°C; V_{DD}=3.0V to 5.5V)

PARAMETER	SYMBOL	CONDITION	MIN	TYP	MAX	UNITS	NOTES
Supply Voltage	V_{DD}	Local Power	+3.0		+5.5	V	1
Pullup Supply Voltage	V_{PU}	Parasite Power	+3.0		+5.5	V	1,2
		Local Power	+3.0		V_{DD}		
Thermometer Error	t_{ERR}	-10°C to +85°C			±0.5	°C	3
		-55°C to +125°C			±2		
Input Logic Low	V_{IL}		-0.3		+0.8	V	1,4,5
Input Logic High	V_{IH}	Local Power	+2.2		The lower of 5.5 or V_{DD} + 0.3	V	1, 6
		Parasite Power	+3.0				
Sink Current	I_L	$V_{I/O}$=0.4V	4.0			mA	1
Standby Current	I_{DDS}			750	1000	nA	7,8
Active Current	I_{DD}	V_{DD}=5V		1	1.5	mA	9
DQ Input Current	I_{DQ}			5		µA	10
Drift				±0.2		°C	11

NOTES:

1. All voltages are referenced to ground.
2. The Pullup Supply Voltage specification assumes that the pullup device is ideal, and therefore the high level of the pullup is equal to V_{PU}. In order to meet the V_{IH} spec of the DS18S20, the actual supply rail for the strong pullup transistor must include margin for the voltage drop across the transistor when it is turned on; thus: $V_{PU\_ACTUAL} = V_{PU\_IDEAL} + V_{TRANSISTOR}$.
3. See typical performance curve in Figure 16
4. Logic low voltages are specified at a sink current of 4 mA.
5. To guarantee a presence pulse under low voltage parasite power conditions, V_{ILMAX} may have to be reduced to as low as 0.5V.
6. Logic high voltages are specified at a source current of 1 mA.
7. Standby current specified up to 70°C. Standby current typically is 3 µA at 125°C.
8. To minimize I_{DDS}, DQ should be within the following ranges: GND ≤ DQ ≤ GND + 0.3V or V_{DD} – 0.3V ≤ DQ ≤ V_{DD}.
9. Active current refers to supply current during active temperature conversions or EEPROM writes.
10. DQ line is high (“hi-Z” state).
11. Drift data is based on a 1000 hour stress test at 125°C with V_{DD} = 5.5V.

AC ELECTRICAL CHARACTERISTICS: NV MEMORY

(-55°C to +100°C; V_{DD}=3.0V to 5.5V)

PARAMETER	SYMBOL	CONDITION	MIN	TYP	MAX	UNITS
NV Write Cycle Time	t_{wr}			2	10	ms
EEPROM Writes	N_{EEWR}	-55°C to +55°C	50k			writes
EEPROM Data Retention	t_{EEDR}	-55°C to +55°C	10			years

AC ELECTRICAL CHARACTERISTICS

(-55°C to +125°C; V_{DD}=3.0V to 5.5V)

PARAMETER	SYMBOL	CONDITION	MIN	TYP	MAX	UNITS	NOTES
Temperature Conversion Time	t_{CONV}				750	ms	1
Time to Strong Pullup On	t_{SPON}	Start Convert T Command Issued			10	µs	
Time Slot	t_{SLOT}		60		120	µs	1
Recovery Time	t_{REC}		1			µs	1
Write 0 Low Time	r_{LOW0}		60		120	µs	1
Write 1 Low Time	t_{LOW1}		1		15	µs	1
Read Data Valid	t_{RDV}				15	µs	1
Reset Time High	t_{RSTH}		480			µs	1
Reset Time Low	t_{RSTL}		480			µs	1,2
Presence Detect High	t_{PDHIGH}		15		60	µs	1
Presence Detect Low	t_{PDLOW}		60		240	µs	1
Capacitance	$C_{IN/OUT}$				25	pF	

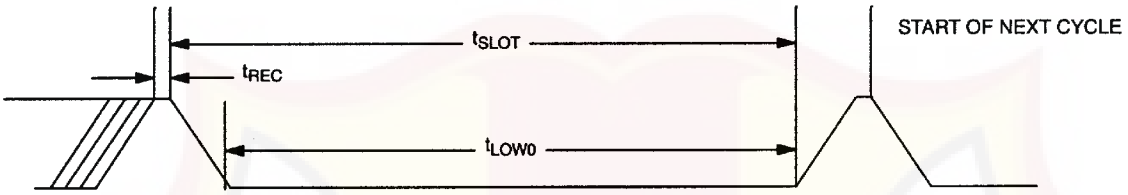
NOTES:

1. Refer to timing diagrams in Figure 17.
2. Under parasite power, if t_{RSTL} > 960 µs, a power on reset may occur.

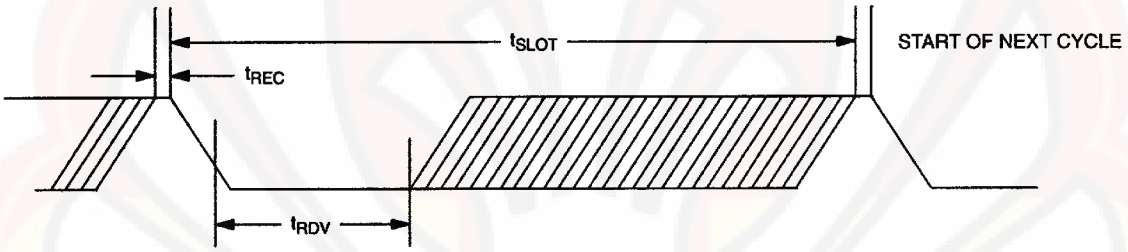
TYPICAL PERFORMANCE CURVE Figure 16

TIMING DIAGRAMS Figure 17

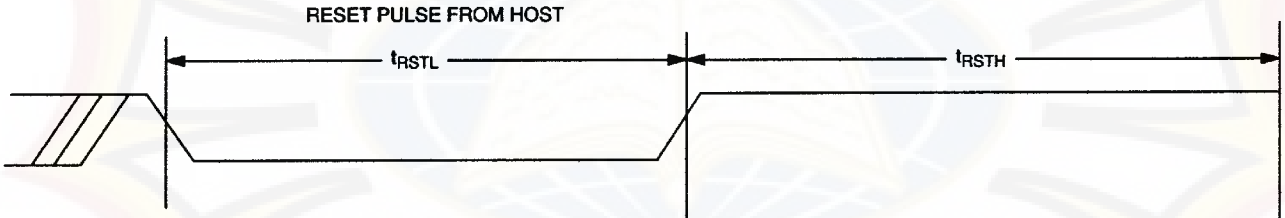
1–WIRE WRITE ZERO TIME SLOT



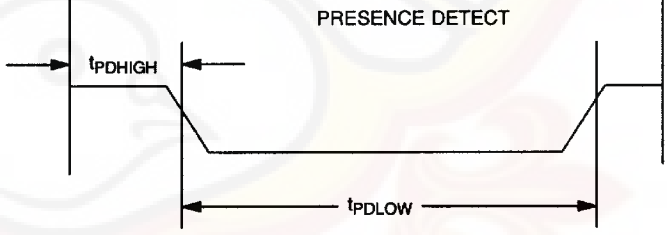
1–WIRE READ ZERO TIME SLOT



1–WIRE RESET PULSE



1–WIRE PRESENCE DETECT



BIODATA

Nama	: Antony Putra Utomo
NRP	: 5103006036
Tempat Lahir	: Pematangsiantar
Tanggal Lahir	: 19 Agustus 1988
Agama	: Kristen
Alamat	: Jl. Pematang SK:5 / no.32 Pematangsiantar

Riwayat Pendidikan

- Tahun 2000, lulus SD Sultan Agung, Pematangsiantar
- Tahun 2003, lulus SLTP Kristen Kalam Kudus, Pematangsiantar
- Tahun 2006, lulus SMA Kristen Kalam Kudus, Pematangsiantar
- Tahun 2006 hingga biodata ini ditulis tercatat sebagai mahasiswa Fakultas Teknik Jurusan Teknik Elektro di Universitas Katolik Widya Mandala Surabaya

*Kontestan Kontes Robot Cerdas Indonesia (KRCI) Regional IV 2009