LUNDS TEKNISKA
HÖGSKOLA
Lunds universitet

DIGITALA PROJEKT

GRAFRITANDE VÄDERSTATION

Erik Ahlmann, I08
Carl Rydholm, I08

Handledare: Bertil Lindvall

Maj 2011

Inledning

Projektet var att skapa en prototyp av en väderstation, kapabel att mäta temperatur och redovisa resultatet i form av en graf på en 128x64-pixel LCD skärm.

Syftet med arbetet var att ge en inblick i svårigheterna och möjligheterna i interaktionen mellan hårdvara och mjukvara. Under projektets gång har man lärt sig baskunskaper inom: datateknik, elektronik, digitalteknik, felsökning samt C-programmering.

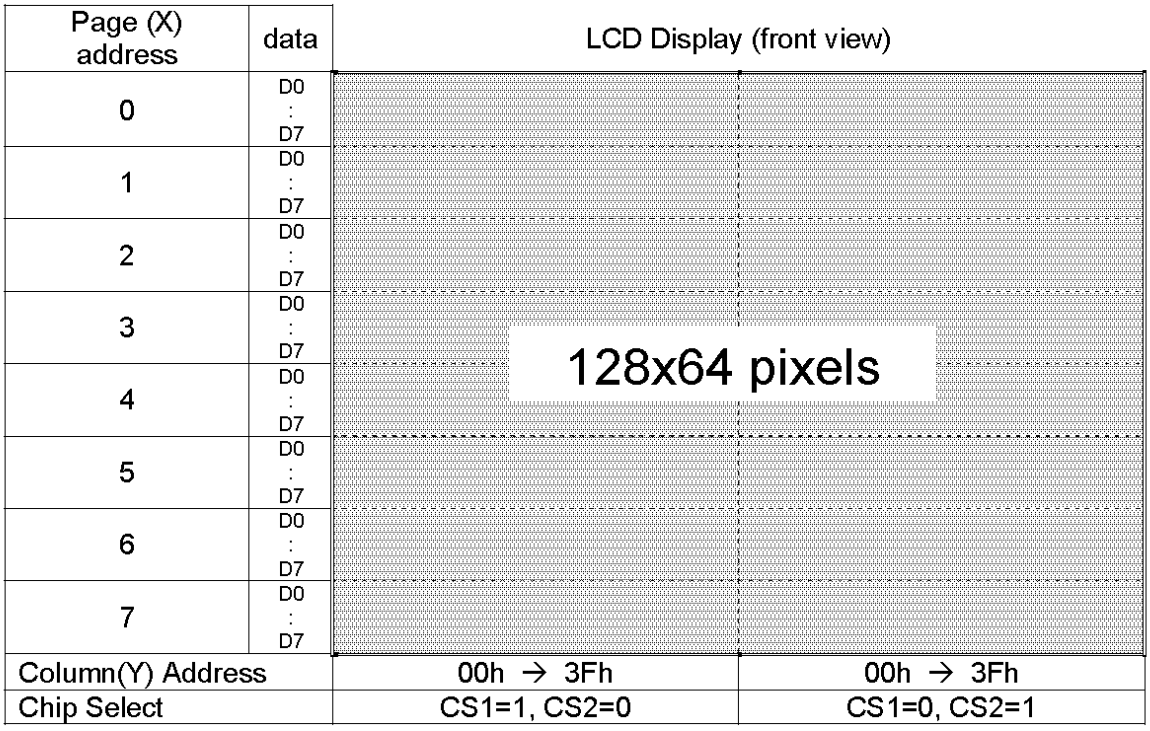
Rapporten innehåller en genomgång av hårdvara, mjukvara, resultat samt problem och lösningar runt dessa områden.

Innehållsförteckning

Hårdvara

Displayen – GDM-12864

Displayen hade 128x 64 pixlar, uppdelad i 2 stycken 64x64-skärmar. För att skriva på den vänstra skärmen valdes pinnen chipselect 1, CS1, och för att skriva på den högra halvan valdes chipselect 2, CS2. Det är omöjligt att skriva till en enskild pixel, istället skrivs det till 8 pixlar i höjdled som placeras i en X-koordinat (0-7) samt en Y-koordinat (0-63). Se figur 1 för pixelplacering.



Figur 1: LCD-skärmens uppdelning av pixlar: Två identiska 64x64-halvor, varje med Y-koordinater från 0 till 63 och X-koordinater från 0 till 7.

Värmesensor- LM335

LM335 är en värmesensor som ger utslag i kelvin. Redovisningen sker i volt, med 10 mV per Kelvin, det vill säga 0 Kelvin motsvarar 0 volt och 2,73 volt motsvarar 273 Kelvin (0 grader Celsius).

Valet av Kelvin-sensor framför en Celsius-sensor gjordes för att lättare representera eventuella temperaturer under 0 grader Celsius.

J-TAG
J-taggen användes för att överföra C-programmet till processorn.

Knappar
2 enklare knappar kopplades in, men implanterades aldrig i koden. Kondensatorer användes för att förhindra eventuell studs vid nedtryckning.

Processor – AVR–ATmega16
Processorn är en 8-bitars ATmega16 med 40 pinnar och 16 kb minne. De 40 pinnarna är uppdelade i 4 portar (A-D) där 4 pinnar var dedikerade för J-taggen.

Displayen kopplades till port B (dataöverföring, 8 bitar) samt port D (kommandoöverföring, 6 bitar) och värmesensorn till port A tillsammans med knapparna. För noggrannare kopplingsschema, se kopplingsschema (se index).

Mjukvara:
Alla kod är skriven i en fil. I detta avsnitt förklarar vi i korta drag hur vi löste de större problemen under kodningsdelen.

AD–omvandlaren
För att överföra den analoga datan från värmesensorn till digital data krävdes att AD-omvandlaren startades. AD-omvandlaren startades genom att sätta 3 register rätt: ADMUX, ADSCRA och SFIOR.

I ADMUX är bit 6-7 vilken strömkälla som ska användas vid konverteringen. I vårt projekt valde vi 5 V från AREF-pinnen. Bit 5, ADLAR, bestämmer hur 10-bitars-resultatet ska presenteras i de två 8-bitars-registerna ADCH och ADCL. De resterande 5 bitarna (5-0) bestämmer vilken pinne som ska läsas ifrån. ADMUX sattes till 0x02, för att läsa från pinne A02 från processorn.

I ADCSRA är framför allt bit 7-5 viktiga; bit 7 måste sätta till 1 för att initiera konverteringen (ADC enable), bit 6 startar AD-omvandligen och bit 5 bestämmer om automatisk omvandling är tillåten. ADCSRA sattes till 0xA0, där bit 6 är en nolla, men ändras vid omvandling genom kommandot ADCSRA |= 0x40.

SFIOR sattes till 0x00 för free running mode.

Värmesensor/omvandling från register till celsius

I metoden get_temp() omvandlas den registrerade datan i registerna ADCL och ADCH till Celsius. ADCL innehöll 8 bitar, medan ADCH innehöll de högsta två. Följaktligen användes formeln:

$$temp = ADCL + ADCH*pow(2, 8)$$

, för sammanslaggning (notera att resterande värden i ADCH är nollor och därför krävs ingen vaskning av bitar).

Resultatet skalades upp 10 gånger för att hantera en decimal. Sedan omvandlades resultatet till Kelvin genom multiplicera med 500 (mättesintervallet var 500 K ty inkopplat 5 V) och dividera med antal bitar i registren, 2^10 = 1024 (ADCL, 8st och ADCH 2st). Det vill säga kvoten (ADCL + ADCH*pow(2, 8))/1024 är en ratio över var i intervallet 0-500 kelvin temperaturen befinner sig.

För att överföra till Celsius subtraheras 2730 (skalat 10 gånger). En kalibrering infördes för att kalibrera mätdata från väderstationen med mer finjusterade temperatursensorer.

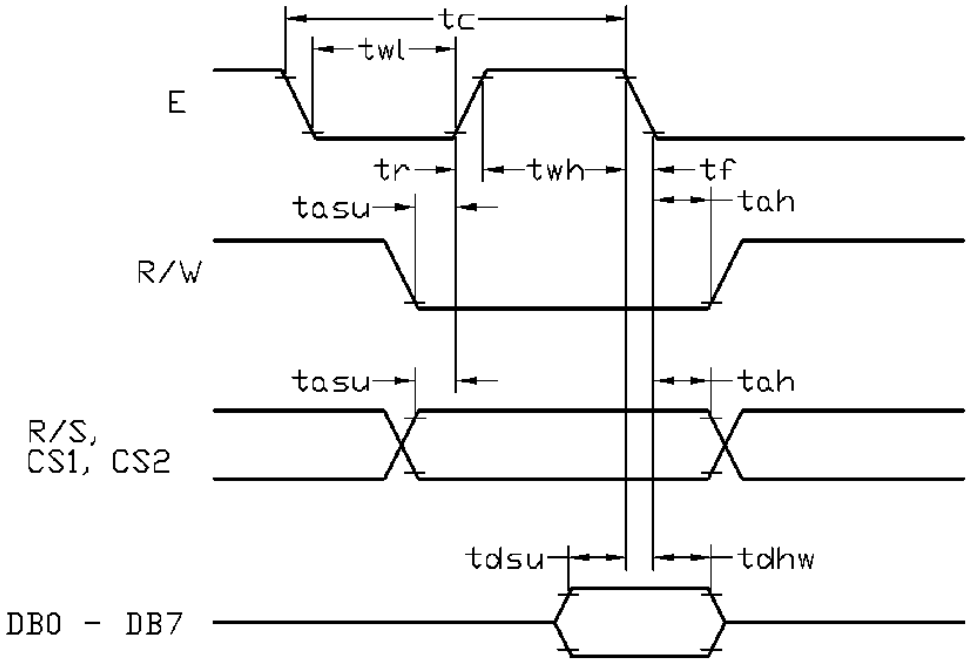
Skärmen

Kontrollen av skärmen bestod av två delar: kommandon och utskrift.

Skärmen: Skicka kommandon

Kommandon används för att sätta på/stänga av skärmen, samt för att specificera var skärmen ska skiva ut de pixlar som skickas in via utskriftsmetoden (under nästa överskrift).

För att skärmen ska läsa ett kommando krävs det att de 6 kommandopinnarna som var kopplade till processorn ändras enligt följande timeingar:



Figur 2: Ordningen som de 6 kommando-bitarna behöver ändras mellan hög och låg för att skärmen ska läsa in ett kommando.

Då datan ska läsas in krävs det att biten R/S är låg.

Om dessa uppfylls så läser skärmen in informationen i de 8 data-bitarna, vilka innehåller information om vilket kommando skärmen ska utföra.

Skärmen: Utskrift av pixel

För att kunna skriva ut en pixel, krävdes det att man först specificerade en ”stående stav”, bestående av 8 pixlar, någonstans på skärmen. Detta gjordes enligt följande två steg:

1. Specificera en Y-koordinat (mellan 0 och 63) i den avsedda halvan (CS1 eller CS2), se Figur 1.
2. Specificera en X-koordinat (mellan 0 och 7) i den avsedda halvan (CS1 eller CS2). Notera att man endast väljer en koordinat i 8 steg, medan skärmen är 64 pixlar i samma dimension. På detta vis väljs endast 8-bitars-”staven”, snarare än en pixel.

Informationen om dessa två steg skickades som tidigare nämnt med kommando-metoden under föregående överskrift.

För att slutligen skriva ut på skärmen, ändras de 6 kommandobitarna åter igen enligt Figur 2. Dock så är biten R/S hög denna gång, för att visa att detta inte är ett kommando. Vilka pixlar som tänds/släcks bestäms av de 8 data-bitarna, var och en kopplad till en pixel i den stående ”staven”.

Algoritm för att rita graf

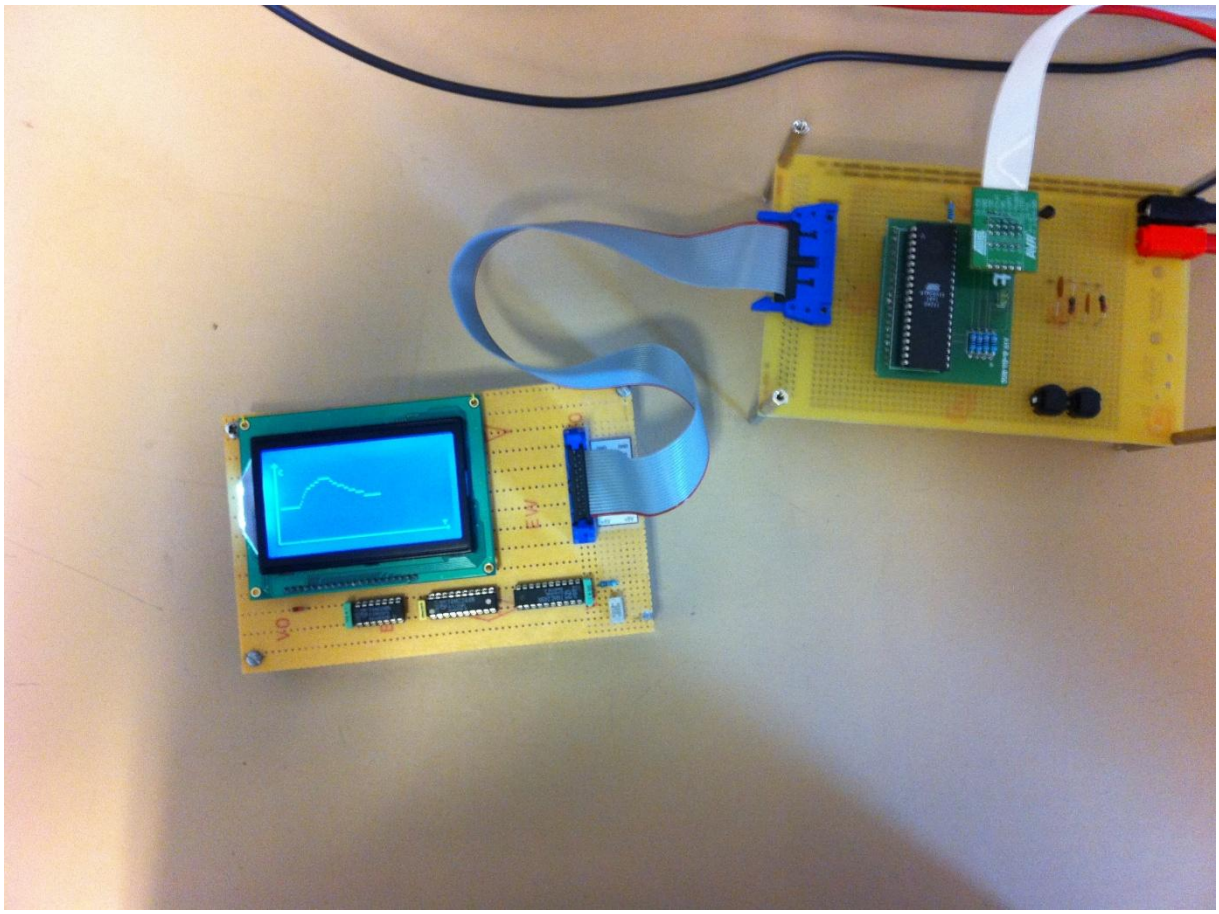
Med hjälp av de övriga metoderna kunde vi läsa in en temperatur, samt skriva ut pixlar på skärmen. För att kunna representera temperaturen som en graf över tiden, gjorde vi följande steg:

1. Vi räknade ut temperaturens relativa position i jämförelse med grafens övre och undre gränser (grafen visade temperaturer från 15 till 30 grader Celsius). Positionen beskrevs i en skala från 0 till 63, för att matcha antalet pixlar som var tillgängliga. För att undvika för grova avrundningar multiplicerades allt med en faktor 100 under beräkningarna.
2. Den relativa positionen delades med 8, vilket angav vilken ”stav” som pixeln befann sig i.
3. Den relativa positionen togs mod(8) (%8), vilket angav vilken pixel i staven som skulle tändas.

En fördröjning användes mellan varje temperaturmätning, som alla skrevs ut på skärmen. En variabel håll ordning på vilken Y-koordinat (den *liggande* dimensionen, se Figur 1) som nästa pixel skulle hamna på. När grafen hade gått över hela skärmen så suddades den ut och började om från början.

Resultat

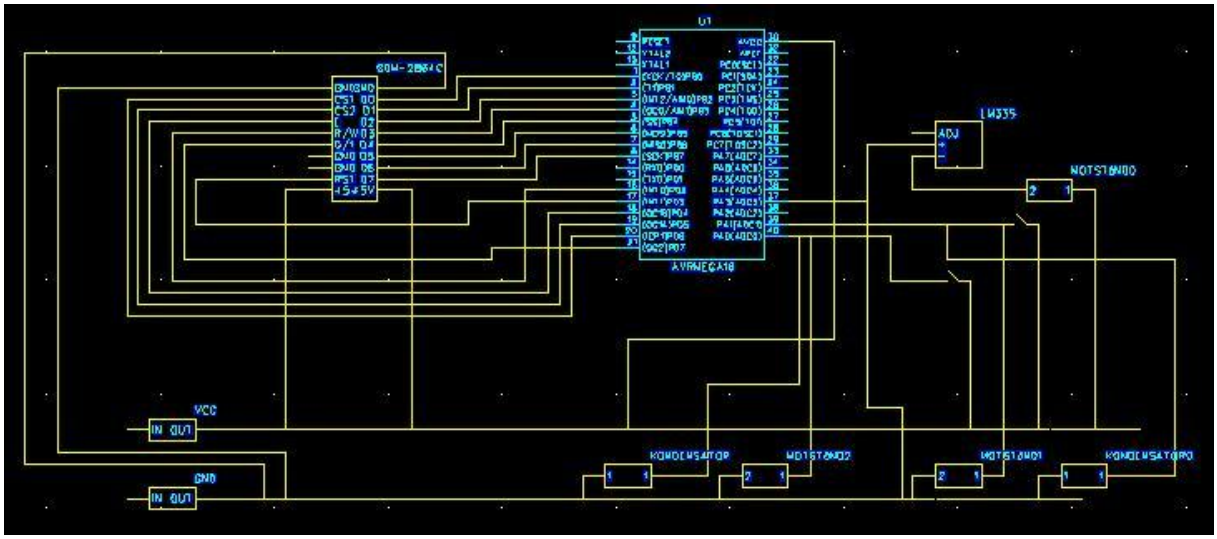
Väderstationen fungerar mycket bra och reagerar på värmeskillnader. Vi känner oss mycket nöjda och har lärt oss mycket. Se figur 3 för utslag av uppvärmning genom beröring under 10 sekunder.



Figur 3: Den färdiga prototypen efter uppvärmning och avsvalning i ca tio sekunder vardera.

Temperaturmätaren är dock väldigt störningskänslig, där kalibreringen kan behöva finjusteras med upp emot 10 Celsius skillnad.

Kopplingschema



Figur 4: Kopplingsschemat för prototypen.

Referenser

Datablad för ATmega16
http://www.eit.lth.se/fileadmin/eit/courses/edi021/datablad/Processors/ATmega16.pdf – senast besökt 2011-05-12

Datablad för GDM 12864
http://www.eit.lth.se/fileadmin/eit/courses/edi021/datablad/Display/GDM12864C.pdf – senast besökt 2011-05-12

Utdelat häfte: ”*Programmera i C*”

Datablad för LM335
http://www.eit.lth.se/fileadmin/eit/courses/edi021/datablad/Sensors/lm335.pdf - senast besökt 2011-05-12

Kod

```
#include <avr/io.h>
#include <util/delay.h>
#include <math.h>


 unsigned short int num1;
 unsigned short int num2;
 unsigned short int high_bound;
```

```
 unsigned short int low_bound;
 unsigned short int counter;


void set_Buttons(void)
{
 DDRA = 0x00;
 return;
}


void init_adc(void)
{
 ADMUX = 0x02;
 ADCSRA = 0xA0;
 SFIOR = 0x00;
 ADCSRA |= 0x40;
 return;
}

void write_cmd(short int CS, short int input)
{
 DDRD = 0xff;
 DDRB = 0xff;
 if (CS == 1) {
  PORTD = 0x3E;
  PORTD &= ~0x10;
  PORTD = 0x42;
  PORTD |= 0x10;
  PORTB = input;
  PORTD &= ~0x10;
  PORTD = 0x2E;
 } else {
  PORTD = 0x5E;
  PORTD &= ~0x10;
  PORTD = 0x22;
  PORTD |= 0x10;
  PORTB = input;
  PORTD &= ~0x10;
  PORTD = 0x4E;
 }
```

```
 return;
}

void write_data(short int CS, short int Xpos, short int Ypos, short int input)
{
 num2 = 0x40;
 num2 |= Ypos;
 write_cmd(CS, num2);
 num1 = 0xB8;
 num1 |= Xpos;
 write_cmd(CS, num1);
 if (CS == 1) {
  PORTD = 0x3E;
  PORTD = 0x2E;
  PORTD = 0x46;
  PORTD = 0x56;
  PORTB = input;
  PORTD = 0x46;
  PORTD = 0x2E;
 } else {
  PORTD = 0x5E;
  PORTD = 0x4E;
  PORTD = 0x26;
  PORTD = 0x36;
  PORTB = input;
  PORTD = 0x26;
  PORTD = 0x4E;
 }
 return;
}

void clear_disp(void) {

 short int i = 0x00;
 short int k = 0x00;
 while (i < 64) {
   write_data(1, 0, i, 0x00);
   write_data(2, 0, i, 0x00);
   write_data(1, 0x01, i, 0x00);
   write_data(2, 0x01, i, 0x00);
   write_data(1, 0x02, i, 0x00);
   write_data(2, 0x02, i, 0x00);
```

```
  write_data(1, 0x03, i, 0x00);
  write_data(2, 0x03, i, 0x00);
  write_data(1, 0x04, i, 0x00);
  write_data(2, 0x04, i, 0x00);
  write_data(1, 0x05, i, 0x00);
  write_data(2, 0x05, i, 0x00);
  write_data(1, 0x06, i, 0x00);
  write_data(2, 0x06, i, 0x00);
  write_data(1, 0x07, i, 0x00);
  write_data(2, 0x07, i, 0x00);
  write_data(1, 0x08, i, 0x00);
  write_data(2, 0x08, i, 0x00);
  i++;
 }
 return;
}


void init_disp(void)
{
 //Horisontell linje
 short int j = 0x03;
 while(j < 64) {
 write_data(1, 0x07, j, 0x20);
 write_data(2, 0x07, (j - 0x03), 0x20);
 j++;
 }
 //Gör en pil bott
 write_data(2, 0x07, 59, 0x70);
 write_data(2, 0x07, 58, 0xF8);

 //Vertikal linje
 write_data(1, 0x00, 0x02, 0xF8);
 write_data(1, 0x01, 0x02, 0xFF);
 write_data(1, 0x02, 0x02, 0xFF);
 write_data(1, 0x03, 0x02, 0xFF);
 write_data(1, 0x04, 0x02, 0xFF);
 write_data(1, 0x05, 0x02, 0xFF);
 write_data(1, 0x06, 0x02, 0xFF);
 write_data(1, 0x07, 0x02, 0x3F);
 //Gör en pil topp
```

```
 write_data(1, 0x00, 0x01, 0x30);
 write_data(1, 0x00, 0x03, 0x30);
 write_data(1, 0x00, 0x00, 0x20);
 write_data(1, 0x00, 0x04, 0x20);
 //Gör ett C
 write_data(1, 1, 6, 0x09);
 write_data(1, 1, 7, 0x09);
 write_data(1, 1, 5, 0x06);
 //Gör ett T
 write_data(2, 7, 56, 0x21);
 write_data(2, 7, 58, 0x21);
 write_data(2, 7, 57, 0x27);
 return;
}

int get_temp(void) {
 short int answer;
 short int kalibrering = 150;
 ADCSRA |= 0x40;
 answer = (ADCL + ADCH*pow(2, 8))*(5000)/(1024) - 2730 + kalibrering;
 return answer;
}

int get_pixel(short int pixel)
{
 if (pixel < 1) {
 return 0x01; }
 else if(pixel < 2){
 return 0x02;}
 else if(pixel < 3){
 return 0x04;}
 else if(pixel < 4){
 return 0x08;}
 else if(pixel < 5){
 return 0x10;}
 else if(pixel < 6){
 return 0x20;}
 else if(pixel < 7){
 return 0x40;}
 else {
 return 0x80;}
}
```

```
void update_temp()
{
 counter++;
 high_bound = 300;
 low_bound = 150;
 short int global = ((high_bound-get_temp())*100)/((high_bound-low_bound)*100/64);
 short int stav = global/8;
 short int pixel = (global)%8;
 _delay_ms(100);
 if(counter<60)
 {
 write_data(1, stav, counter + 4, get_pixel(pixel));
 } else if(counter>=124)
 {
 clear_disp();
 init_disp();
 counter=0;
 } else
 {
 write_data(2, stav, counter+4, get_pixel(pixel));
 }
 return;
}

void main(void)
{

 init_adc();
 set_Buttons();
 write_cmd(1, 0x3F);
 write_cmd(2, 0x3F);
 clear_disp();
 init_disp();
 counter = 0;
   while(1)
 {
 update_temp();
 }
   return;
}
```