Projektrapport

Gadget 2001

- ett övervakningssystem för hem och företag

Ett projekt i kursen Digitala Projekt 2000/2001
av Erik Gredvall och Peter Jansson, e96

Innehållsförteckning

Abstract

We have built an electronic control and alarm system for small business or home use. It was built using the 1-Wire™ technique from Dallas Semiconductor. The microcontroller is the HC11 from Motorola. The system has switches and thermometers, all supervised from the control panel. We used a display with two rows to visualise the status of the system. Each switch can be controlled and read out, and the temperature can also be read out.

When an alarm goes off, the system dials the number to a MiniCall (pager). It then sends an alarm code.

To summarize, it all seems to work well. We have had major problems with fitting the code in to the system and had to prioritise different functions. Although, we are satisfied with the result.

Inledning

Vi valde att bygga ett övervakningssystem för hem eller mindre företag. Detta kom att byggas med Dallas Semiconductor© 1-Wire™ system. Detta innebär att vi på en enda slinga kan lägga in enheter för ID-kontroll, termometrar, larmsensorer, switchar med mera. Syftet med kursen var att lära oss bygga ett mindre fungerande system med microkontroller och perifera enheter.

Specifikation

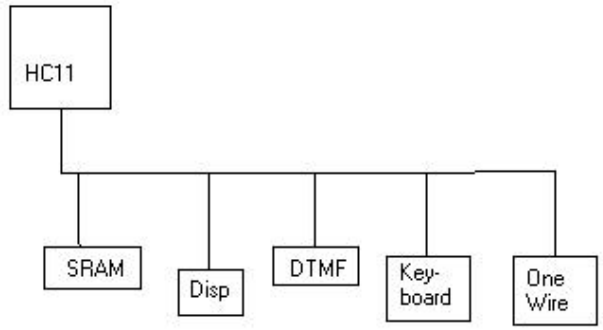
Bygget kommer att baseras på en Motorola MC68HC11ERG processor. Systemets givare kommer att vara enheter från Dallas Semiconductor och arbetar enligt deras 1-Wire™ teknik. Detta innebär att ett antal övervaknings-/kontrollenheter kan kopplas in på endast tre ledningar och att varje enskild enhet kan adresseras.
Enheter att implementera: Två stycken digitala termometrar. Dessa skall kunna ställas in att larma om viss temperatur överskrids. Två stycken switchar att användas till att till exempel starta kaffebryggaren eller tända/släcka lampor. Switcharna skall kunna tidsstyras. Minst två stycken larmslingor för övervakning av till exempel dörrar med brytare eller rum med IR-detektor.
För kommunikationen med enheterna kommer ett litet protokoll att behöva skrivas. Det hela sköts sedan givetvis av HC11:an.
För att kunna kontrollera det hela kommer vi att behöva en knappsats och en display om två eller fyra rader. Här skall tiden kunna ställas in, preferenser för till-/frånslag av switcharna hanteras samt inställning av larmenheterna. Knappsatsen används även för till-/frånslagning av larmet med kod. Displayen visar även tid för ett eventuellt larm.

I mån av tid:
Önskvärt är givetvis att larmet ringer upp en larmoperatör eller en MiniCall och lämnar en sifferkod för vilket larm som utlöst och kanske även tid för larmet.
Att kunna ringa in till sin apparat och styra den via telefon är kul. Därför avser vi implementera detta, och via tonkodning klargöra om vald applikation är till eller från. Det skall gå att slå till/från larmet, till/från kaffebryggaren eller lamporna.
Om det blir ännu mer tid över är det givetvis även roligt att kunna köra det hela från en dator. Därför borde seriell kommunikation med PC implementeras.
Ännu lite roligare är ju givetvis en PalmPilot och dess IR-port. Så om det blir tid, kommer kod att knackas för att lösa även detta…

Teknisk lösning

Hårdvara

Det var alltså Motorolas processor MC68HC11 som fick vårt intresse. Vi byggde med en display, givetvis en processor, ett SRAM, en DTMF-sändare samt enheter från Dallas



Semiconductor att koppla in på vår 1-Wire™-bus.
Skiss över projektet.

Kopplingarna gjordes med virning (kopplingsschema, se bilaga 1). Vissa problem med DTMF och SRAM visade sig bero på att alla ben inte var virade utan ”hängde i luften”. Detta lärde oss att noga kontroller varje ben och inte utgå från att ’det behöver nog inte viras’. Stora problem med att få bort kopplingston vid sändning av DTMF-toner på telelinjen uppkom också. I vårt projekt nyttjas nämligen endast sändning, ej mottagning, varvid vi inte kopplade som tillverkarens schema angav. Efter mycket huvudbry och empiriska tester kom vi till slut fram till en lösning som fungerade. Mer om detta senare.

Vi valde att inrikta oss på Dallas Semiconductors 1-Wire™-system för switchar, temperatur- och larmsensorer i vår övervakningsenhet. Fördelen med 1-wire är att allt styrs via en bus och övervakningssensorer och switchar kan således godtyckligt placeras ut efter två ledningstrådar. Möjligheten till en extra tråd för strömförsörjning till temperatursensorarna vid avläsning av temperaturen utnyttjas inte. Detta medför att så fort vi skall läsa av temperaturen måste busen sättas hög under en halv sekund för att möjliggöra konverteringen. Detta medför inga direkta problem då vi inte läser av temperaturen då larmet är aktivt och pollninng mot switcharna sker.

För att kunna avgöra vilken av 1-wire-enheterna man pratar med har var och en av en enhetsspecifik 64 bitars ROM-kod. Genom att skicka ut rätt kod till enheten så blir den aktiv och de enheter som inte stämmer överens med koden deaktiveras. De enheter som är inaktiva lyssnar inte till någon information som sänds på busen förrän de nollställs med en resetpuls. 1-Wire förutsätter att bussen är en logisk etta (5 V) i vila detta åstadkommes med hjälp av ett pullupmotstånd. Motståndet medger att bussen kan kopplas direkt till jord utan att strömmen igenom densamma blir hög.

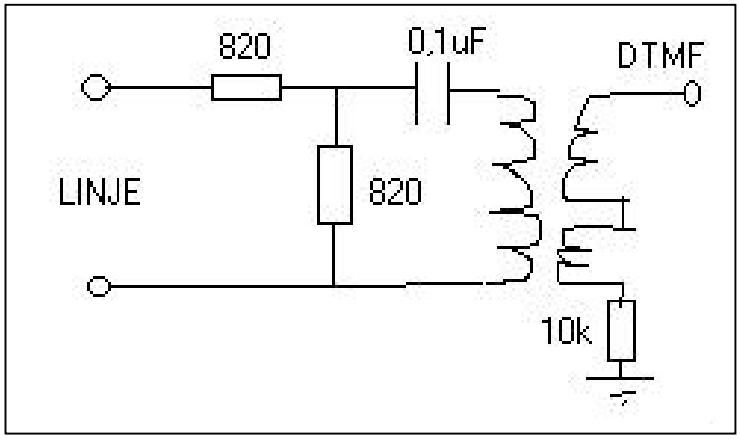
Kommunikationen är sedan relativt enkel. En kort neddragning från 5V till 0V uppfattas av enheterna som en logisk etta och en lång som en logisk nolla. Tiden mellan varje sändning är inte tidskritisk, så man kan i princip sända en bit i dag och nästa i morgon. En tillräckligt lång neddragning til 0V registreras som en resetpuls och det är därför viktigt att ha koll på tidskraven för en logisk: nolla, etta och resetpuls. Resetpulsen har ingen övre tidsbegränsning vilket gör att enheterna nollställs om de blir bortkopplade.

För att sedan läsa från 1-Wire™ enheterna läser man av busen direkt efter man har skickat ut en etta. Detta enkla protokoll ger en robust lösning på kommunikationen och medger också långa ledningstrådar vilket kan vara en fördel i vårt fall.

För att kunna hantera all data som behöver sparas (temperaturer, larmminne mm) implementerade vi givetvis ett SRAM, från Xicor, ett X25128 (16x8bitar). Kommunikationen med minnet var relativt enkel, allt gick enligt protokollet SPI (Serial Peripheral Interface). Kodandet var således relativt smärtfri. Dock var det vissa problem med att CHIP SELECT och HOLD, vilka är aktivt låga, hade virats till etta. Mycket felsökning i koden då det var hårdvaran som felade. Efter att ha lödat om ledningarna rätt, fungerade även denna komponent.

Tonsändaren i projektet är en MT8880C från Mitel. Kommunikationen med DTMF-sändaren löste sig efter noggrant studrande av databladet. Relativt många parametrar att hålla reda på och saker som måste ske i rätt ordning. Inga större problem efter att ha skrivit lite pseudokod.

Dock så är det mycket initieringar som måste göras efter Power Up. Denna programsnutt behöver dock bara köras en gång. Det var även stora problem att få telelinjen att uppfatta tonerna från sändaren. Kopplingstonen kvarstod efter signalering. Efter ett par dagars empiriska tester kom vi dock fram till en koppling som fungerade.



Skiss över koppling mellan DTMF och telelinjen.

Även med DTMF-sändaren trodde vi först att det var fel i koden, men det var ännu en gång hårdvaran som felade. Tillverkarens datablad stämde inte helt med verkligheten om man som vi valde att inte implementera även mottagning av toner.

Det finns även en knappsats i vårt projekt. För att enkelt kunna hantera denna, implementerade vi även en avkodare för 16 tangenters knappsatser. Det blev en MM74C922 från National Semiconductor. Denna lilla tingest fungerar som så att den känner av vilken knapp som tryckts ner i matrisen (4x4). Den tar bort eventuella kontaktstudsar och skickar iväg binärkoden för nedtryckt tangent. Vänstra övre hörnet kodas 0_{10} och nedre högra 15_{10}. Det enda problemet som uppstod med denna komponent var en felvirning från knappsatsen. Två kolumner hade helt enkelt bytt plats, vilket gav oss lite märkliga binärkoder från avkodaren. Det tog dock inte många minuter att ordna till.

Mjukvara

Se bilaga 2, programkod.

Resultat

Det är roligt att se ett fungerande system som är byggt helt från grunden av oss själva. Det vi bär med oss efter kursen är djupare kunskaper om microprocessor, minneshantering, och andra perifera enheter fungerar tillsammans.

Sammanfattning

Kursen är mycket rolig och lärorik. Det är en av de första kurser där man får bygga något helt från grunden. Vi valde att bygga projektet baserat på Motorolas mikroprocessor HC11.

Det har varit stora problem med att få plats med all programkod i vår processor. Mycket av det som vi faktiskt har haft fungerande har vi varit tvungna att kommentera bort för att överhuvudtaget få ett fungerande system.

```
#include "IO-reg.h"

#include "Vectors.h"

#include "intr6811.h"

#include "disp.h"

#include "wire.h"

#include "meny.h"

#include "avbr_clk.h"

#include "dtmf.h"

//#include "minne.c"


char c_temp_1[3], c_temp_2[3], c_temp_3[3], c_temp_4[3];

char c_inne_temp[3], c_ute_temp[3], i, data, temp3, temp4, temp5,
temp6, temp7, temp8, temp9;

char serial_byte_1, serial_byte_2, serial_byte_3, serial_byte_4,
serial_byte_5, serial_byte_6, serial_byte_CRC, fam_code;

char data1, data2, cnt_down_string[3] ;

void keyb_init(void);

void enable_interrupt(void);

int lasse, minnes_adr, cnt_down, inne_temp, ute_temp;


main(void)

{

  keyb_init();

  init_clock();

  swreset_dtmf();

  init_dtmf();
```

```
enable_interrupt();
disp_rad = 1;
kod_count = 0;
armed = 0;
meny_val_1 = 0;
meny_val_2 = 0;
meny_val_3 = 0;
meny_djup = 1;
meny_val_1_max = 3;
disp_init();
disp_clear();
meny_djup = 1;
disp_rad = 1;
meny_rad_max = 6;
meny_djup_max = 6;*/
//Alla charbärande givarvariabler nollställs.
c_inne_temp[0] = ' ';
c_inne_temp[1] = ' ';
c_inne_temp[2] = ' ';
c_ute_temp[0] = ' ';
c_ute_temp[1] = ' ';
c_ute_temp[2] = ' ';
c_temp_1[0] = ' ';
c_temp_1[1] = ' ';
c_temp_1[2] = ' ';
c_temp_2[0] = ' ';
```

```
  c_temp_2[1] = ' ';
  c_temp_2[2] = ' ';
  c_temp_3[0] = ' ';
  c_temp_3[1] = ' ';
  c_temp_3[2] = ' ';
  c_temp_4[0] = ' ';
  c_temp_4[1] = ' ';
  c_temp_4[2] = ' ';
//Menysystemet
//Viloläge
  while (1==1)
  {
    while ((armed_delay==1) & (armed == 0))
    {
      disp_w_data_r(101, " Larm aktiveras ");
      disp_w_data_r(201, " om    sekunder ");
      cnt_down = 30-trenoll_cnt;
      cnt_down_string[0] = temp_to_char2(cnt_down);
      cnt_down_string[1] = temp_to_char3(cnt_down);
      cnt_down_string[2] = ' ';
      disp_w_data_r(405, cnt_down_string);
      /*//Vänte tillstånd
      armed =1;
      disp_clear();
      disp_w_data_r(101, "*Larm aktiverat*");*/
    }
```

```
    while ((alarm_delay==1) & (armed == 1) & (alarm == 0))
    {
      disp_w_data_r(101, " Slå kod, larm  ");
      disp_w_data_r(201, " om    sekunder!");
      cnt_down = 30-trenoll_cnt;
      cnt_down_string[0] = temp_to_char2(cnt_down);
      cnt_down_string[1] = temp_to_char3(cnt_down);
      cnt_down_string[2] = ' ';
      disp_w_data_r(405, cnt_down_string);
    }

    while ((alarm_delay == 0) & (armed == 1) & (alarm == 0))
    {
      disp_clear();
      disp_w_data_r(101, "*Larm aktiverat*");
      if
(wire_switch_read(0x05,0x98,0x63,0x00,0x00,0x00,0x00,0x86) == 0)
        alarm_delay = 1;
      else
        alarm_delay = 0;
      if
(wire_switch_read(0x05,0x45,0x6D,0x00,0x00,0x00,0x00,0xAA) == 0)
      {
        alarm = 1;
      }
    }
```

```
    while (alarm == 1)
    {
      disp_w_data_r(101, "     Larm!      ");
      disp_w_data_r(201, " Ringer minicall");
      if ((alarm_sent == 0) & (alarm == 1)) //Ringer upp MiniCall
      {
        init_send_dtmf();
        send_digit_string("0740114536","01234#"); // # ÄR FÖR ATT
AVSLUTA SÄNDNING TILL MINICALL
        alarm_sent = 1;
      }
    }
    if ((meny_val_1 == 0) & (meny_djup == 1) & (armed == 0))
    {
      //disp_clear();
      armed_delay = 0;
      alarm_delay = 0;
      disp_w_data_r(101, "     >MENY<     ");
      disp_w_data_r(201, "Ute   º Inne   º");
      disp_w_data_r(404,c_ute_temp);
      disp_w_data_r(413,c_inne_temp);
      inne_temp = wire_temp_read(0x10, 0xF1, 0xBD, 0x07, 0x00,
0x00, 0x00, 0x9B);
      ute_temp = wire_temp_read(0x10, 0x95, 0x57, 0x13, 0x00,
0x00, 0x00, 0xD5);
      c_inne_temp[0] = temp_to_char1(inne_temp);
      c_inne_temp[1] = temp_to_char2(inne_temp);
      c_inne_temp[2] = temp_to_char3(inne_temp);
```

```
  c_ute_temp[0] = temp_to_char1(ute_temp);
  c_ute_temp[1] = temp_to_char2(ute_temp);
  c_ute_temp[2] = temp_to_char3(ute_temp);
}
//Grundmeny
if (meny_djup == 2)
{
  if ((meny_val_1 == 1) & (disp_rad == 1))
  {
    disp_w_data_r(101, ">Ärvärden<      ");
    disp_w_data_r(201, " Börvärden      ");
  }
  if ((meny_val_1 == 2) & (disp_rad == 2))
  {
    disp_w_data_r(101, " Ärvärden       ");
    disp_w_data_r(201, ">Börvärden<     ");
  }
  if ((meny_val_1 == 2) & (disp_rad == 1))
  {
    disp_w_data_r(101, ">Börvärden<     ");
    disp_w_data_r(201, " Tid            ");
  }
  if ((meny_val_1 == 3) & (disp_rad == 2))
  {
    disp_w_data_r(101, " Börvärden      ");
    disp_w_data_r(201, ">Tid<           ");
```

```
      }
    }
    //Menyval Ärvärden och Menyval Börvärden
    if (((meny_val_1 == 1) | (meny_val_1 == 2))  & (meny_djup ==
3))
    {
      if ((meny_val_2 == 1) & (disp_rad == 1))
      {
        disp_w_data_r(101, ">Temp<          ");
        disp_w_data_r(201, " Sensorer       ");
      }
      if ((meny_val_2 == 2) & (disp_rad == 2))
      {
        disp_w_data_r(101, " Temp           ");
        disp_w_data_r(201, ">Sensorer<      ");
      }
      if ((meny_val_2 == 2) & (disp_rad == 1))
      {
        disp_w_data_r(101, ">Sensorer<      ");
        disp_w_data_r(201, " Brytare        ");
      }
      if ((meny_val_2 == 3) & (disp_rad == 2))
      {
        disp_w_data_r(101, " Sensorer       ");
        disp_w_data_r(201, ">Brytare<       ");
      }
      if ((meny_val_2 == 3) & (disp_rad == 1))
```

```
      {
        disp_w_data_r(101, ">Brytare<       ");
        disp_w_data_r(201, " Larmnummer     ");
      }
      if ((meny_val_2 == 4) & (disp_rad == 2))
      {
        disp_w_data_r(101, " Brytare        ");
        disp_w_data_r(201, ">Larmnummer<    ");
      }
      {
        disp_w_data_r(101, ">Larmnummer<    ");
        disp_w_data_r(201, " Tid            ");
      }
      if ((meny_val_2 == 5) & (disp_rad == 2))
      {
        disp_w_data_r(101, " Larmnummer     ");
        disp_w_data_r(201, ">Tid<           ");
      }
      if ((meny_val_2 == 5) & (disp_rad == 1))
      {
        disp_w_data_r(101, ">Tid<           ");
        disp_w_data_r(201, " Kod            ");
      }
      if ((meny_val_2 == 6) & ((disp_rad == 2) | (disp_rad ==
1)))
      {
        disp_rad == 2;
```

```
        disp_w_data_r(101, " Tid            ");
        disp_w_data_r(201, ">Kod<           ");
      }*/
    }
    //Menyval Ärvärden-Temp
    if ((meny_val_1 == 1) & (meny_val_2 == 1) & (meny_djup == 4))
    {
      if ((meny_val_3 == 1) & (disp_rad == 1))
      {
        disp_w_data_r(101, ">Ute          °");
        disp_w_data_r(201, " Inne         °");
        disp_w_data_r(313, c_ute_temp);
        disp_w_data_r(413, c_inne_temp);
        inne_temp = wire_temp_read(0x10, 0xF1, 0xBD, 0x07, 0x00,
0x00, 0x00, 0x9B);
        ute_temp = wire_temp_read(0x10, 0x95, 0x57, 0x13, 0x00,
0x00, 0x00, 0xD5);
        c_inne_temp[0] = temp_to_char1(inne_temp);
        c_inne_temp[1] = temp_to_char2(inne_temp);
        c_inne_temp[2] = temp_to_char3(inne_temp);
        c_ute_temp[0] = temp_to_char1(ute_temp);
        c_ute_temp[1] = temp_to_char2(ute_temp);
        c_ute_temp[2] = temp_to_char3(ute_temp);
      }
      if ((meny_val_3 == 2) & (disp_rad == 2))
      {
        disp_w_data_r(101, " Ute          °");
```

```
        disp_w_data_r(201, ">Inne          °");

        disp_w_data_r(313, c_ute_temp);

        disp_w_data_r(413, c_inne_temp);

        inne_temp = wire_temp_read(0x10, 0xF1, 0xBD, 0x07, 0x00,
0x00, 0x00, 0x9B);

        ute_temp = wire_temp_read(0x10, 0x95, 0x57, 0x13, 0x00,
0x00, 0x00, 0xD5);

        c_inne_temp[0] = temp_to_char1(inne_temp);

        c_inne_temp[1] = temp_to_char2(inne_temp);

        c_inne_temp[2] = temp_to_char3(inne_temp);

        c_ute_temp[0] = temp_to_char1(ute_temp);

        c_ute_temp[1] = temp_to_char2(ute_temp);

        c_ute_temp[2] = temp_to_char3(ute_temp);

      }

      if ((meny_val_3 == 2) & (disp_rad == 1))

      {

        disp_w_data_r(101, ">Inne          °");

        disp_w_data_r(201, " Temp1       °");

        disp_w_data_r(313, c_inne_temp);

        disp_w_data_r(413, c_ute_temp);

        inne_temp = wire_temp_read(0x10, 0xF1, 0xBD, 0x07, 0x00,
0x00, 0x00, 0x9B);

        ute_temp = wire_temp_read(0x10, 0x95, 0x57, 0x13, 0x00,
0x00, 0x00, 0xD5);

        c_inne_temp[0] = temp_to_char1(inne_temp);

        c_inne_temp[1] = temp_to_char2(inne_temp);

        c_inne_temp[2] = temp_to_char3(inne_temp);

        c_ute_temp[0] = temp_to_char1(ute_temp);
```

```
        c_ute_temp[1] = temp_to_char2(ute_temp);
        c_ute_temp[2] = temp_to_char3(ute_temp);
      }
      if ((meny_val_3 == 3) & (disp_rad == 2))
      {
        disp_w_data_r(101, " Inne          °");
        disp_w_data_r(201, ">Temp1         °");
        disp_w_data_r(313, c_inne_temp);
        disp_w_data_r(413, c_ute_temp);
        inne_temp = wire_temp_read(0x10, 0xF1, 0xBD, 0x07, 0x00,
0x00, 0x00, 0x9B);
        ute_temp = wire_temp_read(0x10, 0x95, 0x57, 0x13, 0x00,
0x00, 0x00, 0xD5);
        c_inne_temp[0] = temp_to_char1(inne_temp);
        c_inne_temp[1] = temp_to_char2(inne_temp);
        c_inne_temp[2] = temp_to_char3(inne_temp);
        c_ute_temp[0] = temp_to_char1(ute_temp);
        c_ute_temp[1] = temp_to_char2(ute_temp);
        c_ute_temp[2] = temp_to_char3(ute_temp);
      }

      if ((meny_val_3 == 3) & (disp_rad == 1))
      {
        disp_w_data_r(101, ">Temp1         °");
        disp_w_data_r(201, " Temp2         °");
        disp_w_data_r(313, c_ute_temp);
        disp_w_data_r(413, c_inne_temp);
```

```
        inne_temp = wire_temp_read(0x10, 0xF1, 0xBD, 0x07, 0x00,
0x00, 0x00, 0x9B);

        ute_temp = wire_temp_read(0x10, 0x95, 0x57, 0x13, 0x00,
0x00, 0x00, 0xD5);

        c_inne_temp[0] = temp_to_char1(inne_temp);

        c_inne_temp[1] = temp_to_char2(inne_temp);

        c_inne_temp[2] = temp_to_char3(inne_temp);

        c_ute_temp[0] = temp_to_char1(ute_temp);

        c_ute_temp[1] = temp_to_char2(ute_temp);

        c_ute_temp[2] = temp_to_char3(ute_temp);

      }

      if ((meny_val_3 == 4) & (disp_rad == 2))

      {

        disp_w_data_r(101, " Temp1         °");

        disp_w_data_r(201, ">Temp2         °");

        disp_w_data_r(313, c_ute_temp);

        disp_w_data_r(413, c_inne_temp);

        inne_temp = wire_temp_read(0x10, 0xF1, 0xBD, 0x07, 0x00,
0x00, 0x00, 0x9B);

        ute_temp = wire_temp_read(0x10, 0x95, 0x57, 0x13, 0x00,
0x00, 0x00, 0xD5);

        c_inne_temp[0] = temp_to_char1(inne_temp);

        c_inne_temp[1] = temp_to_char2(inne_temp);

        c_inne_temp[2] = temp_to_char3(inne_temp);

        c_ute_temp[0] = temp_to_char1(ute_temp);

        c_ute_temp[1] = temp_to_char2(ute_temp);

        c_ute_temp[2] = temp_to_char3(ute_temp);

      }
```

```
      /*if ((meny_val_3 == 4) & (disp_rad == 1))

      {

        disp_w_data_r(101, ">Temp2         °");

        disp_w_data_r(201, " Temp3         °");

        disp_w_data_r(313, c_inne_temp);

        disp_w_data_r(413, c_ute_temp);

        inne_temp = wire_temp_read(0x10, 0xF1, 0xBD, 0x07, 0x00,
0x00, 0x00, 0x9B);

        ute_temp = wire_temp_read(0x10, 0x95, 0x57, 0x13, 0x00,
0x00, 0x00, 0xD5);

        c_inne_temp[0] = temp_to_char1(inne_temp);

        c_inne_temp[1] = temp_to_char2(inne_temp);

        c_inne_temp[2] = temp_to_char3(inne_temp);

        c_ute_temp[0] = temp_to_char1(ute_temp);

        c_ute_temp[1] = temp_to_char2(ute_temp);

        c_ute_temp[2] = temp_to_char3(ute_temp);

      }

      if ((meny_val_3 == 5) & (disp_rad == 2))

      {

        disp_w_data_r(101, " Temp2         °");

        disp_w_data_r(201, ">Temp3         °");

        disp_w_data_r(313, c_inne_temp);

        disp_w_data_r(413, c_ute_temp);

        inne_temp = wire_temp_read(0x10, 0xF1, 0xBD, 0x07, 0x00,
0x00, 0x00, 0x9B);

        ute_temp = wire_temp_read(0x10, 0x95, 0x57, 0x13, 0x00,
0x00, 0x00, 0xD5);

        c_inne_temp[0] = temp_to_char1(inne_temp);
```

```
        c_inne_temp[1] = temp_to_char2(inne_temp);
        c_inne_temp[2] = temp_to_char3(inne_temp);
        c_ute_temp[0] = temp_to_char1(ute_temp);
        c_ute_temp[1] = temp_to_char2(ute_temp);
        c_ute_temp[2] = temp_to_char3(ute_temp);
      }
      if ((meny_val_3 == 5) & (disp_rad == 1))
      {
        disp_w_data_r(101, ">Temp3         °");
        disp_w_data_r(201, " Temp4         °");
        disp_w_data_r(313, c_ute_temp);
        disp_w_data_r(413, c_inne_temp);
        inne_temp = wire_temp_read(0x10, 0xF1, 0xBD, 0x07, 0x00,
0x00, 0x00, 0x9B);
        ute_temp = wire_temp_read(0x10, 0x95, 0x57, 0x13, 0x00,
0x00, 0x00, 0xD5);
        c_inne_temp[0] = temp_to_char1(inne_temp);
        c_inne_temp[1] = temp_to_char2(inne_temp);
        c_inne_temp[2] = temp_to_char3(inne_temp);
        c_ute_temp[0] = temp_to_char1(ute_temp);
        c_ute_temp[1] = temp_to_char2(ute_temp);
        c_ute_temp[2] = temp_to_char3(ute_temp);
      }
      if ((meny_val_3 == 6) & (disp_rad == 2))
      {
        disp_w_data_r(101, " Temp3         °");
        disp_w_data_r(201, ">Temp4         °");
```

```
        disp_w_data_r(313, c_ute_temp);

        disp_w_data_r(413, c_inne_temp);

        inne_temp = wire_temp_read(0x10, 0xF1, 0xBD, 0x07, 0x00,
0x00, 0x00, 0x9B);

        ute_temp = wire_temp_read(0x10, 0x95, 0x57, 0x13, 0x00,
0x00, 0x00, 0xD5);

        c_inne_temp[0] = temp_to_char1(inne_temp);

        c_inne_temp[1] = temp_to_char2(inne_temp);

        c_inne_temp[2] = temp_to_char3(inne_temp);

        c_ute_temp[0] = temp_to_char1(ute_temp);

        c_ute_temp[1] = temp_to_char2(ute_temp);

        c_ute_temp[2] = temp_to_char3(ute_temp);

      }*/

    }

    //Ärvärden-sensorer

    if ((meny_val_1 == 1) & (meny_val_2 == 2) & (meny_djup == 4))

    {

      if ((meny_val_3 == 1) & (disp_rad == 1))

      {

        disp_w_data_r(101, ">Sensor1<      ");

        disp_w_data_r(201, " Sensor2       ");

        if
(wire_switch_read(0x05,0x45,0x6D,0x00,0x00,0x00,0x00,0xAA))

        {

          disp_w_data_r(116, "/");

        }

        else

        {
```

```
          disp_w_data_r(116, "O");
        }
        if
(wire_switch_read(0x05,0x98,0x63,0x00,0x00,0x00,0x00,0x86))
        {
          disp_w_data_r(216, "/");
        }
        else
        {
          disp_w_data_r(216, "O");
        }
      }
      if ((meny_val_3 == 2) & (disp_rad == 2))
      {
        disp_w_data_r(101, " Sensor1       ");
        disp_w_data_r(201, ">Sensor2<      ");
        if
(wire_switch_read(0x05,0x45,0x6D,0x00,0x00,0x00,0x00,0xAA))
        {
          disp_w_data_r(116, "/");
        }
        else
        {
          disp_w_data_r(116, "O");
        }
        if
(wire_switch_read(0x05,0x98,0x63,0x00,0x00,0x00,0x00,0x86))
        {
```

```
          disp_w_data_r(216, "/");
        }
        else
        {
          disp_w_data_r(216, "O");
        }
      }
      if ((meny_val_3 == 2) & (disp_rad == 1))
      {
        disp_w_data_r(101, ">Sensor2<      ");
        disp_w_data_r(201, " Sensor3       ");
        if
(wire_switch_read(0x05,0x98,0x63,0x00,0x00,0x00,0x00,0x86))
        {
          disp_w_data_r(116, "/");
        }
        else
        {
          disp_w_data_r(116, "O");
        }
        if
(wire_switch_read(0x05,0x45,0x6D,0x00,0x00,0x00,0x00,0xAA))
        {
          disp_w_data_r(216, "/");
        }
        else
        {
```

```
          disp_w_data_r(216, "O");
        }
      }
      if ((meny_val_3 == 3) & (disp_rad == 2))
      {
        disp_w_data_r(101, " Sensor2       ");
        disp_w_data_r(201, ">Sensor3<      ");
        if
(wire_switch_read(0x05,0x98,0x63,0x00,0x00,0x00,0x00,0x86))
        {
          disp_w_data_r(116, "/");
        }
        else
        {
          disp_w_data_r(116, "O");
        }
        if
(wire_switch_read(0x05,0x45,0x6D,0x00,0x00,0x00,0x00,0xAA))
        {
          disp_w_data_r(216, "/");
        }
        else
        {
          disp_w_data_r(216, "O");
        }
      }
      if ((meny_val_3 == 3) & (disp_rad == 1))
```

```
      {
        disp_w_data_r(101, ">Sensor3<      ");
        disp_w_data_r(201, " Sensor4       ");
        if
(wire_switch_read(0x05,0x45,0x6D,0x00,0x00,0x00,0x00,0xAA))
        {
          disp_w_data_r(116, "/");
        }
        else
        {
          disp_w_data_r(116, "O");
        }
        if
(wire_switch_read(0x05,0x98,0x63,0x00,0x00,0x00,0x00,0x86))
        {
          disp_w_data_r(216, "/");
        }
        else
        {
          disp_w_data_r(216, "O");
        }
      }
      if ((meny_val_3 == 4) & (disp_rad == 2))
      {
        disp_w_data_r(101, " Sensor3       ");
        disp_w_data_r(201, ">Sensor4<      ");
        if
(wire_switch_read(0x05,0x45,0x6D,0x00,0x00,0x00,0x00,0xAA))
```

```
        {
          disp_w_data_r(116, "/");
        }
        else
        {
          disp_w_data_r(116, "O");
        }
        if
(wire_switch_read(0x05,0x98,0x63,0x00,0x00,0x00,0x00,0x86))
        {
          disp_w_data_r(216, "/");
        }
        else
        {
          disp_w_data_r(216, "O");
        }
      }
    }
    //Ärvärden-brytare
    if ((meny_val_1 == 1) & (meny_val_2 == 3) & (meny_djup == 4))
    {
      if ((meny_val_3 == 1) & (disp_rad == 1))
      {
        disp_w_data_r(101, ">Brytare1<     ");
        disp_w_data_r(201, " Brytare2      ");
        if
(wire_switch_read(0x05,0x45,0x6D,0x00,0x00,0x00,0x00,0xAA))
```

```
        {
          disp_w_data_r(116, "/");
        }
        else
        {
          disp_w_data_r(116, "O");
        }
        if
(wire_switch_read(0x05,0x98,0x63,0x00,0x00,0x00,0x00,0x86))
        {
          disp_w_data_r(216, "/");
        }
        else
        {
          disp_w_data_r(216, "O");
        }
      }
      if ((meny_val_3 == 2) & (disp_rad == 2))
      {
        disp_w_data_r(101, " Brytare1      ");
        disp_w_data_r(201, ">Brytare2<     ");
        if (meny_val_1 == 1)
        {
          if
(wire_switch_read(0x05,0x45,0x6D,0x00,0x00,0x00,0x00,0xAA))
          {
            disp_w_data_r(116, "/");
```

```
          }
          else
          {
            disp_w_data_r(116, "O");
          }
          if
(wire_switch_read(0x05,0x98,0x63,0x00,0x00,0x00,0x00,0x86))
          {
            disp_w_data_r(216, "/");
          }
          else
          {
            disp_w_data_r(216, "O");
          }
        }
      }
    }
    //Ärvärden & börvärden-Larmnummer
    if (((meny_val_1 == 1) | (meny_val_1 == 2)) & (meny_djup ==
4))
    {
      //Ärvärden & börvärden-Larmnummer
      if (meny_val_2 == 4)
        disp_clear();
      //Ärvärden & börvärden-Tid
      if (meny_val_2 == 5)
        disp_clear();
```

```
  //Ärvärden & börvärden-Kod
  if (meny_val_2 == 6)
    disp_clear();
}
if ((meny_val_1 == 2) & (meny_djup == 4))
{
  //Börvärden-Temp
  if (meny_val_2 == 1)
    disp_clear();
  //Börvärden-Sensorer
  if (meny_val_2 == 2)
    disp_clear();
  //Börvärden-Larmnummer
  if (meny_val_2 == 4)
    disp_clear();
  //Börvärden-Tid
  if (meny_val_2 == 5)
    disp_clear();
}

//börvärden-brytare
if ((meny_val_1 == 2) & (meny_val_2 == 3) & (meny_djup == 4))
{
  if ((meny_val_3 == 1) & (disp_rad == 1))
  {
    disp_w_data_r(101, ">Brytare1<      ");
```

```
        disp_w_data_r(201, " Brytare2       ");
      }
      if ((meny_val_3 == 2) & (disp_rad == 2))
      {
        disp_w_data_r(101, " Brytare1       ");
        disp_w_data_r(201, ">Brytare2<      ");
      }
    }
  }
}
```

//1-wire

```
#include "IO-reg.h"
#include "Vectors.h"
#include "intr6811.h"

void disp_init(void);
void disp_clear(void);
void disp_return_home(void);
void disp_off(void);
void disp_cursor_off(void);
void disp_cursor_blink_on(void);
void disp_two_line(void);
void disp_cursor_blink_off(void);
```

```
void disp_w_data_r(int skriv_start, char CGData[]);

void disp_check_busy(void);

int wire_reset(void);

void wire_one(void);

void wire_zero(void);

void wire_write(int w_data);

int wire_read(void);

int wire_word_read(void);

void keyb_init(void);

void enable_interrupt(void);


int Scratch_1, Scratch_2, Scratch_3, Scratch_4, Scratch_5,
Scratch_6, Scratch_7, Scratch_8, Scratch_9, Scratch_CRC;

int a, serial_byte_1, serial_byte_2, serial_byte_3,
serial_byte_4, serial_byte_5, serial_byte_6, serial_byte_CRC,
ny_data, fam_code, serial, ut_data, bitar, bitar2, data, data1,
data2, data3;


int meny_rad, meny_djup;


main(void)

{

  while(1==1)

  {

    keyb_init();

    enable_interrupt();

    disp_init();

    disp_w_data_r(106, "*MENY*");
```

```
disp_w_data_r(201, "Inne 22º Ute 08º");
for (a=0;a<=400;a++);
disp_w_data_r(201, "Dörr O Fönster C");
for (a=0;a<=400;a++);
data = wire_reset();
wire_write(0x33);
fam_code = wire_word_read();
serial_byte_1 = wire_word_read();
serial_byte_2 = wire_word_read();
serial_byte_3 = wire_word_read();
serial_byte_4 = wire_word_read();
serial_byte_5 = wire_word_read();
serial_byte_6 = wire_word_read();
serial_byte_CRC = wire_word_read();
data = wire_reset();

wire_write(0x55);
wire_write(0x10);
wire_write(0xF1);
wire_write(0x63);
wire_write(0x00);
wire_write(0x00);
wire_write(0x00);
wire_write(0x00);
wire_write(0x86);
wire_write(0x05);
```

```
    wire_write(0xDB);
    wire_write(0x20);
    wire_write(0x36);
    wire_write(0x3A);
    wire_write(0x34);
    wire_write(0x09);
    wire_write(0x86);

    while (1==1);
  }
}


void disp_init(void)
{
  disp_clear();
  disp_two_line();
  disp_cursor_blink_on();

}
void disp_clear(void)
{
  disp_check_busy();
  PORTB = PORTB & 0xEF; // Disable DTMF PB4=0
  PORTB = PORTB & 0xF3; // (R/W)=0 RS=0 PB2=0, PB3=0
  DDRC = 0xFF; //Alla bitar på port c OUT
```

```
  PORTB = PORTB | 0x02; // Enable LCD PB1=1
  PORTC = 0x01; // Data=00000001
  PORTB = PORTB & 0xFD; // Disable LCD PB1=0
}

void disp_cursor_blink_on(void)
{
  disp_check_busy();
  PORTB = PORTB & 0xEF; // Disable DTMF PB4=0
  PORTB = PORTB & 0xF3; // (R/W)=0 RS=0 PB2=0, PB3=0
  DDRC = 0xFF; //Alla bitar på port c OUT
  PORTB = PORTB | 0x02; // Enable LCD PB1=1
  PORTC = 0x0F; // Data=00001111
  PORTB = PORTB & 0xFD; // Disable LCD PB1=0
}

void disp_cursor_blink_off(void)
{
  disp_check_busy();
  PORTB = PORTB & 0xEF; // Disable DTMF PB4=0
  PORTB = PORTB & 0xF3; // (R/W)=0 RS=0 PB2=0, PB3=0
  DDRC = 0xFF; //Alla bitar på port c OUT
  PORTB = PORTB | 0x02; // Enable LCD PB1=1
  PORTC = 0x0E; // Data=00001110
  PORTB = PORTB & 0xFD; // Disable LCD PB1=0
}
```

```
void disp_two_line(void)

{

  disp_check_busy();

  PORTB = PORTB & 0xEF; // Disable DTMF PB4=0

  PORTB = PORTB & 0xF3; // (R/W)=0 RS=0 PB2=0, PB3=0

  DDRC = 0xFF; //Alla bitar på port c OUT

  PORTB = PORTB | 0x02; // Enable LCD PB1=1

  PORTC = 0x3C; // Data=00111100 // Function set 8 bits, 2 line,
5*10 dots

  PORTB = PORTB & 0xFD; // Disable LCD PB1=0

}


void disp_cursor_off(void)

{

  disp_check_busy();

  PORTB = PORTB & 0xEF; // Disable DTMF PB4=0

  PORTB = PORTB & 0xF3; // (R/W)=0 RS=0 PB2=0, PB3=0

  DDRC = 0xFF; //Alla bitar på port c OUT

  PORTB = PORTB | 0x02; // Enable LCD PB1=1

  PORTC = 0x0C; // Data=00001100

  PORTB = PORTB & 0xFD; // Disable LCD PB1=0

}


void disp_off(void)

{

  disp_check_busy();
```

```
  PORTB = PORTB & 0xEF; // Disable DTMF PB4=0
  PORTB = PORTB & 0xF3; // (R/W)=0 RS=0 PB2=0, PB3=0
  DDRC = 0xFF; //Alla bitar på port c OUT
  PORTB = PORTB | 0x02; // Enable LCD PB1=1
  PORTC = 0x08; // Data=00001000
  PORTB = PORTB & 0xFD; // Disable LCD PB1=0
}

void disp_return_home(void)
{
  disp_check_busy();
  PORTB = PORTB & 0xEF; // Disable DTMF PB4=0
  PORTB = PORTB & 0xF3; // (R/W)=0 RS=0 PB2=0, PB3=0
  DDRC = 0xFF; //Alla bitar på port c OUT
  PORTB = PORTB | 0x02; // Enable LCD PB1=1
  PORTC = 0x02; // Data=00000010
  PORTB = PORTB & 0xFD; // Disable LCD PB1=0
}

void disp_w_data_r(int skriv_start, char CGData[])
{
  int a,b;
  b=0;
  disp_check_busy();
  PORTB = PORTB & 0xEF; // Disable DTMF PB4=0
  PORTB = PORTB & 0xF3; // (R/W)=0 RS=0 PB2=0, PB3=0
```

```
  DDRC = 0xFF; //Alla bitar på port c OUT

  PORTB = PORTB | 0x02; // Enable LCD PB1=1

  if ((skriv_start>=101) & (skriv_start<=116))

    PORTC = 0x80+(skriv_start-101); // Sets the DD RAM adress,
första tecknet på andra raden 0x40+skriv_start

  if ((skriv_start>=201) & (skriv_start<=216))

    PORTC = 0xC0+(skriv_start-201); // Sets the DD RAM adress,
första tecknet på andra raden 0x40+skriv_start

  PORTB = PORTB & 0xFD; // Disable LCD PB1=0

  while (CGData[b] != '\0')

    b++;

  for (a=0; a<=(b-1); a++)

  {

    PORTB = PORTB & 0xF3; // (R/W)=0 RS=0 PB2=0, PB3=0

    PORTB = PORTB | 0x02; // Enable LCD PB1=1

    PORTC = 0x10; // Data=00010000 Cursor or display shift,
display shift, shift to the right

    PORTB = PORTB & 0xFB; // (R/W)=0, PB2=0

    PORTB = PORTB | 0x08; // RS=1, PB3=1

    switch (CGData[a])

    {

      case 'A' : PORTC = 0x41;break;

      case 'B' : PORTC = 0x42;break;

      case 'C' : PORTC = 0x43;break;

      case 'D' : PORTC = 0x44;break;

      case 'E' : PORTC = 0x45;break;

      case 'F' : PORTC = 0x46;break;

      case 'G' : PORTC = 0x47;break;
```

```
case 'H' : PORTC = 0x48;break;
case 'I' : PORTC = 0x49;break;
case 'J' : PORTC = 0x4A;break;
case 'K' : PORTC = 0x4B;break;
case 'L' : PORTC = 0x4C;break;
case 'M' : PORTC = 0x4D;break;
case 'N' : PORTC = 0x4E;break;
case 'O' : PORTC = 0x4F;break;
case 'P' : PORTC = 0x50;break;
case 'Q' : PORTC = 0x51;break;
case 'R' : PORTC = 0x52;break;
case 'S' : PORTC = 0x53;break;
case 'T' : PORTC = 0x54;break;
case 'U' : PORTC = 0x55;break;
case 'V' : PORTC = 0x56;break;
case 'W' : PORTC = 0x47;break;
case 'X' : PORTC = 0x58;break;
case 'Y' : PORTC = 0x59;break;
case 'Z' : PORTC = 0x5A;break;
case 'Å' : PORTC = 0xFF;
/*
PORTB = PORTB & 0xF3; // (R/W)=0 RS=0 PB2=0, PB3=0
PORTC = 0x40; //Sets the CG RAM address 01000000
PORTB = PORTB & 0xFD; // Disable LCD PB1=0

PORTB = PORTB | 0x02; // Enable LCD PB1=1
```

```
PORTB = PORTB & 0xFB; // (R/W)=0, PB2=0
PORTB = PORTB | 0x08; // RS=1, PB3=1
PORTC = 0x04; //Sets the CG RAM data 00000100
PORTB = PORTB & 0xFD; // Disable LCD PB1=0

PORTB = PORTB & 0xF3; // (R/W)=0 RS=0 PB2=0, PB3=0
PORTC = 0x41; //Sets the CG RAM address 01000001

PORTB = PORTB & 0xFB; // (R/W)=0, PB2=0
PORTB = PORTB | 0x08; // RS=1, PB3=1
PORTC = 0x00; //Sets the CG RAM data 00000000

PORTB = PORTB & 0xF3; // (R/W)=0 RS=0 PB2=0, PB3=0
PORTC = 0x42; //Sets the CG RAM address 01000010

PORTB = PORTB & 0xFB; // (R/W)=0, PB2=0
PORTB = PORTB | 0x08; // RS=1, PB3=1
PORTC = 0x0E; //Sets the CG RAM data 00001110

PORTB = PORTB & 0xF3; // (R/W)=0 RS=0 PB2=0, PB3=0
PORTC = 0x43; //Sets the CG RAM address 01000011

PORTB = PORTB & 0xFB; // (R/W)=0, PB2=0
PORTB = PORTB | 0x08; // RS=1, PB3=1
PORTC = 0x11; //Sets the CG RAM data 00010001
```

```
PORTB = PORTB & 0xF3; // (R/W)=0 RS=0 PB2=0, PB3=0
PORTC = 0x44; //Sets the CG RAM address 01000100

PORTB = PORTB & 0xFB; // (R/W)=0, PB2=0
PORTB = PORTB | 0x08; // RS=1, PB3=1
PORTC = 0x1F; //Sets the CG RAM data 00011111

PORTB = PORTB & 0xF3; // (R/W)=0 RS=0 PB2=0, PB3=0
PORTC = 0x45; //Sets the CG RAM address 01000101

PORTB = PORTB & 0xFB; // (R/W)=0, PB2=0
PORTB = PORTB | 0x08; // RS=1, PB3=1
PORTC = 0x11; //Sets the CG RAM data 00010001

PORTB = PORTB & 0xF3; // (R/W)=0 RS=0 PB2=0, PB3=0
PORTC = 0x46; //Sets the CG RAM address 01000110

PORTB = PORTB & 0xFB; // (R/W)=0, PB2=0
PORTB = PORTB | 0x08; // RS=1, PB3=1
PORTC = 0x11; //Sets the CG RAM data 00010001
*/
break;
case 'Ä' : PORTC = 0xFF;break;
case 'Ö' : PORTC = 0xEF;break;
case 'a' : PORTC = 0x61;break;
case 'b' : PORTC = 0x62;break;
```

```
case 'c' : PORTC = 0x63;break;
case 'd' : PORTC = 0x64;break;
case 'e' : PORTC = 0x65;break;
case 'f' : PORTC = 0x66;break;
case 'g' : PORTC = 0x67;break;
case 'h' : PORTC = 0x68;break;
case 'i' : PORTC = 0x69;break;
case 'j' : PORTC = 0x6A;break;
case 'k' : PORTC = 0x6B;break;
case 'l' : PORTC = 0x6C;break;
case 'm' : PORTC = 0x6D;break;
case 'n' : PORTC = 0x6E;break;
case 'o' : PORTC = 0x6F;break;
case 'p' : PORTC = 0x70;break;
case 'q' : PORTC = 0x71;break;
case 'r' : PORTC = 0x72;break;
case 's' : PORTC = 0x73;break;
case 't' : PORTC = 0x74;break;
case 'u' : PORTC = 0x75;break;
case 'v' : PORTC = 0x76;break;
case 'w' : PORTC = 0x77;break;
case 'x' : PORTC = 0x78;break;
case 'y' : PORTC = 0x79;break;
case 'z' : PORTC = 0x7A;break;
case 'å' : PORTC = 0xFF;break;
case 'ä' : PORTC = 0xFF;break;
```

```
      case 'ö' : PORTC = 0xEF;break;
      case '0' : PORTC = 0x30;break;
      case '1' : PORTC = 0x31;break;
      case '2' : PORTC = 0x32;break;
      case '3' : PORTC = 0x33;break;
      case '4' : PORTC = 0x34;break;
      case '5' : PORTC = 0x35;break;
      case '6' : PORTC = 0x36;break;
      case '7' : PORTC = 0x37;break;
      case '8' : PORTC = 0x38;break;
      case '9' : PORTC = 0x39;break;
      case '-' : PORTC = 0x2D;break;
      case ' ' : PORTC = 0x10;break;
      case '*' : PORTC = 0x2A;break;
      case 'º' : PORTC = 0xDF;break;
      default: PORTC = 0x10;
      }
    PORTB = PORTB & 0xFD; // Disable LCD PB1=0
  }
  PORTB = PORTB & 0xFD; // Disable LCD PB1=0
}

void disp_check_busy(void)
{
  DDRC = 0x00; //Alla bitar på port c IN
  PORTB = PORTB | 0x04; // (R/W)=1, PB2=1
```

```
  PORTB = PORTB & 0xF7; // RS=0, PB3=0

  PORTB = PORTB | 0x02; // Enable LCD PB1=1

  while ((PORTC & 0x80)==0x80); //Väntar tills busy-flaggan är
noll

  PORTB = PORTB & 0xFD; // Disable LCD PB1=0

}



/*void disp_w_data_l(char CGData)

{

  DDRC = 0xFF; //Alla bitar på port c OUT

  PORTB = PORTB & 0xEF; // Disable DTMF PB4=0

  PORTB = PORTB & 0xF3; // (R/W)=0 RS=0 PB2=0, PB3=0

  PORTB = PORTB | 0x02; // Enable LCD PB1=1

  PORTC = 0x14; // Data=00011100 Cursor or display shift, cursor
move, shift to the right

  PORTB = PORTB & 0xFB; // (R/W)=0, PB2=0

  PORTB = PORTB | 0x08; // RS=0, PB3=1

  PORTC = CGData;

  PORTB = PORTB & 0xFD; // Disable LCD PB1=0

} */



int wire_reset(void)

{

  //Drar ner till noll under 725us (uppmätt), returnerar en etta
om det kom ett svar från någon 1-wire

  int c, w_data;

  PORTA = PORTA & 0xF7; //Bit 3=0 PA3=0

  PACTL = PACTL | 0x08; //Bit 3=1 PA3=OUT
```

```
  for (c=0; c<=50; c++);
  PACTL = PACTL & 0xF7; //Bit 7=0 PA7=IN
  for(c=0;c<=3;c++); //83us
  w_data = PORTA & 0x08;
  for(c=0;c<=23;c++); //Väntar 480us (uppmätt) för recovery
  return (~(w_data >> 3) & 0x01);
}


void wire_one(void)
{
  //Drar ner till noll under 5us
  int d;
  for (d=0; d<=6; d++); //Väntar >120us innan biten sänds (130us
uppmätt)
  PORTA = PORTA & 0xF7; //Bit 3=0 PA3=0
  PACTL = PACTL | 0x08; //Bit 3=1 PA3=OUT
  PACTL = PACTL & 0xF7; //Bit 7=0 PA7=IN
}


void wire_zero(void)
{
  //Drar ner till noll under 74us
  int e;
  for (e=0; e<=6; e++); //Väntar >120us innan biten sänds (130us
uppmätt)
  PORTA = PORTA & 0xF7; //Bit 3=0 PA3=0
  PACTL = PACTL | 0x08; //Bit 3=1 PA3=OUT
```

```
  for (e=0; e<=3; e++);
  PACTL = PACTL & 0xF7; //Bit 7=0 PA7=IN
}


int wire_read(void)
{
  int w_data;
  wire_one();
  w_data = PORTA & 0x08;
  return w_data >> 3;
}


int wire_word_read(void) //Läser av 8 bitar från 1-wire och
returnerar ordet.
{
  int g, w_data, w_word, multip;
  w_word = 0;
  multip = 1;
  for (g=0;g<=7;g++)
  {
    wire_one();
    w_data = PORTA & 0x08;
    w_data = w_data >> 3;
    w_word = w_word + (w_data * multip);
    multip = multip << 1;
  }
  return w_word;
```

```
}


void wire_write(int w_data) //Skriver 8 bitar till 1-wire
{
  int f;
  for (f=0;f<=7;f++)
  {
    if (w_data >> f & 0x01)
      wire_one();
    else
      wire_zero();
  }
}


void keyb_init(void)
{
  TCTL2 = TCTL2 | 0x10; //bit 4=1 Rising edge only
  TCTL2 = TCTL2 & 0xDF; //bit 5=0 Rising edge only
  TMSK1 = TMSK1 | 0x04; //Sätter Timer interrupt Mask Register 1,
TMSK1 bit2=1
}


interrupt void IC1_interrupt(void)
{
  TFLG1 = TFLG1 | 0x04; // Kvitterar interupptet så återgång kan
ske, sätter bit 2 till ett i Timer interupt Flag 1
  disp_w_data_r(210, "Knapp");
```

```
}
```

//Rutiner för display

```
#include "IO-reg.h"
#include "Vectors.h"
#include "intr6811.h"

void disp_init(void);
void disp_clear(void);
void disp_return_home(void);
void disp_off(void);
void disp_cursor_off(void);
void disp_cursor_blink_on(void);
void disp_two_line(void);
void disp_cursor_blink_off(void);
void disp_w_data_r(int skriv_start, char CGData[]);
void disp_check_busy(void);
void disp_w_nummer(int skriv_start, char nummer);
char temp_to_char(int temp);

void disp_init(void)
{
  char k;
  disp_clear();
  disp_two_line();
```

```
disp_cursor_blink_on();

for (k=0; k<=3; k++);
for (k=0; k<=3; k++);
PORTB = PORTB & 0xFD; // Disable LCD PB1=0
PORTB = PORTB & 0xF3; // (R/W)=0 RS=0 PB2=0, PB3=0
PORTB = PORTB | 0x02; // Enable LCD PB1=1
for (k=0; k<=3; k++);
PORTC = 0x40; //Sets the CG RAM address 01000000
PORTB = PORTB & 0xFD; // Disable LCD PB1=0

for (k=0; k<=3; k++);
PORTB = PORTB & 0xFB; // (R/W)=0, PB2=0
PORTB = PORTB | 0x08; // RS=1, PB3=1
PORTB = PORTB | 0x02; // Enable LCD PB1=1
for (k=0; k<=3; k++);
PORTC = 0x04; //Sets the CG RAM data 00000100
PORTB = PORTB & 0xFD; // Disable LCD PB1=0

for (k=0; k<=3; k++);
PORTB = PORTB & 0xF3; // (R/W)=0 RS=0 PB2=0, PB3=0
PORTB = PORTB | 0x02; // Enable LCD PB1=1
for (k=0; k<=3; k++);
PORTC = 0x41; //Sets the CG RAM address 01000001
PORTB = PORTB & 0xFD; // Disable LCD PB1=0
```

```
for (k=0; k<=3; k++);
PORTB = PORTB & 0xFB; // (R/W)=0, PB2=0
PORTB = PORTB | 0x08; // RS=1, PB3=1
PORTB = PORTB | 0x02; // Enable LCD PB1=1
for (k=0; k<=3; k++);
PORTC = 0x00; //Sets the CG RAM data 00000000
PORTB = PORTB & 0xFD; // Disable LCD PB1=0

for (k=0; k<=3; k++);
PORTB = PORTB & 0xF3; // (R/W)=0 RS=0 PB2=0, PB3=0
PORTB = PORTB | 0x02; // Enable LCD PB1=1
for (k=0; k<=3; k++);
PORTC = 0x42; //Sets the CG RAM address 01000010
PORTB = PORTB & 0xFD; // Disable LCD PB1=0

for (k=0; k<=3; k++);
PORTB = PORTB & 0xFB; // (R/W)=0, PB2=0
PORTB = PORTB | 0x08; // RS=1, PB3=1
PORTB = PORTB | 0x02; // Enable LCD PB1=1
for (k=0; k<=3; k++);
PORTC = 0x0E; //Sets the CG RAM data 00001110
PORTB = PORTB & 0xFD; // Disable LCD PB1=0

for (k=0; k<=3; k++);
PORTB = PORTB & 0xF3; // (R/W)=0 RS=0 PB2=0, PB3=0
PORTB = PORTB | 0x02; // Enable LCD PB1=1
```

```
for (k=0; k<=3; k++);
PORTC = 0x43; //Sets the CG RAM address 01000011
PORTB = PORTB & 0xFD; // Disable LCD PB1=0

for (k=0; k<=3; k++);
PORTB = PORTB & 0xFB; // (R/W)=0, PB2=0
PORTB = PORTB | 0x08; // RS=1, PB3=1
PORTB = PORTB | 0x02; // Enable LCD PB1=1
for (k=0; k<=3; k++);
PORTC = 0x11; //Sets the CG RAM data 00010001
PORTB = PORTB & 0xFD; // Disable LCD PB1=0



for (k=0; k<=3; k++);
PORTB = PORTB & 0xF3; // (R/W)=0 RS=0 PB2=0, PB3=0
PORTB = PORTB | 0x02; // Enable LCD PB1=1
for (k=0; k<=3; k++);
PORTC = 0x44; //Sets the CG RAM address 01000100
PORTB = PORTB & 0xFD; // Disable LCD PB1=0

for (k=0; k<=3; k++);
PORTB = PORTB & 0xFB; // (R/W)=0, PB2=0
PORTB = PORTB | 0x08; // RS=1, PB3=1
for (k=0; k<=3; k++);
PORTB = PORTB | 0x02; // Enable LCD PB1=1
PORTC = 0x1F; //Sets the CG RAM data 00011111
```

```
PORTB = PORTB & 0xFD; // Disable LCD PB1=0

for (k=0; k<=3; k++);
PORTB = PORTB & 0xF3; // (R/W)=0 RS=0 PB2=0, PB3=0
PORTB = PORTB | 0x02; // Enable LCD PB1=1
for (k=0; k<=3; k++);
PORTC = 0x45; //Sets the CG RAM address 01000101
PORTB = PORTB & 0xFD; // Disable LCD PB1=0

for (k=0; k<=3; k++);
PORTB = PORTB & 0xFB; // (R/W)=0, PB2=0
PORTB = PORTB | 0x08; // RS=1, PB3=1
PORTB = PORTB | 0x02; // Enable LCD PB1=1
for (k=0; k<=3; k++);
PORTC = 0x11; //Sets the CG RAM data 00010001
PORTB = PORTB & 0xFD; // Disable LCD PB1=0

for (k=0; k<=3; k++);
PORTB = PORTB & 0xF3; // (R/W)=0 RS=0 PB2=0, PB3=0
PORTB = PORTB | 0x02; // Enable LCD PB1=1
for (k=0; k<=3; k++);
PORTC = 0x46; //Sets the CG RAM address 01000110
PORTB = PORTB & 0xFD; // Disable LCD PB1=0

for (k=0; k<=3; k++);
PORTB = PORTB & 0xFB; // (R/W)=0, PB2=0
```

```
PORTB = PORTB | 0x08; // RS=1, PB3=1
PORTB = PORTB | 0x02; // Enable LCD PB1=1
for (k=0; k<=3; k++);
PORTC = 0x11; //Sets the CG RAM data 00010001
PORTB = PORTB & 0xFD; // Disable LCD PB1=0

for (k=0; k<=3; k++);
PORTB = PORTB & 0xF3; // (R/W)=0 RS=0 PB2=0, PB3=0
PORTB = PORTB | 0x02; // Enable LCD PB1=1
for (k=0; k<=3; k++);
PORTC = 0x47; //Sets the CG RAM address 01000111
PORTB = PORTB & 0xFD; // Disable LCD PB1=0

for (k=0; k<=3; k++);
PORTB = PORTB & 0xFB; // (R/W)=0, PB2=0
PORTB = PORTB | 0x08; // RS=1, PB3=1
for (k=0; k<=3; k++);
PORTB = PORTB | 0x02; // Enable LCD PB1=1
PORTC = 0x00; //Sets the CG RAM data 00000000
PORTB = PORTB & 0xFD; // Disable LCD PB1=0

//Ä
for (k=0; k<=3; k++);
PORTB = PORTB & 0xFD; // Disable LCD PB1=0
PORTB = PORTB & 0xF3; // (R/W)=0 RS=0 PB2=0, PB3=0
PORTB = PORTB | 0x02; // Enable LCD PB1=1
```

```
for (k=0; k<=3; k++);
PORTC = 0x48; //Sets the CG RAM address 01001000
PORTB = PORTB & 0xFD; // Disable LCD PB1=0

for (k=0; k<=3; k++);
PORTB = PORTB & 0xFB; // (R/W)=0, PB2=0
PORTB = PORTB | 0x08; // RS=1, PB3=1
PORTB = PORTB | 0x02; // Enable LCD PB1=1
for (k=0; k<=3; k++);
PORTC = 0x0A; //Sets the CG RAM data 00001010
PORTB = PORTB & 0xFD; // Disable LCD PB1=0

for (k=0; k<=3; k++);
PORTB = PORTB & 0xF3; // (R/W)=0 RS=0 PB2=0, PB3=0
PORTB = PORTB | 0x02; // Enable LCD PB1=1
for (k=0; k<=3; k++);
PORTC = 0x49; //Sets the CG RAM address 01001001
PORTB = PORTB & 0xFD; // Disable LCD PB1=0

for (k=0; k<=3; k++);
PORTB = PORTB & 0xFB; // (R/W)=0, PB2=0
PORTB = PORTB | 0x08; // RS=1, PB3=1
PORTB = PORTB | 0x02; // Enable LCD PB1=1
for (k=0; k<=3; k++);
PORTC = 0x00; //Sets the CG RAM data 00000000
PORTB = PORTB & 0xFD; // Disable LCD PB1=0
```

```
for (k=0; k<=3; k++);
PORTB = PORTB & 0xF3; // (R/W)=0 RS=0 PB2=0, PB3=0
PORTB = PORTB | 0x02; // Enable LCD PB1=1
for (k=0; k<=3; k++);
PORTC = 0x4A; //Sets the CG RAM address 01001010
PORTB = PORTB & 0xFD; // Disable LCD PB1=0

for (k=0; k<=3; k++);
PORTB = PORTB & 0xFB; // (R/W)=0, PB2=0
PORTB = PORTB | 0x08; // RS=1, PB3=1
PORTB = PORTB | 0x02; // Enable LCD PB1=1
for (k=0; k<=3; k++);
PORTC = 0x0E; //Sets the CG RAM data 00001110
PORTB = PORTB & 0xFD; // Disable LCD PB1=0

for (k=0; k<=3; k++);
PORTB = PORTB & 0xF3; // (R/W)=0 RS=0 PB2=0, PB3=0
PORTB = PORTB | 0x02; // Enable LCD PB1=1
for (k=0; k<=3; k++);
PORTC = 0x4B; //Sets the CG RAM address 01001011
PORTB = PORTB & 0xFD; // Disable LCD PB1=0

for (k=0; k<=3; k++);
PORTB = PORTB & 0xFB; // (R/W)=0, PB2=0
PORTB = PORTB | 0x08; // RS=1, PB3=1
```

```
PORTB = PORTB | 0x02; // Enable LCD PB1=1
for (k=0; k<=3; k++);
PORTC = 0x11; //Sets the CG RAM data 00010001
PORTB = PORTB & 0xFD; // Disable LCD PB1=0

for (k=0; k<=3; k++);
PORTB = PORTB & 0xF3; // (R/W)=0 RS=0 PB2=0, PB3=0
PORTB = PORTB | 0x02; // Enable LCD PB1=1
for (k=0; k<=3; k++);
PORTC = 0x4C; //Sets the CG RAM address 01001100
PORTB = PORTB & 0xFD; // Disable LCD PB1=0

for (k=0; k<=3; k++);
PORTB = PORTB & 0xFB; // (R/W)=0, PB2=0
PORTB = PORTB | 0x08; // RS=1, PB3=1
for (k=0; k<=3; k++);
PORTB = PORTB | 0x02; // Enable LCD PB1=1
PORTC = 0x1F; //Sets the CG RAM data 00011111
PORTB = PORTB & 0xFD; // Disable LCD PB1=0

for (k=0; k<=3; k++);
PORTB = PORTB & 0xF3; // (R/W)=0 RS=0 PB2=0, PB3=0
PORTB = PORTB | 0x02; // Enable LCD PB1=1
for (k=0; k<=3; k++);
PORTC = 0x4D; //Sets the CG RAM address 01010101
PORTB = PORTB & 0xFD; // Disable LCD PB1=0
```

```
for (k=0; k<=3; k++);
PORTB = PORTB & 0xFB; // (R/W)=0, PB2=0
PORTB = PORTB | 0x08; // RS=1, PB3=1
PORTB = PORTB | 0x02; // Enable LCD PB1=1
for (k=0; k<=3; k++);
PORTC = 0x11; //Sets the CG RAM data 00010001
PORTB = PORTB & 0xFD; // Disable LCD PB1=0

for (k=0; k<=3; k++);
PORTB = PORTB & 0xF3; // (R/W)=0 RS=0 PB2=0, PB3=0
PORTB = PORTB | 0x02; // Enable LCD PB1=1
for (k=0; k<=3; k++);
PORTC = 0x4E; //Sets the CG RAM address 01001110
PORTB = PORTB & 0xFD; // Disable LCD PB1=0

for (k=0; k<=3; k++);
PORTB = PORTB & 0xFB; // (R/W)=0, PB2=0
PORTB = PORTB | 0x08; // RS=1, PB3=1
PORTB = PORTB | 0x02; // Enable LCD PB1=1
for (k=0; k<=3; k++);
PORTC = 0x11; //Sets the CG RAM data 00010001
PORTB = PORTB & 0xFD; // Disable LCD PB1=0

for (k=0; k<=3; k++);
PORTB = PORTB & 0xF3; // (R/W)=0 RS=0 PB2=0, PB3=0
```

```
  PORTB = PORTB | 0x02; // Enable LCD PB1=1
  for (k=0; k<=3; k++);
  PORTC = 0x4F; //Sets the CG RAM address 01001111
  PORTB = PORTB & 0xFD; // Disable LCD PB1=0

  for (k=0; k<=3; k++);
  PORTB = PORTB & 0xFB; // (R/W)=0, PB2=0
  PORTB = PORTB | 0x08; // RS=1, PB3=1
  for (k=0; k<=3; k++);
  PORTB = PORTB | 0x02; // Enable LCD PB1=1
  PORTC = 0x00; //Sets the CG RAM data 00000000
  PORTB = PORTB & 0xFD; // Disable LCD PB1=0
}

void disp_clear(void)
{
  disp_check_busy();
  PORTB = PORTB & 0xEF; // Disable DTMF PB4=0
  PORTB = PORTB & 0xF3; // (R/W)=0 RS=0 PB2=0, PB3=0
  DDRC = 0xFF; //Alla bitar på port c OUT
  PORTB = PORTB | 0x02; // Enable LCD PB1=1
  PORTC = 0x01; // Data=00000001
  PORTB = PORTB & 0xFD; // Disable LCD PB1=0
}

void disp_cursor_blink_on(void)
```

```
{
  disp_check_busy();
  PORTB = PORTB & 0xEF; // Disable DTMF PB4=0
  PORTB = PORTB & 0xF3; // (R/W)=0 RS=0 PB2=0, PB3=0
  DDRC = 0xFF; //Alla bitar på port c OUT
  PORTB = PORTB | 0x02; // Enable LCD PB1=1
  PORTC = 0x0F; // Data=00001111
  PORTB = PORTB & 0xFD; // Disable LCD PB1=0
}

void disp_cursor_blink_off(void)
{
  disp_check_busy();
  PORTB = PORTB & 0xEF; // Disable DTMF PB4=0
  PORTB = PORTB & 0xF3; // (R/W)=0 RS=0 PB2=0, PB3=0
  DDRC = 0xFF; //Alla bitar på port c OUT
  PORTB = PORTB | 0x02; // Enable LCD PB1=1
  PORTC = 0x0E; // Data=00001110
  PORTB = PORTB & 0xFD; // Disable LCD PB1=0
}

void disp_two_line(void)
{
  disp_check_busy();
  PORTB = PORTB & 0xEF; // Disable DTMF PB4=0
  PORTB = PORTB & 0xF3; // (R/W)=0 RS=0 PB2=0, PB3=0
```

```
  DDRC = 0xFF; //Alla bitar på port c OUT
  PORTB = PORTB | 0x02; // Enable LCD PB1=1
  PORTC = 0x3C; // Data=00111100 // Function set 8 bits, 2 line,
5*10 dots
  PORTB = PORTB & 0xFD; // Disable LCD PB1=0
}

void disp_cursor_off(void)
{
  disp_check_busy();
  PORTB = PORTB & 0xEF; // Disable DTMF PB4=0
  PORTB = PORTB & 0xF3; // (R/W)=0 RS=0 PB2=0, PB3=0
  DDRC = 0xFF; //Alla bitar på port c OUT
  PORTB = PORTB | 0x02; // Enable LCD PB1=1
  PORTC = 0x0C; // Data=00001100
  PORTB = PORTB & 0xFD; // Disable LCD PB1=0
}

void disp_off(void)
{
  disp_check_busy();
  PORTB = PORTB & 0xEF; // Disable DTMF PB4=0
  PORTB = PORTB & 0xF3; // (R/W)=0 RS=0 PB2=0, PB3=0
  DDRC = 0xFF; //Alla bitar på port c OUT
  PORTB = PORTB | 0x02; // Enable LCD PB1=1
  PORTC = 0x08; // Data=00001000
  PORTB = PORTB & 0xFD; // Disable LCD PB1=0
```

```
}


void disp_return_home(void)
{
  disp_check_busy();
  PORTB = PORTB & 0xEF; // Disable DTMF PB4=0
  PORTB = PORTB & 0xF3; // (R/W)=0 RS=0 PB2=0, PB3=0
  DDRC = 0xFF; //Alla bitar på port c OUT
  PORTB = PORTB | 0x02; // Enable LCD PB1=1
  PORTC = 0x02; // Data=00000010
  PORTB = PORTB & 0xFD; // Disable LCD PB1=0
}


void disp_w_data_r(int skriv_start, char CGData[])
{
  int a,b;
  b=0;
  disp_check_busy();
  PORTB = PORTB & 0xEF; // Disable DTMF PB4=0
  PORTB = PORTB & 0xF3; // (R/W)=0 RS=0 PB2=0, PB3=0
  DDRC = 0xFF; //Alla bitar på port c OUT
  PORTB = PORTB | 0x02; // Enable LCD PB1=1
  if ((skriv_start>=101) & (skriv_start<=116))
    PORTC = 0x80+(skriv_start-101); // Sets the DD RAM adress,
första tecknet på andra raden 0x40+skriv_start
  if ((skriv_start>=201) & (skriv_start<=216))
```

```
    PORTC = 0xC0+(skriv_start-201); // Sets the DD RAM adress,
första tecknet på andra raden 0x40+skriv_start

  if ((skriv_start>=301) & (skriv_start<=316))

    PORTC = 0x80+(skriv_start-301); // Sets the DD RAM adress,
första tecknet på andra raden 0x40+skriv_start

  if ((skriv_start>=401) & (skriv_start<=416))

    PORTC = 0xC0+(skriv_start-401); // Sets the DD RAM adress,
första tecknet på andra raden 0x40+skriv_start

  PORTB = PORTB & 0xFD; // Disable LCD PB1=0

  if (skriv_start<300) //För att kunna skriva temp till display.

  {

    while (CGData[b] != '\0')

      b++;

  }

  else

    b = 3;

  for (a=0; a<=(b-1); a++)

  {

    PORTB = PORTB & 0xF3; // (R/W)=0 RS=0 PB2=0, PB3=0

    PORTB = PORTB | 0x02; // Enable LCD PB1=1

    PORTC = 0x10; // Data=00010000 Cursor or display shift,
display shift, shift to the right

    PORTB = PORTB & 0xFB; // (R/W)=0, PB2=0

    PORTB = PORTB | 0x08; // RS=1, PB3=1

    switch (CGData[a])

    {

      case '/' : PORTC = 0xFF;break;

      case 'º' : PORTC = 0xDF;break;
```

```
      case '<' : PORTC = 0x7F;break;
      case '>' : PORTC = 0x7E;break;
      case 'Å' : PORTC = 0x00;break;
      case 'Ä' : PORTC = 0x01;break;
      case 'Ö' : PORTC = 0xEF;break;
      case 'å' : PORTC = 0xFF;break;
      case 'ä' : PORTC = 0xE1;break;
      case 'ö' : PORTC = 0xEF;break;
      default: PORTC = CGData[a];
      }
    PORTB = PORTB & 0xFD; // Disable LCD PB1=0
  }
  PORTB = PORTB & 0xFD; // Disable LCD PB1=0
  disp_cursor_off();
}


void disp_check_busy(void)
{
  DDRC = 0x00; //Alla bitar på port c IN
  PORTB = PORTB | 0x04; // (R/W)=1, PB2=1
  PORTB = PORTB & 0xF7; // RS=0, PB3=0
  PORTB = PORTB | 0x02; // Enable LCD PB1=1
  while ((PORTC & 0x80)==0x80); //Väntar tills busy-flaggan är
noll
  PORTB = PORTB & 0xFD; // Disable LCD PB1=0
}
```

```
/*void disp_w_data_l(char CGData)

{

  DDRC = 0xFF; //Alla bitar på port c OUT

  PORTB = PORTB & 0xEF; // Disable DTMF PB4=0

  PORTB = PORTB & 0xF3; // (R/W)=0 RS=0 PB2=0, PB3=0

  PORTB = PORTB | 0x02; // Enable LCD PB1=1

  PORTC = 0x14; // Data=00011100 Cursor or display shift, cursor
move, shift to the right

  PORTB = PORTB & 0xFB; // (R/W)=0, PB2=0

  PORTB = PORTB | 0x08; // RS=0, PB3=1

  PORTC = CGData;

  PORTB = PORTB & 0xFD; // Disable LCD PB1=0

} */


char temp_to_char1(int temp)

{

  char temp_char1[1];

  if (temp>=0)

    temp_char1[0] = ' ';

  else

    temp_char1[0] = '-';

  return temp_char1[0];

}


char temp_to_char2(int temp)

{

  char temp_char2[1];
```

```
  if (((temp>=0) & (temp<=9)) | ((temp<0) & (temp>=-9)))
  {
    temp_char2[0] = ' ';
  }
  else if (((temp>=10) & (temp<=19)) | ((temp<=-10) & (temp>=-
19)))
  {
    temp_char2[0] = '1';
  }
  else if (((temp>=20) & (temp<=29)) | ((temp<=-20) & (temp>=-
29)))
  {
    temp_char2[0] = '2';
  }
  else if (((temp>=30) & (temp<=39)) | ((temp<=-30) & (temp>=-
39)) )
  {
    temp_char2[0] = '3';
  }
  return temp_char2[0];
}


char temp_to_char3(int temp)
{
  char temp_char3[1];
  if ((temp>=0) & (temp<=9))
  {
    temp_char3[0] = temp + 48;
```

```
}
else if ((temp>=10) & (temp<=19))
{
  temp = temp - 10;
  temp_char3[0] = temp + 48;
}
else if ((temp>=20) & (temp<=29))
{
  temp = temp - 20;
  temp_char3[0] = temp + 48;
}
else if ((temp>=30) & (temp<=39))
{
  temp = temp - 30;
  temp_char3[0] = temp + 48;
}
else if ((temp<=-30) & (temp>=-39))
{
  temp = -1*(temp + 30);
  temp_char3[0] = temp + 48;
}
else if ((temp<=-20) & (temp>=-29))
{
  temp = -1*(temp + 20);
  temp_char3[0] = temp + 48;
}
```

```
  else if ((temp<=-10) & (temp>=-19))
  {
    temp = -1*(temp + 10);
    temp_char3[0] = temp + 48;
  }
  else if ((temp<0) & (temp>=-9))
  {
    temp = -1*(temp);
    temp_char3[0] = temp + 48;
  }
  return temp_char3[0];
}



/*
*-----------------*
*DTMF-hantering   *
*-----------------*
*/


#include "dtmf.h"
#include "IO-reg.h" //ÄNDRA TILL IO-regold.h INNAN
DRIFTSÄTTNING!!!
#include "Vectors.h"
#include "intr6811.h"
```

```
void init_dtmf(void);
void pulse_dtmf(void);
void swreset_dtmf(void);
void enable_dtmf(void);
void disable_dtmf(void);
void rs0_hi(void);
void rs0_lo(void);
void read_enable_dtmf(void);
void write_enable_dtmf(void);
void write_bus_dtmf(char data_out_dtmf);
void send_digit_dtmf(char digit);
void send_digit_string(char digit_string[], char alarm_string[]);
void off_hook(void);
void on_hook(void);
void init_send_dtmf(void);
char read_bus_dtmf(void);

void init_send_dtmf(void)
{
  enable_dtmf();
  write_enable_dtmf(); //Startar skrivningen till regA
  rs0_hi();
  write_bus_dtmf(0x0B); //Sätter regA till 1011
  pulse_dtmf();
  disable_dtmf(); //Avslutar skrivningen till regA
  enable_dtmf(); //Startar skrivningen till regB
```

```
  write_bus_dtmf(0x00); //Sätter regB till 0000
  pulse_dtmf();
  read_enable_dtmf();
  rs0_lo();
  disable_dtmf(); //Avslutar skrivningen till regB
}

void send_digit_string(char digit_string[], char alarm_string[])
{
  char a,b,c,d,lena;
  int lars;
  a = 0;
  c = 0;
  while (digit_string[a] != '\0')
    a ++;
  while (alarm_string[c] != '\0')
    c++;
  off_hook();
  for (lars = 0; lars<=30000 ; lars++); //Väntar
  lars = 0;
  for (b = 0 ; b<=(a-1) ; b++)
    send_digit_dtmf(digit_string[b]); //Skickar siffersträngen
  for (lena = 0; lena<=2 ; lena++)
  {
    for (lars = 0; lars<=30000 ; lars++); //Väntar
    lars = 0;
```

```
  }
  for (d = 0; d<=(c-1) ; d++)
    send_digit_dtmf(alarm_string[d]); //Skickar larmsträngen
  lars = 0;
  for (lars = 0; lars<=30000 ; lars++); //Väntar
  lars = 0;
  for (lars = 0; lars<=30000 ; lars++); //Väntar
  lars = 0;
  on_hook();
}

void send_digit_dtmf(char digit)
{
  char stat_dtmf;
  switch (digit)
  {
    case '1' : digit = 0x01; break;
    case '2' : digit = 0x02; break;
    case '3' : digit = 0x03; break;
    case '4' : digit = 0x04; break;
    case '5' : digit = 0x05; break;
    case '6' : digit = 0x06; break;
    case '7' : digit = 0x07; break;
    case '8' : digit = 0x08; break;
    case '9' : digit = 0x09; break;
    case '0' : digit = 0x0A; break;
```

```
    case '*' : digit = 0x0B; break;
    case '#' : digit = 0x0C; break;
    case 'A' : digit = 0x0D; break;
    case 'B' : digit = 0x0E; break;
    case 'C' : digit = 0x0F; break;
    case 'D' : digit = 0x00; break;
  }
  enable_dtmf(); //Startar skrivning till TRANSMITreg
  rs0_lo();
  write_enable_dtmf();
  write_bus_dtmf(digit); //Lägger ut AVKODAD SIFFRA på bussen
  pulse_dtmf();
  disable_dtmf(); //Avslutar skrivningen till TRANSMITreg
  enable_dtmf(); //Påbörjar pollning av STATUSreg
  rs0_hi();
  read_enable_dtmf();
  stat_dtmf = read_bus_dtmf(); //Avslutar pollning STATUSreg
  disable_dtmf(); //Avslutar pollning STATUSreg
  stat_dtmf = stat_dtmf & 0x02; //Maskar bort bit 3 och noll
  while (stat_dtmf != 0x02) //Om pollningen ej visar REDO FÖR NY
SIFFRA, polla igen!
  {
    enable_dtmf(); //Påbörjar pollning av STATUSreg igen
    rs0_hi();
    read_enable_dtmf();
    stat_dtmf = read_bus_dtmf();
    disable_dtmf(); //Avslutar pollning STATUSreg
```

```
    stat_dtmf = stat_dtmf & 0x02; //Maskar bort bit 3
  }
}


void swreset_dtmf(void) //SOFTWARE RESET
{
  char temp;
  enable_dtmf(); //Start första steget mjukvarureset, läs
statusreg
  rs0_hi();
  read_enable_dtmf();
  temp = read_bus_dtmf();
  write_enable_dtmf();
  disable_dtmf(); //Slut första steget
  enable_dtmf(); //Start andra steget, skriv kontrollreg
  write_bus_dtmf(0x00);
  pulse_dtmf();
  disable_dtmf(); //Slut andra steget
  enable_dtmf(); //Början tredje steget, skriv kontrollreg
  write_bus_dtmf(0x00);
  pulse_dtmf();
  disable_dtmf(); //Slut tredje steget
  enable_dtmf(); //Början fjärde steget, skriv kontrollreg
  write_bus_dtmf(0x08);
  pulse_dtmf();
  disable_dtmf(); //Slut fjärde steget
  enable_dtmf(); //Början femte steget, skriv kontrollreg
```

```
  write_bus_dtmf(0x00);
  pulse_dtmf();
  read_enable_dtmf();
  disable_dtmf(); //Slut femte steget.
  enable_dtmf(); //Start sjätte steget, läs statusreg
  rs0_hi();
  read_enable_dtmf();
  temp = read_bus_dtmf();
  rs0_lo();
  write_enable_dtmf();
  disable_dtmf(); //Slut sjätte steget
}


void init_dtmf(void)
{
  PORTB = PORTB & 0x7D; //Nollställer pinne 1 och 7, disablar LCD
sätter FI2 låg
  enable_dtmf();
}


void disable_dtmf(void)
{
  PORTB = PORTB | 0x10; //Ettställer pinne 4, disablar DTMF
[CS(inv)]
}


void enable_dtmf(void)
```

```
{
  PORTB = PORTB & 0xEF; //Nollställer pinne 4, enablar DTMF
[CS(inv)]
}

void read_enable_dtmf(void)
{
  PORTB = PORTB | 0x04; //Sätter pinne 2 PORTB hög
}

void write_enable_dtmf(void)
{
  PORTB = PORTB & 0xFB; //Sätter pinne 2 PORTB låg
}

void rs0_hi(void)
{
  PORTB = PORTB | 0x08; //Sätter pinne 3 PORTB hög
}

void rs0_lo(void)
{
  PORTB = PORTB & 0xF7; //Sätter pinne 3 PORTB låg
}

char read_bus_dtmf(void)
{
```

```
  char data_in_dtmf;
  DDRC= 0x00;//Sätter PORTC till att vara inport
  PORTB = PORTB | 0x80; //Sätter pinne 7, PORTB, hög
  data_in_dtmf = PORTC; //Hämtar datan på PORTC
  PORTB = PORTB & 0x7F; //Sätter pinne 7 låg igen
  data_in_dtmf = data_in_dtmf & 0x0F; //Plockar ut bit 3-0
  return data_in_dtmf;
}

void write_bus_dtmf(char data_out_dtmf)
{
  DDRC = 0xFF;//Sätter PORTC till att vara utport
  PORTC = data_out_dtmf; //Lägger ut data på PORTC
}

void pulse_dtmf(void)
{
  PORTB = PORTB | 0x80; //Sätter pinne 7, PORT B, hög
  PORTB = PORTB & 0x7F; //Sätter pinne 7 låg igen
}

void off_hook(void)
{
  PORTB = PORTB | 0x40; //Sätter pinne 6, PORTB, hög. Drar relät
}
```

```
void on_hook(void)
{
  PORTB = PORTB & 0xBF; //Sätter pinne 6, PORTB, låt. Stänger
relät
}


/*
Minneskontroll.
*/


#include "IO-reg.h"
#include "Vectors.h"
#include "intr6811.h"

void memory_init(void);
void memory_write(int write_adr, int write_data);
void write_memstat(char istr);
int memory_read(int read_adr);
int read_memstat(void);

void memory_init(void)
{
  DDRD = DDRD | 0x38; //MOSI och SCK sätts till utg
  SPCR = 0x5C; //Sätter kontrollregistret till 01011100
}

void memory_write(int write_adr, int write_data)
```

```
{
  int mem_stat, adr_hi, adr_lo, laban; //Variabeldeklaration
  mem_stat = SPSR;
  laban = SPDR;
  if (SPSR == 0x0) //Kollar att Processorns statusregister är
noll
  {
    PORTB = PORTB | 0x20; //Säkerställer hög CS(inv)
    PORTB = PORTB & 0xDF; //Sätter CS(inv) låg
    SPDR = 0x05; //FRÅGA EFTER MINNETS STATUSREGISTER
    while (SPSR==0x0) {} //Vänta på SPIF
    SPDR = 0xFF; //Slask in i DataBuffer för att generera
klockpulser
    while (SPSR==0x0) {} //Vänta på SPIF
    PORTB = PORTB | 0x20; // Sätter CS(inv) hög
    mem_stat = SPDR;
    mem_stat = mem_stat & 0x81; //Nollar allt utom bit 7 och 0
    while (mem_stat!=0x0)
    {
      PORTB = PORTB & 0xDF; //Sätter CS(inv) låg
      SPDR = 0x05; //FRÅGA EFTER MINNETS STATUSREG
      while (SPSR==0x0) {} //Vänta på SPIF
      SPDR = 0xFF; //Slask in i DataBuffer för att generera
klockpulser
      while (SPSR==0x0); //Väntar på SPIF
      PORTB = PORTB | 0x20; //CS(inv) sätts hög.
      mem_stat = SPDR;
      mem_stat = mem_stat & 0x81; //Nollar allt utom bit 7 och 0
```

```
    }
    PORTB = PORTB & 0xDF; //Sätter CS(inv) låg
    SPDR = 0x6; //Lägger ut en WREN till minnet
    while (SPSR==0x0); //Vänta på SPIF
    PORTB = PORTB | 0x20; // Sätter CS(inv) hög
    adr_hi = write_adr >>8; //Plockar ut de höga 8 bitarna ur adr
    adr_lo = write_adr & 0xFF; //Plockar ut de låga 8 bitarna ur
adr
    PORTB = PORTB & 0xDF; //Sätter CS(inv) låg
    SPDR = 0x02; //Skickar en skrivförfågan till minnet
    while (SPSR==0x0); //Vänta på SPIF
    SPDR = adr_hi;
    while (SPSR==0x0); //Vänta på SPIF
    SPDR = adr_lo;
    while (SPSR==0x0); //Vänta på SPIF
    SPDR = write_data;
    while (SPSR==0x0); //Vänta på SPIF
    PORTB = PORTB | 0x20; //Sätter CS(inv) hög igen
    laban = SPSR;
    laban = SPDR;
  }
}

int memory_read(int read_adr)
{
  int adr_hi, adr_lo, read_data, mem_stat; //Variabeldeklaration
  mem_stat = SPSR;
```

```
  mem_stat = SPDR;

  if (SPSR == 0x0) //Kollar att Processorns statusregister är
noll

  {

    PORTB = PORTB | 0x20; //Säkerställer hög CS(inv)

    PORTB = PORTB & 0xDF; //Sätter CS(inv) låg

    SPDR = 0x05; //FRÅGA EFTER MINNETS STATUSREG

    while (SPSR==0x0) {} //Vänta på SPIF

    SPDR = 0xFF; //Slask in i DataBuffer för att generera
klockpulser

    while (SPSR == 0x0); //Vänta på SPIF

    PORTB = PORTB | 0x20; // HÄR SKALL CS(inv) GÅ HÖG!!!

    mem_stat = SPDR;

    mem_stat = mem_stat & 0x81; //Nollar allt utom bit 7 och 0

    while (mem_stat!=0x0)

    {

      PORTB = PORTB & 0xDF; //Sätter CS(inv) låg

      SPDR = 0x05; //FRÅGA EFTER MINNETS STATUSREG

      while (SPSR==0x0) {} //Vänta på SPIF

      SPDR = 0xFF; //Slask in i DataBuffer för att generera
klockpulser

      while (SPSR == 0x0); //Vänta på SPIF

      PORTB = PORTB | 0x20; //CS(inv) sätts hög.

      mem_stat = SPDR;

      mem_stat = mem_stat & 0x81; //Nollar allt utom bit 7 och 0

    }

    adr_hi = read_adr >>8; //Plockar ut de höga 8 bitarna ur adr
```

```
    adr_lo = read_adr & 0xFF; //Plockar ut de låga 8 bitarna ur
adr
    PORTB = PORTB & 0xDF; //Sätter CS(inv) låg
    SPDR = 0x03; //Skickar Read Request
    while (SPSR == 0x0); //Vänta på SPIF
    SPDR = adr_hi;
    while (SPSR==0x0); //Vänta på SPIF
    SPDR = adr_lo;
    while (SPSR==0x0); //Vänta på SPIF
    SPDR = 0xFF; //Slask i databuffern för att få klockor
    while (SPSR==0x0); //Väntar på SPIF
    read_data = SPDR;
    PORTB = PORTB | 0x20; //Sätter CS(inv) hög igen
    read_data = SPDR; //Tilldelar variabeln DATA värdet av SPDR
  }
  return read_data;
}

void write_memstat(char instr)
{
  PORTB = PORTB | 0x20; //Säkerställer hög CS(inv)
  PORTB = PORTB & 0xDF; //Sätter CS(inv) låg
  SPDR = 0x06; //Sätter WEL på minnet
  while (SPSR==0x0); //Vänta på SPIF
  PORTB = PORTB | 0x20; //Sätter hög CS(inv)
  PORTB = PORTB & 0xDF; //Sätter låg CS(inv)
  SPDR = 0x01; //Skicka kommando skriv statusreg
```

```
  while (SPSR==0x0) {} //Vänta på SPIF
  SPDR = instr;
  while (SPSR==0x0); //Vänta på SPIF
  PORTB = PORTB | 0x20; //Sätter CS(inv) hög
}

int read_memstat(void)
{
  int mem_stat;
  PORTB = PORTB | 0x20; //Säkerställer hög CS(inv)
  PORTB = PORTB & 0xDF; //Sätter CS(inv) låg
  SPDR = 0x05; //FRÅGA EFTER MINNETS STATUSREG
  while (SPSR==0x0) {} //Vänta på SPIF
  SPDR = 0xFF; //Slask in i DataBuffer för att generera
klockpulser
  while (SPSR == 0x0); //Vänta på SPIF
  PORTB = PORTB | 0x20; // HÄR SKALL CS(inv) GÅ HÖG!!!
  mem_stat = SPDR;
  return mem_stat;
}
```

//Fungerande klocka

//All denna kod fungerar, men endast vissa delar används på
//grund av minnsesbrist i processorn!!

```
#include "IO-reg.h"
```

```
#include "Vectors.h"
#include "intr6811.h"
#include "disp.h"

//unsigned long Date2ULong(void);
//void Date2String(unsigned long l);

//unsigned long ticks;
unsigned long trenoll_tick;
//unsigned long StartTime;
//unsigned long StopTime;
//char DateString[15];
//char year=1;
//char month=2;
//char day=21;
//char hour=22;
//char min=47;
//char sec=0;
char trenoll_cnt=0;
char alarm=0;
char alarm_delay=0;
//static char months[12] ={31,28,31,30,31,30,31,31,30,31,30,31};

/*void clock(void)
{
  Date2String(Date2ULong());
```

```
}*/


void init_clock(void)
{
    TMSK1 = TMSK1 | 0x80; // Enable OutputCompare1 interrupt
}


interrupt void OC1_interrupt(void){
  TFLG1 = TFLG1 | 0x80; // Ack
  TOC1+=40000;
// Klocka
//  ticks++;
  if ((armed_delay == 1) & (armed == 0)) //HÄR BÖRJAR KODSNUTT
FÖR FÖRDRÖJNING
  {
    trenoll_tick++;
    if (trenoll_tick > 49)
    {
      trenoll_tick = 0;
      trenoll_cnt++;
      if (trenoll_cnt > 29)
      {
        armed = 1;
        //armed_delay = 0;
        trenoll_cnt=0;
      }
    }
```

```
  }
  else if ((armed_delay == 1) & (armed == 1) & (alarm_delay ==
1))
  {
  trenoll_tick++;
    if (trenoll_tick > 49)
    {
      trenoll_tick = 0;
      trenoll_cnt++;
      if (trenoll_cnt > 29)
      {
        alarm = 1;
        //armed_delay = 0;
        trenoll_cnt=0;
      }
    }
  }
  else if ((armed_delay == 0) & (armed == 1))
  {
    trenoll_tick++;
    if (trenoll_tick > 49)
    {
      trenoll_tick = 0;
      trenoll_cnt++;
      if (trenoll_cnt > 29)
      {
        armed = 0;
```

```
        //armed_delay = 0;
        trenoll_cnt=0;
      }
    }
  }
  else
  {}  // HÄR SLUTAR KOD FÖR FÖRDRÖJNING
/*  if (ticks > 49){
    ticks=0;
    sec++;
    if (sec > 59){
      sec=0;
      min++;
        if (min>59){
        min=0;
        hour++;
        if (hour>23){
          hour=0;
          day++;
          if (day > months[month-1]) {
            if (month != 2){
              day=1;
              month++;
            }
            else{  //feb
              if ((year % 4) == 0){
```

```
                if (day>29){
                  day=1;
                  month++;
                }
              }
              else{
                day=1;
                month++;
              }
            } //else feb
          } //day
          if (month>12){
           month=1;
           year++;
          }  //month
        } //hour
      } //min
    } //sec
  }  //ticks*/
} //OC1

/*unsigned long Date2ULong(void) {
unsigned long l;
  l = year;
  l=l<<4; l=l+month;
  l=l<<5; l=l+day;
```

```
  l=l<<5; l=l+hour;
  l=l<<6; l=l+min;
  l=l<<6; l=l+sec;
  return l;
}

void Date2String(unsigned long l) {
char i;
  DateString[1] = (l>>22)&15;  // Månader
  DateString[4] = (l>>17)&31;  // Dagar
  DateString[7] = (l>>12)&31;  // Timmar
  DateString[10] = (l>>6)&63;  // Minuter
  DateString[13] = l&63;       // Sekunder
  for (i=1;i<14;i+=3) {
    DateString[i-1] = 48+DateString[i]/10;
    DateString[i] = 48+DateString[i]%10;
  }
  DateString[2] = '-';
  DateString[5] = ' ';
  DateString[8] = ':';
  DateString[11] = ':';
  DateString[14] = 0x00;
}*/
```