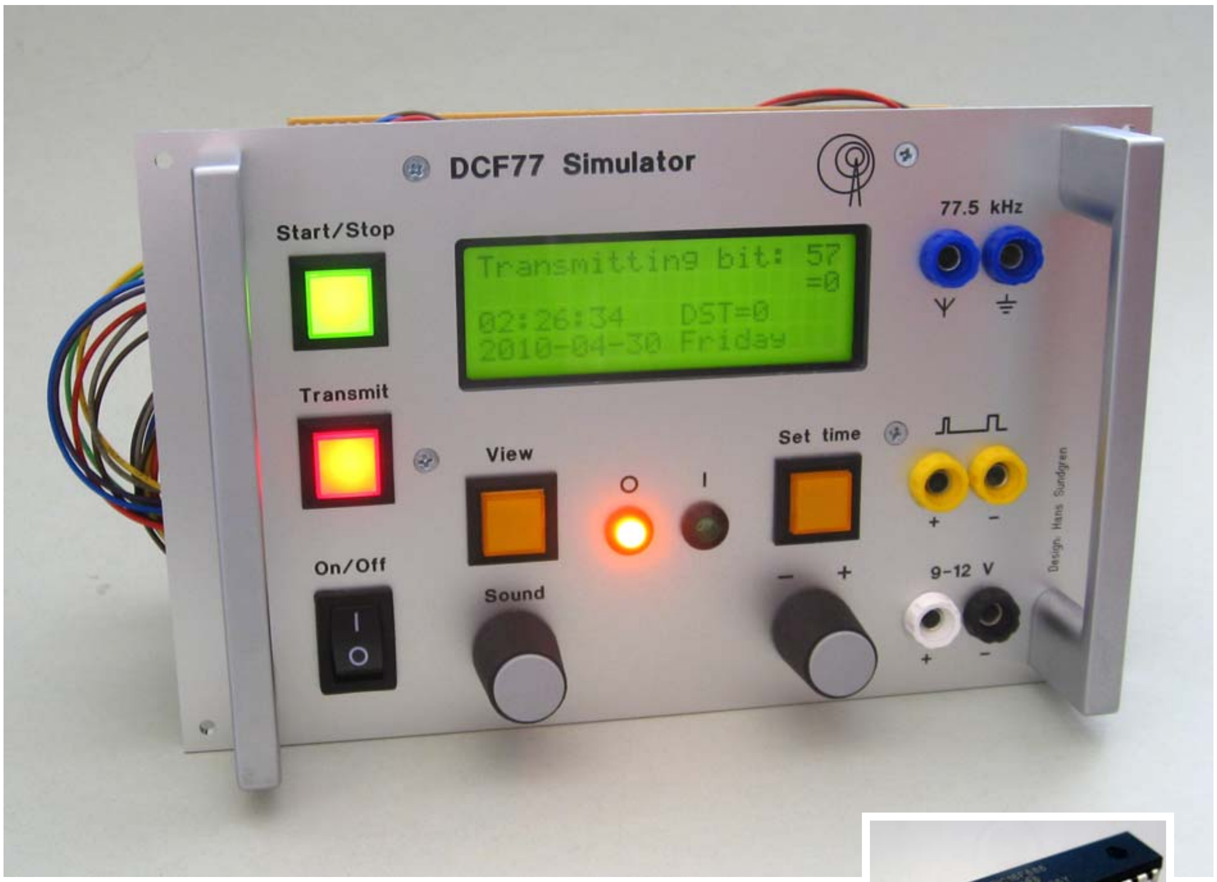
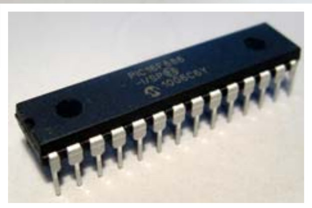
DCF77-simulator

– med display och ljud





Rapport	
Kurs	KTH Tillämpad digitalteknik, kurs IL131V
Lärare	William Sandqvist
Elev	Hans Sundgren
Datum	2010-05-03

Innehåll

1 Allmänt

Detta är en beskrivning av projektuppgiften som Hans Sundgren utförde under vårterminen 2010, som en del av kursen *Tilllämpad digitalteknik* på KTH.

1.1 DCF77 och simulatorn

DCF77 är den tyska långvågssändaren som ger rätt tid till radiostyrda klockor i norra Europa. Denna *DCF77-simulator* skapar ett pulståg identiskt med DCF77 men med en valbar tid, datum och andra parametrar, dvs en falsk tid.

Simulatorn har flera visningslägen för en pedagogisk demonstration av hur DCF77-signalen ser ut. Simulatorn är därmed även en ”demonstrator” som:

- Visar utsända tidparametrar, dvs hur protokollet är uppbyggt.
- Visar hur tidparametrarna representeras av binär BCD-kod.
- Demonstrerar att man kan ställa om radiostyrda klockor, på kort distans, åtminstone med den modulation och det protokoll som används för tillfället.

Simulatorn har två gränssnitt för att styra radiostyrda klockor:

- Elektriskt, 2-tråds
- Radiosignal 77,5 kHz, lågeffekt med någon meters räckvidd.

1.2 Utsignalen

Simulatorn skapar en DCF77 utsignal enligt inställda värden och sändningsprotokoll, se Referensinformation, Protokoll DCF77.

- Logisk ”0” = 100 ms inverterad puls varje sekundstart
- Logisk ”1” = 200 ms inverterad puls varje sekundstart
- Sekund 59: Ingen puls

Den riktiga DCF77-sändaren i Tyskland använder en AM-modulerad signal där normal signal är 100% effekt som minskar till 75% under sekundpulsen.

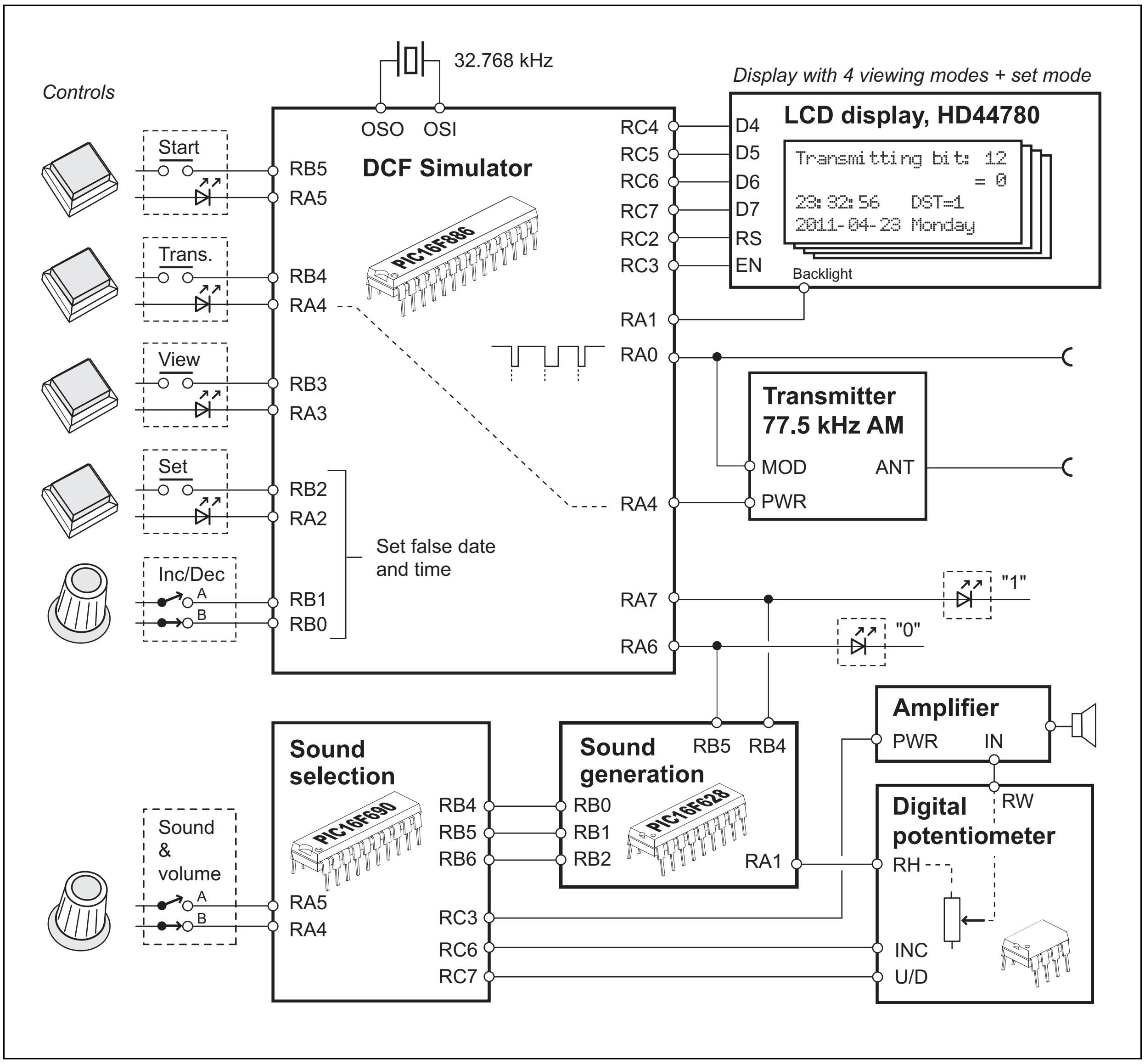
2 Design

2.1 Blockschema

Simulatorn är uppbyggd av en huvudprocessor med två stödprocessorer för att alstra en ljudbild som komplement till textvisning på display. Stödprocessorerna tar ingen aktiv del i själva DCF77-simuleringen.

Förutom de 3 PIC-processorerna och LCD-display ingår följande enheter:

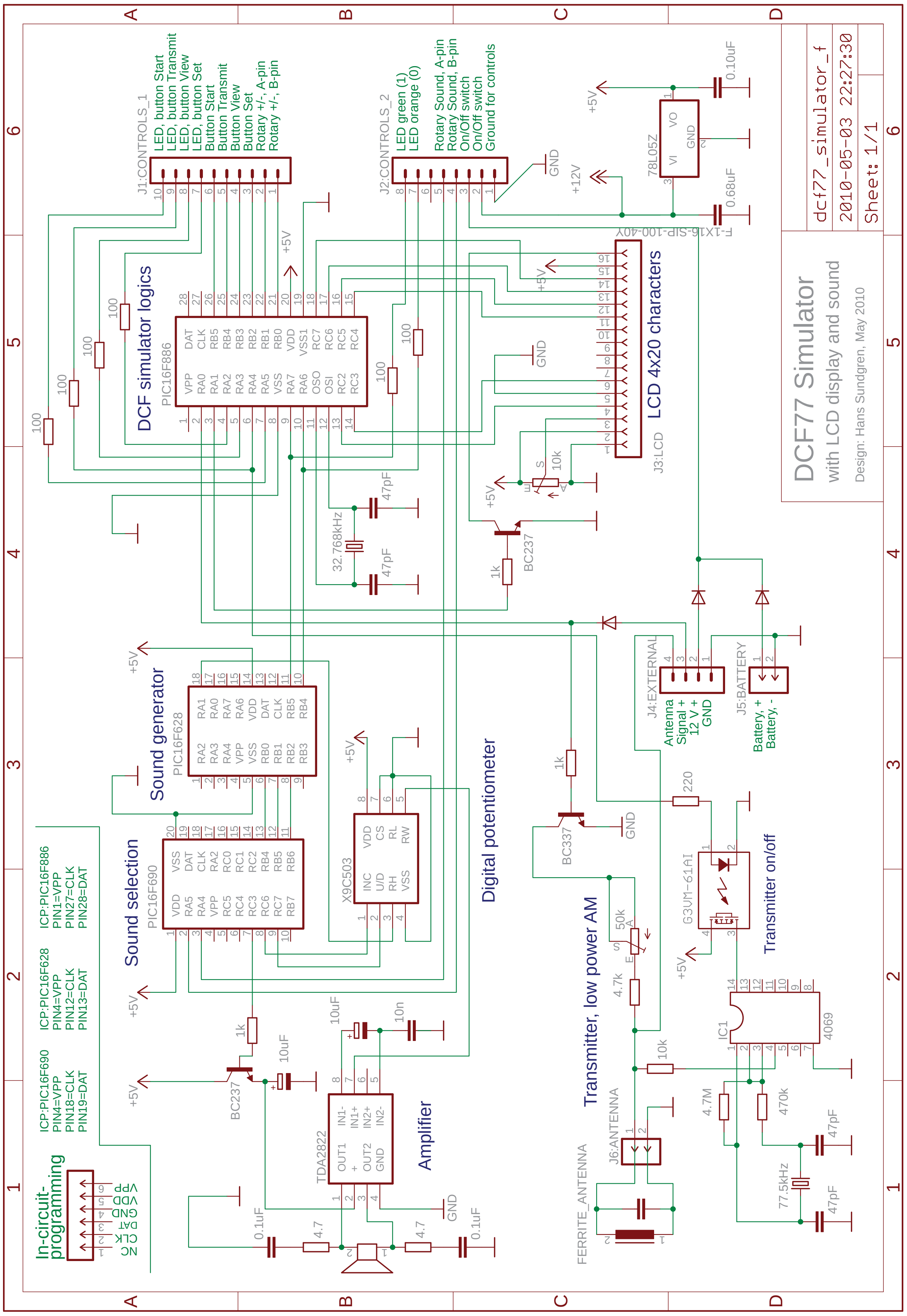
- Digital potentiometer
- Förstärkare
- Lågeffekts-sändare för 77,5 kHz



Blockschema

2.2 Kretsschema

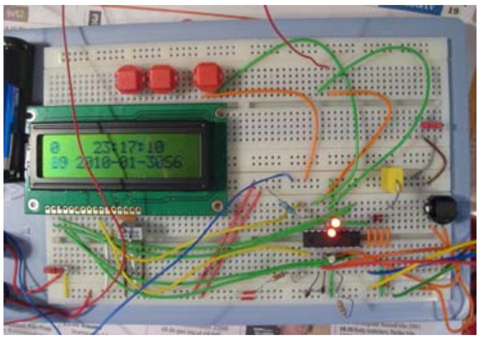
Schemat är ritat i Eagle 5.8.0.



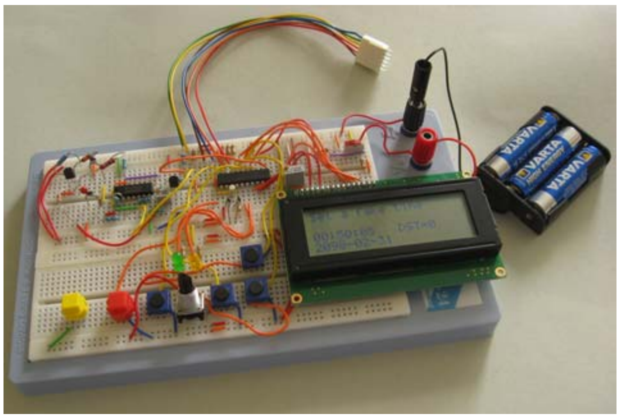
2.3 Uppbyggnad

2.3.1 Utvecklingsversioner

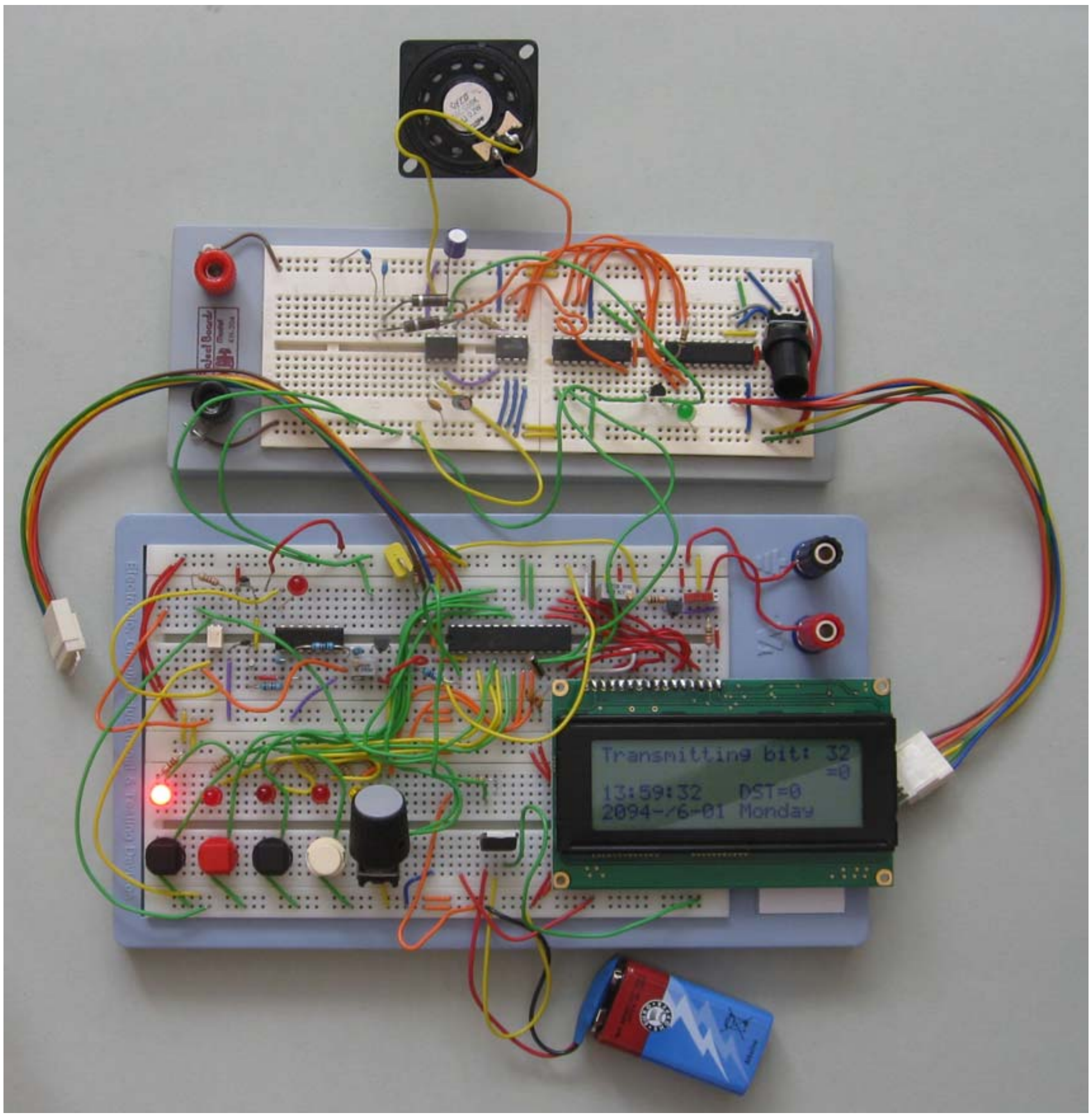
För utveckling, laboration och test av program byggdes simulatorn på kopplingsdäck.



Version A med PIC16F628.



Version B med PIC16F690.

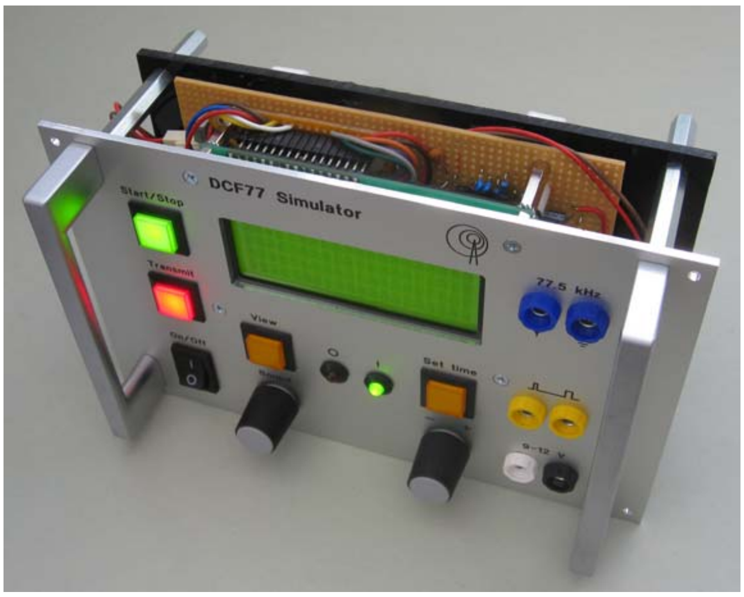


Slutversion C med PIC16F886 och stödprocessorer 628 och 690.

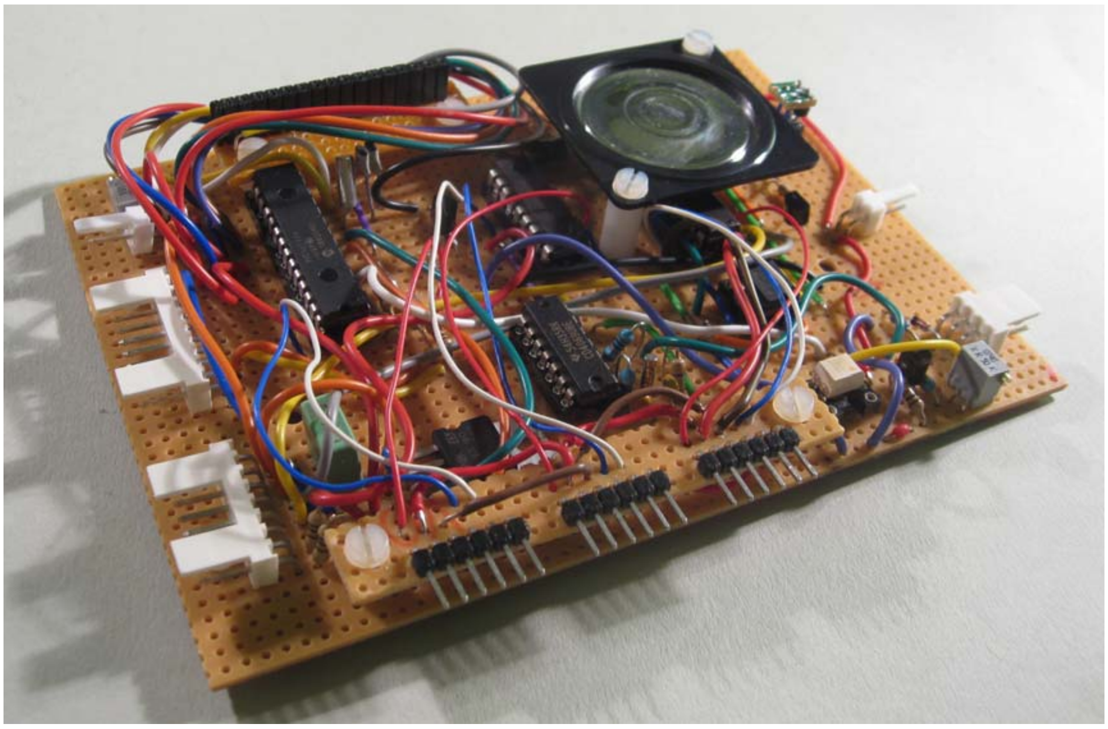
2.3.2 Funktionsprototyp

Funktionsprototypen byggdes upp med elektroniken på ett kretskort med banor (Veroboard). Enheten hänger ihop med frontpanelen i en sandwich-konstruktion:

- Nivå 1: Frontpanel med manöverdon och LCD-display
- Nivå 2: Kretskort med högtalare och stiftanslutningar mot omgivning
- Nivå 3: Batterier och ferritantenn monterade på plastskiva
- Nivå 4: Plastskiva som beröringsskydd



Sandwich-uppbyggnad med stöd av frontpanel.



Kretskortet med stiftlister för PICkit2-programmeraren, en list per processor.

2.4 Användarinterface

2.4.1 LCD-display

Displayen har flera visningslägen enligt nedan.

Startsida vid spänningspåslag

```
Set time and START

23:32:56   DST=1
2011-04-23 Monday
```

Ställa den interna klockan när Set-knappen trycks
Varje enhet markeras och med +/-ratten stegas värdet upp/ner. Nästa enhet markeras vid nästa tryck på SET-knappen.
Enheter: hour, minutes, seconds, dst, year, month, day, weekday.

```
Set a fake time

23:32:56   DST=1
2011-04-23 Monday
```

Normal sändning
Inställd tid enligt den interna klockan stegas upp 1 gång per sekund. Löpnummer på sänd bit samt dess värde stegas upp 1 gång per sekund.

```
Transmitting bit: 12
                  =0
23:32:56   DST=1
2011-04-23 Monday
```

Bitöversikt 60 sekunder
Samtliga 60 bitars värden visas i en översikt. Den bit som sänds för tillfället markeras med en understrykning. Bitschemat uppdateras sekund 0 varje minut.

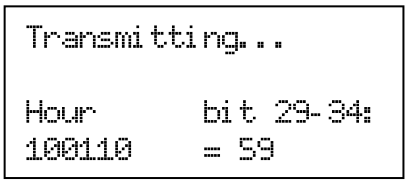
```
00-: 101111110001111
15-: 100110101001011
30-: 100110101001011
45-: 10011010100101-
```

Beskrivning av aktuell sändning
Vid varje given tidpunkt under en minut visas vilka tid-parametrar som sänds.

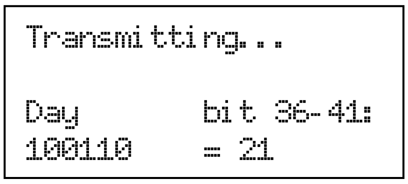
```
Transmitting...

Weather   bit 01-14:
10011010011010
```

```
Transmitting...

DST+      bit 16-19:
1001 = DST inactive
```

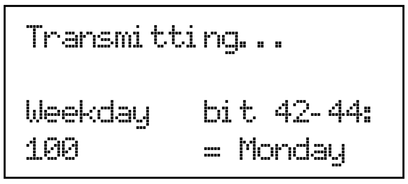
```
Transmitting...

Minute    bit 21-27
1001101   = 59
```



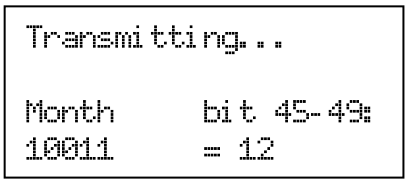
```
Transmitting...

Hour      bit 29-34:
100110    = 59
```



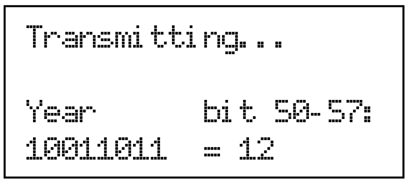
```
Transmitting...

Day       bit 36-41:
100110    = 21
```



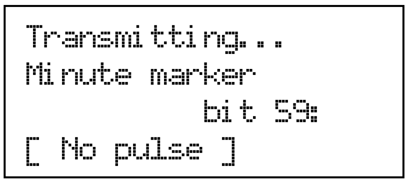
```
Transmitting...

Weekday   bit 42-44:
100       = Monday
```



```
Transmitting...

Month     bit 45-49:
10011     = 12
```



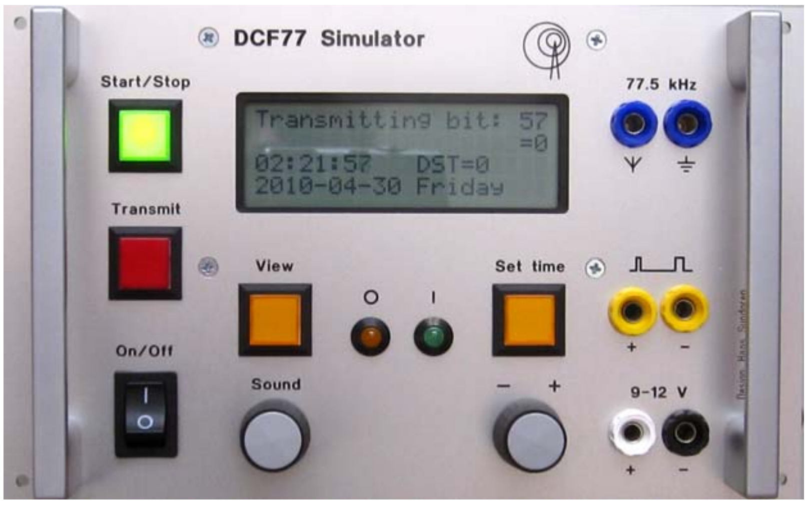
```
Transmitting...

Year      bit 50-57:
10011011  = 12
```



```
Transmitting...
Minute marker
          bit 59:
[ No pulse ]
```

2.4.2 Manövrering

Simulatorn manövreras med tryckknappar och pulsgivare.



- **Start/Stop** tryckknapp
  Start/stopp av den interna ”falsktidsklockan” och utsignalen.
- **Transmit** tryckknapp
  Start/stopp av sändaren för 77.5 kHz.
- **View** tryckknapp
  Stegar igenom de 4 visningslägena.
- **Set time** tryckknapp
  Ställa falsk tid. Efter genomstegning av alla enheter sparas de i EEPROM.
- **Sound** pulsgivare
  Stegar igenom 4 ljudmoder och reglerar ljudvolymen.
- + / - pulsgivare
  Ökar/minskar aktuellt tidvärde i samband med att ställa klockan.

Function	**Unit**	**Mode**			
		Stopped	**Started**	**Set time**	**Set time ready**
Start	**Button**	Active	Active	Blocked	Active
	LED	Flashing	On	Off	Flashing
Transmit	**Button**	Blocked	Active	Blocked	Blocked
	LED	Off	Off/On	Off	Off
View	**Button**	Active	Active	Blocked	Active
	LED	Flashing	Flashing	Off	Flashing
Set time	**Button**	Active	Blocked	Active	Active
	LED	Off	Off	On	Off

Tryckknappars funktion och indikering.

2.5 Hantering av inställd start-tid

Vid spänningssättning hämtas *senast lagrad* start-tid från EEPROM.

Tiden börjar räknas upp när sekvensstart startas med **Start**-knappen. När sekvensstart är stoppad kan "tidsinställningsmod" aktiveras genom att trycka på **Set time**.

Senast inställd starttid lagras i EEPROM när " tidsinställningsmod" avslutas, dvs vid sista tryckningen på **Set time**.

2.6 Ljudkomplement

För att få en tydligare indikation över sända tecken (noll eller ett) kan man välja olika ljudtyper, t ex klick-ljud som en klocka eller toner.

Vid spänningstillslag är ljud avstängt men genom att vrida ratten **Sound** kan man både volymreglera ljudet samt byta ljudtyper.

2.6.1 Volymkontroll

Ratten fungerar som en vanlig volymkontroll.

2.6.2 Byte av ljudtyp

En snabb vridning upp/ner stegar fram ljudtyp. Det finns 5 ljudtyper som stegas fram i ordningsföljd 0-1-2-3-4-0-1-2 osv.

Ljudtyp	**Beskrivning**
0	Inget ljud (förstärkaren avstängd)
1	Klickljud, två toner
2	Pipljud, två toner
3	Inverterad hög ton som matchar sänd bärvåg
4	Inverterad låg ton som matchar sänd bärvåg

2.7 Programvara

2.7.1 Programöversikt

Programmen är skrivna i C och kompilerade med B. Knudsens C-kompilator **Cc5x**. För komplett programkod, se avsnitt Programkod.

Sound generator [PIC16F628]:

```
Main ---------------------------------------------------
 |  Initialization                                        |
 |  Loop                                                  |
 |     Read mode bits and convert to decimal mode         |
 |     If mode = 1                                        |
 |        Play click sound                                |
 |     If mode = 2                                        |
 |        Play other sound                                |
 |     If mode = 3                                        |
 |        Play other sound                                |
 |     If mode = 4                                        |
 |        Play other sound                                |
 |     <can be expanded up to mode = 7>                   |
  -------------------------------------------------------
```

Sound selection [PIC16F690]:

```
  Interrupt: Rotary encoder -----------------------------
 |  Read encoder                                          |
 |     If clockwise                                       |
 |        increase = 1                                    |
 |     If counter-clockwise                               |
 |        decrease = 1                                    |
  -------------------------------------------------------

  Main -------------------------------------------------
 |  Initialization                                        |
 |  Loop                                                  |
 |     If increase                                        |
 |        if (fast)                                       |
 |           quick_up = 1                                 |
 |        else                                            |
 |           increase signal to digital potentiometer     |
 |     If decrease                                        |
 |        if (quick_up = 1)                               |
 |           if (rotation_timeout = 0)                    |
 |              increment sound mode                      |
 |        else                                            |
 |           decrease signal to digital potentiometer     |
 |     If (rotation timeout = 0)                          |
 |        if (time_since_rotation = 170 ms)               |
 |        rotation timeout = 1                            |
  -------------------------------------------------------
```

DCF Simulator [PIC16F886]:

```
 Interrupt: 1 Hz ---------------------------------------
|  If start_mode = 1                                     |
|     Update clock (step up one second)                  |
|     Output pulse-value                                 |
|  Update display                                        |
|  Check buttons                                         |
|     If VIEW-button                                     |
|        Step up display mode 0->1->2->3->0              |
|     If SET-button                                      |
|        Setting > 0                                     |
|         Stop interrupt, jump out to setting sequence   |
 --------------------------------------------------------

 Interrupt: Rotary encoder, during "set clock" ----------
|  Read encoder                                          |
|     If increase                                        |
|        up = 1 and blink LED0                           |
|     If decrease                                        |
|        down = 1 and blink LED1                         |
 --------------------------------------------------------

 Main ---------------------------------------------------
|  Initialization                                        |
|  Read stored time from EEPROM                          |
|  Interrupt is active 1 Hz, but the clock is stopped    |
|  Loop                                                  |
|     Check START-button                                 |
|        If START-button                                 |
|           Toggle start_mode                            |
 --------------------------------------------------------

 Set clock sequence  -----------------------------------
|  Disable 1 Hz interrupt                                |
|  Enable rotary encoder interrupt                       |
|  Step through set sequence, read rotary encoder        |
|     Update and display time values                     |
|  Store set time in EEPROM                              |
|  Enable 1 Hz interrupt                                 |
 --------------------------------------------------------
```

2.7.2 Några programavsnitt

Generering av pulståg, 1 gång per minut

För beräkning av rätt paritetsbitar används en modulo-funktion. För omvandling av decimalt till BCD behövs inget speciellt.

```
min1 = min1 + 1;             // Time to transmit = +1 minute
update_clock ();
bit_value[0] = 0;
bit_value[1] = 0;
bit_value[2] = 1;
bit_value[3] = 0;
bit_value[4] = 0;
bit_value[5] = 0;
bit_value[6] = 0;
bit_value[7] = 0;
bit_value[8] = 0;
bit_value[9] = 0;
bit_value[10] = 0;
bit_value[11] = 0;
bit_value[12] = 0;
bit_value[13] = 0;
bit_value[14] = 0;
bit_value[15] = 0;           // 0 = primary antenna in use
```

```
   bit_value[16] = 0;           // 0 = No DST change in the next hour
   bit_value[17] = dst;         //  DST on/off
   bit_value[18] = 1;           // Timezone, usually opposite of bit 17
   bit_value[19] = 0;           // Leap second insertion
   bit_value[20] = 1;           // Always 1
   bit_value[21] = min1.0;      // Minutes, BCD least significant first
   bit_value[22] = min1.1;
   bit_value[23] = min1.2;
   bit_value[24] = min1.3;
   bit_value[25] = min10.0;
   bit_value[26] = min10.1;
   bit_value[27] = min10.2;

   no_of_ones = 0;
   for (i=21; i<28; i++)
      {
      no_of_ones = no_of_ones+bit_value[i];
      }
   bit_value[28] = no_of_ones%2; // Even parity bit, minute bits
   bit_value[29] = hour1.0;      // Hours, BCD least significant first
   bit_value[30] = hour1.1;
   bit_value[31] = hour1.2;
   bit_value[32] = hour1.3;
   bit_value[33] = hour10.0;
   bit_value[34] = hour10.1;

   no_of_ones = 0;
   for (i=29; i<35; i++)         // Calculate number of "ones"
      {
      no_of_ones = no_of_ones + bit_value[i];
      }
   bit_value[35] = no_of_ones%2; // Even parity bit, hour bits
   bit_value[36] = day1.0;      // Day of month, BCD least significant first
   bit_value[37] = day1.1;
   bit_value[38] = day1.2;
   bit_value[39] = day1.3;
   bit_value[40] = day10.0;
   bit_value[41] = day10.1;
   bit_value[42] = weekday.0;   // Day of week, BCD least significant first,
Mon=1
   bit_value[43] = weekday.1;
   bit_value[44] = weekday.2;
   bit_value[45] = month1.0;    // Month, BCD least significant first
   bit_value[46] = month1.1;
   bit_value[47] = month1.2;
   bit_value[48] = month1.3;
   bit_value[49] = month10.0;
   bit_value[50] = year1.0;     // Year, BCD least significant first
   bit_value[51] = year1.1;
   bit_value[52] = year1.2;
   bit_value[53] = year1.3;
   bit_value[54] = year10.0;
   bit_value[55] = year10.1;
   bit_value[56] = year10.2;
   bit_value[57] = year10.3;

   no_of_ones = 0;
   for (i=36; i<58; i++)        // Calculate number of "ones"
      {
      no_of_ones = no_of_ones + bit_value[i];
      }
   bit_value[58] = no_of_ones%2; // Even parity bit, all transmitted bits
   bit_value[59] = 9;            // No transmission
```

Avkodning av pulsgivare för volym och ljudbild

För beräkning av programmeringsparametrar gjordes följande uppskattning:

- Långsam vridning av ratt = 2 klick per sekund = 500 ms
- Snabb vridning av ratt = 5 klick per sekund = 200 ms

Det teoretiska värdet 200 ms blev 170 ms efter praktiskt test.

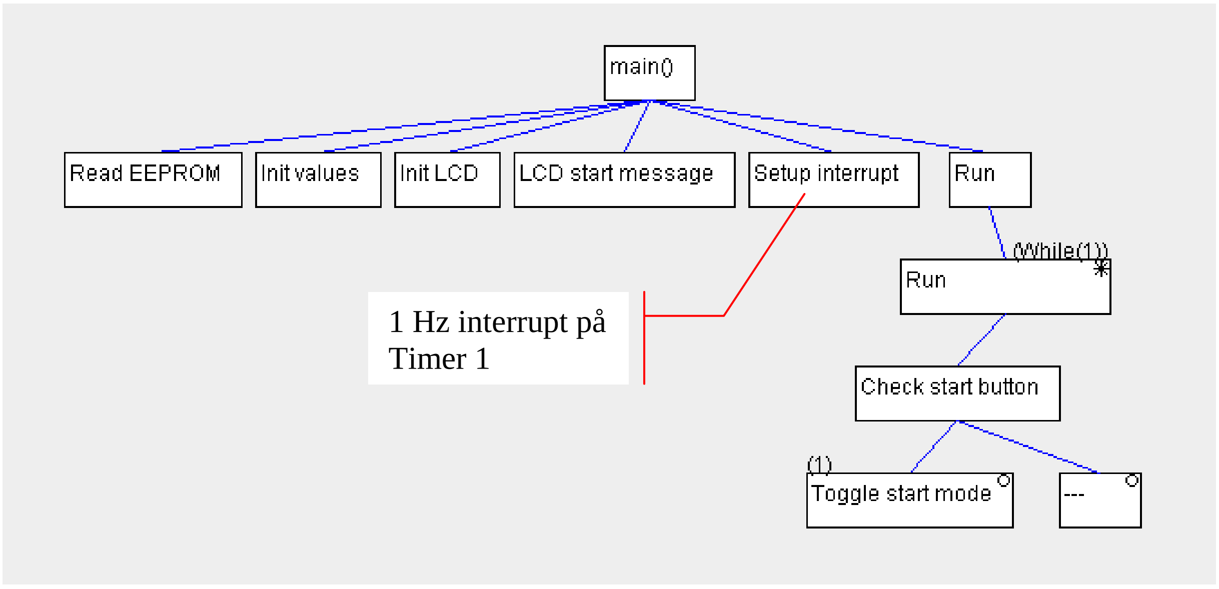
Kommunikation mellan processorer

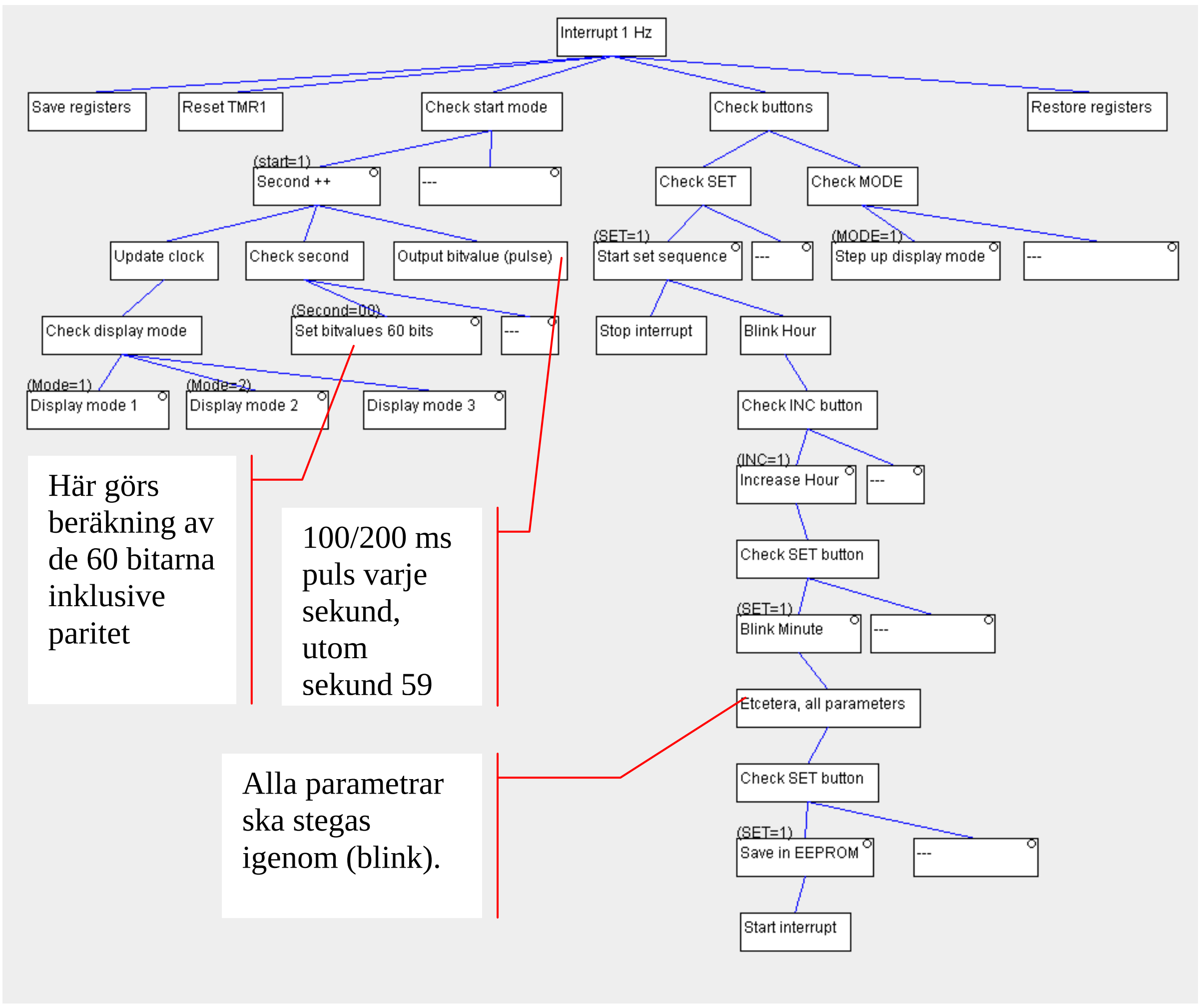
Processorerna för ljudgenerering och för avkodning av pulsgivare kommunicerar med varandra.

PIC:en för avkodning läser av pulsgivaren och stegar upp ljudtyp genom att sätta ett binärt värde på 3 utgångar. PICen för ljudalstring läser av ljudtypen genom motsvarande 3 ingångar. Med 3 I/O-linjer kan 8 ljudtyper definieras av vilka 4 används för tillfället.

2.8 Strukturdiagram JSP

Strukturdiagrammen visar en översikt över de två huvudfunktionerna; ***main*** och ***interrupt 1 Hz*** i huvudprocessorn.





3 Verifiering av funktion

3.1 Verifiering via elektriskt interface

1. Montera isär en DCF77 radiostyrd klocka.
2. Koppla bort mottagaren i klockan, t ex genom att lossa antennen.
3. Identifiera var pulserna går från radiomottagaren till den övriga klock-kretsen. Man kan mäta sig fram med en voltmeter för att söka reda på en pulsad signal med frekvensen 1 Hz.
4. Anslut en ledare till pulsingången och en till jord och anslut sedan dessa till DCF77-simulatorns signalutgång (gula uttag).
5. Starta DCF77-simulatorn.
6. Tryck **Set time** och ställ in en falsk tid. Tryck därefter **Start**.
7. Nollställ den radiostyrda klockan genom Reset-knapp eller genom att ta bort batteriet en kort stund.
8. Vänta några minuter för att den anslutna radioklockan ska avkoda signalen och synkronisera sin interna klocka.

Korrekt funktion innebär:

- Den anslutna klockan ska visa önskad falsk tid.
- Simulatorns klocka och den externa klockan ska stega upp sekunderna helt synkroniserade.
- Genom att trycka på **View** ska man kunna skifta visningslägen över avsänd data.

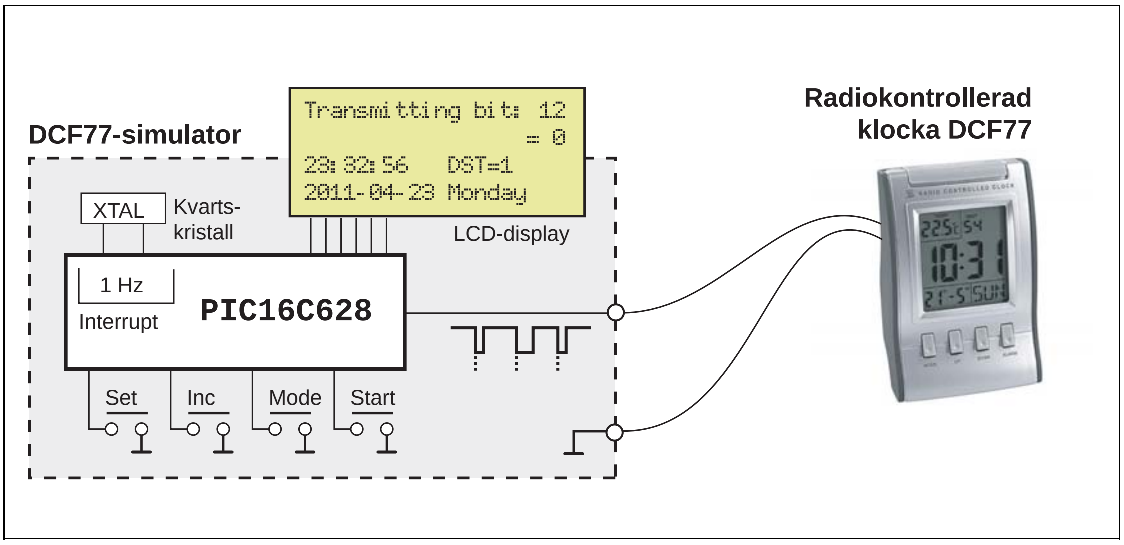
3.2 Verifiering via radio-interface

1. Placera en eller flera radiostyrda klockor runt simulatorn.
2. Starta DCF77-simulatorn
3. Tryck **Set time** och ställ in en falsk tid. Tryck därefter **Start**.
4. Starta sändaren genom att trycka **Transmit**.
5. Nollställ den radiostyrda klockan genom Reset-knapp eller genom att ta bort batteriet en kort stund.
6. Vänta några minuter för att radioklockan ska avkoda signalen och synkronisera sin interna klocka.

4 Projektets genomförande

4.1 Ursprungsplanen

Ursprungsplanen innebar en simulator med elektriskt interface och 4 styrknappar.



Blockschema över den ursprungliga simulatorn, enligt ”Begäran om programmeringsuppgift” den 7 februari

4.2 Utökning av funktion

Projektdefinitionen har under arbetets gång utökats, dels för att större PIC-processor behövdes, dels för att testa lite fler PIC-funktioner.

Processorn byttes ut 2 gånger eftersom fler I/O-pinnar krävdes för ett mer användarvänligt HMI (Human Machine Interface). För att generera de olika vyerna på LCD-displayen krävdes dessutom mer minne än det 2 k som ingår i ursprungsprocessorn PIC16F628. Slutligen infördes också ljudalstring som komplement till den visuella presentationen.

Datum	Processor	LCD-display	Öka/minska	Volym-reglering	Ljudalstring
7 feb	PIC16F628	2x16 tecken	Knappar	-	-
10 feb		4x20 tecken		-	-
21 feb	PIC16F690			-	-
14 mar			Pulsgivare	-	-
19 mar	PIC16F886			Potentiometer	Analog oscillator
18 apr				PIC16F690 + pulsgivare	PIC16F628

Projektets successiva utökning av funktion.

4.3 Slutsatser

Hans har fått följande insikter under projektets genomförande:

- Att visa många olika saker på en LCD-display kan kräva stort programminne.
- Det behöves fler I/O-pinnar än ursprungsplanen.
- PIC-processorn ska inte belastas med för många interrupt.
- Det går bra att använda flera sammankopplade processorer. Man behöver inte belasta en processor med ”allt”. Processorer är billiga.

Test av funktion i köket.

Pappa Sture Sundgren (87 assisterar vid avstämning av ferritantenn för 77,5 kHz..

Hans lägger dit sista lödningen.

4.4 Komponentleverantörer

Följande leverantörer har använts:

Frontpanel: *Schaeffer AG*. Panelen ritades upp i *Front Panel Designer* och laddades sedan ner till leverantören i Tyskland för fräsning och gravering.

PIC-processorer: *Kjell & Company*

Elektronik: *Elfa samt Microkit*

Handtag samt fästelement: *Clas Ohlson*

5 Programkod

Simulatorn är skriven i C-kod och nedan visas hela programkoden för de tre processorerna:

- PIC16F628
- PIC16F690
- PIC16F886

Programkoden är den aktuella för demonstrationen den 4 maj.

5.1 Sound generator code PIC16F628

```
/* FILE / DATE:     tone_generator_0502.c / 2010-05-02
   DESCRIPTION:     Tone generator for DCF simulator

   MICRO-PROCESSOR: PIC16F628
   COMPILER:        B Knudsen Cc5x C-compiler - not ANSI-C

   EXTERNAL HW:     Output amplifier
                    3 pins on PIC16F690 (Sound selection)
                    2 pins on PIC16F886 (Sound trigger)

   DESIGNER:        Hans Sundgren

   RAM usage:       4 bytes (3 local), 220 bytes free
   CODE WORDS:      181 code words (8 %)


CHIP CONNECTIONS ********************************************
                      ________ _______
                     |        \/       |
                     |    PIC16F628    |
                  ---|RA2  1    18  RA1|->- Sound output
                  ---|RA3  2    17  RA0|---
                  ---|RA4  3    16  RA7|---
                  ---|RA5  4    15  RA6|---
              GND ---|Vss  5    14  Vdd|--- +5V
     Sound mode.1 ->-|RB0  6    13  RB7|---
     Sound mode.2 ->-|RB1  7    12  RB6|---
     Sound mode.3 ->-|RB2  8    11  RB5|-<- Trigger.0
                  ---|RB3  9    10  RB4|-<- Trigger.1
                     |_________________|


INPUT: SOUND TRIGGER (fron PIC16F886)
 ___________________________________________________________
| Type                | Function                  | Pin     |
|---------------------|-----------------------------------|
| "Null" sound        | Trigger 0 sound (RA6)     | RB5     |
| "One" sound         | Trigger 1 sound (RA7)     | RB4     |
|_____________________|___________________________|_________|


INPUT: SOUND SELECTION (fron PIC16F690)
 ___________________________________________________________
| Type                | Function                  | Pin     |
|---------------------|-----------------------------------|
| Sound mode 1 bit    | Interface (RB4)           | RB0     |
| Sound mode 2 bit    | Interface (RB5)           | RB1     |
| Sound mode 3 bit    | Interface (RB6)           | RB2     |
|_____________________|___________________________|_________|


OUTPUT: SOUND to amplifier
 ___________________________________________________________
| Type                | Function                  | Pin     |
|---------------------|-----------------------------------|
| Digital output      | Sound                     | RA1      |
|_____________________|___________________________|__________|



PROGRAM OVERVIEW  *******************************************

    Main ------------------------------------------------------
   |  Initialization                                           |
   |  Loop                                                     |
   |     Read mode bits and convert to decimal mode            |
   |     If mode = 1                                           |
   |        Play click sound                                   |
   |     If mode = 2                                           |
   |        Play other sound                                   |
   |     If mode = 3                                           |
   |        Play other sound                                   |
   |     If mode = 4                                           |
   |        Play other sound                                   |
   |     <can be expanded up to mode = 7>                      |
    -------------------------------------------------------

   */


/* _______________________________ INCLUDE __________________________ */

#include "16f628.h"
#include "int16CXX.h"


/* _______________________________ CONFIGURATION ____________________ */

#pragma config |= 0x3f10         // use internal 4MHz oscillator


/* _______________________________ GLOBAL VARIABLES _________________ */

char mode;                       // Ordered sound mode from PIC "Sound selection"


/* _______________________________ FUNCTIONS ________________________ */

void delay( char millisec);
void delay250us ( char millisec);
void delay10( char );

void tone_a (void);
void tone_b (void);


/* _______________________________ I/O PIN DEFINITIONS ____ */

#pragma bit TONE   @ PORTA.1     // Main output to transmitter

#pragma bit MODE_1  @ PORTB.0    // Sound selection bit 1
#pragma bit MODE_2  @ PORTB.1    // Sound selection bit 2
#pragma bit MODE_3  @ PORTB.2    // Sound selection bit 3
#pragma bit TRIG_0  @ PORTB.4    // "Null" signal from DCF simulator
#pragma bit TRIG_1  @ PORTB.5    // "One" signal from DCF simulator


/* _______________________________ MAIN FUNCTION start ________________________ */


void main(void)
{

   TRISA = 0b0000.0000;           // A1 = output
   TRISB = 0b0011.0111;           // B0, B1, B4, B5 = inputs
   OPTION.7 = 0;                 // Activate internal pullup resistors
   CM0=1; CM1=1;  CM2=1;         // No comparators on PORTA

   while(1)
   {
      mode.0 = MODE_1;           // Read mode from Sound Selection PIC
      mode.1 = MODE_2;
      mode.2 = MODE_3;

      if (mode == 1)
      {
         char s;
         if (TRIG_0 == 1)
         {
            for (s=0; s<6; s++)
            {
               tone_a ();
            }
            delay10(20);          // Wait for TRIG_0 to reset to 0
         }
         if (TRIG_1 == 1)
         {
            for (s=0; s<6; s++)
            {
               tone_b ();
            }
            delay10(25);          // Wait for TRIG_1 to reset to 0
            }
         }
      if (mode == 2)
      {
         while (TRIG_0 == 1)
         {
               tone_a ();
         }
         while (TRIG_1 == 1)
         {
               tone_b ();
         }
      }
      if (mode == 3)
      {
         if (TRIG_0 == 0  && TRIG_1 == 0)
         {
               tone_b ();
         }
      }
      if (mode == 4)
      {
         if (TRIG_0 == 0  && TRIG_1 == 0)
         {
               tone_a ();
         }
      }
   }
}


/* _______________________________ FUNCTIONS ________________________ */


void tone_a (void)
   {
      nop();
      delay(2);
      TONE = 1;
      delay(2);
      TONE = 0;
   }

void tone_b (void)
   {
      nop();
      delay(1);
      TONE = 1;
      delay(1);
      TONE = 0;
   }

void delay10( char n)
{
    char i;
    OPTION = 7;
    do  {
        i = TMR0 + 39; /* 256 microsec * 39 = 10 ms */
        while ( i != TMR0) ;
    } while ( --n > 0);
}

void delay ( char millisec)
/*
  Delays a multiple of 1 milliseconds at 4 MHz
  using the TMR0 timer
*/


{
   OPTION = 2;  /* prescaler divide by 8        */
   do  {
       TMR0 = 0;
       while ( TMR0 < 125)   /* 125 * 8 = 1000  */
            ;
   } while ( -- millisec > 0);
}

void delay250us ( char millisec)
{
   char m;
   char k;
   for (m=0; m<millisec; m++)
   {
         for (k=0; k<250; k++)
         {
         nop();
         }
   }
}
```

5.2 Sound selection code PIC16F690

```
/* FILE / DATE:     dcf_sound_selection_0502.c / 2010-05-02
   DESCRIPTION:     Sound selector logics for DCF simulator

   MICRO-PROCESSOR: PIC16F690
   COMPILER:        B Knudsen Cc5x C-compiler - not ANSI-C

   EXTERNAL HW:     Digital potentiometer X9C503
                    Rotary encoder
                    3 pins on PIC16F628 (Sound generator)

   DESIGNER:        Hans Sundgren
   CREDIT:          William Sandqvist william@kth.se for rotary encoder reading

   RAM usage:       10 bytes (7 local), 246 bytes free
   CODE WORDS:      238 code words (5 %)


CHIP CONNECTIONS ***********************************************
                      ________  _______
                     |        \/       |
                     |     16F690      |
              +5V ---|Vdd  1    28  Vss|---GND
        Encoder.B ->-|RA5  2    27  RA0|---
        Encoder.A ->-|RA4  3    26  RA1|---
                  ---|RA3  4    25  RA2|---
                  ---|RC5  5    24  RC0|---
                  ---|RC4  6    23  RC1|---
 Amplifier on/off -<-|RC3  7    22  RC2|---
  Digital pot.INC -<-|RC6  8    21  RB4|->- Sound mode.1 to PIC
  Digital pot.U/D -<-|RC7  9    20  RB5|->- Sound mode.2 to PIC
                  ---|RB7 10    19  RB6|->- Sound mode.3 to PIC
                     |_________________|


INPUT: ROTATING DIAL type A/B/Ground
 __________________________________________________________
| Type                 | Function                  | Pin      |
|---------------------|------------------------------------|
| Rotary encoder A     | Sound selection           | RA4      |
| Rotary encoder B     | Sound selection           | RA5      |
|_____________________|__________________________|_________|


OUTPUT: AMPLIFIER ON/OFF
 __________________________________________________________
| Type                 | Function                  | Pin      |
|---------------------|------------------------------------|
| Digital output       | Voltage to amplifier      | RC3      |
|_____________________|__________________________|_________|


OUTPUT: DIGITAL POTENTIOMETER (to amplifier)
 __________________________________________________________
| Type                 | Function                  | Pin      |
|---------------------|------------------------------------|
| Dig. potentiometer   | Change signal (INC)       | RC6      |
| Dig. potentiometer   | Up/Down change (U/D)      | RC7      |
|_____________________|__________________________|_________|


OUTPUT: SOUND GENERATOR INTERFACE TO PIC16F628
 __________________________________________________________
| Type                 | Function                  | Pin      |
|---------------------|------------------------------------|
| Sound mode bit 1     | Interface (RB0)           | RB4      |
| Sound mode bit 2     | Interface (RB1)           | RB5      |
| Sound mode bit 3     | Interface (RB2)           | RB6      |
|_____________________|__________________________|_________|


PROGRAM OVERVIEW  *******************************************

    Interrupt: Rotary encoder ----------------------------
   |  Read encoder                                        |
   |     If clockwise                                     |
   |        increase = 1                                  |
   |     If counter-clockwise                             |
   |        decrease = 1                                  |
    -------------------------------------------------------

    Main -------------------------------------------------
   |  Initialization                                      |
   |  Loop                                                |
   |     If increase                                      |
   |        if (fast)                                     |
   |           quick_up = 1                               |
   |        else                                          |
   |           increase signal to digital potentiometer   |
   |     If decrease                                      |
   |        if (quick_up = 1)                             |
   |           if (rotation_timeout = 0)                  |
   |              increment sound mode                    |
   |        else                                          |
   |           decrease signal to digital potentiometer   |
   |     If (rotation timeout = 0)                        |
   |        if (time_since_rotation = 170 ms)             |
   |        rotation timeout = 1                          |
    -------------------------------------------------------
   */

/* _______________________________ INCLUDE & CONFIGURATION ________________ */

#include "16f690.h"
#include "int16CXX.H"                    // Required for interrupt
#pragma config |= 0x00D4                 // Use internal 4MHz oscillator


/* _________________________ GLOBAL VARIABLES _____________________________ */

bit quick_up;       // Indicator for fast clockwise rotation
bit rot_timeout;    // Timeout for "fast" has expired
bit increase;       // Clockwise rotation
bit decrease;       // Counter-clockwise rotation
char mode;          // Sound mode 1 -> 2 -> 3 -> 0 -> 1 -> ...
char transit;       // Store transitions for encoder


/* _______________________________ FUNCTIONS ______________________________ */

interrupt int_server(void);
void inc_pulse (void);
void delay250us ( char millisec);
void delay( char millisec);


/* _______________________________ I/O PIN DEFINITIONS ____________________ */

#pragma bit AMPL    @ PORTC.3    // Amplifier voltage supply
#pragma bit INC     @ PORTC.6    // Step change digital potentiometer
#pragma bit U_D     @ PORTC.7    // Up/Down digital potentiometer

#pragma bit ROT_A   @ PORTA.5    // Rotating encoder A
#pragma bit ROT_B   @ PORTA.4    // Rotating encoder B


/* _______________________________ INTERRUPT FUNCTION _____________________ */

#pragma origin 4                 // Special interrupt instruction CC5x
```

```
interrupt int_server(void)
{
   int_save_registers            // W, STATUS (and PCLATH)

   if( RBIF == 1 )               // Rotary encoder interrupt
   {
      delay(1);                  // Debounce rotary switches
      transit.0 = ROT_A;         // Read rotary encoder value
      transit.1 = ROT_B;
      if( transit == 0b00.01 )   // Compare value
      {
         decrease = 1;
      }
      if( transit == 0b01.00 )
      {
         increase = 1;
      }
      transit.2 = transit.0;     // Replace old with new
      transit.3 = transit.1;
      RBIF = 0;                  // Reset interrupt flag
   }
   int_restore_registers         // W, STATUS (and PCLATH)
}


/* _______________________________ MAIN FUNCTION start _____________________ */

void main(void)
{
   ANSEL=0;                      // Disable analog signals on port C
   ANSELH=0;

   TRISA = 0b0011.0000;          // A.4 and A.5 inputs encoder
   TRISB = 0b0000.0000;          // B.4, B.5 and B.6 outputs
   TRISC=  0b0000.0000;             // C3, C.6 and C.7 outputs

   OPTION.7 = 0;
   WPUA.4 = 1;                   // Weak pullup input Encoder.A
   WPUA.5 = 1;                   // Weak pullup input Encoder.B
   IOCA.4  = 1;                  // Enable interrupt on Encoder.A
   IOCA.5  = 1;                  // Enable interrupt on Encoder.B

/* Definition of interrupt 1 Hz
   00.xx.x.x.x.x --
   xx.11.x.x.x.x Prescale 1/8    // 1,000,000 / 8 = 125,000
   xx.xx.1.x.x.x TMR1-oscillator is on
   xx.xx.x.1.x.x - (clock input synchronization)
   xx.xx.x.x.0.x Use internal clock 4.000.000/4 = 1.000.000
   xx.xx.x.x.x.0 TIMER1 is ON

*/
   T1CON = 0b00.11.0.1.0.1 ;
   /* CCPR = CLOSC * Timer1Period = 32768*1 = 32768 */
   /* Desired interval check = 200 ms = 5 Hz --> 125.000/25.000 = 5 Hz*/
   /* CCPR = 32768 = 0x8000 CCPR1H = 0x80, CCPR1L = 0x00 */
   /* CCPR = 25000 = 0x61A8 CCPR1H = 0x61, CCPR1L = 0xA8 */

   transit = 0;                  // Set start values
   quick_up = 0;
   rot_timeout = 1;
   mode = 0;
   increase = 0; decrease = 0;

   PORTB= 0b0000.0000;           // Sound mode 0 = silent to start with
   AMPL = 0;                     // Amplifier off

   RABIE   = 1;                  /* local enable  */

   TMR1H = 0x00;                 // Reset timer
   TMR1L = 0x00;
   GIE     = 1;
```

```
while (1)                    // Infinite loop
{
   if (increase == 1)
   {
      U_D = 1;               // Upwards direction
      inc_pulse ();
      increase = 0;

      if (rot_timeout == 0)   // If short time since last
      {
         quick_up = 1;
      }
      rot_timeout =0;
      TMR1ON = 0;             // Stop timer
      TMR1H = 0x00;           // Reset timer
      TMR1L = 0x00;
      TMR1ON = 1;             // Start timer
   }

   if (decrease == 1)
   {
      if (quick_up == 1)
      {
         quick_up = 0;
         decrease = 0;

         if (rot_timeout ==0)
         {
            mode = mode +1;
            if (mode == 5)
            {
               mode = 0;
            }
            if (mode ==0)
            {
            AMPL = 0;
               PORTB= 0b0000.0000;
            }
            if (mode ==1)
            {
            AMPL = 1;
               PORTB= 0b0001.0000;
            }
            if (mode ==2)
            {
            PORTB= 0b0010.0000;
            }
            if (mode ==3)
            {
            PORTB= 0b0011.0000;
            }
            if (mode ==4)
            {
            PORTB= 0b0100.0000;
            }
         }
         else
         {
            U_D = 0;
            delay250us(2);
            inc_pulse ();
         }
      }
      else
      {
         U_D = 0;
         delay250us(2);
         inc_pulse ();
         decrease = 0;
         rot_timeout = 0;
         TMR1ON = 0;          // Stop timer
         TMR1H = 0x00;        // Reset timer
```

```
            TMR1L = 0x00;
            TMR1ON = 1;          // Start timer
         }
      }
      if (rot_timeout == 0)
      {
         if (TMR1H > 0x55)       // Check if 170 ms expired since last rotation
                                 // 0x55 = 01010101
                                 // 01010101.00000000 = 21760
                                 // 125,000 / 21,760 = 5.7 = 170 ms
         {
            rot_timeout = 1;
            TMR1ON = 0;          // Stop timer
            TMR1H = 0x00;        // Reset timer
            TMR1L = 0x00;
         }
      }
   }
}

void inc_pulse (void)
{
   INC = 1;
   delay250us(1);
   INC = 0;
}

/* _______________________________ DELAY functions _____________________ */

void delay ( char millisec)
/*
  Delays a multiple of 1 milliseconds at 4 MHz
  using the TMR0 timer
*/


{
   OPTION = 2;  /* prescaler divide by 8        */
   do  {
       TMR0 = 0;
       while ( TMR0 < 125)   /* 125 * 8 = 1000  */
            ;
   } while ( -- millisec > 0);
}


void delay250us ( char millisec)
{
   char m;
   char k;
   for (m=0; m<millisec; m++)
   {
         for (k=0; k<250; k++)
         {
         nop();
         }
   }
}
```

5.3 DCF77 Simulator code PIC16F886

```
/* FILE / DATE:     dcf_sim_.c / 2010-05-02
   DESCRIPTION:     DCF Simulator

   MICRO-PROCESSOR: PIC16F886
   COMPILER:        B Knudsen Cc5x C-compiler - not ANSI-C

   EXTERNAL HW:     LCD Display
                    4 pushbuttons with LED
                    Rotating encoder
                    2 LEDs

   DESIGNER:        Hans Sundgren
   CREDIT:          William Sandqvist william@kth.se for rotary encoder reading

   RAM usage:       172 bytes (12 local), 196 bytes free
   CODE WORDS:      4450 code words (54 %)


CHIP CONNECTIONS ********************************************
                        _________ ________
                       |         \/        |
                       |       16F886      |
                      -|RE3   1     28  RB7|-
     Output.PULSE--<-|RA0   2     27  RB6|-
     LCD.backlight-<-|RA1   3     26  RB5|-<-Btn.START
      LED.SET------<-|RA2   4     25  RB4|-<-Btn.TRANSMIT
      LED.VIEW-----<-|RA3   5     24  RB3|-<-Btn.VIEW
      LED.TRANSMIT-<-|RA4   6     23  RB2|-<-Btn.SET
      LED.START----<-|RA5   7     22  RB1|-<-Encoder.A
               GND---|Vss   8     21  RB0|-<-Encoder.B
           LED.one-<-|RA7   9     20  Vdd|---+5V
          LED.null-<-|RA6  10     19  Vss|---GND
   XTAL 32,768 kHz---|OSO  11     18  RC7|->-LCD.D7
   XTAL 32,768 kHz---|OSI  12     17  RC6|->-LCD.D6
            LCD.RS-<-|RC2  13     16  RC5|->-LCD.D5
            LCD.EN-<-|RC3  14     15  RC4|->-LCD.D4
                       |___________________|


DISPLAY VIEWS, 1 -> 2 -> 3 -> 4 -> 1 -> ...
 ____________________________________________
| View | Shows                 | LCD backlight |
|------|----------------------|---------------|
| 1    | Time and bit display | No            |
| 2    | Time and bit display | Yes           |
| 3    | Bit overview         | Yes           |
| 4    | Bit description      | Yes           |
|______|______________________|_______________|


PUSH-BUTTONS
 ____________________________________________________________
| Button     | Color  | Pin | Lamp, pin: lit                  |
|------------|--------|-----|--------------------------------|
| Start/Stop | Green  | RB5 | RA5 | On during clock running   |
| Transmit   | Red    | RB4 | RA4 | Blink on output signal    |
| View       | Blue   | RB3 | RA3 | Blink when pressed        |
| Set        | Yellow | RB2 | RA2 | On during set clock       |
|____________|________|_____|________________________________|


LED INDICATORS
 ____________________________________________________________
| LED        | Color  | Pin | Lit during...                   |
|------------|--------|-----|--------------------------------|
| 0          | Yellow | RA7 | Output = 0                      |
|            |        |     | Decrement during clock setting |
|            |        |     |                                 |
| 1          | Green  | RA6 | Output = 1                      |
|            |        |     | Increment during clock setting |
|____________|________|_____|_________________________________|


ROTATING DIALS
 _________________________________________________________________
| Type            |  Function                        | Pin      |
|-----------------|----------------------------------|----------|
| Rotary encoder  | Increase/decrease clock setting  | RB0/RB1  |
|_________________|__________________________________|__________|



PROGRAM OVERVIEW  *******************************************

    Interrupt: 1 Hz --------------------------------------
   |  If start_mode = 1                                    |
   |     Update clock (step up one second)                 |
   |     Output pulse-value                                |
   |  Update display                                       |
   |  Check buttons                                        |
   |     If VIEW-button                                    |
   |        Step up display mode 0->1->2->3->0             |
   |     If SET-button                                     |
   |        Setting > 0                                    |
   |         Stop interrupt, jump out to setting sequence  |
    ------------------------------------------------------

    Interrupt: Rotary encoder, during "set clock" ----------
   |  Read encoder                                         |
   |     If increase                                       |
   |        up = 1 and blink LED0                          |
   |     If decrease                                       |
   |        down = 1 and blink LED1                        |
    ------------------------------------------------------

    Main -------------------------------------------------
   |  Initialization                                       |
   |  Read stored time from EEPROM                         |
   |  Interrupt is active 1 Hz, but the clock is stopped   |
   |  Loop                                                 |
   |     Check START-button                                |
   |        If START-button                                |
   |           Toggle start_mode                           |
    ------------------------------------------------------

    Set clock sequence  ----------------------------------
   |  Disable 1 Hz interrupt                               |
   |  Enable rotary encoder interrupt                      |
   |  Step through set sequence, read rotary encoder       |
   |     Update and display time values                    |
   |  Store set time in EEPROM                             |
   |  Enable 1 Hz interrupt                                |
    ------------------------------------------------------
   */


/* _______________________________ INCLUDE __________________________ */
#include "16F886.h"
#include "int16CXX.H"

/* _______________________________ CONFIGURATION ____________________ */

#pragma config WDTE=0, FOSC=4, PWRTE=1

/* _______________________________ GLOBAL VARIABLES _________________ */


bit start_mode;     // Clock running or not
char display_mode;  // Display alternatives: 1=clock, 2=overview, 3=details
char set_mode;      // Setting mode; 0=no set, 1=min, 2=hour, 3=day, etc.

char tick_pos;      // Current position in the bit stream
char tick_pos_1;
char tick_pos_10;
char bit_val;
char transit;       // Store rotary encoder transitions
bit up;
bit down;
bit tranmit_on;


char stored_time[14]; // Time data stored in EEPROM
char START_pressed;
char SET_pressed;
char no_of_ones;      // Counter of total no. of ones for parity bit setting
char first;
bit trans_mode;       //Transmitting on/off

//#pragma rambank 1


#pragma rambank 1

char bit_value[60]; // The one-minute bit stream to output

int sec1;           // DCF77 clock, second units
int sec10;          // DCF77 clock, second tens
int min1;           // DCF77 clock, minute units
int min10;          // DCF77 clock, minute tens
int hour1;          // DCF77 clock, hour units
int hour10;         // DCF77 clock, hour tens
int day1;           // DCF77 clock, day units
int day10;          // DCF77 clock, day tens
int month1;         // DCF77 clock, month units
int month10;        // DCF77 clock, month tens
int year1;          // DCF77 clock, year units (200X)
int year10;         // DCF77 clock, year tens  (20x0)
char weekday;       // DCF77 clock, day of week, starting Monday
char dst;           // DCF77 clock, Daylights saving time on/off


#pragma rambank 2

char c_pos[60];


/* _______________________________ FUNCTIONS _____________ */

void delay(char millisec);
void delay10(char n);
void delay250us(char millisec);

void get_bitvalue();            // Get the value of the current bit
void output_bitvalue();         // Send the bitvalue to LED and output

void set_bitvalue (void);       // Set bit stream once per minute based on clock

void pixel_def( void );
void lcd_init( void );          // Initalize the LCD display
void lcd_putchar( char data );  // Display one character on LCD
void lcd_putint( char value );
void lcd_strobe( void );
void lcd_startmsg(void);

void lcd_string ( char * s );

void lcd_command ( char cmd );

void put_eedata( void );
void get_eedata( void );
void update_display_mode_3 (void);
void update_display_mode_4 (void);

char text1( char );
char text2( char );
char text3( char );
char text4( char );
char text5( char );
char text6( char );
char text11( char );
char text12( char );
char text13( char );
char text14( char );
char text15( char );
char text16( char );
char text17( char );


void blink_led (char id);

void blink_null (void);
void blink_one (void);

void update_clock (void);
void update_minutes (void);

void set_clock (void);

void blink_empty (char digits);
void blink_digits (char dig1, char dig2);

void check_btn (void);

void display_time(void);

void display_weekday (void);
void display_dst (void);

void display_3_pos (void);
void display_4_pos (void);

//void display_bit_range (char s1, char s2, char t1, char t2);

interrupt int_server(void);

#define CLEAR     0b0.0000001

#define POS_1_0   0b1.0000000    // LCD display line 1
#define POS_1_1   0b1.0000001
#define POS_1_4   0b1.0000100
#define POS_1_5   0b1.0000101
#define POS_1_6   0b1.0000110
#define POS_1_7   0b1.0000111
#define POS_1_8   0b1.0001000
#define POS_1_9   0b1.0001001
#define POS_1_10  0b1.0001010
#define POS_1_11  0b1.0001011
#define POS_1_12  0b1.0001100
#define POS_1_13  0b1.0001101
#define POS_1_14  0b1.0001110
#define POS_1_15  0b1.0001111
#define POS_1_16  0b1.0010000
#define POS_1_17  0b1.0010001
#define POS_1_18  0b1.0010010
#define POS_1_19  0b1.0010011

#define POS_2_0   0b1.1000000    // LCD display line 2
#define POS_2_1   0b1.1000001
#define POS_2_2   0b1.1000010
#define POS_2_3   0b1.1000011
#define POS_2_4   0b1.1000100
#define POS_2_5   0b1.1000101
```

```
#define POS_2_6   0b1.1000110
#define POS_2_7   0b1.1000111
#define POS_2_8   0b1.1001000
#define POS_2_9   0b1.1001001
#define POS_2_10  0b1.1001010
#define POS_2_11  0b1.1001011
#define POS_2_12  0b1.1001100
#define POS_2_13  0b1.1001101
#define POS_2_14  0b1.1001110
#define POS_2_15  0b1.1001111
#define POS_2_16  0b1.1010000
#define POS_2_17  0b1.1010001
#define POS_2_18  0b1.1010010
#define POS_2_19  0b1.1010011

#define POS_3_0   0b1.0010100     // LCD display line 3
#define POS_3_1   0b1.0010101
#define POS_3_2   0b1.0010110
#define POS_3_3   0b1.0010111
#define POS_3_4   0b1.0011000
#define POS_3_5   0b1.0011001
#define POS_3_6   0b1.0011010
#define POS_3_7   0b1.0011011
#define POS_3_8   0b1.0011100
#define POS_3_9   0b1.0011101
#define POS_3_10  0b1.0011110
#define POS_3_11  0b1.0011111
#define POS_3_12  0b1.0100000
#define POS_3_13  0b1.0100001
#define POS_3_14  0b1.0100010
#define POS_3_15  0b1.0100011
#define POS_3_16  0b1.0100100
#define POS_3_17  0b1.0100101
#define POS_3_18  0b1.0100110
#define POS_3_19  0b1.0100111

#define POS_4_0   0b1.1010100      // LCD display line 4
#define POS_4_1   0b1.1010101
#define POS_4_2   0b1.1010110
#define POS_4_3   0b1.1010111
#define POS_4_4   0b1.1011000
#define POS_4_5   0b1.1011001
#define POS_4_6   0b1.1011010
#define POS_4_7   0b1.1011011
#define POS_4_8   0b1.1011100
#define POS_4_9   0b1.1011101
#define POS_4_10  0b1.1011110
#define POS_4_11  0b1.1011111
#define POS_4_12  0b1.1100000
#define POS_4_13  0b1.1100001
#define POS_4_14  0b1.1100010
#define POS_4_15  0b1.1100011
#define POS_4_16  0b1.1100100
#define POS_4_17  0b1.1100101
#define POS_4_18  0b1.1100110
#define POS_4_19  0b1.1100111


/* _______________________________ I/O PIN DEFINITIONS ____ */

#pragma bit PULSE @ PORTA.0     // Output signal to transmitter
#pragma bit T_ON  @ PORTB.7     // Transmitter on/off

#pragma bit LED0  @ PORTA.6     // LED indication "zero"
#pragma bit LED1  @ PORTA.7     // LED indication "one"

#pragma bit START @ PORTB.5     // Button START
#pragma bit TRANS @ PORTB.4     // Button TRANMIT
#pragma bit VIEW  @ PORTB.3     // Button VIEW
#pragma bit SET   @ PORTB.2     // Button SET
#pragma bit ROTA  @ PORTB.0     // Rotating encoder A
#pragma bit ROTB  @ PORTB.1     // Rotating encoder B
```

```
#pragma bit I_START @ PORTA.5   // Indicator START button
#pragma bit I_TRANS @ PORTA.4   // Indicator TRANMIT button
#pragma bit I_VIEW  @ PORTA.3   // Indicator VIEW button
#pragma bit I_SET   @ PORTA.2   // Indicator SET button

#pragma bit D4    @ PORTC.4     // LCD control
#pragma bit D5    @ PORTC.5     // LCD control
#pragma bit D6    @ PORTC.6     // LCD control
#pragma bit D7    @ PORTC.7     // LCD control
#pragma bit RS    @ PORTC.2     // LCD control
#pragma bit EN    @ PORTC.3     // LCD control
#pragma bit LCDLI @ PORTA.1     // LCD backlight

//#pragma bit start_VIEW @ 0x20.7


#pragma origin 4                // Special interrupt instruction CC5x
#pragma sharedAllocation        // Special interrupt instruction CC5x


/* _______________________________ INTERRUPT FUNCTION _______________________ */

interrupt int_server(void)
{
   int_save_registers                 // W, STATUS (and PCLATH)
   char sv_FSR = FSR;

   if(RBIF==1)                        // Rotary encoder interrupt
   {
      char m;
      char k;
      for (m=0; m<2; m++)             // debounce
      {
         for (k=0; k<250; k++)
         {
         nop();
         nop();
         }
      }
      transit.0=ROTB;                 // Read rotary encoder value
      transit.1=ROTA;
      if( transit==0b00.01 )          // Compare value
      {
         down=1;
      }
      if( transit==0b01.00 )
      {
         up=1;
      }
      transit.2=transit.0;            // Replace old with new
      transit.3=transit.1;
      RBIF = 0;                       // Reset interrupt flag
   }
   if (TMR1IF)                        // Timer 1 overflow interrupt
   {
      TMR1IF = 0;                     // Reset overflow flag
      TMR1H  = 0x80;                  // Reset counter
      TMR1L  = 0x00;

      if (start_mode == 0)
      {
         trans_mode=0;                // Switch off transmitter
         I_TRANS=0;
         T_ON = 0;

         I_START=1; delay250us(10);  I_START=0; delay250us(30); // Blink Start
button (x3)
         I_START=1; delay250us(10);  I_START=0; delay250us(30);
         I_START=1; delay250us(10);  I_START=0; delay250us(30);
      }
      if (start_mode ==1)
      {
         I_START=1;
         tick_pos++;                 // Step up tick_pos one second (not needed )
         sec1++;                     // Step up one second
         update_clock ();            // Main clock updated every second

         if (tick_pos == 60)
         {
            tick_pos = 0;
         }

         if (tick_pos == 0)
         {
            set_bitvalue();          // Set all 60 bits with initial valu
            first=0;
            if (display_mode == 3)
            {
                update_display_mode_3 ();  //Update bits
            }
         }
         output_bitvalue();          // Binary pulse on output
         }
      if (display_mode == 1 || display_mode == 2)
      {
         display_time ();
      }
      check_btn ();
   }
   FSR = sv_FSR;
   int_restore_registers          // W, STATUS (and PCLATH)
}


/* _______________________________ MAIN FUNCTION start ________________________ */

void main(void)
{
   ANSEL=0;                 // Initialisation to enable PORTC as digital I/O
   ANSELH=0;

   CM1CON0 = 0 ;            // Behvös den??
   CM2CON0 = 0 ;            // Behvös den??

   TRISA = 0b0000.0000;     //  Port A are all outputs
   TRISC=  0b0000.0000;     //  Port C are all outputs
   TRISB = 0b0011.1111;       //  Port B are mainly inputs

   OPTION.7 = 0;
   WPUB.0 = 1;              //  Weak pullup Encoder.B input
   WPUB.1 = 1;              //  Weak pullup Encoder.A input
   WPUB.2 = 1;              //  Weak pullup SET input
   WPUB.3 = 1;              //  Weak pullup VIEW input
   WPUB.4 = 1;              //  Weak pullup TRANSMIT input
   WPUB.5 = 1;              //  Weak pullup START input

   IOCB.0  = 1;             //  Enable interrupt on Encoder.B
   IOCB.1  = 1;             //  Enable interrupt on Encoder.A
   IOCB.4  = 1;             //  Enable interrupt on TRANSMIT input

   I_START = 0;
   I_TRANS = 0;
   I_VIEW = 0;
   I_SET = 0;
   LED0 = 0;
   LED1 = 0;

   RBIE   = 1;              /* local enable  */

   get_eedata ();                  // Get the string of time data from EEPROM ...
   hour10  =stored_time[0];        // ... and store in variables
   hour1   =stored_time[1];
   min10   =stored_time[2];
   min1    =stored_time[3];
   dst     =stored_time[6];
   year10  =stored_time[7];
   year1   =stored_time[8];
   month10 =stored_time[9];
   month1  =stored_time[10];
   day10   =stored_time[11];
   day1    =stored_time[12];
   weekday =stored_time[13];

   sec1 = 9;                      // Sec = 59
   sec10 = 5;                     // Sec = 59

   tick_pos = 59;
   display_mode = 1;              // Show clock at start + bitvalue

   trans_mode = 0;                // Transmitter off
   set_mode = 0;
   first=0;

   lcd_init();                     // Initiate the LCD
   LCDLI=0;                        // No backlight to start with

   char i;

   lcd_command (POS_1_0);
   for(i=0; i<16; i++) lcd_putchar(text1(i));  // display start text 1
   delay10(100);

   lcd_command (POS_2_0);
   for(i=0; i<16; i++) lcd_putchar(text2(i));  // display start text 2
   delay10(100);
   lcd_command (CLEAR);          // Clear display

   /* Definition of interrupt 1 Hz
   00.xx.x.x.x.x --
   xx.00.x.x.x.x Prescale 1/1
   xx.xx.1.x.x.x TMR1-oscillator is on
   xx.xx.x.0.x.x - (clock input synchronization)
   xx.xx.x.x.1.x Use external clock 32.768
   xx.xx.x.x.x.1 TIMER1 is ON
   */

   T1CON = 0b00.00.1.1.1.1 ;
   /* CCPR = CLOSC * Timer1Period = 32768*1 = 32768 */
   /* CCPR = 32768 = 0x8000 CCPR1H = 0x80, CCPR1L = 0x00 */

   //TMR1IE = 0;

   //TMR1H = 0x00;      //Short time to initial interrupt after Start
   //TMR1L = 0x11;
   /* Timer1Period is now 1 s */

//***********************************************************

   //lcd_command (0b0000.0000);                   // Clear display
                       //next interrupt should start at 00
   start_mode = 0;

   PULSE = 1;
   LED0 = 0;
   LED1 = 0;

   TMR1H = 0xEE;  //Short time to initial interrupt after Start
   TMR1L = 0x00;
   TMR1IE = 1;                // Enable TMR1 interrupt
   GIE = 1;                   // Interrupts allowed
   PEIE = 1;
   //display_time ();


/*
```

```
   while (1)                             // Infinite loop, wait for button press
   {
      while( START == 0) ;                // Button pressed
      {
         start_mode = ! start_VIEW;         // Start/stop signal
         delay10 (1);

         if (display_mode == 1 || display_mode == 2)
         {
            //lcd_command (CLEAR);          // Clear display
         }
      }
      while( START == 1 ) ;               // Button up
      {
         delay10 (1);
         //lcd_command (CLEAR);          // Clear display

      }



   }

*/
   while (1)
   {



   if (START==0)
      {
         if (START_pressed==0)
         {
            start_mode = ! start_mode;         // Start/stop signal
            delay10 (1);
            if (display_mode == 1 || display_mode == 2)
            {
               lcd_command (POS_1_0);
               for(i=0; i<20; i++) lcd_putchar(text5(i));  // display start text 1

               lcd_command (POS_2_18);          // move cursor to "line 2, position
18
               lcd_putchar('=');
            }
            START_pressed=1;
            //if (start_mode == 1)
            //{
            //   TMR1IE = 1;                // Enable TMR1 interrupt
            //}
            //else
            //{
            //   TMR1IE = 0;                // Enable TMR1 interrupt
            //}
         }
         if (START==1)
         {
            delay10 (1);
            START_pressed=0;
         }

      }

/*
    if (SET == 0)               // Button pressed
      {
         if (SET_pressed==0)
         {
            set_mode =  1;         // Step up display VIEW
            delay10 (1);
            lcd_command(0b0000.1100);  /* display on, cursor off, blink off */
            lcd_command (CLEAR);          // Clear display'
            SET_pressed=1;
            if (set_VIEW==1)
            {
               set_clock ();
            }
         }

      //set_clock ();                  // Funkar inte

         if (SET==1)
         {
            delay10 (1);
            SET_pressed=0;
         }

      }

*/
   }


   //}
      //if (VIEW==down)
      //{

}


/* ________________________ TRANSMITTER OUTPUT FUNCTION start
________________________ */

void output_bitvalue(void)
{
   char i;
   bit_val = bit_value[tick_pos];

   if (display_mode == 1 || display_mode == 2)
   {
      tick_pos_1 = tick_pos%10;         // Get the single digit for display
      tick_pos_10 = tick_pos/10;        // Get the single digit for display
      lcd_command (0b1001.0010);                    // reposition to "line 2,
position 18
      lcd_putchar(48+tick_pos_10);
      lcd_putchar(48+tick_pos_1);
   }

   if (display_mode == 3)
   {
      update_display_mode_3 ();
   }

   if (display_mode == 4)                 // Endast markering tillsivdare
   {
      update_display_mode_4 ();
   }

   if (bit_val == 0)
      {
      if (display_mode == 1 || display_mode == 2)
         {
         lcd_command (0b1101.0011);                    // reposition to "line 1,
position 19
         lcd_putchar('0');
         }

      LED0 = 1;
      if (trans_mode==1)
      {
         I_TRANS = 0;
      }
      PULSE = 0;
      delay10(10);
      LED0 = 0;
      if (trans_mode==1)
      {
         I_TRANS = 1;
      }
      PULSE = 1;
      }


   if (bit_val == 1)
   {
      if (display_mode == 1 || display_mode == 2)
      {

         lcd_command (0b1101.0011);                    // reposition to "line 1,
position 19
      //lcd_command (0b1000.0000);                     // reposition to "line 1,
position 0

         //RS = 1;  // LCD in character-VIEW
         lcd_putchar('1');
         }

      LED1 = 1;
       if (trans_mode==1)
      {
         I_TRANS = 0;
      }
      PULSE = 0;
      delay10(20);

      LED1 = 0;
      if (trans_mode==1)
      {
         I_TRANS = 1;
      }
      PULSE = 1;
      }



   if (bit_val == 9)    //last second before new minute, no pulse
      {

      if (display_mode == 1 || display_mode == 2)
         {
         //RS = 0;  // LCD in command-VIEW
         //lcd_putchar( 0b00000010 ); //Cursor hoME
         //RS = 1;  // LCD in character-VIEW
         lcd_command (0b1101.0011);                    // reposition to "line 1,
position 19
         lcd_putchar('-');
         }
      //PULSE2 = 0;
      }



}



void check_btn (void)
{
char i;
   if (VIEW == 0)              // Button pressed
   {
      I_VIEW=1; delay250us(30);  I_VIEW=0;  // Blink Start button

      display_mode =  display_mode+1;         // Step up display VIEW

      //lcd_command (CLEAR);          // Clear display
      first=0;

      if (display_mode==5)
      {
         display_mode = 1;
      }
      if (display_mode == 1 || display_mode == 2)
      {
         lcd_command (POS_1_0);
         for(i=0; i<20; i++) lcd_putchar(text5(i));  // display start text 1

         lcd_command (POS_2_0);
         for(i=0; i<20; i++) lcd_putchar(text6(i));  // display start text 1

         lcd_command (POS_2_18);          // move cursor to "line 2, position 18
         lcd_putchar('=');

         lcd_command(0b0000.1100);  /* display on, cursor off, blink off */
         //lcd_command (CLEAR);          // Clear display'
         LCDLI=0;

      }
      if (display_mode == 2 || display_mode == 3 || display_mode == 4)
      {
         LCDLI=1;

      }
      if (display_mode==4)
      {
         //LCDLI=1;
         lcd_command(0b0000.1100);  /* display on, cursor off, blink off */
         lcd_command (CLEAR);          // Clear display'

      }
   }
   if (SET == 0)               // Button pressed
   {
      if (start_mode == 0)
       {
         set_mode =  1;         // Step up display VIEW
         lcd_command(0b0000.1100);  /* display on, cursor off, blink off */
         lcd_command (CLEAR);          // Clear display'
         //set_clock ();                   // Funkar inte

         if (set_mode==1)
         {
            set_clock ();
         }
      }
   }



   if (TRANS == 0)               // Button pressed
   {
      if (start_mode == 1)         // Transmit button active only during run
      {
         I_TRANS=1; delay250us(30);  I_TRANS=0;  // Blink Start button
         trans_mode = ! trans_mode;         // Start/stop signal
         delay10 (1);
         if (trans_mode == 1)
         {
            I_TRANS=1;
            T_ON = 1;
         }
         if (trans_mode == 0)
         {
            I_TRANS=0;
           T_ON = 0;
         }

}
```

```
    }


/*
    if (SET == 0)               // Button pressed
    {
         if (SET_pressed==0)
         {
            set_mode =  1;         // Step up display VIEW
            delay10 (1);
            lcd_command(0b0000.1100);  /* display on, cursor off, blink off */
            lcd_command (CLEAR);          // Clear display'
            SET_pressed=1;
            if (set_VIEW==1)
            {
               set_clock ();
            }
         }

      //set_clock ();                   // Funkar inte

       if (SET==1)
       {
          delay10 (1);
          SET_pressed=0;
       }

    }
*/



}




void blink_empty (char digits)     // Erases a number of LCD characters
{
   char i;
   for (i=0; i<digits; i++)
   {
      lcd_putchar(' ');
   }
   delay10 (30);                    // Wait 500 ms
}


void blink_digits (char dig1, char dig2)    // Erases a number of LCD characters
{
   lcd_putchar(48+dig1);            // 48 = 0011.0000
   lcd_putchar(48+dig2);
   delay10(20);
}



/* ________________________ DCF77 CLOCK start ________________________ */

void update_clock (void)       // Updates the clock data with one second
{
   if (sec1>9)
   {
      sec1 = 0;
      sec10++;
   }
      if (sec10 > 5)
   {
      sec10 = 0;
      min1++;
   }
   update_minutes ();
```

```
}


void update_minutes (void)      // Clock over minute > hour > day > month > year
{
   if(min1>9)
   {
      min1 = 0;
      min10++;
   }
   if(min10>5)
   {
      min10 = 0;
      hour1++;
   }
   if(hour1>9)
   {
      hour1 = 0;
      hour10++;
   }
   if(hour10 >2){
      hour10 = 0;
   }
   if(hour10 >=2 && hour1>3)
   {
      hour1 = 0;
      hour10 = 0;
      day1++;
   }
   if(day1>9){
      day1 = 0;
      day10++;
   }
   if(day10 >=2 && day1>1)
   {
      day1 = 1;
      day10 = 0;
      month1++;
   }
   if(month1>9)
   {
      month1 = 0;
      month10++;
   }
   if(month10 >=1 && month1>2)
   {
      month1 = 1;
      month10 = 0;
      year1++;
   }
   if(year1>9)
   {
      year1 = 0;
      year10++;
   }
}


void display_time ( void )
{

   lcd_command (POS_3_0);                  // reposition to line 1

   lcd_putint(hour10);
   lcd_putint(hour1);
   lcd_putchar(':');
   lcd_putint(min10);
   lcd_putint(min1);
   lcd_putchar(':');
   lcd_putint(sec10);
   lcd_putint(sec1);
   lcd_putchar(' ');
   lcd_putchar(' ');
   lcd_putchar(' ');
   lcd_putchar('D');
   lcd_putchar('S');
   lcd_putchar('T');
   lcd_putchar('=');
   lcd_putint(dst);


   lcd_command (POS_4_0);                  // reposition to line 1
   lcd_putchar('2');
   lcd_putchar('0');
   lcd_putint(year10);         // 48 = 0011.0000
   lcd_putint(year1);
   lcd_putchar('-');
   lcd_putint(month10);
   lcd_putint(month1);
   lcd_putchar('-');
   lcd_putint(day10);
   lcd_putint(day1);
   lcd_putchar(' ');

   if (weekday == 1)
   {
      lcd_putchar('M');
      lcd_putchar('o');
      lcd_putchar('n');
      lcd_putchar('d');
      lcd_putchar('a');
      lcd_putchar('y');
      lcd_putchar(' ');
   }

   if (weekday == 2)
   {
      lcd_putchar('T');
      lcd_putchar('u');
      lcd_putchar('e');
      lcd_putchar('s');
      lcd_putchar('d');
      lcd_putchar('a');
      lcd_putchar('y');
      lcd_putchar(' ');
      lcd_putchar(' ');
   }
   if (weekday == 3)
   {
      lcd_putchar('W');
      lcd_putchar('e');
      lcd_putchar('d');
      lcd_putchar('n');
      lcd_putchar('e');
      lcd_putchar('s');
      lcd_putchar('d');
      lcd_putchar('a');
      lcd_putchar('y');
   }
   if (weekday == 4)
   {
      lcd_putchar('T');
      lcd_putchar('h');
      lcd_putchar('u');
      lcd_putchar('r');
      lcd_putchar('s');
      lcd_putchar('d');
      lcd_putchar('a');
      lcd_putchar('y');
      lcd_putchar(' ');


   }
   if (weekday == 5)
```

```
   {
      lcd_putchar('F');
      lcd_putchar('r');
      lcd_putchar('i');
      lcd_putchar('d');
      lcd_putchar('a');
      lcd_putchar('y');
      lcd_putchar(' ');
      lcd_putchar(' ');
   }
   if (weekday == 6)
   {
      lcd_putchar('S');
      lcd_putchar('a');
      lcd_putchar('t');
      lcd_putchar('u');
      lcd_putchar('r');
      lcd_putchar('d');
      lcd_putchar('a');
      lcd_putchar('y');
   }

   if (weekday == 7)
   {
      lcd_putchar('S');
      lcd_putchar('u');
      lcd_putchar('n');
      lcd_putchar('d');
      lcd_putchar('a');
      lcd_putchar('y');
      lcd_putchar(' ');
      lcd_putchar(' ');
   }
}



/* ________________________ SET TRANSMIT BIT VALUES________________________ */

void set_bitvalue (void)            // Set new bitvalues before a new minute
   {
   char i;
   min1 = min1 + 1;                 // Time to transmit = +1 minute
   //update_minutes ();   // Replaced update_time 7 feb
   update_clock ();
   bit_value[0] = 0;
   bit_value[1] = 0;
   bit_value[2] = 1;
   bit_value[3] = 0;
   bit_value[4] = 0;
   bit_value[5] = 0;
   bit_value[6] = 0;
   bit_value[7] = 0;
   bit_value[8] = 0;
   bit_value[9] = 0;
   bit_value[10] = 0;
   bit_value[11] = 0;
   bit_value[12] = 0;
   bit_value[13] = 0;
   bit_value[14] = 0;
   bit_value[15] = 0;               // 0 = primary antenna in use
   bit_value[16] = 0;               // 0 = No DST change in the next hour
   bit_value[17] = dst;             //  DST on/off
   bit_value[18] = 1;               // Timezone, usually opposite of bit 17
   bit_value[19] = 0;               // Leap second insertion (not used in
simulator)
   bit_value[20] = 1;               // Always 1
   bit_value[21] = min1.0;          // Minutes, BCD least significant first
   bit_value[22] = min1.1;
   bit_value[23] = min1.2;
   bit_value[24] = min1.3;
   bit_value[25] = min10.0;
   bit_value[26] = min10.1;
   bit_value[27] = min10.2;

   no_of_ones = 0;
   for (i=21; i<28; i++)
      {
      no_of_ones = no_of_ones+bit_value[i];
      }
   bit_value[28] = no_of_ones%2;    // Even parity bit, minute bits
   bit_value[29] = hour1.0;           // Hours, BCD least significant first
   bit_value[30] = hour1.1;
   bit_value[31] = hour1.2;
   bit_value[32] = hour1.3;
   bit_value[33] = hour10.0;
   bit_value[34] = hour10.1;

   no_of_ones = 0;
   for (i=29; i<35; i++)            // Calculate number of "ones"
      {
      no_of_ones = no_of_ones + bit_value[i];
      }
   bit_value[35] = no_of_ones%2;    // Even parity bit, hour bits
   bit_value[36] = day1.0;          // Day of month, BCD least significant first
   bit_value[37] = day1.1;
   bit_value[38] = day1.2;
   bit_value[39] = day1.3;
   bit_value[40] = day10.0;
   bit_value[41] = day10.1;
   bit_value[42] = weekday.0;       // Day of week, BCD least significant first,
Mon=1
   bit_value[43] = weekday.1;
   bit_value[44] = weekday.2;
   bit_value[45] = month1.0;        // Month, BCD least significant first
   bit_value[46] = month1.1;
   bit_value[47] = month1.2;
   bit_value[48] = month1.3;
   bit_value[49] = month10.0;
   bit_value[50] = year1.0;         // Year, BCD least significant first
   bit_value[51] = year1.1;
   bit_value[52] = year1.2;
   bit_value[53] = year1.3;
   bit_value[54] = year10.0;
   bit_value[55] = year10.1;
   bit_value[56] = year10.2;
   bit_value[57] = year10.3;

   no_of_ones = 0;
   for (i=36; i<58; i++)            // Calculate number of "ones"
      {
      no_of_ones = no_of_ones + bit_value[i];
      }
   bit_value[58] = no_of_ones%2;    // Even parity bit, all transmitted bits
   bit_value[59] = 9;               // No transmission

   min1 = min1-1;                   // Reset to real time
   if(min1<0)                       // For case "00.00"
   {
      min1 = 9;
      min10--;
   }
   if(min10 < 0)
   {
      min10 = 5;
      hour1--;
   }
   if(hour1 < 0)
   {
      hour1 = 3;
      hour10--;
   }
   if(hour10<0){
      hour10 = 2;
   }
   // MORE FUNCTIONS required to correct tyhe day, month, year
   if (display_mode ==3)
   {
      update_display_mode_3 ();
   }

}


void put_eedata( void )
{
/* Put global s[] -string in EEPROM-data */
char i, c;
for(i = 0;  i <14; i++)
    {
      c = stored_time[i];
      /* Write EEPROM-data sequence                          */
      EEADR = i;          /* EEPROM-data adress 0x00 => 0x7F */
      EEDATA = c;         /* data to be written              */
      WREN = 1;           /* write enable                    */
      EECON2 = 0x55;      /* first Byte in comandsequence    */
      EECON2 = 0xAA;      /* second Byte in comandsequence   */
      WR = 1;             /* write                           */
      while( EEIF == 0) ; /* wait for done (EEIF=1)          */
      WREN = 0;           /* write disable - safety first    */
      EEIF = 0;           /* Reset EEIF bit in software      */
      /* End of write EEPROM-data sequence                   */
      //if(stored_time == '\0') break;
    }
}



void get_eedata( void )
{
/* Get EEPROM-data and put it in global s[] */
char i;
for(i = 0;  i <14; i++)
    {
      /* Start of read EEPROM-data sequence                */
      EEADR = i;       /* EEPROM-data adress 0x00 => 0x7F  */
      RD = 1;          /* Read                             */
      stored_time[i] = EEDATA;   /* data to be read                  */
      /* End of read EEPROM-data sequence                  */
      //if( s[i] == '\0') break;
    }
}





/* ________________________ LCD TEXT STORAGE________________________ */

char text1( char x)                         // this is the way to store a sentence
{
   skip(x);                                 /* internal function CC5x.  */
   #pragma return[] = "DCF77 Simulator "    // 16-teckens rad
}

char text2( char x)                          // this is the way to store a sentence
{
   skip(x);                                 /* internal function CC5x.  */
   #pragma return[] = "Set clock, start"    // 8 chars max!
}

char text3( char x)                          // this is the way to store a sentence
{
   skip(x);                                 /* internal function CC5x.  */
   #pragma return[] = "Set a fake time "    // 8 chars max!
```

```
}

char text4( char x)                          // this is the way to store a sentence
{
   skip(x);                                 /* internal function CC5x.  */
   #pragma return[] = "Time is set. START  "    // 8 chars max!
}
char text5( char x)                          // this is the way to store a sentence
{
   skip(x);                                 /* internal function CC5x.  */
   #pragma return[] = "Transmitting bit:   "    // 8 chars max!
}

char text6( char x)                          // this is the way to store a sentence
{
   skip(x);                                 /* internal function CC5x.  */
   #pragma return[] = "                    "    // 20 empty characters (erase)
}
char text7( char x)                          // this is the way to store a sentence
{
   skip(x);                                 /* internal function CC5x.  */
   #pragma return[] = "bit"    // 20 empty characters (erase)
}




/* ________________________ LCD INITIALIZE________________________ */

void lcd_init( void ) // must be run once before using the display
{
   delay(40);                 // give LCD time to settle
   RS = 0;                    // LCD in command-VIEW
   lcd_putchar(0b0011.0011);  /* LCD starts in 8 bit VIEW          */
   lcd_putchar(0b0011.0010);  /* change to 4 bit VIEW              */
   lcd_putchar(0b0010.1000);  /* two line (8+8 chars in the row)   */
   lcd_putchar(0b0000.1100);  /* display on, cursor off, blink off */
   lcd_putchar(0b0000.0001);  /* display clear                     */
   lcd_putchar(0b0000.0110);  /* ROTArement VIEW, shift off         */
   RS = 1;                     // LCD in character-VIEW
}


/* ________________________ LCD PUT CHARACTER________________________ */


void lcd_putchar( char data )
{
   // Must set LCD-VIEW before calling this function
   // RS = 1 LCD in character-VIEW
   // RS = 0 LCD in command-VIEW

   D7 = data.7;         // upper Nibble
   D6 = data.6;
   D5 = data.5;
   D4 = data.4;
   EN = 0; nop(); nop(); EN = 1; delay(1);

   D7 = data.3;         // lower Nibble
   D6 = data.2;
   D5 = data.1;
   D4 = data.0;
   EN = 0; nop(); nop(); EN = 1; delay(2);
}



void lcd_command ( char cmd )       // Write command to LCD
{
   RS = 0;
   lcd_putchar( cmd );
   RS = 1;                          // Back to character VIEW
   delay(2);
}

void lcd_string ( char * s )       // Write command to LCD
{
   RS = 1;
   while (*s) lcd_putchar(*s++);
   delay(2);
}

void lcd_putint( char value )
{
  lcd_putchar(value+48);
}




/* ____ DELAY CAUSING PROBLEMS SOMETIMES ____ */


void delay ( char millisec)
/*
  Delays a multiple of 1 milliseconds at 4 MHz
  using the TMR0 timer
*/


{
   OPTION = 2;  /* prescaler divide by 8        */
   do  {
       TMR0 = 0;
       while ( TMR0 < 125)   /* 125 * 8 = 1000  */
            ;
   } while ( -- millisec > 0);
}




void delay250us ( char millisec)
{
   char m;
   char k;
   for (m=0; m<millisec; m++)
   {
         for (k=0; k<250; k++)
         {
         nop();
         nop();
         }
   }
}



void delay10 ( char n)
/*
  Delays a multiple of 10 milliseconds using the TMR0 timer
  Clock : 4 MHz   => period T = 0.25 microseconds
  1 IS = 1 Instruction Cycle = 1 microsecond
  error: 0.16 percent
*/
{
    char i;

    OPTION = 7;
    do  {
        i = TMR0 + 39; /* 256 microsec * 39 = 10 ms */
        while ( i != TMR0)
            ;
    } while ( --n > 0);
}


#pragma codepage 1

void display_3_pos (void)
{

   c_pos[0]  = POS_1_5;
   c_pos[1]  = POS_1_6;
   c_pos[2]  = POS_1_7;
   c_pos[3]  = POS_1_8;
   c_pos[4]  = POS_1_9;
   c_pos[5]  = POS_1_10;
   c_pos[6]  = POS_1_11;
   c_pos[7]  = POS_1_12;
   c_pos[8]  = POS_1_13;
   c_pos[9]  = POS_1_14;
   c_pos[10] = POS_1_15;
   c_pos[11] = POS_1_16;
   c_pos[12] = POS_1_17;
   c_pos[13] = POS_1_18;
   c_pos[14] = POS_1_19;

   c_pos[15] = POS_2_5;
   c_pos[16] = POS_2_6;
   c_pos[17] = POS_2_7;
   c_pos[18] = POS_2_8;
   c_pos[19] = POS_2_9;
   c_pos[20] = POS_2_10;
   c_pos[21] = POS_2_11;
   c_pos[22] = POS_2_12;
   c_pos[23] = POS_2_13;
   c_pos[24] = POS_2_14;
   c_pos[25] = POS_2_15;
   c_pos[26] = POS_2_16;
   c_pos[27] = POS_2_17;
   c_pos[28] = POS_2_18;
   c_pos[29] = POS_2_19;

   c_pos[30] = POS_3_5;
   c_pos[31] = POS_3_6;
   c_pos[32] = POS_3_7;
   c_pos[33] = POS_3_8;
   c_pos[34] = POS_3_9;
   c_pos[35] = POS_3_10;
   c_pos[36] = POS_3_11;
   c_pos[37] = POS_3_12;
   c_pos[38] = POS_3_13;
   c_pos[39] = POS_3_14;
   c_pos[40] = POS_3_15;
   c_pos[41] = POS_3_16;
   c_pos[42] = POS_3_17;
   c_pos[43] = POS_3_18;
   c_pos[44] = POS_3_19;

   c_pos[45] = POS_4_5;
   c_pos[46] = POS_4_6;
   c_pos[47] = POS_4_7;
   c_pos[48] = POS_4_8;
   c_pos[49] = POS_4_9;
   c_pos[50] = POS_4_10;
   c_pos[51] = POS_4_11;
   c_pos[52] = POS_4_12;
   c_pos[53] = POS_4_13;
   c_pos[54] = POS_4_14;
   c_pos[55] = POS_4_15;
   c_pos[56] = POS_4_16;
   c_pos[57] = POS_4_17;
   c_pos[58] = POS_4_18;
   c_pos[59] = POS_4_19;
```

```
}


void display_4_pos (void)
{
   c_pos[0]  = POS_2_0;
   c_pos[1]  = POS_2_1;
   c_pos[2]  = POS_2_2;
   c_pos[3]  = POS_2_3;
   c_pos[4]  = POS_2_4;
   c_pos[5]  = POS_2_5;
   c_pos[6]  = POS_2_6;
   c_pos[7]  = POS_2_7;
   c_pos[8]  = POS_2_8;
   c_pos[9]  = POS_2_9;
   c_pos[10] = POS_2_10;
   c_pos[11] = POS_2_11;
   c_pos[12] = POS_2_12;
   c_pos[13] = POS_2_13;
   c_pos[14] = POS_2_14;

   c_pos[15] = POS_1_1;
   c_pos[16] = POS_2_0;
   c_pos[17] = POS_2_1;
   c_pos[18] = POS_2_2;
   c_pos[19] = POS_2_3;
   c_pos[20] = POS_1_1;
   c_pos[21] = POS_2_0;
   c_pos[22] = POS_2_1;
   c_pos[23] = POS_2_2;
   c_pos[24] = POS_2_3;
   c_pos[25] = POS_2_4;
   c_pos[26] = POS_2_5;
   c_pos[27] = POS_2_6;
   c_pos[28] = POS_3_1;
   c_pos[29] = POS_2_0;

   c_pos[30] = POS_2_1;
   c_pos[31] = POS_2_2;
   c_pos[32] = POS_2_3;
   c_pos[33] = POS_2_4;
   c_pos[34] = POS_2_5;
   c_pos[35] = POS_3_1;
   c_pos[36] = POS_2_0;
   c_pos[37] = POS_2_1;
   c_pos[38] = POS_2_2;
   c_pos[39] = POS_2_3;
   c_pos[40] = POS_2_4;
   c_pos[41] = POS_2_5;
   c_pos[42] = POS_2_0;
   c_pos[43] = POS_2_1;
   c_pos[44] = POS_2_2;

   c_pos[45] = POS_2_0;
   c_pos[46] = POS_2_1;
   c_pos[47] = POS_2_2;
   c_pos[48] = POS_2_3;
   c_pos[49] = POS_2_4;
   c_pos[50] = POS_2_0;
   c_pos[51] = POS_2_1;
   c_pos[52] = POS_2_2;
   c_pos[53] = POS_2_3;
   c_pos[54] = POS_2_4;
   c_pos[55] = POS_2_5;
   c_pos[56] = POS_2_6;
   c_pos[57] = POS_2_7;
   c_pos[58] = POS_2_0;
   c_pos[59] = POS_2_0;

}
```

```
void update_display_mode_3 (void)
{
   display_3_pos ();                      // load the cursor position for display 2
   lcd_command (c_pos[tick_pos]);        // move cursor to current bit position
   if(first==0)
   {
      lcd_command (0b0000.1110);              // Cursor on, blink off
      lcd_command (0b0000.0110);              // ROTArement, not shifted

      lcd_command (POS_1_0);                   // reposition to line 1
      lcd_putchar('0');
      lcd_putchar('0');
      lcd_putchar('-');
      lcd_putchar(':');
      lcd_putchar(' ');
      char i;
      for(i=0; i<15; i++)                    // display binary values 1-14
      {
         lcd_putint(bit_value[i]);
      }
      lcd_command (POS_2_0);
      lcd_putchar('1');
      lcd_putchar('5');
      lcd_putchar('-');
      lcd_putchar(':');
      lcd_putchar(' ');
      for(i=15; i<30; i++)                    // display binary values 15-29
      {
         lcd_putint(bit_value[i]);
      }
      lcd_command (POS_3_0);
      lcd_putchar('3');
      lcd_putchar('0');
      lcd_putchar('-');
      lcd_putchar(':');
      lcd_putchar(' ');
      for(i=30; i<45; i++)                    // display binary values 30-44
      {
         lcd_putint(bit_value[i]);
      }
      lcd_command (POS_4_0);
      lcd_putchar('4');
      lcd_putchar('5');
      lcd_putchar('-');
      lcd_putchar(':');
      lcd_putchar(' ');
      for(i=45; i<59; i++)                    // display binary values 45-58
      {
         lcd_putint(bit_value[i]);
      }
      lcd_putchar('-');                       // bit 59 always no signal
      first=1;
   }
}




void update_display_mode_4 (void)
{
   display_4_pos ();                      // Update cursor positions for this
display
   char i;
   lcd_command (0b0000.1110);             // Cursor on, blink off
   lcd_command (0b0000.0110);             // ROTArement, not shifted
   lcd_command (POS_1_11);
   lcd_putchar('b');
   lcd_putchar('i');
   lcd_putchar('t');

   if(tick_pos > 0 && tick_pos < 15)
   {

      lcd_command (POS_3_0);
      lcd_putchar(' ');
      lcd_putchar(' ');
      lcd_putchar(' ');
      lcd_putchar(' ');
      lcd_putchar(' ');
      lcd_putchar(' ');
      lcd_putchar(' ');
      lcd_putchar(' ');
      lcd_putchar(' ');
      lcd_putchar(' ');

      lcd_command (POS_1_0);
      lcd_putchar('W');
      lcd_putchar('e');
      lcd_putchar('a');
      lcd_putchar('t');
      lcd_putchar('h');
      lcd_putchar('e');
      lcd_putchar('r');
      lcd_putchar(' ');
      lcd_putchar(' ');
      lcd_putchar(' ');
      lcd_putchar(' ');

      lcd_command (POS_1_14);
      lcd_putchar(' ');
      lcd_putchar('0');
      lcd_putchar('1');
      lcd_putchar('-');
      lcd_putchar('1');
      lcd_putchar('4');

      lcd_command (POS_2_0);

      for(i=0; i<15; i++)
      {
         lcd_putint(bit_value[i]);
      }
   }

   if (tick_pos > 15 && tick_pos < 20)
   {
      lcd_command (POS_1_0);
      lcd_putchar('D');
      lcd_putchar('S');
      lcd_putchar('T');
      lcd_putchar('+');
      lcd_putchar(' ');
      lcd_putchar(' ');
      lcd_putchar(' ');

      lcd_command (POS_1_15);
      lcd_putchar('1');
      lcd_putchar('6');
      lcd_putchar('-');
      lcd_putchar('1');
      lcd_putchar('9');

      lcd_command (POS_2_0);
      for(i=16; i<20; i++)
      {
         lcd_putint(bit_value[i]);
      }

      for(i=0; i<11; i++)
      {
         lcd_putchar(' ');
      }
   }
```

```
if (tick_pos == 21)
{
   lcd_command (POS_1_0);
   lcd_putchar('M');
   lcd_putchar('i');
   lcd_putchar('n');
   lcd_putchar('u');
   lcd_putchar('t');
   lcd_putchar('e');

   lcd_command (POS_1_15);
   lcd_putchar('2');
   lcd_putchar('1');
   lcd_putchar('-');
   lcd_putchar('2');
   lcd_putchar('7');

   lcd_command (POS_2_0);
   for(i=21; i<28; i++)
   {
      lcd_putint(bit_value[i]);
   }
   lcd_command (POS_2_8);
   lcd_putchar('=');
   lcd_putchar(' ');
   lcd_putint(min10);
   lcd_putint(min1);
   lcd_putchar(' ');

}

if(tick_pos == 29)
{
   lcd_command (POS_1_0);
   lcd_putchar('H');
   lcd_putchar('o');
   lcd_putchar('u');
   lcd_putchar('r');
   lcd_putchar(' ');
   lcd_putchar(' ');

   lcd_command (POS_1_15);
   lcd_putchar('2');
   lcd_putchar('9');
   lcd_putchar('-');
   lcd_putchar('3');
   lcd_putchar('4');

   lcd_command (POS_2_0);
   for(i=29; i<35; i++)
   {
      lcd_putint(bit_value[i]);
   }
   lcd_putchar(' ');

   lcd_command (POS_2_7);
   lcd_putchar('=');
   lcd_putchar(' ');
   lcd_putint(hour10);
   lcd_putint(hour1);
   lcd_putchar(' ');
}

if(tick_pos == 36)
{

   lcd_command (POS_1_0);
   lcd_putchar('D');
   lcd_putchar('a');
   lcd_putchar('y');
```

```
      lcd_putchar(' ');


      lcd_command (POS_1_15);
      lcd_putchar('3');
      lcd_putchar('6');
      lcd_putchar('-');
      lcd_putchar('4');
      lcd_putchar('1');

      lcd_command (POS_2_0);

      for(i=36; i<42; i++)
      {
         lcd_putint(bit_value[i]);
      }
      lcd_putchar(' ');
      lcd_command (POS_2_7);
      lcd_putchar('=');
      lcd_putchar(' ');
      lcd_putint(day10);
      lcd_putint(day1);
   }
   if(tick_pos == 42)
   {
      lcd_command (POS_1_0);
      lcd_putchar('W');
      lcd_putchar('e');
      lcd_putchar('e');
      lcd_putchar('k');
      lcd_putchar('d');
      lcd_putchar('a');
      lcd_putchar('y');


      lcd_command (POS_1_15);
      lcd_putchar('4');
      lcd_putchar('2');
      lcd_putchar('-');
      lcd_putchar('4');
      lcd_putchar('4');

      lcd_command (POS_2_0);
      lcd_putint(bit_value[42]);
      lcd_putint(bit_value[43]);
      lcd_putint(bit_value[44]);
      lcd_putchar(' ');
      lcd_putchar(' ');
      lcd_putchar(' ');


      lcd_command (POS_2_4);
      lcd_putchar('=');
      lcd_putchar(' ');
      lcd_putchar('D');
      lcd_putchar(' ');
      lcd_putchar(' ');
      lcd_putchar(' ');
      lcd_putchar(' ');
   }

   if(tick_pos ==45)
   {

      lcd_command (POS_1_0);
      lcd_putchar('M');
      lcd_putchar('o');
      lcd_putchar('n');
      lcd_putchar('t');
      lcd_putchar('h');
      lcd_putchar(' ');
      lcd_putchar(' ');
      lcd_command (POS_1_15);
      lcd_putchar('4');
      lcd_putchar('5');
      lcd_putchar('-');
      lcd_putchar('4');
      lcd_putchar('9');

      lcd_command (POS_2_0);
      for(i=45; i<50; i++)
      {
         lcd_putint(bit_value[i]);
      }

      lcd_command (POS_2_6);
      lcd_putchar('=');
      lcd_putchar(' ');
      lcd_putint(month10);
      lcd_putint(month1);

   }
   if(tick_pos == 50)
   {
      lcd_command (POS_1_0);
      lcd_putchar('Y');
      lcd_putchar('e');
      lcd_putchar('a');
      lcd_putchar('r');
      lcd_putchar(' ');

      lcd_command (POS_1_15);
      lcd_putchar('5');
      lcd_putchar('0');
      lcd_putchar('-');
      lcd_putchar('5');
      lcd_putchar('7');

      lcd_command (POS_2_0);
      for(i=50; i<58; i++)
      {
         lcd_putint(bit_value[i]);
      }

      lcd_command (POS_2_9);
      lcd_putchar('=');
      lcd_putchar(' ');
      lcd_putchar('(');
      lcd_putchar('2');
      lcd_putchar('0');
      lcd_putchar(')');
      lcd_putint(year10);
      lcd_putint(year1);
   }

   if(tick_pos > 58 || tick_pos < 01)
   {

      lcd_command (POS_1_0);
      lcd_putchar('M');
      lcd_putchar('i');
      lcd_putchar('n');
      lcd_putchar('u');
      lcd_putchar('t');
      lcd_putchar('e');
      lcd_putchar('M');
      lcd_putchar('a');
      lcd_putchar('r');
      lcd_putchar('k');

      lcd_command (POS_1_15);
      lcd_putchar(' ');
      lcd_putchar('5');
      lcd_putchar('9');

      lcd_command (POS_2_0);
      lcd_putchar('[');
      lcd_putchar('N');
      lcd_putchar('o');
      lcd_putchar(' ');
      lcd_putchar('p');
      lcd_putchar('u');
      lcd_putchar('l');
      lcd_putchar('s');
      lcd_putchar('e');
      lcd_putchar(']');
      lcd_putchar(' ');
      lcd_putchar(' ');
      lcd_putchar(' ');
      lcd_putchar(' ');
      lcd_putchar(' ');
      lcd_putchar(' ');
      lcd_putchar(' ');
   }
   lcd_command (c_pos[tick_pos]);        // move cursor to current bit position
}

#pragma codepage 2

/* ________________________ SET CLOCK start ________________________ */


void set_clock (void)
{
   I_SET=1;          // SET button indicator on
   I_START=0;        // START button off
   I_TRANS=0;        // TRANS button indicator on
   LED0 = 0;
   LED1 = 0;

   T_ON = 0;         // Switch off transmitter
   trans_mode=0;

   transit = 0;
   up = 0;
   down = 0;
   start_mode = 0;

   TMR1IE = 0;                // Disable TMR1 interrupt
   T1CON.0 = 0 ;    // Stop the oscillator
   TMR1IF = 0;

   RBIE   = 1;   /* local enable  */
   GIE     = 1;   /* global enable */

   char i;
   lcd_command (0b00000010);             // Cursor home
   lcd_command (0b0000.1110);            // Cursor on, blink off

   for(i=0; i<16; i++) lcd_putchar(text3(i));  // display start text 1

   display_time();
   delay10(9);

   lcd_command (POS_3_0);           // Reposition to hours position
   blink_empty (2) ;                // Blink hours

   while (set_mode ==1)
   {
      if (up ==1)
      {
         up = 0;
         blink_one ();
         hour1++;
         if(hour1>9)
         {
            hour1 = 0;
            hour10++;
         }
         if(hour10 >= 2 && hour1 > 3)
         {
           hour1=0;
           hour10=0;
         }

      }
      if (down ==1)
      {
         down = 0;
         blink_null ();
         hour1--;

         if(hour1<0)
         {
            hour1 = 9;
            hour10--;
         }

         if(hour10<0)
         {
            hour10 = 2;
            hour1 = 3;
         }

      }
      lcd_command (POS_3_0);                  // reposition to hour position ...
      lcd_putint(hour10);                     // ... and display hours
      lcd_putint(hour1);

      if (SET == 0)                           // If SET-button is pressed
      {
         set_mode = 2;                        // Move to minutes VIEW
         lcd_command (POS_3_3);               // Reposition to minutes position
         blink_empty (2);                     // Blink minutes
      }
   }
   while (set_mode ==2)
   {
      if (up ==1)
      {
         up = 0;
         blink_one ();
         min1++;

         if(min1>9)  {
            min1=0;
            min10++; }

         if(min10>5) {
            min10=0; }
      }

      if (down ==1)
      {
         down = 0;
         blink_null ();
         min1--;

         if(min1<0)  {
            min1=9;
            min10--; }

         if(min10<0) {
            min10=5; }
      }
      lcd_command (POS_3_3);                  // reposition to line 1
      lcd_putint(min10);
      lcd_putint(min1);


      if (SET == 0)
      {
         set_mode = 3;                        // Move to seconds VIEW
         lcd_command (POS_3_6);           // Goto seconds position
         blink_empty (2) ;                    // Blink seconds
      }
   }

   while (set_mode ==3)
   {
      if (up ==1)
      {
         up = 0;
         blink_one ();
         sec1++;
         if(sec1>9)
         {
            sec1 = 0;
            sec10++;
         }
         if(sec10>5)
         {
            sec10=0;
         }
      tick_pos = sec10*10 + sec1;
      }

      if (down ==1)
      {
         down = 0;
         blink_null ();
         sec1--;

         if(sec1<0)  {
            sec1=9;
            sec10--; }

         if(sec10<0) {
            sec10=5; }
      }
      tick_pos = sec10*10 + sec1;
      lcd_command (POS_3_6);                  // reposition to line 1
      lcd_putint(sec10);
      lcd_putint(sec1);

      if (SET == 0)
      {
         set_mode = 4;                        // Move to seconds VIEW
         lcd_command (POS_3_15);           // Goto seconds position
         blink_empty (1) ;                    // Blink seconds
      }
   }
   while (set_mode ==4)
   {
      if ((up ==1) || (down == 1))
      {
         up = 0;
         blink_one ();
         blink_null ();
         down = 0;

         //dst = ! dst;
         dst++;
         if(dst>1)
         {
            dst = 0;
         }
      }
      lcd_command (POS_3_15);                    // reposition to line 1
      lcd_putint(dst);


      if (SET == 0)
      {
         set_mode = 5;                          // Move to minutes VIEW
         lcd_command (POS_4_0);            // Reposition to minutes position
         blink_empty (4) ;
      }
   }


   while (set_mode ==5)
   {

      if (up ==1)
      {
         up = 0;
         blink_one ();
         year1++;
         if(year1>9)
         {
            year1 = 0;
            year10++;
         }
         if(year10>9)
         {
            year10=0;
         }
      }


      if (down ==1)
      {
         down = 0;
         blink_null ();
         year1--;
         if(year1<0)
         {
            year1 = 9;
            year10--;
         }
         if(year10<0)
         {
            year10=9;
         }
      }

      lcd_command (POS_4_0);                    // reposition to line 1
      lcd_putchar('2');
      lcd_putchar('0');
      lcd_putint(year10);
      lcd_putint(year1);

      if (SET == 0)
      {
         set_mode = 6;                          // Move to minutes VIEW
         lcd_command (POS_4_5);            // Reposition to minutes position
         blink_empty (2) ;
      }
   }


   while (set_mode ==6)
    {
      if (up ==1)
      {
         up = 0;
         blink_one ();
         month1++;
         if(month1>9)
         {
            month1 = 0;
            month10++;
         }
         if(month10==1 && month1>2)
         {
            month10 = 0;
            month1 = 1;
         }
      }

      if (down ==1)
      {
         down = 0;
         blink_null ();
         month1--;
         if(month1<0)
         {
            month1 = 9;
            month10--;
         }
         if(month10<1 && month1<1)
         {
            month10 = 1;
            month1 = 2;
         }
      }

      lcd_command (POS_4_5);                  // reposition to line 1
      lcd_putint(month10);
      lcd_putint(month1);


      if (SET == 0)
      {
         set_mode = 7;                        // Move to minutes VIEW
         lcd_command (POS_4_8);           // Reposition to minutes position
         blink_empty (2) ;
      }
   }


   while (set_mode ==7)
    {

      if (up ==1)
      {
         up = 0;
         blink_one ();
         day1++;
         if(day1>9)
         {
            day1 = 0;
            day10++;
         }
         if(day10 >2 && day1>1)
         {
            day1 = 1;
            day10 = 0;
         }
      }


      if (down ==1)
      {
         down = 0;
         blink_null ();
         day1--;
         if(day1<0)
         {
            day1 = 9;
            day10--;
         }
         if(day10<1 && day1<1)
         {
            day1 = 1;
            day10 = 3;              // Måste justeras
         }
      }

      lcd_command (POS_4_8);                  // reposition to line 1
      lcd_putint(day10);
      lcd_putint(day1);


      if (SET == 0)
      {
         set_mode = 8;                        // Move to minutes VIEW
         lcd_command (POS_4_11);           // Reposition to weekday position
         blink_empty (9) ;
      }
   }


   while (set_mode ==8)
   {

      if (up ==1)
      {
         up = 0;
         blink_one ();
         weekday++;
         if (weekday > 7)
         {
            weekday = 1;
         }
      }


      if (down ==1)
      {
         down = 0;
         blink_null ();
         weekday--;
         if (weekday <1)
         {
            weekday = 7;
         }
      }
   lcd_command (POS_4_11);
   if (weekday == 1)
   {
      lcd_putchar('M');
      lcd_putchar('o');
      lcd_putchar('n');
      lcd_putchar('d');
      lcd_putchar('a');
      lcd_putchar('y');
      lcd_putchar(' ');
   }

   if (weekday == 2)
   {
      lcd_putchar('T');
      lcd_putchar('u');
      lcd_putchar('e');
      lcd_putchar('s');
      lcd_putchar('d');
      lcd_putchar('a');
      lcd_putchar('y');
      lcd_putchar(' ');
      lcd_putchar(' ');
   }
   if (weekday == 3)
   {
      lcd_putchar('W');
      lcd_putchar('e');
      lcd_putchar('d');
      lcd_putchar('n');
      lcd_putchar('e');
      lcd_putchar('s');
      lcd_putchar('d');
      lcd_putchar('a');
      lcd_putchar('y');
   }
   if (weekday == 4)
   {
      lcd_putchar('T');
      lcd_putchar('h');
      lcd_putchar('u');
      lcd_putchar('r');
      lcd_putchar('s');
      lcd_putchar('d');
      lcd_putchar('a');
      lcd_putchar('y');
      lcd_putchar(' ');


   }
   if (weekday == 5)
   {
      lcd_putchar('F');
      lcd_putchar('r');
      lcd_putchar('i');
      lcd_putchar('d');
      lcd_putchar('a');
      lcd_putchar('y');
      lcd_putchar(' ');
      lcd_putchar(' ');
   }
   if (weekday == 6)
   {
      lcd_putchar('S');
      lcd_putchar('a');
      lcd_putchar('t');
      lcd_putchar('u');
      lcd_putchar('r');
      lcd_putchar('d');
      lcd_putchar('a');
      lcd_putchar('y');
   }

   if (weekday == 7)
   {
      lcd_putchar('S');
      lcd_putchar('u');
      lcd_putchar('n');
      lcd_putchar('d');
      lcd_putchar('a');
      lcd_putchar('y');
      lcd_putchar(' ');
      lcd_putchar(' ');
   }


      if (SET == 0)
      {
         delay10 (10);                        // Debounce
         set_mode=0;
      }
   }

   stored_time[0]=hour10;     // Update string with set time
   stored_time[1]=hour1;
   stored_time[2]=min10;
   stored_time[3]=min1;
   stored_time[4]=sec10;
   stored_time[5]=sec1;
   stored_time[6]=dst;
   stored_time[7]=year10;
   stored_time[8]=year1;
   stored_time[9]=month10;
   stored_time[10]=month1;
   stored_time[11]=day10;
   stored_time[12]=day1;
   stored_time[13]=weekday;

   put_eedata ();              // Store set time in EEPROM

   lcd_command (0b0000.0001);                 // Cursor hoME
   for(i=0; i<20; i++) lcd_putchar(text4(i));  // display text 4
   display_mode = 1;                          // display VIEW should be 1 again
   lcd_command(0b0000.1100);                  // display on, cursor off, blink off
*/


   //Lägg in senast använd display mode här

   GIE = 0;                         // Interrupts llowed

   RBIE   = 0;                     /* local enable  */
   TMR1H = 0x80;                    // Reset counter
   TMR1L = 0x00;
   TMR1IF = 0;                      // Enable interrupt
   T1CON.0 = 1 ;                    // Start the oscillator

   TMR1IE = 1;                      // Enable interrupt
   PEIE = 1;                     // Interrupts llowed
   GIE = 1;                     // Interrupts llowed

   I_SET=0;
}

void blink_one (void)
{
   LED1=1;
   delay250us(1);
   LED1=0;
}
void blink_null (void)
{
   LED0=1;
   delay250us(1);
   LED0=0;
}
```

6 Referensinformation

6.1 Protokoll DCF77

Bit	Symbol	Exempel[1)]	Beskrivning
0	M	0	**Start av minut**, alltid 0
1-14	väder	0110...	**Väder-information** från Meteo, mm
15	R	0	**Indikering av onormal sändning**, normalt 0
16	A1	0	**Annonsering av sommartid** (DST), sätts till ”1” timmen före ändring.
17-18	Z1, Z2	0 1	01 = normaltid 10 = sommartid
19	A2	0	**Annonsering av skottsekund**, sätts till ”1” timmen före ändring
20	S	1	**Indikering av start** på tidinformation, alltid ”1”.
21-27	minuter	1 0 1 0 0 0 0	**Minuter 00-59** som BCD, 7 bitar, lsb först [1-2-4-8 | 10-20-40-xx]
28	P1	0	**Paritetsbit**, jämn paritet över bit 21-28
29-34	timmar	1 0 0 0 0 1	**Timmar 00-23** som BCD, 6 bitar, lsb först [1-2-4-8 | 10-20-xx-xx]
35	P2	1	**Paritetsbit**, jämn paritet över bit 29-35
36-41	dag	0 0 1 0 0 1	**Dag 01-31** som BCD, 6 bitar, lsb först [1-2-4-8 | 10-20-xx-xx]
42-44	veckodag	1 1 1	**Veckodag** som BCD, 3 bitar, lsb först [1-2-4-x] 1 = måndag
45-49	månad	0 1 0 0 1	**Månad 01-12** som BCD, 5 bitar, lsb först [1-2-4-8 | 10-xx-xx-xx]
50-57	år	1 1 1 0 1 0 0 0	**År 00-99** som BCD, 8 bitar, lsb först [1-2-4-8 | 10-20-40-80] Århundradet ”20” är undersförstått.
58	P3	1	**Paritetsbit**, jämn paritet över bit 36-58
59	-	-	**Markering av ny minut**, ingen puls

[1)] Exempelvärden för **2017-12-24, 21:05, DST=0, söndag**