**Elektro- och informationsteknik**

# Projektrapport

## System för övervakning och styrning

**Jon Axelsson et06ja3**
**Per Söderberg et06ps2**

**2009 - 08 - 24**

## Innehållsförteckning

# Inledning

Målet med projektet har varit att bygga en prototyp för grunden till ett system för hemautomation. Det är tänkt att vara uppbyggt kring en huvudenhet som står i anslutning till en PC via serieporten samt ett antal underenheter via radio. Projektet har resulterat i färdiga prototyper av en huvudenhet med tillhörande mjukvara, två underenheter med tillhörande mjukvara samt mjukvara till en PC.

Underenheterna är tänka att placeras i närheten av det som önskas styras eller mätas. De är försedda med två analoga ingångar, tre digitala ingångar och tre digitala utgångar.

Huvudenheten skickar respektive tar emot data från underenheterna cykliskt och vid varje cykel skickar den över data till PC:n.

PC:n fungerar som användargränssnitt, här kan de digitala utångarnas värde bestämmas och mätdata kan behandlas och studeras. Allt handhavande av det färdiga systemet sker via MATLAB.

# Genomförande

## Huvudenhet

### Hårdvara

Huvudenheten är uppbyggd runt en Motorola MC68008-processor med lämpligt valda periferienheter. Fullständigt kopplingsschema finns i bilaga 1.

- Programminne - 27C256
- RAM-minne - TC551001BPL-70L
- Realtidsklocka - ICM7170
- UART - 68C681CP
- Nivåkonverterare - MAX232
- Radiomodul - OLIMEX MOD-NR24LR
- Styrlogik - PALCE22V10

Huvudoscillatorn är byggd med en 3,6864 Mhz-kristall och en inverterare, 74HC04N.Huvudoscillatorns frekvens har valts med hänsyn till UART-kretsen.

För adressering använder sig MC68008 av 20 bitar. RAM-minnet är på 128 kB alltså behöver det 17 adressbitar. Resterande 3 bitar används för att dela upp adressrymden i 8 lika stora delar. Varje periferienhet som adresserats av processorn har fått vars ett sådant block. Detta innebär att det bara är RAM-minnet som använder hela sitt block av adresser. Detta är inget problem eftersom det finns fler block än periferienheter. De två första blocken används av programminne respektive RAM-minne. Det tredje är reserverat för eventuell expansion. I efterföljande block ligger UART, RTC och SPI i uppräknad ordning.

Programminnets $\overline{CE}$ (Chip Enable) är aktivt låg och har valts att alltid vara aktiv. När data ska läsas ut från minnet används $\overline{OE}$ (Output Enable). Denna lösning gör att utläsningen av data går lite snabbare och blir enklare men i gengäld drar lite mer ström då minnet inte används.

RAM-minnet fungerar på ungefär samma sätt och har liknande anslutningar så när som på att det går att skriva till det genom att låta $R/\overline{W}$ (Read/Write) vara låg. I schemaverktyget kallas anslutningen för $\overline{WE}$ (Write Enable).

Realtidsklockan används för att förknippa samplad data med när den samplades. För att läsa vad tiden är används $\overline{RD}$ (Read), avbrott används alltså inte. När tiden ska ställas in och vid konfigurationen används $\overline{WR}$ (Write). $\overline{CS}$ (Chip Select) används som förväntat för att välja klockan när den är efterfrågad. Realtidsklockan är försedd med en kristall på 32.768 kHz och kringkomponenter enligt dess datablad för att göra denna justerbar.

UART-kretsen används huvudsakligen för seriekommunikation med PC:n. Dess anslutningar för kommunikation med processorn är $\overline{CS}$, $R/\overline{W}$ och *DTACKN* (Data Acknowledge Negative). Det finns dessutom två anslutningar för att hantera avbrott *INTERN* (Interrupt Negative) *IACKN* (Interrupt Acknowledge Negative). UART kretsen har även en inbyggd räknare/timer och en 8 bitas utgång som används för att generera signaler till styrlogiken, SPI-kommunikationskretsarna och tre lysdioder. För att UART-kretsen ska kunna kommunicera via serieporten behövs det en signalnivåkonverterare, MAX232.

För att kommunicera med radiomodulen används SPI (Serial Peripheral Interface). SPI är en standard för att kommunicera seriellt och har så kallad full duplex. Det fungerar så att en master-enhet genererar en klocksignal, *CLK* och en dataström på utsignalen kallad *MOSI* (Master Output, Slave Input). En slavenhet tar emot klocksignalen och dataströmmen samtidigt som den skickar en annan dataström på signalen kallad *MISO* (Master Input, Slave Output). Kommunikationen sker alltså i båda riktningarna samtidigt. Vidare finns en styrsignal $\overline{CS}$ som masterenheten styr, denna används huvudsakligen vid kommunikation mellan mer än 2 enheter.

För att Huvudenheten ska kunna kommunicera via SPI behövs något som implementerar denna standard. Kretsarna som används är två skiftregister och två räknare. 74LS166N är en parallell till seriellomvandlare, denna gör att data kan laddas från databussen och sedan på begäran skickas ut seriellt. Kretsens $\overline{CLEAR}$ och *CLOCK INHIBIT* är satta till fasta värden så att kretsen alltid är funktionell. Resterande signaler är den seriella utgången *Q*, den negativt flanktriggade *CLK*-ingången och *SHIFT* /$\overline{LOAD}$. *SHIFT* /$\overline{LOAD}$ bestämmer om data ska laddas eller klockas ut. 74HC595 används för omvandlingen från seriell till parallell och har insignalerna $\overline{OE}$, $\overline{SRCLR}$ (Shift

Register Clear) som är satt hög, *RCLK* (Register Clock) som uppdaterar utgångarna med vad som finns inklockat och *SRCLK* (Shift Register Clock) som klockar in den data som finns på *SER*-ingången.

För att generera en klocka till dessa två kretsar och SPI-kommumikationen med radiomodulen delas huvudoscillatorn ner med en fyrabitars räknare och resultatet blir en signal som har en frekvens som är 16 gånger lägre. Signalen används sedan till nästa räknares klockingång. Denna räknare ser till att det endast kommer 8 klockpulser på förfrågan då det är 8 bitar som ska klockas över åt gången. Båda räknarna är kopplade så att de alltid är funktionella. För att starta uppräkningen används *ENT* (Enable T) som förutom att tillåta räknaren att räkna ser till att *RCO* (Ripple-Carry Output) går att aktivera. Denna används för att stoppa räknaren. Den minst signifikanta biten från räknarens utgång används som klocka till SPI-kommunikationen. För att starta och stoppa räknaren används styrlogiken som finns beskriven på sida 5 i rapporten.

Radiomodulen har förutom de fyra tidigare nämnda SPI signalerna även *CE* och *IRQ* (Interrupt Request). *CE* bestämmer vilket tillstånd som radiomodulen befinner sig i. Tillståndsgrafen finns i databladet. *IRQ* använder radiomodulen för att uppmärksamma processorn när det hänt något.

För att realisera den styrlogik som behövs används två PAL-kretsar. De har vardera 12 ingångar och 10 anslutningar som kan konfigureras som in eller utgångar. Konfigureras en anslutning som utgång går denna att använda internt som ytterligare en insignal, vilket har utnyttjats för att göra tillstånd i logiken.

Insignaler till PAL-krets 1

| Anslutning (används i källkoden) | Enhet | Namn |
|---|---|---|
| 1 | Huvudoscillator | CLK |
| 2 | CPU | ADR 17 |
| 3 | CPU | ADR 18 |
| 4 | CPU | ADR 19 |
| 5 | CPU | $R/\overline{W}$ |
| 6 | CPU | $\overline{DS}$ (Data Strobe) |
| 7 | CPU | $\overline{AS}$ (Adress Strobe) |
| 8 | UART | INTERN |
| 9 | UART | DTACK |
| 10 | RADIO | IRQ |
| 11 | SPI | RCO** |
| 13 | SPI / UART* | START_SPI_TR** |

*Enheten ”SPI / UART” indikerar att det är en signal till SPI-kommunikationen men att den genereras från UART-kretsen.

Utsignaler från PAL-krets 1

| Anslutning (används i källkoden) | Enhet | Namn |
|---|---|---|
| 14 | CPU | $\overline{DTACK}$ |
| 15 | CPU | $\overline{IPL1}$ |
| 16 | CPU | $\overline{IPL0}/\overline{IPL2}$ |
| 17 | SPI | ENT** |
| 18 | RTC | $\overline{WR}$ |
| 19 | RCT | $\overline{CS}$ |
| 20 | RTC | $\overline{RD}$ |
| 21 | Delay RTC DTACK | DTACK_DELAY1*** |
| 22 | Delay RTC DTACK | DTACK_DELAY2*** |
| 23 | 74HC595 | *RCLK* ** |

**Vid en puls på START_SPI_TR ska det genereras 8 klockpulser till SPI-klockan. Detta ser styrlogiken till genom att starta räknaren med ENT-signalen när pulsen kommer. När räknaren har genererat 8 klockpulser till SPI:n kommer dess RCO att gå hög vilket gör att logiken kommer att inaktivera räknaren genom att låta ENT gå låg. I samband med detta kommer en aktiveringspuls på *RCLK* som uppdaterar utgångarna på skiftregistret för serie till parallellomvandlingen med den inkommen data från SPI kommunikationen. För att uppnå detta realiserades tillståndsgrafen nedan. Notera att det är ENT och *RCLK* som bestämmer tillståndet i grafen.

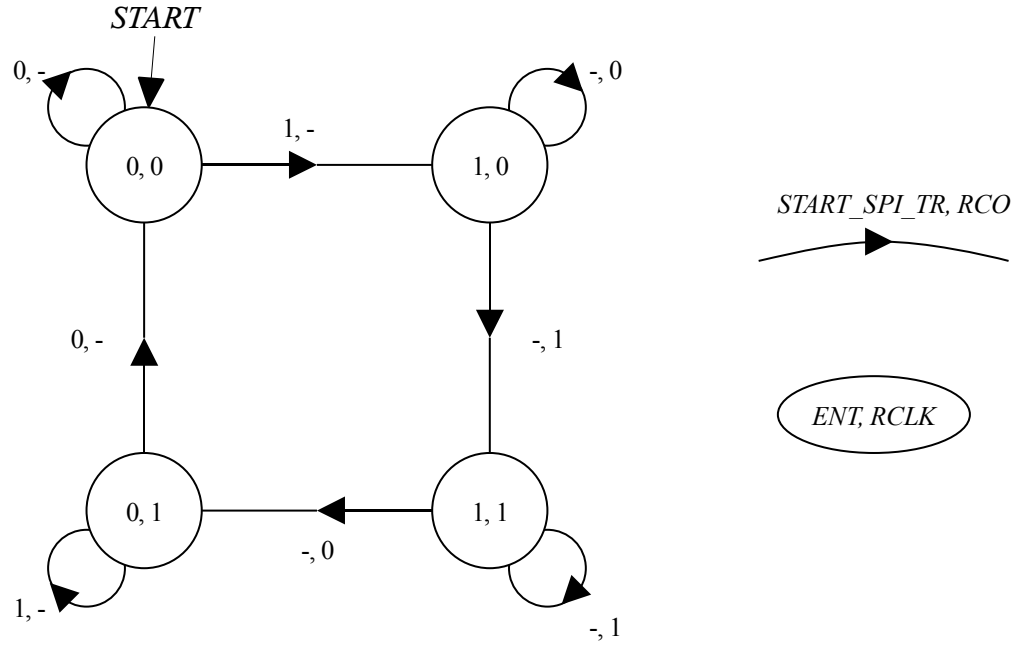



*Illustration 1: Tillståndsgraf*

***Varje gång processorn kommunicerar med en periferienhet så kommer den att få en svarssignal på ingången $\overline{DTACK}$. Eftersom realtidsklockan behöver lite mer tid på sig än övriga kretsar fördröjs svarssignalen med DTACK_DELAY1 och 2.

Insignaler till PAL-krets 2

| Anslutning (används i källkoden) | Enhet | Namn |
|---|---|---|
| 1 | CPU | ADR 1 |
| 2 | CPU | ADR 2 |
| 3 | CPU | ADR 3 |
| 4 | CPU | ADR 17 |
| 5 | CPU | ADR 18 |
| 6 | CPU | ADR 19 |
| 7 | CPU | $R/\overline{W}$ |
| 8 | CPU | $\overline{DS}$ (Data Strobe) |
| 9 | CPU | $\overline{AS}$ (Adress Strobe) |
| 10 | CPU | FC0 |
| 11 | CPU | FC1 |
| 13 | CPU | FC2 |
| 23 | SPI | SPICLK |

Utsignaler från PAL-krets 2

| Anslutning (används i källkoden) | Enhet | Namn |
|---|---|---|
| 14 | CPU | $\overline{VPA}$ |
| 15 | RAM | $\overline{OE}$ |
| 16 | RAM | $R/\overline{W}$ |
| 17 | Programminne | $\overline{OE}$ |
| 18 | SPI | $SHIFT/\overline{LOAD}$ |
| 19 | SPI | $\overline{OE}$ |
| 20 | | |
| 21 | UART | $\overline{CS}$ |
| 22 | UART | $IACKN$ |

De logiska uttrycken för PAL-kretsarna finns i bilaga 3.

**Mjukvara**

För att programmet som körs i processorn ska kunna kommunicera med sina periferienheter på ett smidigt sätt finns drivrutiner till realtidsklocka, UART och radiomodulen.

Mjukvaran för huvudenheten finns i bilaga 4 och arbetar enligt flödesschemat nedan. Vidare beskrivning av flödesschemat följer på nästa sida.

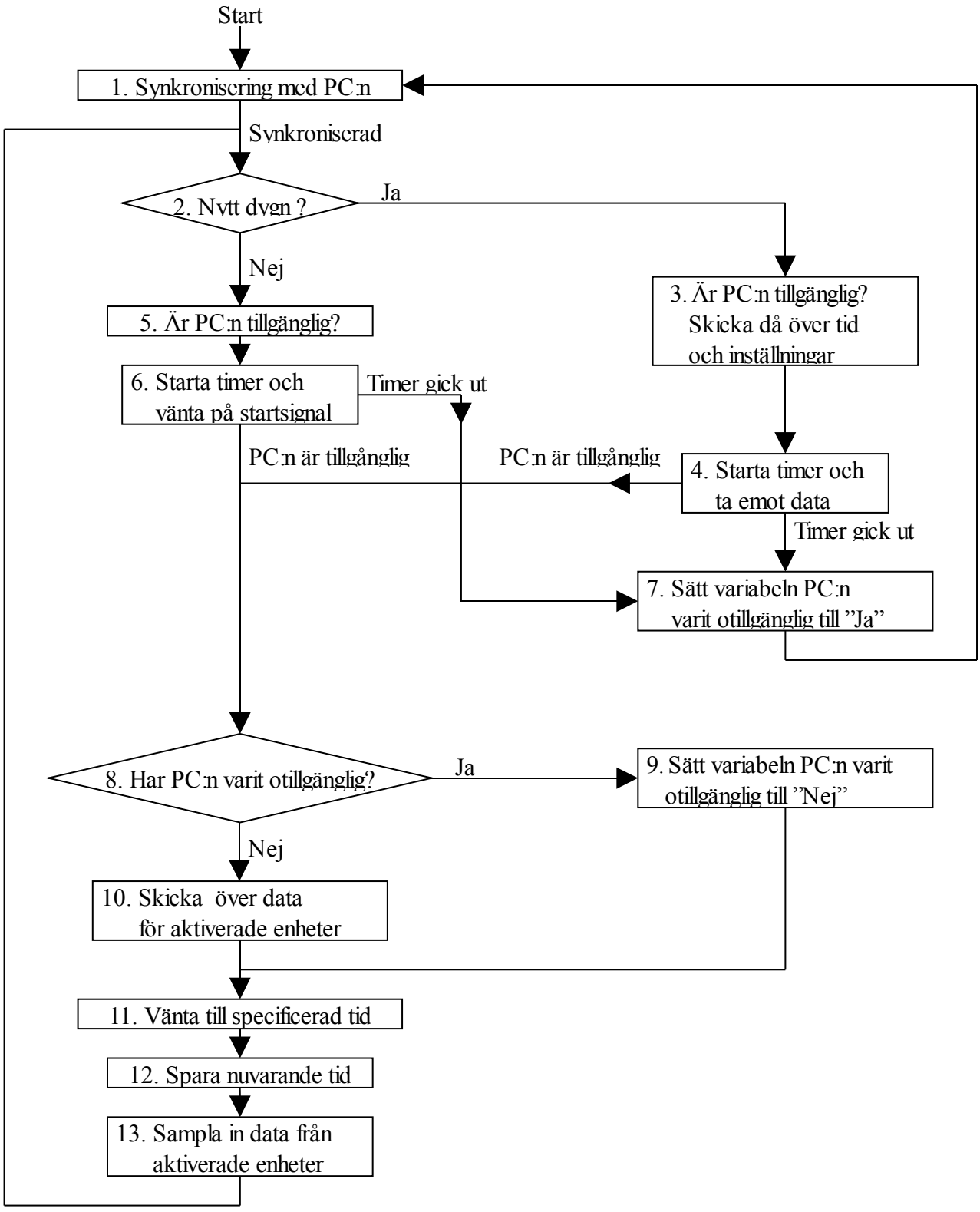



*Illustration 2: Pseudokod för huvudenheten.*

1. Vid start försöker huvudenheten synkronisera sig med PC:n för att definiera paketstorleken som används.

2. Skulle det vara ett nytt dygn så utför programmet 3. Är det inte det så utför programmet 4.

3. Här skickas först ett meddelande för kontrollera att kommunikationen med PC:n fungerar korrekt. I meddelandet skickas även information om att realtidsklockan ska synkroniseras mot PC:ns klocka samt att inställningar för underenheterna ska skickas över.

4. En timer startas och om PC:n svarar så synkroniseras realtidsklockan och inställningar överförs, sedan utför programmet 8. Går timern ut så innebär detta att seriekommunikationen men PC:n inte fungerar korrekt och 7 utförs.

5. Här skickas först ett meddelande för kontrollera att kommunikationen med PC:n fungerar korrekt.

6. En timer startas och om PC:n svara så utför programmet 8. Annars utför 7.

7. En variabel som håller reda på om kommunikationen med PC:n inte fungerat uppdateras och programmet återgår till 1.

8. Har det förekommit störningar i kommunikationen med PC:n så utförs 9. Annars utförs 10.

9. Variabeln som håller reda på om kommunikationen med PC:n inte fungerat uppdateras och programmet går vidare till 11. Anledningen till att programmet gör på detta sättet är att det inte finns någon data att skicka över till PC:n från början, innan den har samplats från underenheterna.

10. Samlad data från de aktiverade underenheterna skickas vidare till PC:n.

11. Programmet väntar in en specificerad tid innan det fortskrider till 12.

12. Ett tidssampel tas från realtidsklockan och programmet fortsätter till 13.

13. Underenheterna cyklas igenom för att sätta digitala utgångar och sampla ingångarna. När det är färdigt körs cykeln om igen från 2.

Eftersom det är tänkt att ingångarna ska samplas till exempel var 10 min kan inte den kommunikationen användas för att styra till exempel en glödlampa, det skulle ta oanvändbart lång tid. Därför kan 11 och 13 avbrytas tillfälligt för att uppdatera de digitala utgångarna för en specifik enhet. All radiokommunikations sker dock på samma sätt och finns beskriven på sida 10 i rapporten.

## Underenhet

### Hårdvara

Underenheten är uppbyggda runt en AVRMEGA16. Dess enda periferienhet är radiomodulen OLIMEX MOD-NR24LR och dess spänningsmatning som är ordnad med en LP3855ET. Som huvudoscillator används AVR:ens inbyggda oscillatior. För ytterligare detaljer och koppling se kopplingsschema i bilaga 2.

### Mjukvara

Mjukvaran för underenheten finns i bilaga 5 och arbetar enligt flödesschemat nedan. Vidare beskrivning av flödesschemat följer nedan och en beskrivning av radiokommunikationen och finns på sida 10 i rapporen.

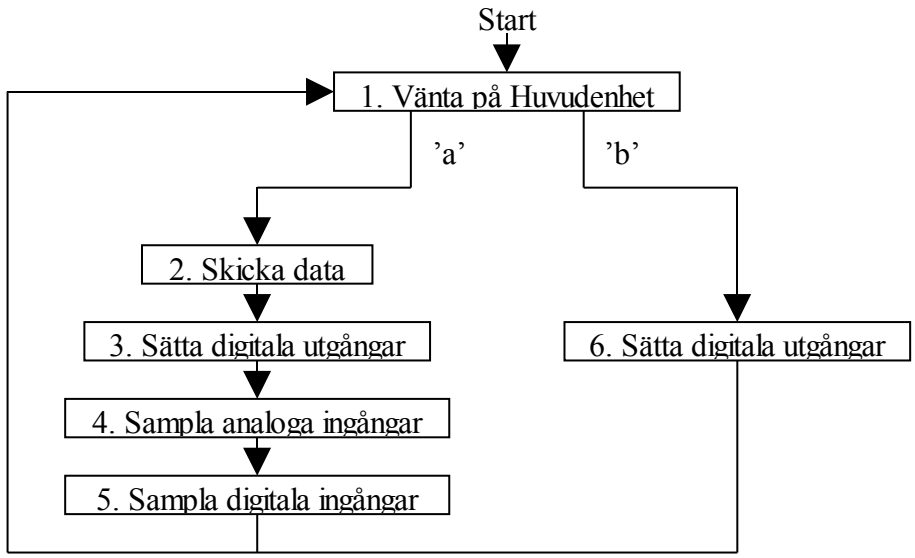



*Illustration 3: Pseudokod för huvudenheten.*

1.Vid start ställer sig underenheten och väntar på att den ska bli anropad via radio. Beroende på om första byten är tecknet *a* eller *b* så går programmet vidare till 2 eller 6.

2.Här skickas data som finns i vektorn för utgående data. Observera att den initialt är noll.

3.Här används mottagen data för att sätta underenhetens digitala utgångar till rätt värde.

4.Här samplas analoga värden med hjälp av den inbyggda AD-omvandlaren, efter samplingen filtreras värdet. Filtret finns där för att reservera kombinationerna 0x00 och 0xFF och gör att värdet 0x00 blir 0x01 och att 0xFF blir 0xFE, övriga värden lämnas oberörda. Två samplingar behöver göras då det bara finns en AD-omvandlare som delas på ingångarna. Data läggs i avsedda bytes i vektorn för utgående datapaket.

5.Här samplas värden från de digitala ingångarna. Ingång 1 läggs som lsb i avsedd byte i vektorn för utgående data, övriga ingångar läggs i bitarna 2 och 3. Övriga bitar skrivs alltid till 01010 för att reservera sekvensen 0x00. Programmet återgår sedan till 1.

6.Här används mottagen data för att sätta underenhetens digitala utgångar till rätt värde. Programmet återgår sedan direkt till 1 igen.

## Radiokommunikation

Radiomodulerna har en del automatiska funktioner som används eftersom det förenklar arbetet för processorn den är ansluten till. Till att börja med används så kallat *auto acknowledgement* så att den mottagande enheten bekräftar att den tagit emot meddelandet. Om inte *auto acknowledgement* har kommit inom 500 µs så sänds paketet om igen. Denna omsändning upprepas maximalt 15 gånger varefter sändningen betraktas som misslyckad. Paketlängden för den data som skickas är alltid inställd på fyra bytes. Eftersom radiomodulerna inte stödjer användande av 126 enheter samtidigt så löses detta genom att alla enheter får en unik adress på 5 bytes och alltid är konfigurerade som mottagare. När huvudenheten ska kommunicera med en underenhet ändrar den sin sändar- och mottagaradress och går sedan över till att vara en sändare en stund för att kommunicera med en underenhet. Huvudenheten går tillbaka till att vara mottagare samtidigt som underenheten går över till att vara sändare en stund om den har något att sända. Varje undermodul tilldelas ett tal mellan 1 och 126 som motsvarar en adressen. Eftersom det är önskvärt att adressen växlar mellan 1 och 0 ett antal gånger och inte börjar med samma bitmönster som den så kallade *preambel*:n så manchesterkodas talet och läggs i ett 32-bitarstal. Innan 32-bitarstalet läggs 0xE7. Manchesterkodningen går till enligt:

- Är msb i 16-bitarstalet 0 så läggs 01 som msb och efterföljande bit i 32-bitarstalet.
- Är msb i 16-bitarstalet 1 så läggs 10 som msb och efterföljande bit i 32-bitarstalet.

Samma mönster används för efterföljande bitar. Det är alltså till synes inte adressrymden som begränsar antalet underenheter till 126.

Paketen som skickas från huvudenheten till en underenhet via radio:

| | | | |
|---|---|---|---|
| Escape-sekvens ’a’ | Reserverad | Reserverad | Digitala utgångar |
| Escape-sekvens ’b’ | Reserverad | Reserverad | Digitala utgångar |

Paketet som skickas från en underenhet till huvudenheten via radio:

| | | | |
|---|---|---|---|
| Escape-sekvens ’A’ | A/D-värde på ingång 1 | A/D-värde på ingång 2 | Digitala ingångar |

## PC

PC:n skulle som tidigare nämnt fungera som användargränssnitt. Via programmet skulle man ställa in:

- Vilka underenheter som huvudenheten ska förvänta sig är aktiva.
- Vad som är anslutet till underenheternas ingångar.
- Vilket värde underenheternas utgångar ska ha.

Det skulle dessutom vara flexibelt för att göra till exempel olika sorters plottar. På grund av sin flexibilitet skulle programmet oavsett hur det skrevs bli ganska avancerat ur användarsynpunkt. Därför föll valet på ett program där funktionen begränsas huvudsakligen av användarens kreativiteten, men dock till viss del även kunskap, MATLAB.

Programmet som konstruerats för detta projektet är ett ett fönsterbaserat program med två stycken areor för att rita grafer, en för analoga ingångar och en för digitala ingångar. I fönstret finns även en del där man gör inställningar för varje enhet och kopplar upp och ned kommunikationen med huvudenheten. Under tiden programmet körs sparas insamlad data i en fil och när det stängs av sparas inställningarna för var enhet i en annan fil. Nästa gång programmet startar läses inställningarna och data in och graferna ritas på nytt.

Programmet använder sig av MATLAB:s inbyggda funktioner för att skapa ett GUI (Guided User Interface), skriva och läsa från filer och kommunicera via serieporten. När programmet startar skapar det användargränssnittet och läser in data från filerna. Resten av exekveringen är händelsestyrd vilket innebär att programmet börja exekvera kod när användaren trycker på en knapp eller huvudenheten kommunicerar via serieporten. I bilaga 4 finns hela källkoden för MATLAB-programmet.

**Seriekommunikation**

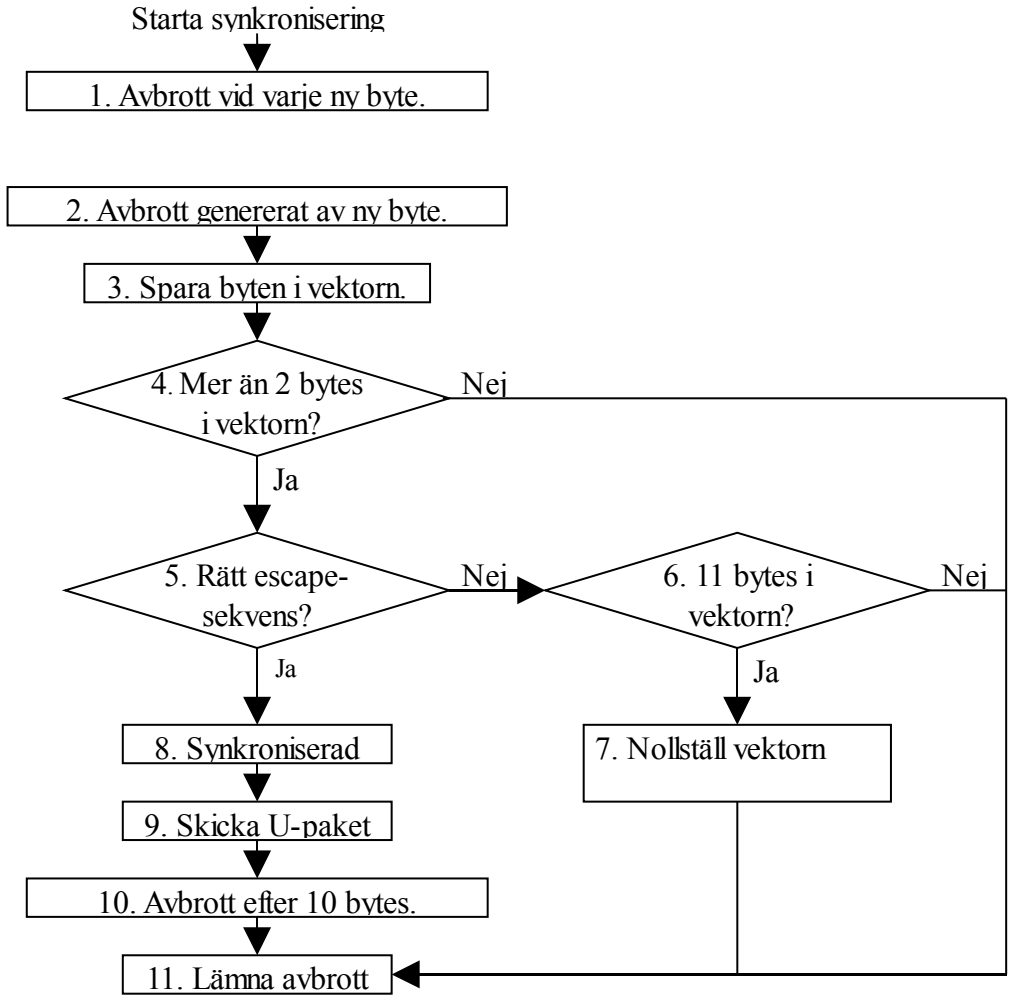



*Illustration 4: Pseudokod för synkronisering med MATLAB.*

1.En förutsättning för att synkroniseringen ska fungera är att det genereras ett avbrott vid varje ny byte som tas emot via serieporten. Därför görs här inställningar om att det ska genereras ett avbrott vid varje ny byte.

2.När en ny byte kommit genereras ett avbrott och programmet kommer hit. Programmet fortsätter till 3.

3.Den nya byten läggs till sist i en vektor varefter programmet fortsätter till 4.

4.Är det mer än två bytes i vektorn utförs 5 annars utförs 11.

5.Är det rätt escape-sekvens (som är 0xFF följt av 0x00) utförs 8 annars utförs 6.

6.Är det mer än 11 bytes i vektorn utförs 7 annars utförs 11.

7.Innehållet i vektorn raderas för att den inte ska bli för stor . Därefter går programmet vidare till 11.

8.MATLAB är nu synkroniserat med huvudenheten och programmet fortsätter till 9.

9.Ett U-paket skickas från MATLAB för att tala om för huvudenheten att de är synkroniserade.

10.Eftersom enheterna nu är synkroniserade ändras paketstorleken till 10 bytes (8 bytes data och 2 bytes escape-sekvens).

11.Avbrottsrutinen lämnas och programmet återgår till där det blev avbrutet.

För att kunna kommunicera via serieporten skapades ett enkelt protokoll som följs efter det att huvudenheten och PC:n har synkroniserats.

Paket som skickas från huvudenheten till PC:n:

| Uppgift | Byte 0 | Byte 1 | Byte 2 | Byte 3 | Byte 4 | Byte 5 | Byte 6 | Byte 7 |
|---|---|---|---|---|---|---|---|---|
| PC tillgänglig | 'A'+100 | 0x5F – begär tid<br>0x50 – inget | 0x5F – begär inställningar<br>0x50 – inget | | | | | |
| Tidsstämpel | 'T'+100 | Sec | Min | H | | | | |
| Data paket | Första enheten | A/D 1 | A/D 2 | Dig. in | Andra enheten | A/D 1 | A/D 2 | Dig. In |

Dessutom läggs en escape-sekvens bestående av två bytes, 0xFF och 0x00, efter vart paket så att PC:n kan kontrollera så att allting fortfarande är synkroniserat.

Paket som skickas från PC:n till huvudenheten:

| Uppgift | Byte 0 | Byte 1 | Byte 2 | Byte 3 | Byte 4 | Byte 5 | Byte 6 | Byte 7 |
|---|---|---|---|---|---|---|---|---|
| Inställningar | A | Enhet 1 | Enhet 2 | Enhet 3 | Enhet 4 | Enhet 5 | Enhet 6 | Enhet 7 |
| Inställningar | B | Enhet 8 | Enhet 9 | Enhet 10 | Enhet 11 | Enhet 12 | Enhet 13 | Enhet 14 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| Inställningar | R | Enhet 120 | Enhet 121 | Enhet 122 | Enhet 123 | Enhet 124 | Enhet 125 | Enhet 126 |
| Sätt utgångar | S | Enhet | Data | | | | | |
| Tid | T | Sec | Min | H | Dag | Mån | År | Starta klockan |
| PC synkroniserad | U | U | U | U | U | U | U | U |
| PC inte synkroniserad | V | V | V | V | V | V | V | V |
| Startsignal | W | W | W | W | W | W | W | W |

## Resultat och utvärderande diskussion

Resultatet uppfyller kort sagt de krav som ställs på prototypen i kravspecifikationen. Under projektets gång när resultatet närmat sig steg för steg visade det sig dock att den ursprungliga kravspecifikationen inte var helt perfekt utan kommer att kompletteras för en eventuell nyare version.

Denna projektkursen har inneburit praktiskt arbete i form av planerande, konstruerande och montering av hårdvara följt av skrivande av mjukvara samt sist men absolut inte minst, felsökning av både hårdvara och mjukvara. Studenter med mänskliga egenskaper som läser denna kursen kommer lära sig mycket förutom den ovärderliga erfarenheten av att ha genomfört ett projekt av denna typ. Detta kan till exempel vara vikten av noggrannhet vid

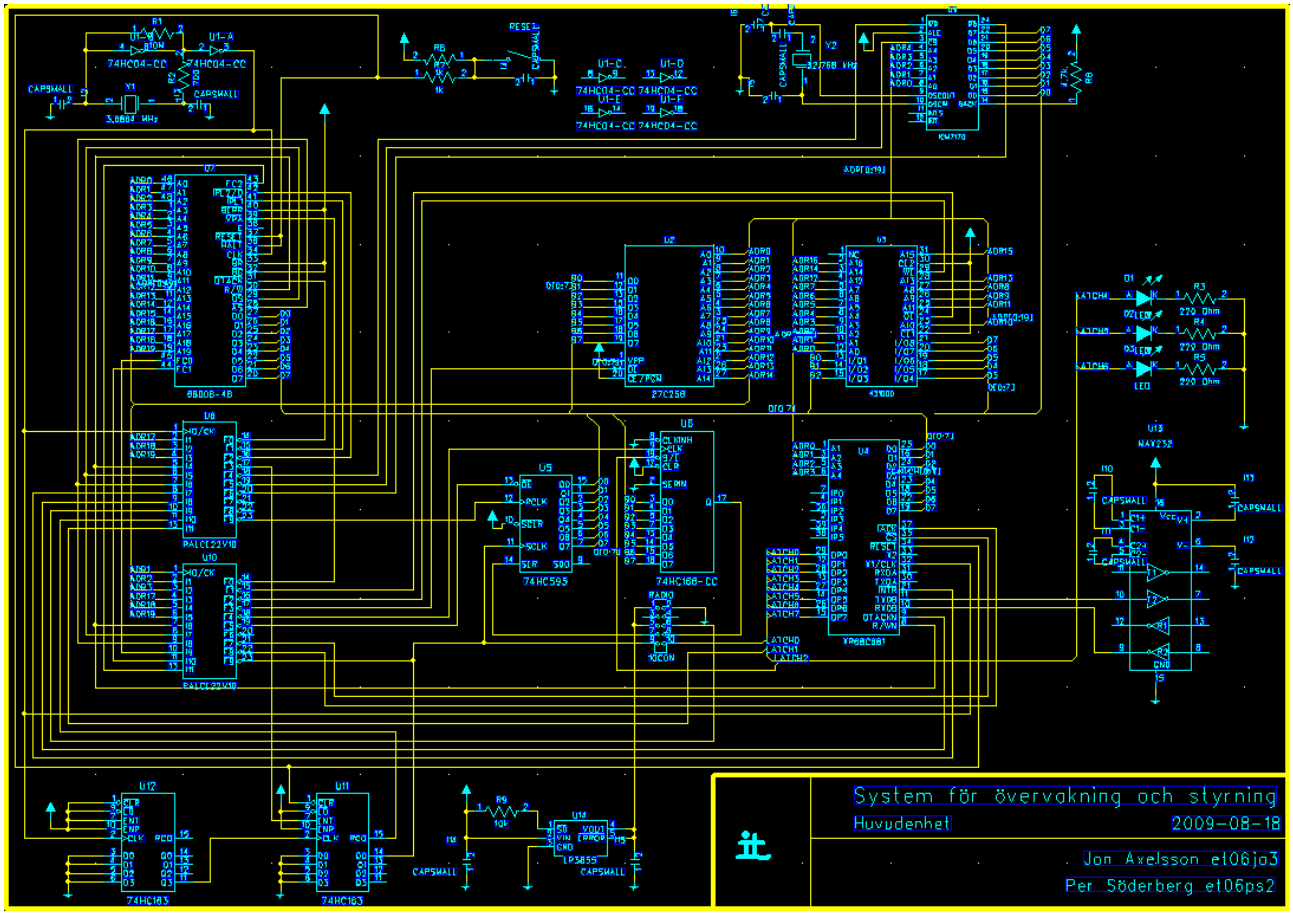
- Databladsläsning

- Planering av hårdvara och mjukvara

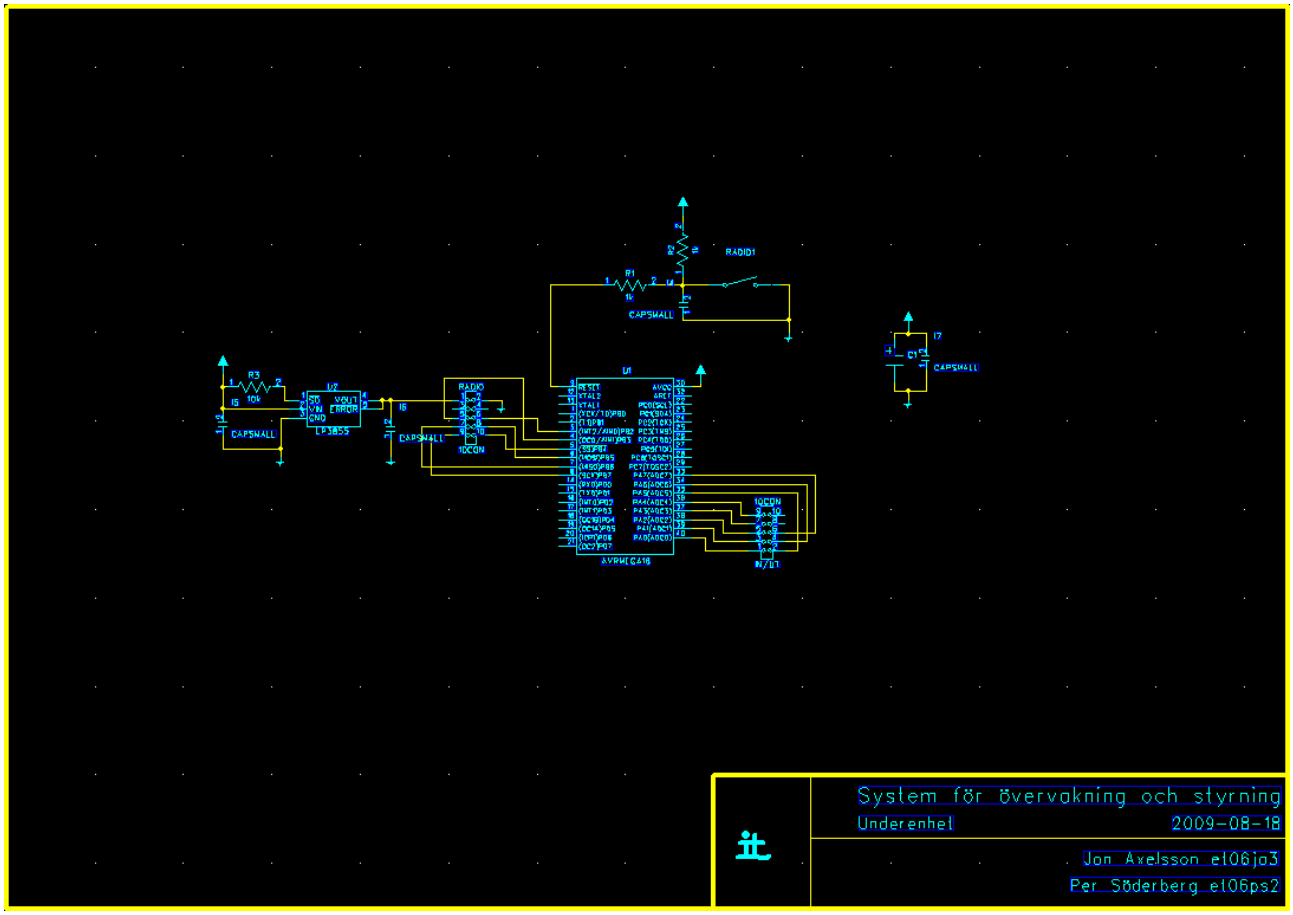
- Montering av hårdvara

För att inte tala om fördelarna med samarbete som bland annat innebär att många misstag i planeringen och fel helt kan undvikas och att tiden det tar att identifiera och åtgärda ett fel kraftigt kan minskas även om denna är mycket oberäknelig vilket såklart påverkar hela tidsplanen.

# Bilagor




*Bilaga 1: Kopplingsschema för huvudenhet.*




*Bilaga 2: Kopplingschema för underenhet.*

Bilaga 3: Källkod för PAL-kretsar

PAL1.pld

```
Title             JAPS
Pattern                 Pal1
Revision          0
Author                  Per
Date              27-02-09
device 22v10
CLK          1
A17          2
A18          3   'Adressbitar
A19          4
RWN          5   'Read/Write from CPU
DSN          6   'Data strobe from CPU
ASN          7   'Adress strobe from CPU
INTRN        8   'Interrupt from UART
DTACKN             9   'Data ack. from UART
IRQ          10   'Interrupt radio
RCO          11   'RCO from counter for SPI clock
GND          12
STARTS             13   'Start SPI Transfer control signal from CPU (via UART)
DTACK        14   'DTACK to CPU
IPL1         15   'Interrupt input to CPU
IPL02        16   'Interrupt input to CPU
ENT          17   'Enable SPI
WRN          18   'Write to RTC
CSN          19   'Chip select to RTC
RDN          20   'Read to RTC
DTACKD1            21
DTACKD2            22
Q            23   'Tillståndsvariabel
start
DTACK /= /A18 * /A19 * /ASN + A17 * /A18 * A19 * RWN * /ASN + A17 * /A18 * A19
* /RWN * /DSN * /ASN + /DTACKN + DTACKD2;
IPL1 /= INTRN * /IRQ;
IPL02 = INTRN;
ENT := /Q * STARTS + ENT * RCO + ENT * /Q;   '/ENT * STARTS + ENT * /RCO;
WRN /= /A17 * /A18 * A19 * /RWN * /DSN * /ASN;
CSN /= /A17 * /A18 * A19 * /ASN;
RDN /= /A17 * /A18 * A19 * RWN * /ASN;
DTACKD1 := /CSN;
DTACKD2 := /WRN + DTACKD1;       '
Q := Q * STARTS + ENT * RCO + ENT * Q;
end
```

```
PAL2.pld

Title             JAPS
Pattern                Pal2
Revision          0
Author                 Per
Date              27-02-09
device 22v10
A1                1
A2                2
A3                3
A17               4
A18               5  'Adressbitar
A19               6
RWNC              7  'Read/Write from CPU
DSN               8  'Data strobe from CPU
ASN               9  'Adress strobe from CPU
FC0               10
FC1               11
GND               12  'Processor status from CPU
FC2               13
VPA               14  'Autovector
OENR              15  'Output enable to RAM
WRN               16  'Write enable to RAM
OENE              17  'Output enable to EPROM
SPICLK2                18  'SPI clock to parallell in ser. out   Shitft/loadN
to SPI
OENS              19  'Output enable to SPI
RWNU              20  'Read/write to UART
CSN               21  'Chip selcet to UART
IACKN             22  'Interrup ack. to UART
SPICLK                 23

start
VPA /= /A1 * A2 * /A3 * /ASN * FC0 * FC1 * FC2;
OENR /= A17 * /A18 * /A19 * RWNC * /ASN;
WRN /= A17 * /A18 * /A19 * /RWNC * /DSN * /ASN;
OENE /= /A17 * /A18 * /A19 * RWNC * /ASN;
SPICLK2 /= A17 * /A18 * A19 * /RWNC * /DSN * /ASN + SPICLK;
OENS /= A17 * /A18 * A19 * RWNC * /ASN;
RWNU = A17 * A18 * /A19 * RWNC * /ASN;
CSN /= A17 * A18 * /A19 * /DSN * /ASN;
IACKN /= A1 * /A2 * A3 * /ASN * FC0 * FC1 * FC2;
end
```

Bilaga 4: Källkod för MC68008 huvudenhet

main.c

```
/*************************************************************************
                            main.c
                            ------
   Detta ar huvudprogrammet som ar skrivet i ANSI C. Exekveringen av hela
   programpaketet borjar i pmain.68k (lage __main).

   exp4() anropas fran assemblyprogrammet exp4.68k vid avbrott.

   _avben() anropar avben.68k vilket tillater avbrott fran PI/T.
*************************************************************************/


#include "uart_spi.h"
#include "rtc.h"
#include "radio.h"

unsigned short    lastdata[126][3], /*Collected data from the slave units*/
                        timea[3],   /*Two arrays to store timemarks*/
                        timeb[3],
                        packet[4],
                        units[126]; /*Setings: 7:EN unit, 6:Enhet kontaktbar,
                                               5:EN ad1, 4:EN ad2, 3:reserverad,
                                               2-0:data to digital out */

unsigned short    timearray,              /*Specifies witch timearray to use*/
                        unit,
                        u_base,           /*Varaiabels used in UART interrupt*/
                        u_recived,
                        u_recive_data,/*7:4-Type of data, 3:0-Nbrof bytes left*/
                        u_update_unit,
                        u_counter_int,
                        u_sent_unit,
                        r_status,   /*Statusregister in radiomodule*/
                        x_status,   /*7:3-reserved, 2-radiotransmission faild,
                                        1-radiotransmission succsseded,
                                        0-radiomodule recived a packet */
                        x_returned,
                        old_sampletime,
                        start_communication,
                        pc_con,                         /*PC is connected*/
                        ask_pc;                         /*Send qustion to PC*/

unsigned short addr;

void delay()
{     /* long delay */
      int mips, avr;
      for(mips=0; mips<100; mips++){
            for(avr=0; avr<255; avr++){
            }
      }
}

void delay2()
{     /* short delay */
      int mips, avr;
      for(mips=0; mips<10; mips++){
                  for(avr=0; avr<255; avr++){
                  }
      }
}

unsigned short examin_trans()
{     /* Examins x_status to determin what hapend with the radiotransmission. */
      if(x_status & 2){
```

```
            x_status = x_status & 0xFD;
            return 1;
      }

      if(x_status & 4){
            x_status = x_status & 0xFB;
            return 2;
      }
      return 0;
}

unsigned short examin_result()
{     /* Examins x_status to determin if the radiomodule recived a packet. */
      if(x_status & 1){
            x_status = x_status & 0xFE;
            return 1;
      }
      return 0;
}

void update_unit()
{     /* Updates the output of a slaveunit specified by u_update_unit */
      set(6);
      while(u_recive_data & 0x0F){ }
      CE_LOW();
      /*Adresserna går från 1 till 126*/
      radio_change_addr(RX_ADDR_P0, u_update_unit);
      radio_config_tx();
      packet[0] = 'b';
      packet[3] = units[u_update_unit-1]  & 0x07;
      x_status=0;
      radio_transmit(&packet[0]);
      while(!x_status){ }
      x_returned = examin_trans();
      switch(x_returned){
            case 1:
                        /* Succsseded */
                        reset(6);
                        break;
            case 2:
                        /* Unit not found. */
                        break;
      }
      u_update_unit=0;
}

void init_main()
{     /* Initialaze variabels */
      packet[0] = 0;
      packet[1] = 1;
      packet[2] = 2;
      packet[3] = 3;
      for(unit=0; unit<126; unit++){
            units[unit]=0;
      }
      timearray=0;
      unit=0;
      u_base=0;
      u_recived=0;
      u_recive_data=0;
      u_update_unit=0;
      u_counter_int=0;
      u_sent_unit=0;
      r_status=0;
      x_status=0;
      x_returned=0;
      start_communication=0;
      pc_con=0;
      ask_pc=0;
      addr=0;
```

```
}

void main()
{
      init_main();

      init_uart_spi();
      init_RTC();
      sec = 00;
      min = 15;
      h = 0;           /* This will make the baseunit to ask the PC for time
                          and settings when the syncronisation is finished.  */
      day = 26;
      month = 3;
      year = 9;
      set_time();
      start_RTC();

      /* Blink with status LED:s at startup. */
      set(4);
      set(5);
      set(6);
      delay();
      reset(4);
      reset(5);
      reset(6);

      /* Configure radiomodule. */
      CE_LOW();
      radio_setup(4, 17);
      radio_change_addr(RX_ADDR_P0, addr);
      radio_config_rx();

      CE_HIGH();        /* Set radiomodule in RX-mode. */

      _avben();         /* Enable interrupt */

      greta:
            if(pc_con == 0){  /* Send sync signlas */
                  set(4);
                  delay();
                  transmitt(0xFF);
                  delay2();
                  transmitt(0);
                  delay();
                  delay();
                  reset(4);
                  delay();
                  delay();
            }

            if(ask_pc){
                  delay();
                  delay();
                  transmitt('A'+100);
                  if(h == 0){
                        transmitt(0x5F); /*Ask for time. */
                        delay2();
                        transmitt(0x5F); /*Ask for all settings for all units.*/
                  }else{
                        transmitt(0x50);  /*Do not ask for time. */
                        delay2();
                        transmitt(0x50);  /*Do not ask for settings. */
                  }
                  transmitt(0x50);
                  delay2();
                  transmitt(0x50);
                  transmitt(0x50);
                  delay2();
                  transmitt(0x50);
```

```
            transmitt(0x50);
            delay2();
            transmitt(0xFF);
            transmitt(0);
            delay2();

            /* Set and start timer and wait for PC to answer.
               This is done two times to get enough delay to
               transfer all the settings if that is needed.  */
            set_counter(0xFF, 0xFF);
            start_counter();
            while(!start_communication && !u_counter_int){ }
            stop_counter();
            if(u_counter_int){
                  pc_con=0;
                  u_counter_int=0;
            }
            set_counter(0xFF, 0xFF);
            start_counter();
            while(!start_communication && !u_counter_int){ }
            stop_counter();
            if(u_counter_int){
                  pc_con=0;
                  u_counter_int=0;
            }
            ask_pc=0;
      }

      if(start_communication){
            if(pc_con>1){  /* Send time stamp and collected data to PC */
                  transmitt('T'+100);
                  if(timearray){
                        transmitt(timea[0]);
                  delay2();
                        transmitt(timea[1]);
                        transmitt(timea[2]);
                  }else{
                        transmitt(timeb[0]);
                  delay2();
                        transmitt(timeb[1]);
                        transmitt(timeb[2]);
                  }
                  delay2();
                  transmitt(0);
                  transmitt(0);
                  delay2();
                  transmitt(0);
                  transmitt(0);
                  delay2();
                  transmitt(0xFF);
                  transmitt(0);
                  delay2();
                  u_sent_unit=0;
                  for(unit=0; unit<126; unit++){
                        if(units[unit] & 0x80){
                              transmitt(unit+1);
                              transmitt(lastdata[unit][0]);
                              delay2();
                              transmitt(lastdata[unit][1]);
                              transmitt(lastdata[unit][2]);
                              delay2();
                              u_sent_unit++;
                        }
                        if(u_sent_unit == 2){
                              transmitt(0xFF);
                              transmitt(0);
                              delay2();
                              u_sent_unit=0;
                        }
                  }
```

```
      if(u_sent_unit){
            transmitt(0);
            transmitt(0);
            delay2();
            transmitt(0);
            transmitt(0);
            delay2();
            transmitt(0xFF);
            transmitt(0);
            delay2();
            u_sent_unit=0;
      }
}else{      /* Send time and data next time */
      pc_con = 2;
}

/* Wait for the next time to collect data from slave units */
while(sec%5 | old_sampletime == sec){
      set_counter(0x3F, 0xFF);
      start_counter();
      while(!u_counter_int){
            if(u_update_unit){  /* Uppdate the unit. */
                  update_unit();
            }
      }
      u_counter_int=0;
      get_time();
}
old_sampletime = sec;
/* Make a new timestamp. */
get_time();
if(timearray){
      timeb[0] = sec;
      timeb[1] = min;
      timeb[2] = h;
      timearray=0;
}else{
      timea[0] = sec;
      timea[1] = min;
      timea[2] = h;
      timearray=1;
}

/* Cycle through all units and
   sample those that are activiated. */
set(5);
reset(6);
for(unit=0; unit<126; unit++){
      if(units[unit] & 0x80){
            CE_LOW();
            /* Adresses is from 1 to 126 */
            radio_change_addr(RX_ADDR_P0, unit+1);
            radio_config_tx();
            packet[0] = 'a';
            packet[3] = units[unit]  & 0x07;
            x_status=0;
            radio_transmit(&packet[0]);
            while(!x_status){}
            x_returned = examin_trans();
            switch(x_returned){
                  case 1:
                        set_counter(0x33, 0x33);
                        start_counter();
                        while(!x_status && !u_counter_int){}
                              stop_counter();
                        if(u_counter_int){
                              set(6);
                              /*
                              Unit not found. Ack recived but
                              no answer before timer ran out.
```

```
                                                */
                                                u_counter_int=0;
                                          }else{
                                                if(examin_result()){
                                                      if(packet[0] == 'A'){
                                                            /*Data processing.*/
                                                            lastdata[unit][0] =
                                                                  packet[1];
                                                            lastdata[unit][1] =
                                                                  packet[2];
                                                            lastdata[unit][2] =
                                                                  packet[3];
                                                      }else{
                                                            set(6);
                                                            /*
                                                            Answer but no A
                                                            in first byte.
                                                            */
                                                      }
                                                }else{
                                                      set(6);
                                                      /* Unit not responding,
                                                         but it sent an ack
                                                         recently. */
                                                }
                                          }
                                          break;
                                    case 2:
                                          set(4);
                                          /* Unit not found. */
                                          break;
                                    }
                                    if(u_update_unit){
                                          update_unit();    /* Uppdate the unit. */
                                    }
                              }
                        }
                        reset(5);

                        ask_pc=1;
                        start_communication=0;
                  }     /* start_communication */

      goto greta;         /* Endless loop */

}

int exp2()
{   /* Interruptprogram for radiomodule*/
      CE_LOW();
      r_status = radio_status();
      if(r_status & RX_DR){   /* Recived a packet */
            radio_recieve(&packet[0]);
            radio_rw_combo(W_REGISTER, STATUS, RX_DR);
            x_status = x_status | 1;
      }

      if (r_status & TX_DS) { /* Succsseded to transmitt a packet. */
            radio_rw_combo(W_REGISTER, STATUS, r_status|TX_DS);
            x_status = x_status | 2;
      }

      if (r_status & MAX_RT) {      /* Faild to transmitt a packet. */
            radio_rw_combo(W_REGISTER, STATUS, r_status|MAX_RT);
            radio_rw(FLUSH_TX, NOP);
            x_status = x_status | 4;
      }

      if (r_status & TX_FULL) {     /* Faild to transmitt a packet
                                       three times in a row.        */
```

```
            radio_rw(FLUSH_TX, NOP);
            x_status = x_status | 4;
      }
      radio_config_rx();
      CE_HIGH();
      return(0);
}

int exp5()
{     /* Interruptprogram for UART*/
      unsigned short exeption = get_exeption();
      if(exeption & CHBRR){   /* Sign recived */
            u_recived = recive();
            if(u_recive_data & 0x0F){
            /* The byte belongs to the current packet that is being recived. */
                  switch(u_recive_data & 0xF0){
                        case 0x10:        /* A...R-packet */
                              units[u_base-(u_recive_data & 0x0F)] = u_recived;
                              break;
                        case 0x20:        /* S-packet */
                              if((u_recive_data & 0x0F) == 0x07){
                                    u_update_unit = u_recived;
                              }else if((u_recive_data & 0x0F) == 0x06){
                                    units[u_update_unit-1] = u_recived;
                              }
                              break;
                        case 0x30:        /* T-packet */
                              switch(u_recive_data & 0x0F){
                                    case 0x07:
                                          sec = u_recived;
                                          break;
                                    case 0x06:
                                          min = u_recived;
                                          break;
                                    case 0x05:
                                          h = u_recived;
                                          break;
                                    case 0x04:
                                          day = u_recived;
                                          break;
                                    case 0x03:
                                          month = u_recived;
                                          break;
                                    case 0x02:
                                          year = u_recived;
                                          break;
                                    case 0x01:
                                          set_time();
                                          start_RTC();
                                          break;
                                    }
                                    break;
                  }
                  u_recive_data--;
            }else{
                  if(u_recived >= 'A' && u_recived <= 'R'){
                        u_recive_data = 0x17;
                        u_base = 7*(u_recived - 'A')+7;
                  }else if(u_recived == 'S'){
                        u_recive_data = 0x27;
                  }else if(u_recived == 'T'){
                        u_recive_data = 0x37;
                  }else if(u_recived == 'U'){
                        if(!pc_con){
                              pc_con=1;
                        }
                        ask_pc=1;
                  }else if(u_recived == 'V'){
                        pc_con=0;
                  }else if(u_recived == 'W'){
```

```
                        start_communication=1;
                }
        }

    }
    if(exeption & CR){            /* Timer ran out. */
        stop_counter();
        u_counter_int=1;
    }
    if(exeption & CBTR){    /* Channel B transmitter ready */
        if(in_txfifo){
            transmitt_fifo();
        }else{
            stop_tx_int();
        }
    }
    return(0);
}
```

uart_spi.h

```
#ifndef UART_SPI
#define UART_SPI

	#define	CR		0x08
	#define CBTR	0x10
	#define	CHBRR 0x20

	extern unsigned short in_txfifo;

	void init_uart_spi();

	void transmitt_fifo();

	void transmitt(char data);

	void stop_tx_int();

	unsigned short recive();

	unsigned short get_exeption();

	void set(short pin);

	void reset(short pin);

	void reset_all();

	unsigned short start_counter();

	unsigned short stop_counter();

	void set_counter(unsigned short upper, unsigned short lower);

	unsigned short spi_tr(unsigned short data);

	void spi_delay();

	void CE_puls();

#endif
```

uart_spi.c

```
#include "uart_spi.h"

#define INT_MASK (CHBRR | CBTR | CR)   /*0b00111000;*/
#define INT_MASK2 (CHBRR | CR)
#define TxRDY_MASK 0x04

/*                                          write_read
                    Address*/
#option sep_on class uart_pekare
static unsigned short int     *MRA_MRA    = (unsigned short int *) 0x60000,
                              *CSRA_SRA   = (unsigned short int *) 0x60001,
                              *CRA        = (unsigned short int *) 0x60002,
                              *THRA_RHRA  = (unsigned short int *) 0x60003,
                              *ACR_IPCR   = (unsigned short int *) 0x60004,
                              *IMR_ISR    = (unsigned short int *) 0x60005,
                              *CTUR_CTU   = (unsigned short int *) 0x60006,
                              *CTLR_CTL   = (unsigned short int *) 0x60007,
                              *MRB_MRB    = (unsigned short int *) 0x60008,
                              *CSRB_SRB   = (unsigned short int *) 0x60009,
                              *CRB        = (unsigned short int *) 0x6000A,
                              *THRB_RHRB  = (unsigned short int *) 0x6000B,
                              *IVR_IVR    = (unsigned short int *) 0x6000C,
                              *OPCR_IP    = (unsigned short int *) 0x6000D,
                              *RESETOP    = (unsigned short int *) 0x6000E,
                              *SETOP      = (unsigned short int *) 0x6000F;


static unsigned short int     *SPI        = (unsigned short int *) 0xA0000;
#option sep_off

unsigned short in_txfifo, txdata[8];

void CSN_LOW()
{
      *RESETOP = 0x01;
}

void CSN_HIGH()
{
      *SETOP = 0x01;
}

void CE_LOW()
{
      *RESETOP = 0x08;
}

void CE_HIGH()
{
      *SETOP = 0x08;
}

void init_uart_spi()
{
      *IMR_ISR    = 0;  /*No interrupts*/
      *CRA        = 0x10;     /*Reset MR pointer*/
      *CRB        = 0x10;
      *CRA        = 0x20;     /*Reset reciver*/
      *CRB        = 0x20;
      *CRA        = 0x30;     /*Reset transmitter*/
      *CRB        = 0x30;

      *MRA_MRA    = 0x13; /*Set OP0 to output, recive ready interrupt,
                                    no parity, 8 bits per character*/
      *MRA_MRA    = 0x07;     /*Normal channel mode, 1 stop bit*/
      *CSRA_SRA   = 0xBB; /*9600 baud rate recive and transmitt*/

      *MRB_MRB    = 0x13; /*Set OP1 to output, recive ready interrupt,
```

```
                                    no parity, 8 bits per character*/
      *MRB_MRB    = 0x07;     /*Normal channel mode, 1 stop bit*/
      *CSRB_SRB   = 0xBB; /*9600 baud rate recive and transmitt*/

      *OPCR_IP    = 0;  /*Set OP2-OP7 to outputs*/
      *ACR_IPCR   = 0x30;     /*Counter clock source 1x clock of
                                    Channel B transmitter*/

      *IVR_IVR    = 0x40;     /*Interrupt vector*/
      *IMR_ISR    = INT_MASK;  /*Interrupt mask register*/

      *CRB        = 0x05; /*Start channel B riciver and transmitter*/
      reset_all();
      in_txfifo   = 0;
      *SETOP            = 0x01; /*CSN radiomodul*/
}

void transmitt_fifo()
{
      unsigned short i;
      *THRB_RHRB = txdata[0];
      in_txfifo--;
      for(i=0; i<in_txfifo; i++){
            txdata[i] = txdata[i+1];
      }
}

void transmitt(char data)
{
      if(*CSRB_SRB | TxRDY_MASK){
            if(in_txfifo){
                  if(in_txfifo <= 8){
                        txdata[in_txfifo++] = data;
                        transmitt_fifo();
                  }else{
                        /* signalera att USART-kommunikationen är dålig,
                           nere eller slö. */
                  }
            }else{
                  *THRB_RHRB = data;
            }
      }else{
            if(in_txfifo){
                  if(in_txfifo <= 8){
                        txdata[in_txfifo++] = data;
                  }else{
                        /* signalera att USART-kommunikationen är dålig,
                           nere eller slö. */
                  }
            }else{
                  txdata[in_txfifo++] = data;
                  *IMR_ISR    = INT_MASK;
            }
      }
}

void stop_tx_int()
{
      *IMR_ISR    = INT_MASK2;
}

unsigned short recive()
{
      return *THRB_RHRB;
}

unsigned short get_exeption()
{
      return *IMR_ISR;
}
```

```
void set(short pin)
{
      *SETOP = 1<<pin;
}

void reset(short pin)
{
      *RESETOP = 1<<pin;
}

void reset_all()
{
      *RESETOP = 0xff;
}

unsigned short start_counter()
{
      return *RESETOP;
}

unsigned short stop_counter()
{
      return *SETOP;
}

void set_counter(unsigned short upper, unsigned short lower)
{
      *CTUR_CTU = upper;
      *CTLR_CTL = lower;
}

unsigned short spi_tr(unsigned short data)
{
      *RESETOP    = 0x04;     /*SPI skiftregister med seriellt ut kan laddas*/
      *SPI        = data;     /*Laddar skiftregistret med data*/
      *SETOP            = 0x04;     /*SPI skiftregister kan nu skicka ut datan
seriellt*/
      *RESETOP    = 0x02;     /*Startar SPI överföringen*/
      *SETOP            = 0x02;     /*Nollställer Start SPI Transfer*/
      spi_delay();
      data = *SPI;
      return(data);
}

void spi_delay()
{
      *RESETOP    = 0;
      *RESETOP    = 0;
      *RESETOP    = 0;
}

void CE_puls()
{
      *SETOP = 1<<3;
      *RESETOP = 1<<3;
}
```

rtc.h

```
#ifndef RTC
#define RTC

      extern unsigned short   sec, min, h, day, month, year;

      void init_RTC();

      void start_RTC();

      void stop_RTC();

      void set_time();

      void get_time();

#endif
```

rtc.c

```
#include "rtc.h"

#option sep_on class rtc_pekare
static unsigned short int     *RCOPYTIME  = (unsigned short int *) 0x80000,
                              *RH         = (unsigned short int *) 0x80001,
                              *RMIN       = (unsigned short int *) 0x80002,
                              *RSEC       = (unsigned short int *) 0x80003,
                              *RMONTH     = (unsigned short int *) 0x80004,
                              *RDAY       = (unsigned short int *) 0x80005,
                              *RYEAR      = (unsigned short int *) 0x80006,
                              *RDAYOFWEEK = (unsigned short int *) 0x80007,
                              *RINT       = (unsigned short int *) 0x80010,
                              *RCOMMAND   = (unsigned short int *) 0x80011;
#option sep_off

unsigned short    sec, min, h, day, month, year;

void init_RTC()
{
      *RINT       = 0;
      *RCOMMAND   = 0x04;
}

void start_RTC()
{
      *RCOMMAND   = 0x0C;
}

void stop_RTC()
{
      *RCOMMAND   = 0x04;
}

void set_time()
{
      *RCOMMAND   = 0x04; /*Stops RTC*/
      *RCOPYTIME  = 0;
      *RSEC       = sec;
      *RMIN       = min;
      *RH               = h;
      *RDAY       = day;
      *RMONTH           = month;
      *RYEAR            = year;
}

void get_time()
{
      if(*RCOPYTIME){}
      h                 = *RH;
      min               = *RMIN;
      sec               = *RSEC;
      month       = *RMONTH;
      day               = *RDAY;
      year        = *RYEAR;
}
```

radio.h

```
#ifndef RADIO_H
#define RADIO_H

	/* command definitions */
	#define R_REGISTER		0x00	/* 0b00000000 */
	#define W_REGISTER		0x20	/* 0b00100000 */
	#define R_RX_PAYLOAD		0x61	/* 0b01100001 */
	#define W_TX_PAYLOAD		0xA0	/* 0b10100000 */
	#define FLUSH_TX		0xE1	/* 0b11100001 */
	#define FLUSH_RX		0xE2	/* 0b11100010 */
	#define REUSE_TX_PL		0xE3	/* 0b11100011 */
	#define ACTIVATE		0x50	/* 0b01010000 */
	#define R_RX_PL_WID		0x60	/* 0b01100000 */
	#define W_ACK_PAYLOAD		0xA8	/* 0b10101000 */
	#define W_TX_PAYLOAD_NO_ACK	0xB0	/* 0b10110000 */
	#define NOP			0xFF	/* 0b11111111 */

	/* register definitions */
	#define CONFIG			0x00
	#define EN_AA			0x01
	#define EN_RXADDR		0x02
	#define SETUP_AW		0x03
	#define SETUP_RETR		0x04
	#define RF_CH			0x05
	#define RF_SETUP		0x06
	#define STATUS			0x07
	#define OBSERVE_TX		0x08
	#define CD			0x09
	#define RX_ADDR_P0		0x0a
	#define RX_ADDR_P1		0x0b
	#define RX_ADDR_P2		0x0c
	#define RX_ADDR_P3		0x0d
	#define RX_ADDR_P4		0x0e
	#define RX_ADDR_P5		0x0f
	#define TX_ADDR			0x10
	#define RX_PW_P0		0x11
	#define RX_PW_P1		0x12
	#define RX_PW_P2		0x13
	#define RX_PW_P3		0x14
	#define RX_PW_P4		0x15
	#define RX_PW_P5		0x16
	#define FIFO_STATUS		0x17
	#define DYNPD			0x1c
	#define FEATURE			0x1d

	/* CONFIG register bitmasks */
	#define MASK_RX_DR		(1<<6)
	#define MASK_TX_DS		(1<<5)
	#define MASK_MAX_RT		(1<<4)
	#define EN_CRC			(1<<3)
	#define CRCO			(1<<2)
	#define PWR_UP			(1<<1)
	#define PRIM_RX			(1<<0)

	/* EN_AA register bitmasks */
	#define AA_ON				0x3f
	#define AA_OFF				0x00

	/* EN_RXADDR register bitmasks */
	#define     ERX_P1				(1<<1)
	#define     ERX_P0				(1<<0)

	/* RF_SETUP register bitmasks */
	#define RF_DR				(1<<3)

	/* STATUS register bitmasks */
	#define RX_DR			(1<<6)
	#define TX_DS			(1<<5)
```

```
      #define MAX_RT                  (1<<4)
      #define RX_P_NO                 (1<<3)|(1<<2)|(1<<1)
      #define TX_FULL                 (1<<0)

      /* adress defenitions */
      #define TX_ADDR_P0                        0x55
      #define TX_ADDR_P1                        0xC2

      /* functions */
      unsigned short radio_rw(unsigned short cmd, unsigned short data);
      unsigned short radio_rw_combo(unsigned short cmd, unsigned short extra,
unsigned short data);
      void radio_setup(unsigned short plength, unsigned short channel);
      void radio_config_tx();
      void radio_config_rx();
      unsigned short radio_status();
      unsigned short radio_change_addr(unsigned short pipe, unsigned short
addr);
      unsigned short radio_transmit(unsigned short *packet);
      unsigned short radio_recieve(unsigned short *packet);

#endif
```

radio.c

```
#include "radio.h"
#include "uart_spi.h"

unsigned short radio_packet_length;

unsigned short radio_buffer[32];
unsigned int inadr, outadr, temp;

unsigned short radio_rw(unsigned short cmd, unsigned short data)
{
        unsigned short status, tmp;
        CSN_LOW();
        status = spi_tr(cmd);
        if (cmd == R_RX_PAYLOAD) {
                tmp = spi_tr(NOP);
                CSN_HIGH();
                return tmp;
        }
        if (cmd == W_TX_PAYLOAD) {
                spi_tr(data);
                CSN_HIGH();
                return status;
        }
        CSN_HIGH();
        return status;
}

unsigned short radio_rw_combo(unsigned short cmd, unsigned short reg, unsigned
short data)
{
        unsigned short compound,ret, i;
        CSN_LOW();
        compound = cmd | reg;
        ret = spi_tr(compound);
        if (cmd == R_REGISTER) {
                if (reg==RX_ADDR_P0 || reg==RX_ADDR_P1 || reg==TX_ADDR) {
                        for (i=0; i!=5; i++) {
                                radio_buffer[i] = spi_tr(NOP);
                        }
                } else {
                        ret = spi_tr(NOP);
                }
        } else if (cmd == W_REGISTER) {
                if (reg==RX_ADDR_P0 || reg==RX_ADDR_P1 || reg==TX_ADDR) {
                        for (i=0; i!=5; i++) {
                                spi_tr(radio_buffer[i]);
                        }
                } else {
                        spi_tr(data);
                }
        }
        CSN_HIGH();
        return ret;
}

void radio_setup(unsigned short plength, unsigned short channel)
{
      CE_LOW();
      radio_rw(FLUSH_RX, NOP);
      radio_rw(FLUSH_TX, NOP);
      radio_rw_combo(W_REGISTER, EN_AA, AA_ON);
      radio_rw_combo(W_REGISTER, EN_RXADDR, ERX_P0);
      /* retransmitt delay of 500µs */
      radio_rw_combo(W_REGISTER, SETUP_RETR, 0x17);
      radio_rw_combo(W_REGISTER, RF_CH, channel);
      /* 2 Mbps data rate, full TX output power */
      radio_rw_combo(W_REGISTER, RF_SETUP, RF_DR | 3<<1);
      /* P0 packet length of 1 byte */
```

```
      radio_rw_combo(W_REGISTER, RX_PW_P0, plength);
      radio_rw_combo(W_REGISTER, RX_PW_P1, plength); /* P1 ... */
      radio_packet_length = plength;
}

void radio_config_tx()
{
        radio_rw_combo(W_REGISTER, CONFIG, EN_CRC | CRCO | PWR_UP);
}

void radio_config_rx()
{
        radio_rw_combo(W_REGISTER, CONFIG, EN_CRC | CRCO | PWR_UP | PRIM_RX);
}

unsigned short radio_status()
{
        return radio_rw(NOP, NOP);
}

unsigned short radio_change_addr(unsigned short pipe, unsigned int inaddr)
{
      unsigned short i, manbit=0, manbyte=0;
      radio_buffer[4] = 0xE7;
      for(i=0; i<4; i++){
            radio_buffer[i] = 0;
      }
      for(i=0; i<16; i++){
            radio_buffer[manbyte] = radio_buffer[manbyte] | ((1 & inaddr>>i) +
1)<<manbit;
            manbit=manbit+2;
            if(manbit > 6){
                  manbit=0;
                  manbyte++;
            }
      }
      radio_rw_combo(W_REGISTER, pipe, 0);
      return radio_rw_combo(W_REGISTER,TX_ADDR, 0);
}

unsigned short radio_transmit(unsigned short *packet)
{
        unsigned short status, i;
        CSN_LOW();
        status = spi_tr(W_TX_PAYLOAD);
        for (i=0; i<radio_packet_length; i++) {
                spi_tr(packet[i]);
        }
        CSN_HIGH();
            CE_puls();
        return status;
}

unsigned short radio_recieve(unsigned short *packet)
{
        unsigned short status, i;
        CSN_LOW();
        status = spi_tr(R_RX_PAYLOAD);
        for (i=0; i<radio_packet_length; i++) {
                packet[i] = spi_tr(NOP);
        }
        CSN_HIGH();
        return status;
}
```

Bilaga 5: Källkod för AVR underenhet

main.c

```
#include <avr/io.h>
#include <avr/interrupt.h>
#define F_CPU 8000000UL
#include <util/delay.h>

#include "radio.h"

#define     ME 0x02

uint8_t transmit, rx_packet[4] = {0, 0, 0, 0}, tx_packet[4] = {0, 0, 0, 0};

uint8_t filter(uint8_t in)
{   //Filter that eliminates 0xFF and 0x00.
      if(in == 0xFF){
            return 0xFE;
      }
      if(in == 0x00){
            return 0x01;
      }
      return in;
}

int main(void)
{

    DDRA = 0b11100000;  //PORT A setup
      DDRC = 0b00000001;      //PORT C setup
      ADMUX = (1<<REFS0)|(1<<ADLAR);                        //ADMUX=0b01100000;
      ADCSRA = (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);//ADCSRA=0b10000111;
      GICR = (1<<INT2);
      MCUCR = 0;

      unsigned int addr=ME;

      transmit = 0;

      // Configure radiomodule.
      radio_setup(4, 17);
      radio_change_addr(RX_ADDR_P0, addr);
      radio_config_rx();

      // Enable global interrupts.
      sei();

      // Reset PORTA and blink with LED at PORTC at startup.
      PORTA = 0;
    PORTC = 1;
      _delay_ms(500);
    PORTC = 0;

      // Set radiomodule in RX-mode.
      CE_HIGH();

      // Endless loop.
    for (;;) {
            if(transmit){
                  if(rx_packet[0] == 'a'){                  //'a'-packet recived.
                        _delay_ms(3);
                        CE_LOW();
                        radio_config_tx();
                        radio_transmit(&tx_packet[0]);

                        PORTA = (rx_packet[3]&0x07)<<5;

                        tx_packet[0] = 'A';
```

```
                        ADCSRA = ADCSRA & ~(1<<ADIF);
                        ADMUX = ADMUX & 0xF0;
                        ADCSRA = ADCSRA | (1<<ADSC);
                        while( !(ADCSRA & (1<<ADIF)) ){ }
                        tx_packet[1] = filter(ADCH);

                        ADCSRA = ADCSRA & ~(1<<ADIF);
                        ADMUX = (ADMUX & 0xF0) + 1;
                        ADCSRA = ADCSRA | (1<<ADSC);
                        while( !(ADCSRA & (1<<ADIF)) ){ }
                        tx_packet[2] = filter(ADCH);

                        tx_packet[3] = ((PINA&0x1C)>>2) | 0x50;

                  }else if(rx_packet[0] == 'b'){      //'b'-packet recived.
                        PORTA = (rx_packet[3]&0x07)<<5;
                        _delay_ms(4);
                  }
                  CE_HIGH();
                  transmit=0;
                  PORTC=0;
            }
            _delay_ms(1);
    }

}

ISR(INT2_vect)
{
      uint8_t status;
      CE_LOW();
      status = radio_status();

      if(status & RX_DR){     // Recived a packet.
            radio_recieve(&rx_packet[0]);
            radio_rw_combo(W_REGISTER, STATUS, status|RX_DR);
            transmit=1;
            PORTC=1;
      }

      if (status & TX_DS){    // Succsseded to transmitt a packet.
            radio_rw_combo(W_REGISTER, STATUS, status|TX_DS);
      }

      if (status & MAX_RT){   // Faild to transmitt a packet.
            radio_rw_combo(W_REGISTER, STATUS, status|MAX_RT);
      }

      if (status & TX_FULL){  // Faild to transmitt a packet three times in a
row.
            radio_rw(FLUSH_TX, NOP);
      }
      radio_config_rx();
      CE_HIGH();
}
```

radio.h

```
#ifndef RADIO_H
#define RADIO_H

#include <avr/io.h>

#define RADIO_DDR DDRB
#define RADIO_PORT PORTB
#define RADIO_PIN PINB

#define RADIO_CE PB3
#define RADIO_IRQ PB2
#define RADIO_CSN PB4

// command definitions
#define R_REGISTER              0b00000000
#define W_REGISTER              0b00100000
#define R_RX_PAYLOAD            0b01100001
#define W_TX_PAYLOAD            0b10100000
#define FLUSH_TX                0b11100001
#define FLUSH_RX                0b11100010
#define REUSE_TX_PL             0b11100011
#define ACTIVATE                0b01010000
#define R_RX_PL_WID             0b01100000
#define W_ACK_PAYLOAD           0b10101000
#define W_TX_PAYLOAD_NO_ACK     0b10110000
#define NOP                     0b11111111

// register definitions
#define CONFIG                  0x00
#define EN_AA                   0x01
#define EN_RXADDR               0x02
#define SETUP_AW                0x03
#define SETUP_RETR              0x04
#define RF_CH                   0x05
#define RF_SETUP                0x06
#define STATUS                  0x07
#define OBSERVE_TX              0x08
#define CD                      0x09
#define RX_ADDR_P0              0x0a
#define RX_ADDR_P1              0x0b
#define RX_ADDR_P2              0x0c
#define RX_ADDR_P3              0x0d
#define RX_ADDR_P4              0x0e
#define RX_ADDR_P5              0x0f
#define TX_ADDR                 0x10
#define RX_PW_P0                0x11
#define RX_PW_P1                0x12
#define RX_PW_P2                0x13
#define RX_PW_P3                0x14
#define RX_PW_P4                0x15
#define RX_PW_P5                0x16
#define FIFO_STATUS             0x17
#define DYNPD                   0x1c
#define FEATURE                 0x1d

// config register bitmasks
#define MASK_RX_DR              (1<<6)
#define MASK_TX_DS              (1<<5)
#define MASK_MAX_RT             (1<<4)
#define EN_CRC                  (1<<3)
#define CRCO                    (1<<2)
#define PWR_UP                  (1<<1)
#define PRIM_RX                 (1<<0)

/* EN_AA register bitmasks */
#define AA_ON                               0x3f
#define AA_OFF                              0x00
```

```
/* EN_RXADDR register bitmasks */
#define     ERX_P1                              (1<<1)
#define     ERX_P0                              (1<<0)

/* RF_SETUP register bitmasks */
#define RF_DR                               (1<<3)

// status register bitmasks
#define RX_DR                   (1<<6)
#define TX_DS                   (1<<5)
#define MAX_RT                  (1<<4)
#define RX_P_NO                 (1<<3)|(1<<2)|(1<<1)
#define TX_FULL                 (1<<0)

/* FIFO_STATUS register bitmasks */
#define TX_EMPTY                    (1<<4)

/* feature register bitmasks */
#define EN_DYN_ACK              (1<<0)

/* adress defenitions */
#define TX_ADDR_P0                          0x55
#define TX_ADDR_P1                          0xC2

/* Macros */
#define CSN_LOW() RADIO_PORT &=~ (1<<RADIO_CSN)
#define CSN_HIGH() RADIO_PORT |= (1<<RADIO_CSN)

#define CE_LOW() RADIO_PORT &=~ (1<<RADIO_CE)
#define CE_HIGH() RADIO_PORT |= (1<<RADIO_CE)

#define RADIO_IRQ_BIT (RADIO_PIN & (1<<RADIO_IRQ))

/* functions */
void spi_setup(void);
uint8_t spio(uint8_t b);
uint8_t radio_rw(uint8_t cmd, uint8_t data);
uint8_t radio_rw_combo(uint8_t cmd, uint8_t extra, uint8_t data);
void radio_setup(uint8_t plength, uint8_t channel);
void radio_config_tx(void);
void radio_config_rx(void);
uint8_t radio_status(void);

uint8_t radio_change_addr(uint8_t pipe, unsigned int addr);
uint8_t radio_transmit(uint8_t *packet);
uint8_t radio_recieve(uint8_t *packet);

#endif
```

radio.c

```
#include "radio.h"

#define F_CPU 8000000UL
#include <util/delay.h>

static uint8_t radio_packet_length = 1;

uint8_t radio_buffer[32];

void spi_setup(void)
{
        // setup SPI pins' io config
        DDRB |= (1<<PB4)|(1<<PB5)|(1<<PB7);
        DDRB &=~ (1<<PB6);
        PORTB |= (1<<PB4);

        // SPI enable, master mode, fcpu/4 (=4MHZ @ 16MHZ fcpu)
        SPCR |= (1<<SPE)|(1<<MSTR);
}

uint8_t spio(uint8_t b)
{
        SPDR = b;
        while (!(SPSR & (1<<SPIF)));
        return SPDR;
}

uint8_t radio_rw(uint8_t cmd, uint8_t data)
{
        uint8_t status, tmp;
        CSN_LOW();
        status = spio(cmd);
        if (cmd == R_RX_PAYLOAD) {
                tmp = spio(NOP);
                CSN_HIGH();
                return tmp;
        }
        if (cmd == W_TX_PAYLOAD) {
                spio(data);
                CSN_HIGH();
                return status;
        }
        CSN_HIGH();
        return status;
}

uint8_t radio_rw_combo(uint8_t cmd, uint8_t reg, uint8_t data)
{
        uint8_t compound,ret, i;
        CSN_LOW();
        compound = cmd | reg;
        ret = spio(compound);
        if (cmd == R_REGISTER) {
                if (reg==RX_ADDR_P0 || reg==RX_ADDR_P1 || reg==TX_ADDR) {
                        for (i=0; i!=5; i++) {
                                radio_buffer[i] = spio(NOP);
                        }
                } else {
                        ret = spio(NOP);
                }
        } else if (cmd == W_REGISTER) {
                if (reg==RX_ADDR_P0 || reg==RX_ADDR_P1 || reg==TX_ADDR) {
                        for (i=0; i!=5; i++) {
                                spio(radio_buffer[i]);
                        }
                } else {
                        spio(data);
               }
```

```
        }
        CSN_HIGH();
        return ret;
}

void radio_setup(uint8_t plength, uint8_t channel)
{
        spi_setup();

        // setup radio control signals' io config
        RADIO_DDR |= (1<<RADIO_CE)|(1<<RADIO_CSN); // outputs
        RADIO_DDR &=~ (1<<RADIO_IRQ);              // inputs
        CE_LOW();
        radio_rw(FLUSH_RX, NOP);
        radio_rw(FLUSH_TX, NOP);
        radio_rw_combo(W_REGISTER, EN_AA, AA_ON);
        radio_rw_combo(W_REGISTER, EN_RXADDR, ERX_P0);
        /* retransmitt delay of 500µs */
        radio_rw_combo(W_REGISTER, SETUP_RETR, 0x1F);
        radio_rw_combo(W_REGISTER, RF_CH, channel);
        /* 2 Mbps data rate, full TX output power */
        radio_rw_combo(W_REGISTER, RF_SETUP, RF_DR | 3<<1);
        /* P0 packet length of 1 byte */
        radio_rw_combo(W_REGISTER, RX_PW_P0, plength);
        radio_rw_combo(W_REGISTER, RX_PW_P1, plength); /* P1 ... */
        radio_packet_length = plength;

}

void radio_config_tx(void)
{
        radio_rw_combo(W_REGISTER, CONFIG, EN_CRC | CRCO | PWR_UP);
}

void radio_config_rx(void)
{
        radio_rw_combo(W_REGISTER, CONFIG, EN_CRC | CRCO | PWR_UP | PRIM_RX);
}

uint8_t radio_status(void)
{
        return radio_rw(NOP, NOP);
}

uint8_t radio_change_addr(uint8_t pipe, unsigned int inaddr)
{
      uint8_t i, manbit=0, manbyte=0;
      radio_buffer[4] = 0xE7;
      for(i=0; i<4; i++){
            radio_buffer[i] = 0;
      }
      for(i=0; i<16; i++){
            radio_buffer[manbyte] = radio_buffer[manbyte] | ((1 & inaddr>>i) +
1)<<manbit;
            manbit=manbit+2;
            if(manbit > 6){
                  manbit=0;
                  manbyte++;
            }
      }
      radio_rw_combo(W_REGISTER, pipe, 0);
      return radio_rw_combo(W_REGISTER,TX_ADDR, 0);
}

uint8_t radio_transmit(uint8_t *packet)
{
        uint8_t status, i;
        CSN_LOW();
        status = spio(W_TX_PAYLOAD);
        for (i=0; i<radio_packet_length; i++) {
```

```
                spio(packet[i]);
        }
        CSN_HIGH();

        CE_HIGH();
        _delay_us(20);
        CE_LOW();
        return status;
}

uint8_t radio_recieve(uint8_t *packet)
{
        uint8_t status, i;
        CSN_LOW();
        status = spio(R_RX_PAYLOAD);
        for (i=0; i<radio_packet_length; i++) {
                packet[i] = spio(NOP);
        }
        CSN_HIGH();
        return status;
}
```

Bilaga 6: Källkod för MATLAB program

control_system.m

```
function control_system
    % Variables for serial comunication.
    global serport;
    global not_synced;
    global tx_packet;

    % Variables for data storage.
    global units;
    global timestamp;
    global data;
    global timestep;
    global data_file;

    % Variables for GUI and plots.
    global axes1;
    global axes2;
    global led;
    global first_value;
    global last_hot_timestep;

    % Variables only used in plotPower.
    global oldPower;

    % Initilize all variabels.
    % Variables for serial comunication.
    serport = serial('COM1' , 'BaudRate', 9600, 'Parity', 'none', 'StopBits',
1);
    serport.BytesAvailableFcnCount = 1;
    serport.BytesAvailableFcnMode = 'byte';
    serport.BytesAvailableFcn = @serial_Callback;
    not_synced=1;
    tx_packet = zeros(1,8, 'uint8');
    packet = zeros(1,10, 'uint8');

    % Variables for data storage.
    dimOfUnits = [5 126];
    units = zeros(5,126, 'uint8');  % | settings   | sensortype | value dig1 |
                                    % | value dig2 | value dig3 | x126 units |
                                    %  settings: 7:EN unit, 6:Unit contact,
                                    %            5:EN ad1, 4:EN ad2, 3:reserved,
                                    %            2-0:data to digital out
    dimOfTimestamp = [4 1];
    timestamp = zeros(4,4600, 'uint8'); %|day |hour |min |sec | x4600 timesteps

    dimOfData = [3 126];
    data = zeros(3,126,4600, 'uint8');%|ad1 |ad2 |dig | x126units x4600timesteps

    dimOfEscape = [3 1];
    escsekv = [1; 2; 3;];

    timestep = 1;
    error = 0;
    clk = clock';

    % Variables for GUI and plots.
    last_hot_timestep = 1;
    first_value = zeros(1,126, 'uint8');
```

```
% Open file for settings.
units_file_string = 'units.sfoos';
units_file = fopen(units_file_string, 'r+');
if(units_file < 0)  % If file dosen't exist, create it.
    units_file = fopen(units_file_string, 'w+');
else
    % Read file content to the matrix units.
    temp = fread(units_file,dimOfUnits);
    if(size(temp) == dimOfUnits)
        units = temp;
        temp = fread(units_file,dimOfEscape);
        if(size(temp) == dimOfEscape)
            if(temp == escsekv)
            else
                disp('Fel escape sekvens, units')
                error = 1;
            end
        elseif(size(temp) == [0 0])
            disp('ingen escape sekvens')
        else
            disp('Fel på escape sekvensens längd, units')
            error = 1;
        end
    elseif(size(temp) == [0 0])
        disp('ingen fil')
    else
        disp('Fel på data matrisen, units')
        error = 1;
    end
    if(error)
        % An error have occurred.
    else
    end
end

% Create and then hide the GUI as it is being constructed.
f = figure('Visible','off','Position',[0,0,1024,768]);

% Construct the components of the GUI.
box_start = 700;
box_dec=28;
tagnbr = 1;
tagstr = num2str(tagnbr);
sensors = {'None','Temp','Humidity'};
% Gray boxes
uicontrol(  'Style','text',...
            'String',' ',...
            'Position',[0,728,359,40]);
uicontrol(  'Style','text',...
            'String',{'   '},...
            'Position',[0,0,30,728]);
% Base station button and indicator.
baseStation = uicontrol('Style','togglebutton',...
                        'String','Connect to base station',...
                        'Position',[30,748,160,20],...
                        'Callback',{@baseStation_Callback});
led = uicontrol(        'Style','text',...
                        'BackgroundColor', 'r',...
                        'Position',[200,748,20,20]);
% Labels for the settings columns.
```

```
    uicontrol(  'Style','text',...
                'String','Enable',...
                'Position',[20,728,35,15]);
    uicontrol(  'Style','text',...
                'String','AD1',...
                'Position',[70,728,30,15]);
    uicontrol(  'Style','text',...
                'String','AD2',...
                'Position',[150,728,30,15]);
    uicontrol(  'Style','text',...
                'String','Digital out',...
                'Position',[203,728,60,15]);
    uicontrol(  'Style','text',...
                'String','P1',...
                'Position',[275,728,15,15]);
    uicontrol(  'Style','text',...
                'String','P2',...
                'Position',[305,728,15,15]);
    uicontrol(  'Style','text',...
                'String','P3',...
                'Position',[335,728,15,15]);

    % Create settings for 20 units.
    for c=1:20
        uicontrol(  'Style','text',...
                    'String',tagstr,...
                    'Tag',tagstr,...
                    'Position',[0,box_start,30,15]);
        uicontrol(  'Style','checkbox',...
                    'Tag',tagstr,...
                    'Value', bitand(units(1,c), 128)/128,...
                    'Position',[35,box_start,15,15],...
                    'Callback',{@en_Callback});
        uicontrol(  'Style','popupmenu',...
                    'String',sensors,...
                    'Tag',tagstr,...
                    'Value', 1+bitand(units(2,c), 240)/16,...  %240 - 0b11110000
                    'Position',[55,box_start,70,20],...
                    'Callback',{@ad1_menu_Callback});
        uicontrol(  'Style','popupmenu',...
                    'String',sensors,...
                    'Tag',tagstr,...
                    'Value', 1+bitand(units(2,c), 15),...      %15  - 0b00001111
                    'Position',[130,box_start,70,20],...
                    'Callback',{@ad2_menu_Callback});
        uicontrol(  'Style','checkbox',...
                    'Tag',strcat(tagstr, '1'),...
                    'Value', bitand(units(1,c), 1),...
                    'Position',[210,box_start,15,15],...
                    'Callback',{@digout_Callback});
        uicontrol(  'Style','checkbox',...
                    'Tag',strcat(tagstr, '2'),...
                    'Value', bitand(units(1,c), 2)/2,...
                    'Position',[225,box_start,15,15],...
                    'Callback',{@digout_Callback});
        uicontrol(  'Style','checkbox',...
                    'Tag',strcat(tagstr, '3'),...
                    'Value', bitand(units(1,c), 4)/4,...
                    'Position',[240,box_start,15,15],...
                    'Callback',{@digout_Callback});
        uicontrol(  'Style','edit',...
```

```
                'String', num2str(10 * units(3, c)),...
                'Tag',strcat(tagstr, '1'),...
                'Position',[265,box_start,30,15],...
                'Callback',{@digin_Callback});
    uicontrol(  'Style','edit',...
                'String', num2str(10 * units(4, c)),...
                'Tag',strcat(tagstr, '2'),...
                'Position',[297,box_start,30,15],...
                'Callback',{@digin_Callback});
    uicontrol(  'Style','edit',...
                'String', num2str(10 * units(5, c)),...
                'Tag',strcat(tagstr, '3'),...
                'Position',[329,box_start,30,15],...
                'Callback',{@digin_Callback});
    box_start=box_start-box_dec;
    tagnbr = tagnbr+1;
    tagstr = num2str(tagnbr);
end  %for-loop

% Create plot areas.
axes1 = axes('Units','Pixels','Position',[389,402,635,362]);
axes2 = axes('Units','Pixels','Position',[389,20,635,362]);

% Move the GUI to the center of the screen.
movegui(f,'center')

% Assign the GUI a name to appear in the window title.
% Make the GUI visible.
set(f,  'Name','System för övervakning och styrning',...
        'CloseRequestFcn',@closeGUI,...
        'Visible','on');

% Open the data file for this month.
data_file_string = strcat(num2str(clk(1:2)'), '.sfoos');
data_file = fopen(data_file_string, 'r+');
if(data_file < 0)  % If file dosen't exist, create it.
    data_file = fopen(data_file_string, 'w+');
else
    % Read file content to the matrices timestamp and data.
    while(1)
        temp = fread(data_file,dimOfTimestamp);
        if(size(temp) == dimOfTimestamp)
            timestamp(:,timestep) = temp;
        elseif(size(temp) == [0 0])
            disp('slut på fil')
            break;
        else
            disp('Fel på tidsstämpeln, data')
            error = 1;
            break;
        end
        temp = fread(data_file,dimOfData);
        if(size(temp) == dimOfData)
            data(:,:,timestep) = temp;
            temp = fread(data_file,dimOfEscape);
            if(size(temp) == dimOfEscape)
                if(temp == escsekv)
                    timestep = timestep+1;
                else
                    disp('Fel escape sekvens, data')
                    error = 1;
```

```
                end
            elseif(size(temp) == [0 0])
                disp('slut på fil ingen escape sekvens')
                error = 1;
                break;
            else
                disp('Fel på escape sekvensens längd, data')
                error = 1;
                break;
            end
        elseif(size(temp) == [0 0])
            disp('slut på fil men en tidstämbep lästes in')
            error = 1;
            break;
        else
            disp('Fel på data matrisen, data')
            error = 1;
            break;
        end
    end
    if(error)
        % An error have occurred.
    else
        % Plot the data that is read from the file.
        new_timestep = timestep;
        timestep = timestep-1;
        plotAxis([clk(1:2)' double(timestamp(:,timestep))']);
        if(timestep < 22)
            startstep = 2;
            last_hot_timestep = 1;
        else
            startstep = timestep-21;
            last_hot_timestep = startstep-1;
        end
        for timestep = startstep:new_timestep-1
            plotPower;
            for unit=1:126
                if(bitand(units(1,unit), 128)/128)
                    plotTemp(unit);
                end
            end
        end
        timestep = new_timestep;
        last_hot_timestep = timestep;
        disp('ritat klart')
    end
end
first_value = 7*ones(1,126);

%  Callbacks for control_system is located in separate files exept for
%  serial_Callback and closeGUI.

% This callback handels the serial communication.
function serial_Callback(source,eventdata)
    if(not_synced)
        packet(not_synced) = fread(serport, 1);
        not_synced = not_synced + 1;
        if(not_synced >= 3)
            if(packet(not_synced-2) == 255 && packet(not_synced-1) == 0)
                % The syncronisation seems to have succeeded.
                tx_packet = 'U'.*ones(1,8);
```

```
                    fwrite(serport, tx_packet);
                    set(led, 'BackgroundColor', 'g');
                    fclose(serport);
                    serport.BytesAvailableFcnCount = 10;
                    fopen(serport);
                    not_synced = 0;
                elseif(not_synced == 11)
                    set(led, 'BackgroundColor', 'r');
                    not_synced = 1;
                end
            end
        else
            packet(1:10) = fread(serport, 10);
            if(packet(9) == 255 && packet(10) == 0)
                if(packet(1) < 127)  % Data from units is recived.
                    % First unit.
                    unit = packet(1);
                    data(1,unit,timestep) = packet(2);
                    data(2,unit,timestep) = packet(3);
                    data(3,unit,timestep) = packet(4);
                    plotTemp(unit);
                    % Potential second unit.
                    unit = packet(5);
                    if(unit)
                        data(1,unit,timestep) = packet(6);
                        data(2,unit,timestep) = packet(7);
                        data(3,unit,timestep) = packet(8);
                        plotTemp(unit);
                    end
                else
                    switch(packet(1))
                        case ('T'+100),  % A new timestamp is recived
                            fwrite(data_file, timestamp(:,timestep));
                            fwrite(data_file, data(:,:,timestep));
                            fwrite(data_file, escsekv);
                            timestep = timestep+1;
                            temp = clock';
                            if(temp(2) ~= clk(2))
                                % New month, save data...
                                fclose(data_file);
                                % and create a new file.
                                clk = clock';
                                data_file_string=strcat(num2str(clk(1:2)'),
'.sfoos');
                                data_file = fopen(data_file_string, 'w+');
                            else
                                timestamp(1, timestep) = temp(3);
                                timestamp(2, timestep) = packet(4);
                                timestamp(3, timestep) = packet(3);
                                timestamp(4, timestep) = packet(2);
                            end
                            plotAxis(clock);
                            plotPower;
                        case ('A'+100),  % Motorola asks if there is a PC.
                            % Answer the Motorola that there is a PC.
                            tx_packet = 'U'.*ones(1,8);
                            fwrite(serport, tx_packet);
                            if(packet(2) == 95)  %95 - 0x5F
                                % Motorola wants to know the time.
                                temp = clock;
                                tx_packet(1) = 'T';
```

```
                                for c = 2:6
                                    tx_packet(c) = uint8(temp(8-c));
                                end
                                tx_packet(7) = uint8(mod(temp(1),1000));
                                tx_packet(8) = 0;
                                fwrite(serport, tx_packet);
                            end
                            if(packet(3) == 95)  %95 - 0x5F
                                % Motorola wants to know the settings for
                                % all the units.
                                for c = 0:17
                                    tx_packet(1) = 'A'+ c;
                                    for n = 2:8
                                        tx_packet(n) = units(1, (7*c + n-1));
                                    end
                                    fwrite(serport, tx_packet);
                                end
                            end
                            tx_packet = 'W'.*ones(1,8);
                            fwrite(serport, tx_packet);
                        end
                    end
                else
                    % The escape sequence was wrong and the syncronisation is
                    % considerd lost.
                    set(led, 'BackgroundColor', 'r');
                    tx_packet = 'V'.*ones(1,8);
                    fwrite(serport, tx_packet);
                    fclose(serport);
                    serport.BytesAvailableFcnCount = 1;
                    fopen(serport);
                    not_synced=1;
                end
            end
        end

        % This callback executes when the user hits the X (close) button.
        function closeGUI(src,evnt)
            % Save settings to file.
            fclose(units_file);
            units_file = fopen(units_file_string, 'w+');
            fwrite(units_file, units);
            fwrite(units_file, escsekv);

            % Close all files and serialport.
            fclose(data_file);
            fclose(units_file);
            fclose(serport);
            delete(serport);
            delete(gcf);
        end
    end
```

plotAxis.m

```
function plotAxis(clk)
    global timestamp;
    global timestep;
    global axes1;
    global axes2;

    axes(axes1)
    tempa = clk;
    tempb = [tempa(1:2) double(timestamp(:,timestep)')];
    tempb(6) = tempb(6)+2;
    tempa = tempb;
    tempa(5) = tempa(5)-2;
    axis([datenum(tempa) datenum(tempb) 0 255])
    datetick('x','keeplimits');

    axes(axes2)
    axis([datenum(tempa) datenum(tempb) 0 5000])
    datetick('x','keeplimits');
end
```

plotTemp.m

```
function plotTemp(unit)
    global units;   %content in the first byte: 7:EN unit, 6:Unit contact,
                    %5:EN ad1, 4:EN ad2, 3:reserved, 2-0:data to digital out
    global timestamp;
    global data;
    global timestep;
    global first_value;
    global axes1;

    if(timestep > 1)
        axes(axes1)
        hold on
        tempc = clock;
        tempt1 = [tempc(1:2) double(timestamp(:,timestep-1)')];
        tempt2 = [tempc(1:2) double(timestamp(:,timestep)')];
        tiden = [datenum(tempt1);datenum(tempt2)];
        if(bitand(units(1, unit), uint8(32)))
            if(bitand(first_value(unit),1))
                first_value(unit) = bitand(first_value(unit), 254);
                                                        %254 - 0b11111110
            else
                datan = [data(1,unit,timestep-1);data(1,unit,timestep)];
                plot(tiden,datan,'b');
            end
        end
        if(bitand(units(1, unit), uint8(16)))
            if(bitand(first_value(unit),2))
                first_value(unit) = bitand(first_value(unit), 253);
                                                        %253 - 0b11111101
            else
                datan = [data(2,unit,timestep-1);data(2,unit,timestep)];
                plot(tiden,datan,'r');
            end
        end
        drawnow
        hold off
    end
end
```

plotPower.m

```
function plotPower
    global units;   % settings (first byte): 7:EN unit, 6:Unit contact,
                    % 5:EN ad1, 4:EN ad2, 3:reserved, 2-0:data to digital out
    global timestamp;
    global data;
    global timestep;
    global last_hot_timestep;
    global axes2;
    global oldPower;

    if(timestep-last_hot_timestep > 2)
        axes(axes2)
        hold on
        tempc = clock;
        tempt1 = [tempc(1:2) double(timestamp(:,timestep-2)')];
        tempt2 = [tempc(1:2) double(timestamp(:,timestep-1)')];
        tiden = [datenum(tempt1);datenum(tempt2)];
        datan = double(0);
        for c=1:126
            if(bitand(units(1,c), 128))
                datan=datan...
                      +double(bitand(data(3,c,timestep-1),1))*10*units(3,c)...
                      +double(bitand(data(3,c,timestep-1),2))*5*units(4,c)...
                      +double(bitand(data(3,c,timestep-1),4))*2.5*units(5,c);
            end
        end
        plot(tiden,[oldPower datan],'r');
        drawnow
        oldPower = datan;
        hold off
    else
        if(timestep-last_hot_timestep == 1)
            datan = 0;
            for c=1:126
                if(bitand(units(1,c), 128))
                    datan=datan...
                      +double(bitand(data(3,c,timestep-1),1))*10*units(3,c)...
                      +double(bitand(data(3,c,timestep-1),2))*5*units(4,c)...
                      +double(bitand(data(3,c,timestep-1),4))*2.5*units(5,c);
                end
            end
            oldPower = datan;
        end
    end
end
```

baseStation_Callback.m

```
function baseStation_Callback(source,eventdata)
    global serport;
    global not_synced;
    global timestep;
    global last_hot_timestep;
    global first_value;
    global led;

    if(get(source, 'Value'))
        % Connect Base station
        set(source, 'String', 'Disconnect from base station');
        if(size(serport.status) == [1 6])
            fopen(serport);
            set(led, 'BackgroundColor', 'y');
            if serport.BytesAvailable > 0        % clear buffer
                fread(serport, serport.BytesAvailable);
            end
        end
    else
        % Disconnect Base station
        first_value = 7*ones(1,126);
        last_hot_timestep = timestep;
        tx_packet = 'V'.*ones(1,8);
        fwrite(serport, tx_packet);
        fclose(serport);
        serport.BytesAvailableFcnCount = 1;
        not_synced=1;
        set(led, 'BackgroundColor', 'r');
        set(source, 'String', 'Connect to base station');
    end
end
```

en_Callback.m

```
function en_Callback(source,eventdata)
    global serport;
    global units;

    unit = str2double(get(source, 'Tag'));
    if(get(source, 'Value'))
        units(1, unit) = bitset(units(1, unit), 8);
    else
        units(1, unit) = bitand(units(1, unit), uint8(127)); %127 - 0b01111111
    end

    if(size(serport.status) == [1 4])
        tx_packet = zeros(1,8, 'uint8');
        tx_packet(1) = 'S';
        tx_packet(2) = unit;
        tx_packet(3) = units(1, unit);
        fwrite(serport, tx_packet);
    end
end
```

ad1_menu_Callback.m

```
function ad1_menu_Callback(source,eventdata)
    global units;

    unit = str2double(get(source, 'Tag'));
    val = get(source,'Value')-1;
    units(2,unit)=bitor(bitand(units(2,unit),uint8(15)),bitshift(uint8(val),4));
                                                    %15 - 0b00001111
    if(val)
        units(1, unit) = bitset(units(1, unit), 6);
    else
        units(1, unit) = bitand(units(1, unit), uint8(223)); %223 - 0b1101111
    end
end
```

ad2_menu_Callback.m

```
function ad2_menu_Callback(source,eventdata)
    global units;

    unit = str2double(get(source, 'Tag'));
    val = get(source,'Value')-1;
    units(2, unit) = bitor(bitand(units(2, unit), uint8(240)), uint8(val));
                                                        %240 - 0b11110000
    if(val)
        units(1, unit) = bitset(units(1, unit), 5);
    else
        units(1, unit) = bitand(units(1, unit), uint8(239)); %239 - 0b1110111
    end
end
```

digout_Callback.m

```
function digout_Callback(source,eventdata)
    global serport;
    global units;

    unit = get(source, 'Tag') + 0;
    pin = str2double([unit(length(unit)) '']);
    unit = str2double([unit(1:length(unit)-1) '']);
    if(get(source, 'Value'))
        units(1, unit) = bitset(units(1, unit), pin);
    else
        units(1, unit) = bitand(units(1, unit), uint8(255-2^(pin-1)));
    end

    if(size(serport.status) == [1 4])
        if(bitand(units(1,unit), 128)/128)
            tx_packet = zeros(1,8, 'uint8');
            tx_packet(1) = 'S';
            tx_packet(2) = unit;
            tx_packet(3) = units(1, unit);
            fwrite(serport, tx_packet);
        end
    end
end
```

digin_Callback.m

```
function digin_Callback(source,eventdata)
    global units;

    unit = get(source, 'Tag') + 0;
    pin = str2double([unit(length(unit)) '']);
    unit = str2double([unit(1:length(unit)-1) '']);
    units(pin+2, unit) = uint8(str2double(get(source, 'String'))/10);
    set(source, 'String', num2str(10 * units(pin+2, unit)));
end
```

# Appendix

## Kravspecifikation

Målet med det valda projektet är att bygga ett system som förenklar styrning och övervakning av valfria saker, till exempel att slå av och på elförbrukare eller övervaka temperaturer. Systemet ska bestå av en huvudenhet, ett antal mindre moduler som kan placeras nära platsen där styrning/övervakning önskas samt en pc.

De små modulerna kommer att vara baserade på ATmega16. De ska ha minst en analog ingång för att kunna hantera olika typer av sensorer till exempel temperatursensorer. De ska ha minst en utgång för att kunna styra till exempel ett relä.

Huvudenheten kommer att vara baserad på MC86008. Denna ska kommunicera med de små modulerna trådlöst med hjälp av Zigbee. (I värsta fall kabel, typ cat 5). Den ska även kommunicera med pc:n och överföra insamlad data dit, samt ta emot data och överföra till modulerna.

Pc:n ska fungera som användargränssnitt. Här ifrån ska man kunna styra till exempel elförbrukare, eller titta på statistik över insamlad data.

## Om tid finnes

Fler in och utgångar på de små modulerna för att kunna hantera/styra fler saker och utföra mer avancerade uppgifter.

Fjärrkontroll för att styra enklare funktioner via pc:n

Snyggt användargränssnitt, kanske även möjlighet att styra/övervaka via internet.

Ingång för kommunikation med elmätare för att föra statistik över total elförbrukning.

Komma på ett bättre namn.

# Källförteckning

## Böcker:

Lindvall, Bertil (2008) Utvecklingssystem för MC68008, Lund

Kernighan, Brian W, Ritchie, Dennis M, (1988) The C Programming language, New Jersey

## Hemsidor:

http://blinkdagger.com/matlab/matlab-gui-tutorial-close-gui-confirmation

http://blinkdagger.com/matlab/matlab-gui-graphical-user-interface-tutorial-for-beginners

http://www.asciitabell.se/

http://www.diyembedded.com/

## Datablad:

http://www.eit.lth.se/fileadmin/eit/courses/edi021/datablad/Analog/voltage/lp3855.pdf

http://www.eit.lth.se/fileadmin/eit/courses/edi021/datablad/Logik/74HC/74HC04.pdf

http://www.eit.lth.se/fileadmin/eit/courses/edi021/datablad/Logik/74HC/74HC163.pdf

http://www.eit.lth.se/fileadmin/eit/courses/edi021/datablad/Logik/74LS/74LS166.pdf

http://www.eit.lth.se/fileadmin/eit/courses/edi021/datablad/Logik/74HC/74HC595.pdf

http://www.eit.lth.se/fileadmin/eit/courses/edi021/datablad/Periphery/Communication/max232-233.pdf

http://www.eit.lth.se/fileadmin/eit/courses/edi021/datablad/Logik/Programmable/Palce22v10.pdf

http://www.eit.lth.se/fileadmin/eit/courses/edi021/datablad/Periphery/RTC/ICM7170.pdf

http://www.eit.lth.se/fileadmin/eit/courses/edi021/datablad/Memory/Eprom/NM27C256.pdf

http://www.eit.lth.se/fileadmin/eit/courses/edi021/datablad/Memory/Sram/431000.pdf

http://www.eit.lth.se/fileadmin/eit/courses/edi021/datablad/Periphery/Communication/SCN68681.pdf

http://www.eit.lth.se/fileadmin/eit/courses/edi021/datablad/Processors/68000UM.pdf

http://www.nordicsemi.no/files/Product/data_sheet/nRF24L01_Product_Specification_v2_0.pdf

http://www.eit.lth.se/fileadmin/eit/courses/edi021/datablad/Processors/ATmega16.pdf