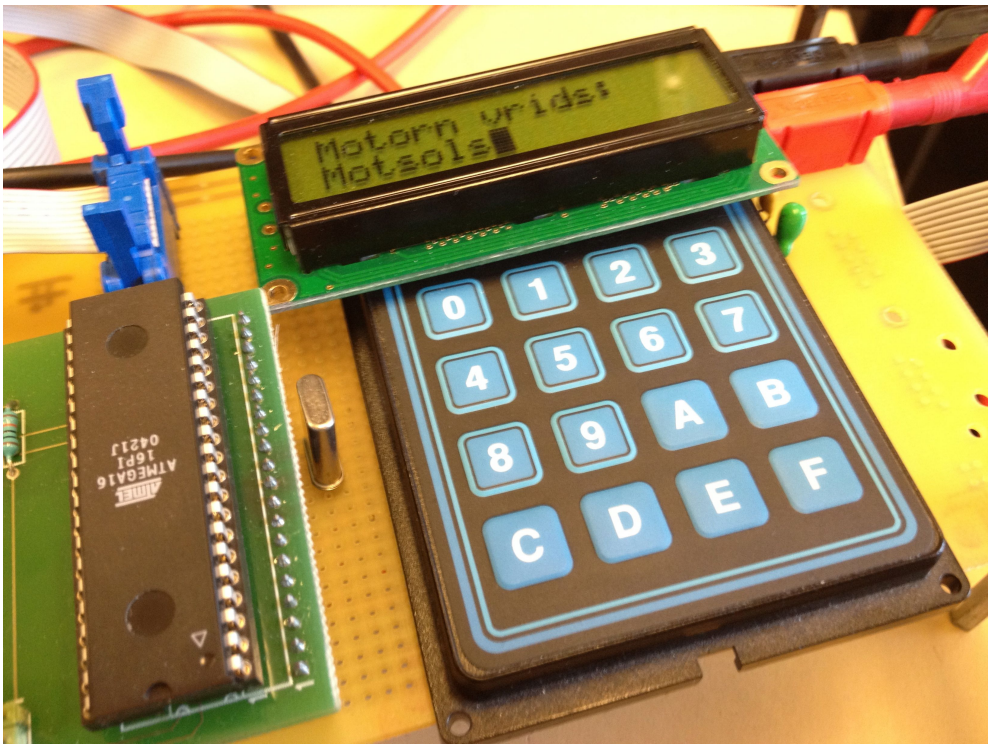
# Tidsstyrd persiennöppnare

Projektrapport i Digitala Projekt - EITF11,

Elektro- och informationsteknik, Lunds Tekniska Högskola

Carl Agmén, Tobias Gard och Viktor Kjellin. I-09




## ABSTRACT

With the general sleeping-in student in mind this project had the goal set on constructing a timer/alarm and directly remote controlled shade opener, for those hard-to-get-up mornings. Due to a lack of needed pre-knowledge about hardware-software programming and integration, combined with time factors, some of the features, such as the alarm by time and wireless connection, had to be down prioritised in favour of getting a final prototype with the desired main feature; to control a servo motor using a direct or time delayed signal. The project has taught us a lot about the difficulties of planning and building electronic constructions, and will therefore most likely give us a greater respect for people working with this that we will encounter in our future professional careers.

# Inledning

Den stereotype studenten beskrivs ofta som en nattaktiv varelse som föredrar att spendera dygnets första ljusa timmar i en mjuk säng, i ett mörkt rum. I de fall det ändå hade varit fördelaktigt att istället befinna sig på en mer produktiv ort, är botemedlet mot det där mysigt mörka rummet enkelt; Varde ljus!

Målet med kursen Digitala projekt är att konstruera något mer eller mindre användbart med hjälp av mikroprocessorn ATmega16 och övriga erforderliga komponenter. Med bakgrund i föregående stycke satte dessa tre morgontrötta teknologer upp målet att konstruera den revolutionerande timerinställda persiennöppnaren, med total avsaknad av snooze-knapp.

# Teori

## Kravspecifikation

För att få översikt på vad som behövde göras, och få en bild av grundkonstruktionen sattes ett antal krav för uppbyggnad och funktion hos den färdiga konstruktionen.

### Ursprungliga krav

Konstruktionen består av två separata enheter som tillsammans bildar den slutgiltiga konstruktionen. De båda enheterna kommunicerar trådlöst. De två enheterna är:

- En motorenhet som förutom processor består av en servomotor och trådlös mottagare. Motorenheten står för den fysiska öppningen av persiennen.

- En kontrollenhet som förutom processor består av huvudkomponenterna en knappsats, display samt en trådlös sändare. Kontrollenheten skickar signaler till servoenheten när det är dags att öppna eller stänga persiennen.

### *Funktionella krav*

- Kontrollenheten ska kunna skicka en trådlös signal till motorn som vrider persiennen i rätt position.
- Motorn alternativt kontrollenheten ska veta när persiennen är öppen eller stängd så att samma kommando inte ska kunna ges två gånger.
- Signalen ska kunna skickas på ett bestämt klockslag eller efter en utsatt tid. (Klocka och timer)
- Persiennen ska manuellt, genom direkttryck, kunna öppnas och stängas med kontrollenheten.

- Kontrollenheten ska hela tiden veta vad klockan är. (Extern kristall)
- Alla användarinställningar (alarmtid exempelvis) ska kunna ställas in med knappsatsen och visualiseras i displayen.
- Det ska gå att ställa in aktuellt klockslag på kontrollenheten.

### Användargränssnitt

I displayen visas om nedräkning/alarm är aktiverat och i så fall när öppning/stängning kommer ske, antingen som klockslag eller nedräkning.

### Knappval

- **0-9:** Siffror för inmatning av alarmtid.
- **A:** På startsida, gå till inställning av alarmtid. Vid inställning av alarmtid, bekräfta alarmtid och gå tillbaka till startmeny.
- **B:** Direktstyrning av motor, motsols.
- **C:** Markör vänster vid inställning av alarmtid.
- **D:** Markör höger vid inställning av alarmtid.
- **E:** Nollställning av alarmtid, åter till startsida.
- **F:** Direktstyrning av motor, medsols.

### Inkluderade komponenter

#### Kontrollenhet

- Mikroprocessor AVR, ATmega16
- 16x2-raders Alfanumerisk LCD Display, Sharp LM162
- Steglös spänningsomvandlare för displaykontrast
- 16-tangenters knappsats, Grayhill 88BB2-072
- 16-key Encoder, MM54C922
- Sändarkomponent till Single Chip 2.4GHz Transceiver, nRF24L01
- Extern kristall, 16 MHz
- Kondensatorer, 1,5 resp 15 farad
- Spänningsomvandlare 3,3v

#### Motorenhet

- Mikroprocessor AVR, ATmega16
- Mottagarenhet till Single Chip 2.4GHz Transceiver, nRF24L01
- Mini High-Speed Digital Servo, GS-D9257

## Genomförande

### Förberedelser

Arbetet med persiennöppnaren inleddes med att göra en grov skiss över vilka komponenter som behövdes och vilka krav som skulle ställas på den färdiga prototypen. Detta resulterade i kravspecifikation och kopplingsschema etcetera som beskrivits ovan.

### Montering och testning

Utefter det framarbetade kopplingsschemat monterades de olika komponenterna på två kretskort genom lödning och trådvirning. Ledningar innehållandes 5V löddes fast på kretskortens kontaktpunkter och komponenter, medan övriga ledningar virades på komponenternas pinnar med ändamålsenligt verktyg. Under arbetets gång förändrades konstruktionen vid några tillfällen vilket gjorde att gruppen fick ytterligare träning i trådvirande.

Efter första montering testades de enskilda komponenterna genom debugging i Studio 4 på dator där processorn kopplades in i datorn med hjälp av en AVR JTAGICE mkII. Debuggingen blev förutom återkoppling på om kopplingarna planerats och utförts riktigt, även vägen att gå för att lära sig hur komponenterna skulle styras och i förlängningen hur mjukvaran skulle utformas.

### Mjukvaruprogrammering

Koden som lades in på mikroprocessorn skrevs i olika filer beroende på funktion (main/display/key-pad etc.), allt i Studio 4 på dator. Källkoden i sin helhet finns i appendix 1.

## Resultat

Till följd av diverse motgångar samt större svårigheter än vi förutsett med konstruktionen får den färdiga persiennöppnaren något färre funktioner än planerat. Huvudfunktionen att kunna styra en motor direkt eller med tidsfördröjning är dock fortfarande uppfylld.

Konstruktionen består av två enheter där kontrollenheten uppfyller ursprungliga krav, bortsett från att tidsstyrningen endast består av en timer. Alltså kan man inte ställa in ett specifikt klockslag, men väl räkna ut hur lång tid man vill ställa in på egen hand. Motorenheten fungerar enligt de uppsatta kraven, men är för tillfället inte kopplad till en fysisk persienn. Det som är fastställt är att styrningen av själva servomotorn fungerar.

Till slut gick vi även ifrån den trådlösa kopplingen mellan enheterna. Den trådlösa sändaren respektive mottagaren monterades initialt, men på grund av tidsbrist ansåg vi att det var viktigare att få en fungerande slutprototyp än att den skulle fungera trådlöst och använde oss istället av en trådbunden SPI-koppling, som är en buss för synkron seriekommunikation bestående av en master- och en slaveenhet likt de trådlösa komponenterna. Därmed blir kommunikationen mellan enheterna i praktiken som den trådlösa, men stabilare och enklare.

## Diskussion

Under kursens gång har vi lärt oss mycket om hur elektronisk apparatur faktiskt fungerar och framförallt komplexiteten som ligger bakom de flesta av våra vardagsprodukter.

Största lärdomen gruppen har tagit av kursen är antagligen enkelt uttryckt att det är komplicerat och ligger mycket jobb bakom att framställa elektriska konstruktioner. Även om vi i framtiden inte kommer genomföra liknande projekt på egen hand kommer vi säkerligen besitta en större respekt och förståelse för de personer som vi möter som gör det.

# Referenser

## Datablad och manualer


- Mikroprocessor AVR, ATmega16, http://www.eit.lth.se/fileadmin/eit/courses/edi021/datablad/Processors/ATmega16.pdf
- 16x2-raders Alfanumerisk LCD Display, Sharp LM162, http://www.eit.lth.se/fileadmin/eit/courses/edi021/datablad/Display/LCD.pdf
- 16x2-raders Alfanumerisk LCD Display, Sharp LM162, http://www.eit.lth.se/fileadmin/eit/courses/edi021/datablad/Periphery/Other/MM54C922.pdf
- Single Chip 2.4GHz Transceiver, nRF24L01, nRF24L01_Product_Specification_v2_0.pdf, hämtad från http://www.nordicsemi.com/eng/Products/2.4GHz-RF/nRF24L01
- Mini High-Speed Digital Servo, GS-D9257, http://www.pololu.com/catalog/product/2140/faqs

# Appendix

## Källkod

### Källkod kontollenheten

```
#include <avr/io.h>
#include "lcd.h"
#include <avr/interrupt.h>
#include <util/delay.h>

#define F_CPU 1000000UL

char buttonPressed;
int hour1, hour2, min1, min2, s1, s2, cursorPlace, motorPlace, aPressed;
#define DD_MOSI     PINB3
#define DD_SCK      PINB5
#define DDR_SPI     PORTB
#define CSN_LOW() PORTB &= ~ (1<<PB4);
#define CSN_HIGH() PORTB |= (1<<PB4);

ISR(INT0_vect) { //Interupt när knapp på tangentbordet trycks ner
    DDRA = 0x00;
    _delay_ms(10);
   PORTD &= ~_BV(PD3);           //Skickar Output Enable när INT0 triggas
   _delay_ms(10);
    buttonPressed = PINA & 0x0F;
    if(aPressed == 1){
        if(buttonPressed >= 0 && buttonPressed <= 9){
            set_time(buttonPressed);
        }
        if(buttonPressed == 10){
            confirm_time();
        }
        if(buttonPressed == 11){
            motor_turn(1);
        }
        if(buttonPressed == 12){
            cursor_left();
        }
        if(buttonPressed == 13){
            cursor_right();
        }
        if(buttonPressed == 14){
            reset_program();
        }
        if(buttonPressed == 15){
            motor_turn(2);
        }
    } else if(aPressed == 0){
        if(buttonPressed >= 0 && buttonPressed <= 9){
            return;
        }
        if(buttonPressed == 10){
            _delay_ms(10);
            set_alarm();
        }
        if(buttonPressed == 11){
            motor_turn(1);
        }
        if(buttonPressed == 12){
            return;
        }
        if(buttonPressed == 13){
```

```
}
```

```
        if(buttonPressed == 14){
            reset_program();
        }
        if(buttonPressed == 15){
            motor_turn(2);
        }
    } else {
        if(buttonPressed >= 0 && buttonPressed <= 9){
             return;
        }
        if(buttonPressed == 10){
            return;
        }
        if(buttonPressed == 11){
            motor_turn(1);
        }
        if(buttonPressed == 12){
            return;
        }
        if(buttonPressed == 13){
            return;
        }
        if(buttonPressed == 14){
            reset_program();
        }
        if(buttonPressed == 15){
            motor_turn(2);
        }
    }
}

void set_alarm() //Visar "sätt alarm"-menyn
{
    disp_alarm();
    hour1 = 0;
    hour2 = 0;
    min1 = 0;
    min2 = 0;
    s1 = 0;
    s2 = 0;
    cursorPlace = 0;
    disp_time(hour1, hour2, min1, min2, s1, s2);
    disp_cursorFirst(); //Sätter markören längs ned till vänster
    aPressed++;
    return;
}

void motor_turn(val) //Vrider motorn, 1 motsols, 2 medsols.
{
    disp_motorTurning(val);
    if(val == 1 ){
        SPI_MasterTransmit(0x01);
    } else if(val == 2){
        SPI_MasterTransmit(0x02);
    }
}

void cursor_left() //Flyttar markören 1 steg åt vänster
{
    if(cursorPlace == 0){
        return;
    }
    cursorPlace--;
    _delay_ms(10);
    disp_cursorShift(1); //move cursor left
    if(cursorPlace == 2 || cursorPlace == 5){
        cursor_left();
    }
    return;
}
```

```
void cursor_right() //Flyttar markören 1 steg åt höger
{
    if(cursorPlace == 7){
        return;
    }
    cursorPlace++;
    _delay_ms(10);
    disp_cursorShift(2); //move cursor right
    if(cursorPlace == 2 || cursorPlace == 5){
        cursor_right();
    }
    return;
}

void reset_program() //Startar om programmet, nollställer timer osv
{
    hour1 = 0;
    hour2 = 0;
    min1 = 0;
    min2 = 0;
    s1 = 0;
    s2 = 0;
    cursorPlace = 0;
    aPressed = 0;
    TCNT1 = 0;
    disp_Welcome();
}

void show() //Visar inskrivna siffror
{
    disp_alarm();
    disp_time(hour1, hour2, min1, min2, s1, s2);
    return;
}

void confirm_time() //Startar nedräkning
{
    disp_alarmSet();
    disp_time(hour1, hour2, min1, min2, s1, s2);
    aPressed++;
    startCountdown();
    return;
}

void set_time(int val) //Sätter vald tid på vald plats
{
    if(cursorPlace == 0){
        hour1 = val;
        show();
    } else if(cursorPlace == 1){
        hour2 = val;
        show();
    } else if(cursorPlace == 3){
        if(val >= 6){
            min1 = 6;
            min2 = 0;
        } else {
            min1 = val;
        }
        show();
    } else if(cursorPlace == 4){
        min2 = val;
        if(min1 == 6){
            min2 = 0;
        }
        show();
```

```
    } else if(cursorPlace == 6){
        if(val >= 6){
            s1 = 6;
            s2 = 0;
        } else {
            s1 = val;
        }
        show();
    } else if(cursorPlace == 7){
        s2 = val;
        if(s1 == 6){
            s2 = 0;
        }
        show();
    }
    cursorPlace = 0;
    return;
}

void startCountdown()
{
    sei();
    TCCR1B |= (1 << CS10); // Set up timer
    int i = 0;
    for (;;){
        if (TCNT1 >= 49999){
            TCNT1 = 0; // Reset timer value
            i++;
            if(i > 160){
                s2--;
                if(s2 == -1){
                    s2 = 9;
                    s1--;
                    if(s1 == -1){
                        s1 = 5;
                        min2--;
                        if(min2 == -1){
                            min2 = 9;
                            min1--;
                            if(min1 == -1){
                                min1 = 5;
                                hour2--;
                                if(hour2 == -1){
                                    hour2 = 9;
                                    hour1--;
                                    if(hour1 == -1){
                                        break;
                                    }
                                }
                            }
                        }
                    }
                }
                disp_alarmSet();
                disp_time(hour1, hour2, min1, min2, s1, s2);
                i = 0;
            }
        }
    }
    disp_timeUp();
    SPI_MasterTransmit(0x03);
}


void SPI_MasterTransmit(char cDATA)
{
    CSN_LOW(); //Slaveselect low
    SPDR = cDATA; //Skriver till buffert för att starta SPI
    while(!(SPSR & (1<<SPIF))); //Fortsätter till lyckad sändning
   CSN_HIGH();          }//Slaveselect high
```

```
void main(void)

{

    _delay_ms(50);
    hour1 = 0;
    hour2 = 0;
    min1 = 0;
    min2 = 0;
    s1 = 0;
    s2 = 0;
    cursorPlace = 0;
    aPressed = 0;

    DDRA = 0xFF;
    DDRD = 0xFB;
   GICR = 1<<INT0;                  // Enable INT0
   MCUCR = 1<<ISC01 | 1<<ISC00;     // Trigger INT0 on all activity
   GIFR = 1<<INTF0;                 // External Interupt Flag 0
   SREG = 0x80;                     // Global Interupt Set

    DDR_SPI = (1<<DD_MOSI)|(1<<DD_SCK);
   DDRB = (1<<DDB5)|(0<<DDB6)|(1<<DDB7)|(1<<DDB4);       //Config SPI-ports
   DDRB &=~ (1<<PB6);
   SPCR = (1<<SPE)|(1<<MSTR)|(1<<SPR0)|(1<<SPR1);    //Enable SPI and set to Master, clo
ckrate to fck/16

    _delay_ms(50);
    disp_init();
    disp_home();
    disp_Welcome();

    CSN_HIGH();

    sei();          //start interupts

    while(1)
    {
    }

}
```

## Källkod lcd

```
#include <avr/io.h>
#include "lcd.h"
#include <avr/interrupt.h>
#define F_CPU 1000000UL
#include <util/delay.h>

char t1, t2, t3, t4, t5, t6;
int t;
char d;

void write_cmd(char val) {
    _delay_ms(5);
    PORTA=val;
    _delay_ms(5);
    PORTD=0xCB;
    PORTD=0x8B;
}

void disp_writeCh(char val) {
    _delay_ms(5);
    PORTA=val;
    PORTD=0xEB;
    PORTD=0x8B;
}

void disp_clear() {
    _delay_ms(10);
    write_cmd(0x01); //clear display
    _delay_ms(10);
   write_cmd(0x38);    //function set
   _delay_ms(10);
}
void disp_init() {

    disp_clear();
    _delay_ms(5);
    write_cmd(0x0F); //display on
    _delay_ms(5);
    write_cmd(0x06); //entry mode set;
}
void disp_home(){
    write_cmd(0x03);
}

void disp_Welcome()
{
    DDRA = 0xFF;
    disp_clear();
    _delay_ms(10);
    disp_writeCh('W');
    disp_writeCh('e');
    disp_writeCh('l');
    disp_writeCh('c');
    disp_writeCh('o');
    disp_writeCh('m');
    disp_writeCh('e');
    disp_writeCh('!');
    _delay_ms(10);
    write_cmd(0xC0); //Byter rad
    _delay_ms(10);
```

```
    disp_writeCh('A');
    disp_writeCh(':');
    disp_writeCh('S');
    disp_writeCh('A');
    disp_writeCh(' ');
    disp_writeCh('B');
    disp_writeCh(':');
    disp_writeCh('C');
    disp_writeCh('C');
    disp_writeCh('W');
    disp_writeCh(' ');
    disp_writeCh('F');
    disp_writeCh(':');
    disp_writeCh('C');
    disp_writeCh('W');
}

void disp_alarm()
{
    DDRA = 0xFF;
    disp_clear();
    _delay_ms(10);
    disp_writeCh('S');
    disp_writeCh('e');
    disp_writeCh('t');
    disp_writeCh(' ');
    disp_writeCh('a');
    disp_writeCh('l');
    disp_writeCh('a');
    disp_writeCh('r');
    disp_writeCh('m');
    disp_writeCh(':');
    write_cmd(0xC0); //Byter rad
    _delay_ms(10);
}

void disp_time(int hour1, int hour2, int min1, int min2, int s1, int s2)
{
    DDRA = 0xFF;
    t1 = 0x30 + hour1;
    t2 = 0x30 + hour2;
    t3 = 0x30 + min1;
    t4 = 0x30 + min2;
    t5 = 0x30 + s1;
    t6 = 0x30 + s2;
    _delay_ms(10);
    disp_writeCh(t1);
    disp_writeCh(t2);
    disp_writeCh(':');
    disp_writeCh(t3);
    disp_writeCh(t4);
    disp_writeCh(':');
    disp_writeCh(t5);
    disp_writeCh(t6);
    disp_cursorFirst();
}

void disp_alarmSet()
{
    DDRA = 0xFF;
    disp_clear();
    _delay_ms(10);
    disp_writeCh('T');
    disp_writeCh('i');
    disp_writeCh('m');
    disp_writeCh('e');
    disp_writeCh('r');
    disp_writeCh(' ');
    disp_writeCh('s');
    disp_writeCh('e');
```

```
    disp_writeCh('!');
    write_cmd(0xC0); //Byter rad
    _delay_ms(10);
}


void disp_cursorFirst() //Sätter markören på första tecknet på undre raden
{
    DDRA = 0xFF;
    _delay_ms(10);
    write_cmd(0x02); //Cursor home
    _delay_ms(10);
    write_cmd(0xC0); //Byter rad
}
void disp_cursorShift(int i) //Flyttar markören, i = 1 vänster, i = 2 höger
{
    DDRA = 0xFF;
    if(i == 1){
        _delay_ms(10);
        write_cmd(0x10); //Markör vänster
    } else if(i == 2){
        _delay_ms(10);
        write_cmd(0x14); //Markör höger
    }
}

void disp_motorTurning(int i) //Skriver ut att motorn vrids
{
    DDRA = 0xFF;
    disp_clear();
    disp_writeCh('M');
    disp_writeCh('o');
    disp_writeCh('t');
    disp_writeCh('o');
    disp_writeCh('r');
    disp_writeCh('n');
    disp_writeCh(' ');
    disp_writeCh('v');
    disp_writeCh('r');
    disp_writeCh('i');
    disp_writeCh('d');
    disp_writeCh('s');
    disp_writeCh(':');
    disp_cursorFirst();
    if(i == 1){
        disp_writeCh('M');
        disp_writeCh('o');
        disp_writeCh('t');
        disp_writeCh('s');
        disp_writeCh('o');
        disp_writeCh('l');
        disp_writeCh('s');
    } else if(i == 2){
        disp_writeCh('M');
        disp_writeCh('e');
        disp_writeCh('d');
        disp_writeCh('s');
        disp_writeCh('o');
        disp_writeCh('l');
        disp_writeCh('s');
    }
}

void disp_timeUp()
{
    DDRA = 0xFF;
    disp_clear();
    _delay_ms(10);
    disp_writeCh('T');
```

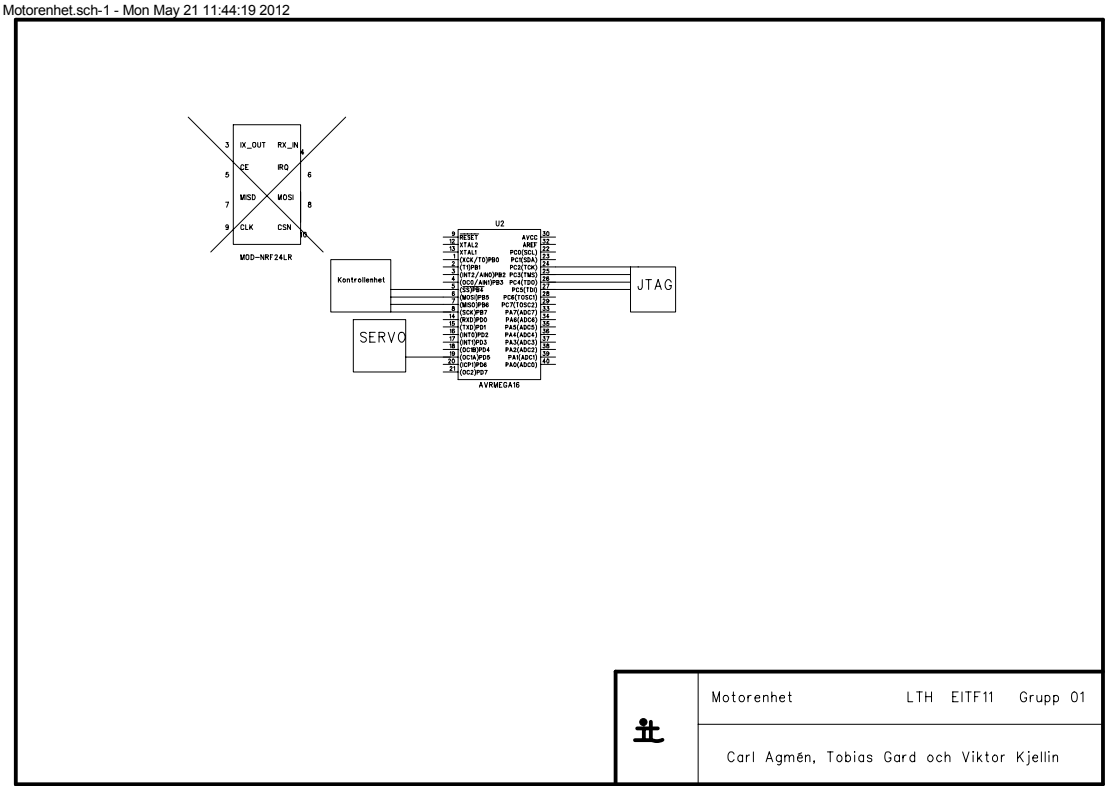
## Källkod motorenheten

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

#define F_CPU 1000000UL


#define DD_MISO      PINB6
#define DD_MOSI      PINB5
#define DD_SCK       PINB7
#define DDR_SPI      PORTB

#define ECHO          0x00
#define CW            0x01
#define CCW           0x02

int tmp = 1500; //Sätter standardvärde för PWM-puls


void SPI_SlaveInit(void)
{
    DDR_SPI = (1<<DD_MISO);
    SPCR = (1<<SPE);

}

char SPI_SlaveReceive(void)
{
    while(!(SPSR & (1<<SPIF)))
    ;
    return SPDR; //return dataregister
}

void turn_motor(int steps)
{


    if(steps > tmp)
    {
        for(int i = tmp; i<steps; i++)
            {
                _delay_ms(5);
                OCR1A = ICR1 - i;
            }
        tmp = steps;
    } else if(steps < tmp)
        {
        for(int i = tmp; i>steps; i--)
            {
                _delay_ms(5);
                OCR1A = ICR1 - i;
            }
        tmp = steps;
        }
}


void clockwise(void) //Sätt clockpuls till ~2 ms
{
    turn_motor(700);
}
```

```
int counter_clockwise(void) //Sätt clockpuls till ~1 ms
{
    turn_motor(2220);
}


         int main(void)
{
    _delay_ms(20);
    DDRB |= (1<<PB5) | (0<<PB6) | (1<<PB7) | (1<<PB4); //MOSI, SCK, SS = input pga slav
e
    DDRD = 0xFF;
    TCCR1A |=1<<WGM11 | 1<<COM1A1| 1<<COM1A0;
    TCCR1B |=1<<WGM12 | 1<<WGM13 | 1<<CS10;

    ICR1 = 19999;

    DDR_SPI = (1<<DD_MISO);
    SPCR =(1<<SPE)|(1<<SPR0);

    _delay_ms(200);


    while(1)
        {
            char bit;
            bit = SPI_SlaveReceive();
            switch(bit) {
                case 0x01:
                    clockwise();
                    break;
                case 0x02:
                    counter_clockwise();
                    break;
                case 0x03: //ser till så att servon inte vrider sig åt samma håll som s
ist
                    if(tmp==2220)
                    {
                        clockwise();
                        break;
                    } else if (tmp==700)
                    {
                        counter_clockwise();
                        break;
                    } else
                    {
                        clockwise();
                        break;
                    }
            }
        }
}
```

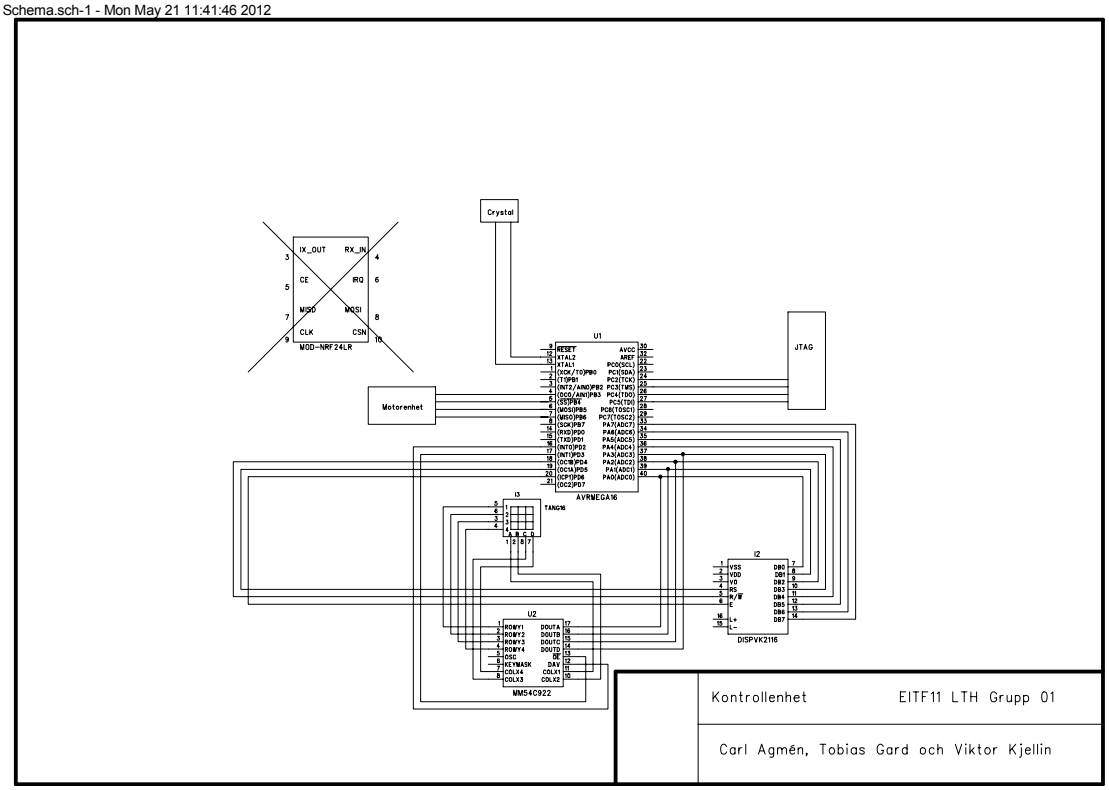
## Kopplingschema

Motorenhet.sch-1 - Mon May 21 11:44:19 2012




*Kopplingsschema över motorenheten.*

Schema.sch-1 - Mon May 21 11:41:46 2012




*Kopplingsschema över kontrollenheten.*