LUNDS TEKNISKA HÖGSKOLA
Lunds universitet

Department of Information Technology

Digital Project 5p

040119 - 040302

Automatic Light Control

Per Hyttfors, e99phy
Niklas Göransson, e98ng

Abstract

One of the most important elements in a pleasant atmosphere or a good work environment in a room is the proper lighting. The trend of compact living among students force a single room to serve as both a good study environment, as well as dining room, bedroom, and TV room. Our goal is to ease this adjustment of the lights in a room for different occasions with the use of a programmable remote control system.

This report describes the development of a prototype of this control system. Our control system uses IR to transmit the different setups to receivers connected to the lights. These different setups can be programmed and changed through a remote control panel with buttons and a numeric display. The report includes choice of components, circuit board design and program code.

Innehållsförteckning

1 Inledning

En av de viktigaste faktorerna för att skapa en trevlig atmosfär eller en bra arbetsmiljö i ett rum är belysningen. Speciellt så måste ett typiskt studentrum fungera både som studieplats, matrum, sovrum, tv-rum med mera. För att underlätta inställning av de lampor som används vid de olika tillfällena ville vi bygga en fjärrkontroll med möjlighet att lagra olika ljusprogram för olika situationer. Ett tryck på en knapp och med ens så ska lamporna vara inställda enligt ens önskemål.

1.1 Kravspecifikation

Fjärrkontrollen skall:

- Minnas åtta olika program för åtta olika lampor, även efter omstart.
- Skicka ut instruktioner, trådlöst, till dessa lampor via samma kanal [e g kunna adressera olika mottagare].
- Visa vilket program som är aktiverat för tillfället.
- Lära sig nya program via ett enkelt programmeringsgränssnitt.

Mottagaren skall:

- Ta emot ett datapaket trådlöst
- Känna igen sin adress.
- Utföra det den blir uppmanad till [e g tända eller släcka en lampa].
- Vara så liten som möjligt och kunna drivas av ett batteri.

1.2 Funktionsbeskrivning

Kretsen fungerar i korthet som följer.

När strömmen slås på skickas det program som är sparat i position ett ut till mottagarna. Man kan nu välja på att byta program med knapp ett eller två, för att skicka ut det valda programmet trycker man på ”Select”– knappen. När kretsen har skickat ut information till alla mottagare tänds ”Select”– dioden för att visa att programnumret på displayen är det aktuella ljusprogrammet.
Eller så kan man välja att programmera en ny ljussättning. Man väljer då vilken position man vill redigera och trycker på ”Program”– knappen. Nu skiftar displayen ifrån att visa vilket program som är valt till att visa vilken lampa som går att ändra. För att indikera att man är i programmeringsläge tänds ”Program”– dioden. Genom att trycka på knapp ett eller två så väljs vilken

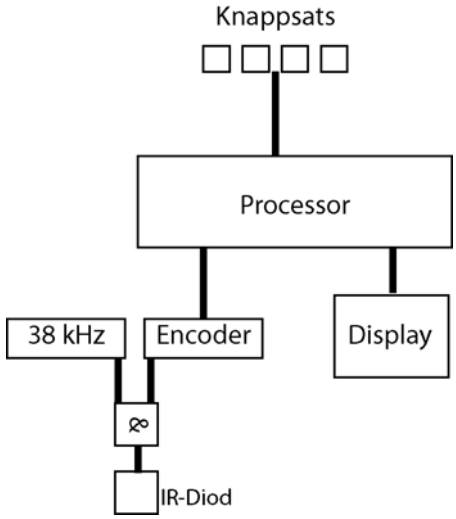
lampa som skall redigeras. För att tända eller släcka den valda lampan i det aktuella programmet används ”Select” - knappen. När man ändrar en lampa tänds eller släcks den direkt när ”Select”– knappen trycks ned. För att gå ut programmeringsläget trycks ”Program”– knappen ner igen.

1.3 Genomförande

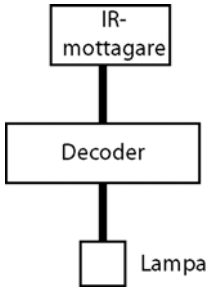
Efter att ha gått igenom våra olika alternativ så valde vi att använda Atmels AVR processor som kärna i vår konstruktion. Det är en modern processor som används i en mängd olika industriella tillämpningar, dessutom så var Per sedan tidigare bekant med AVR processorn och dess utvecklingsverktyg. Vidare så tillhandahöll institutionen ett encoder/decoder par ifrån Motorola som passade våra krav på IR delen utmärkt.
För att bygga ihop vår konstruktion använde vi institutionens färdiga kretskort och gammal hederlig virning. Vi satte dit en byggsten åt gången, verifierade att den fungerade så som vi önskade och fortsatte i detta mönster.

2 Hårdvaran

I denna del beskrivs de viktigaste delarna av vår konstruktion samt hur de samverkar. För ett utförligt kretsschema se Appendix A .



Figur 1. Blockskiss över kontrollpanelen.

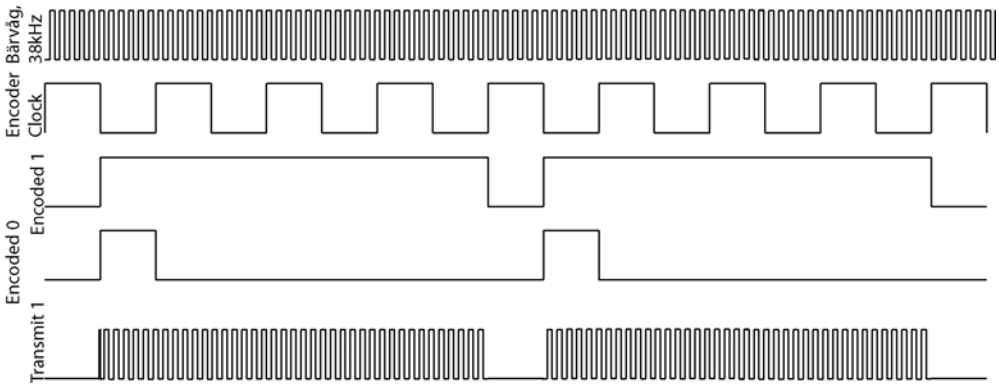


Figur 2. Blockskiss över mottagaren.

2.1 Signalering via IR

De flesta, på marknaden existerande, IR-komponenter är avstämda för en frekvens kring 38kHz. I vårt fall gäller detta mottagaren IS1U60 ifrån Sharp. Så för att kunna kommunicera med den var vi tvungna att generera en bärvåg med denna frekvens. En del olika alternativ fanns att tillgå, men vi valde att generera vågen med hjälp av en 74LS324, Voltage Controled oscillator [VCO]. Det är sedan encoderns uppgift att modulera denna våg på ett för mottagaren lämpligt sätt.

Motorola-kretsarna använder sig av vågtyperna som är illustrerade nedan. Ett kritiskt moment vid denna typ av signalering är att encodern och decodern skall vara väl avstämda med avseende på den interna klockan. En sänd etta skall tolkas som precis en etta, inte som 1½ eller 2 och så vidare. Mer om detta under rubrik 2.4.



Figur 3. Skiss över en IR signal och dess komponenter.

2.2 Komponenter

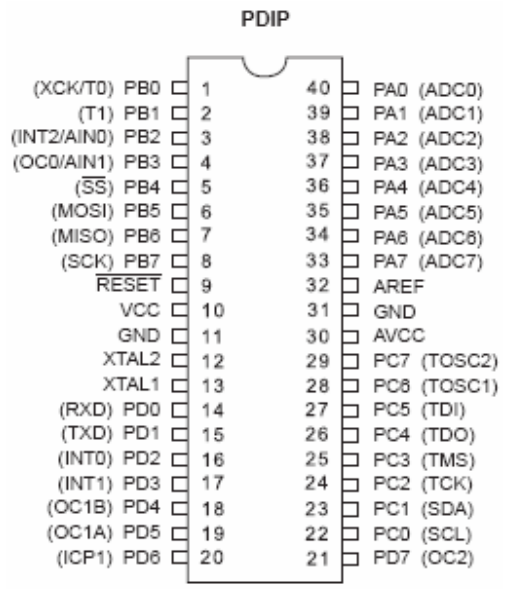
Nedan följer en uppräkning och en förklaring på de viktigaste komponenterna.

2.3 Atmel AVR

Processorn innehåller de flesta av de funktioner som vår konstruktion består av, digitala in- och utgångar, minne (1Kb SRAM, 512byte EERAM och 16Kb flash programminne).

In- och utgångarna är grupperade i fyra block (A-D) om åtta portar:

- A-blocket består av A/D ingångar men de går även att programmera som digitala in- eller utgångar, vilket vi valde att göra. Vi använder A-portarna för att kommunicera med IR encodern (MC145026).
- B-blocket är digitala in- eller utgångar. B-portarna styr 7-segmentet, program- och selectdioden.
- C-blocket är digitala in- eller utgångar. C-portarna används i vårt fall endast för att ansluta J-TAG gränssnittet.
- D-blocket är även de in- eller utgångar. D-portarna fungerar i vårt fall som ingångar för knapparna.



Figur 4. AVR Atmel 16 benkonfiguration.

2.4 Motorola 145026/27

Encoder/decoder paret är speciellt avsett för kommunikation via IR. Med hjälp av dem kan man särskilja 32 olika adresser och skicka fyra bitar data per sändning. Med hjälp av ett externt RC nät så ställs de interna klockorna i kretsarna. Vi valde att arbeta med en frekvens på 2kHz.

Encodern fungerar som följer:
När transfer enable [TE] blir låg skickas informationen från adress- och dataingångarna ut seriellt på Digital Out. Ettor och nollor kodas enligt figur 1. Informationen skickas två gånger för att mottagaren skall kunna verifiera att sändningen är korrekt.

Decodern fungerar så här:
Efter det att det första datapaketet tagit emot lagras det internt för att jämföras med nästkommande paket. Om innehållet i de båda stämmer överens så läggs informationen ut på datautgångarna och Valid Transfer[VT] går hög. Utgångarna kommer sedan att hålla dessa värden tills det att ny, giltig, data tagits emot.

Speciellt om inkopplingen av encodern i vår krets är att vi bara använder fyra bitar i processorn för att adressera en mottagare. Därför vi har virat en av adressbitarna till en logisk etta. På samma sätt använder vi bara en av databitarna och de övriga tre är virade till noll. Vi har även valt att sätta en inverterare på TE eftersom utgången på processorn är låg under hela uppstarten.

2.5 74LS324

324:an används för att generera den bärvåg som behövs för att skicka information via IR-dioden. Genom att koppla en potentiometer mellan V+ och Frequens Control ingången kan önskad frekvens lätt ställas in.

2.6 IR-komponenter

IS1U60 Remote Control Detector är avstämd för en frekvens på 38kHz. Funktionen är att ta emot IR-signalen och filtrera bort 38kHz vågen och skicka ut en TTL kompatibel etta när den tar emot en våg och en nolla då den inte tar emot något.

TSUS5400 IR-Photo Diode är helt enkelt en IR-diod. Det som skiljer sig ifrån en vanlig diodinkoppling i vår krets är dimensioneringen för att köra så mycket ström som möjligt igenom dioden. Eftersom den bara är öppen under väldigt korta tider så tål den upp emot en 1A.

3 Mjukvaran

Nedan följer en beskrivning av mjukvaran samt dess utvecklingsmiljö. Även information om hur portarna på AVR processorn är konfigurerade av programmet.

3.1 Utvecklingsmiljö

Till hjälp vid utvecklingen med AVR processorn användes Atmels AVR JTAG ICE. JTAG ICE är ett komplett verktyg som möjliggör full programmering samt debug av samtliga 8 bitas AVR processorer med JTAG interface. JTAG styrs genom programvaran AVR Studio som är har en integrerad utvecklingsmiljö för att skriva och debugga AVR applikationer. AVR Studio har även en assembler/disassembler samt en mycket bra simulator av AVR processorer. Denna simulator användes flitigt vid utvecklingen när JTAG inte fanns att tillgå. Som C-kompilator användes AVR-GCC som är en mycket bra "Open source" mjukvara och finns tillgängligt till stor sett samtliga plattformar.

3.2 Gränssnitt, I/O

Tabell 1 visar hur portarna på AVR processorn är konfigurerade av programmet. Samtliga portar som är konfigurerade som ingångar har den interna pull-up resistorn aktiverad. Initiering av portarna sker i huvudprogrammets *intit()* funktion.

Port	Pin	Function	I/O
A	0	Hardware address	O
A	1	Hardware address	O
A	2	Hardware address	O
A	3	Hardware address	O
A	4	Data	O
A	5	Transmitt enable	O
B	0	Display	O
B	1	Display	O
B	2	Display	O
B	3	Display	O
B	4	Select LED	O
B	5	Latch enable	O
B	6	Program LED	O
D	0	(-) button	I
D	1	(+) button	I
D	2	Select button	I
D	3	Program button	I

Tabell 1. AVR ATmega16 portkonfiguration.

3.3 Lagring av data

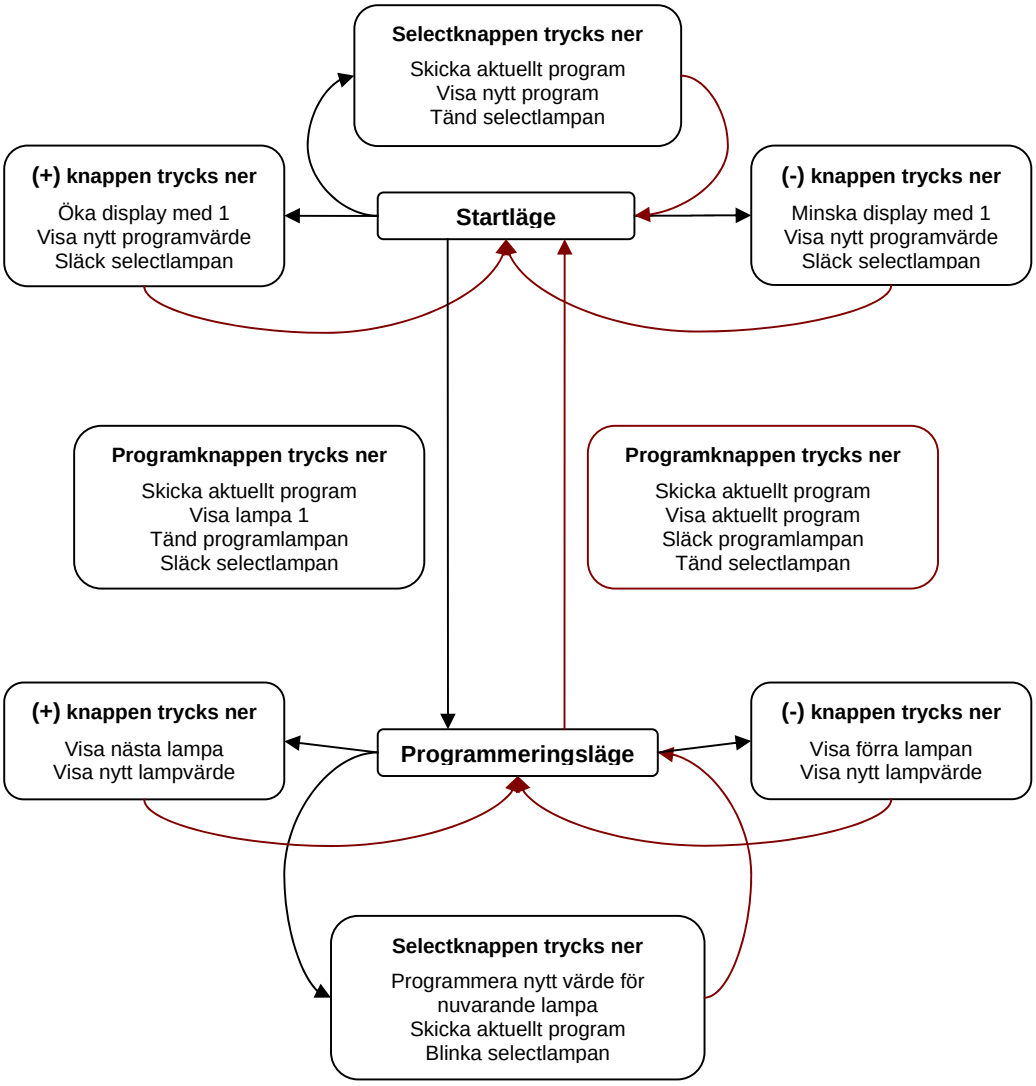
För att spara programinställningarna för de olika lamporna räcker det med att använda det interna 512 byte stora EEPROM minnet. Denna typ av minne lämpar sig särskilt bra då data sparas även då strömmen bryts till kontrollpanelen och minnet. Lagringen av de 9 olika programmen lagras linjärt i minnet, dvs. först de 9 lamporna för det första programmet, sedan de 9 lamporna för nästkommande program osv. Således så representeras varje lampa av en individuell adress i minnet för varje program.

3.4 Programkod

Här följer en förklaring av de olika programkodernas funktion och betydelser. Samtliga koder finns välkommenterade i sin helhet i Appendix B.

alc_v3.c

Detta är huvudprogrammet och innehåller den fullständiga koden bortsett från *timerpause()* funktionen som är importerad från Procyon AVRlib biblioteket. Programkoden börjar med att definierar olika värden till lätt använda och logiska namn. Detta möjliggör även att man enkelt senare kan ändra värden utan att behöva söka igenom och ändra i hela koden. Även globala variabler skapas som skall kunna nås av samtliga funktioner i koden. Sedan skapas en rad olika funktioner som anropas senare av huvudprogrammet. Huvudprogrammet startar med att köra en *init()* funktion som initialiserar samtliga portar som skall användas samt skapa den fasta adresslistan med de olika lampornas hårdvaruadress. Program 1 visas och sätts som aktuellt program och sänds ut till samtliga lampor. Programmet går sedan in i starttillståndet, som enligt flödesdiagrammet (Figur 5) nedan kallas startläge.



Figur 5. Flödesdiagram för huvudprogram.

Programmet befinner sig i starttillståndet så länge som programknappen inte har blivit ner tryckt. För att veta om en knapp har blivit ner tryckt så körs funktionen *button_poll()* som pollar in-portarna och kolla efter ändringar. Anledningen att polling används istället för interrupts är att programmet inte är tidskritiskt på något sätt. Det är även inte är önskvärt att låta någon knapptryckning avbryta en exekvering av kod, t ex vid uppdatering av displayen. De olika knapptryckningarnas funktion beror på i vilket tillstånd programmet befinner sig i. I startläget så väljs önskat program med (+) och (-) knapparna och sänds ut genom att trycka på selectknappen. Sändning av program görs genom att funktionen *send_program()* körs. Funktionen hämtar data ur minnet för samtliga lampor för det aktuella programmet och sänder ut dessa data till respektive

adress under några ms. Displayen visar i starttillståndet hela tiden programnumret. För att visa en siffra eller tända/släcka en lampa anropas någon av funktionerna *display_dot(),display_progdot()* eller *display()*, där den sistnämnda funktionen sänder ut angiven siffra till 7segmentet. När programknappen trycks ner i starttillståndet så byts läget och displayen representerar nu i stället lamporna i det aktuella programmet som visas av de nuvarande lamporna i rummet. Man kan sedan tända och släcka respektive lampa genom att trycka på selectknappen. Val av lampa görs med (+) och (-) knapparna. För att ändra data för aktuell lampa läses och uppdateras EEPROM- minnet på den aktuella adressen med ny data. Funktionerna som tar hand om detta är *EEPROM_write()*, *EEPROM_read()* och *EEPROM_address()*. Den sistnämnda funktionen används enbart för att beräkna den korrekta adressen givet önskat program och lampa. När sedan programknappen trycks ner igen när programmet befinner sig i programmeringsläget, så återgår programmet till starttillståndet efter att ha sänt ut det aktuella programmet en sista gång. Displayen återgår till att visa programnummer istället för lampnummer.

global.h

Denna kod används för att definiera globala projekt inställningar som används av Procyon AVRlibs funktioner då dessa är skrivna för att kunna användas till samtliga AVR processorer. Processorns klockhastighet definieras här för ATmega16 som 16Mhz vilket används senare av timerfunktionerna.

timer.c/timer.h

Procyon AVRlib timerfunktioner som utnyttjar den interna klockan hos AVR processorn. Funktionen som användes var timerPause() som pausar för exakt antal givna ms. En enkel pause funktion skulle kunna ha skapats av oss själva men varför uppfinna hjulet igen. AVRlib är en mycket behändig och lättanvänd skara av färdiga funktioner för många vanliga användningsområden som man ofta stöter på. Mer information om AVRlib går att finna under sektionen referenser.

4 Resultat

Resultatet blev en fungerande prototyp som kan styra två lampor. Vi har inte implementerat 230V reläerna eller batteridriften av mottagarna.

5 Förbättringsförslag

Följande punkter är förslag till förbättringar av konstruktionen.

- Förbättring av IR delen, vi besitter inte alla färdigheter för att konstruera en optimal IR-signaleringskanal.
- Eftersom vi lätt kan skicka fyra bitar data till mottagaren så hade man kunnat styra en ljusdimmer med hjälp av konstruktionen.
- Istället för att använda IR hade man med fördel kunnat använda radiovågor. Encoder/decoder kretsarna fungerar även till radioöverföring, men det blir betydligt mer analog konstruktion än med IR.

6 Slutsats

Slutligen så kan man konstatera att det gick bra att konstruera en krets som uppfyllde kravspecifikationen. Vårt val av processor och andra komponenterna visade sig även uppfylla våra krav. En förfining av prototypen skulle utan problem gå att använda i ett studentrum, av en vanlig student.

Kursen har varit mycket bra och givande, speciellt så var det kul att få komma med idén till projektet själv.

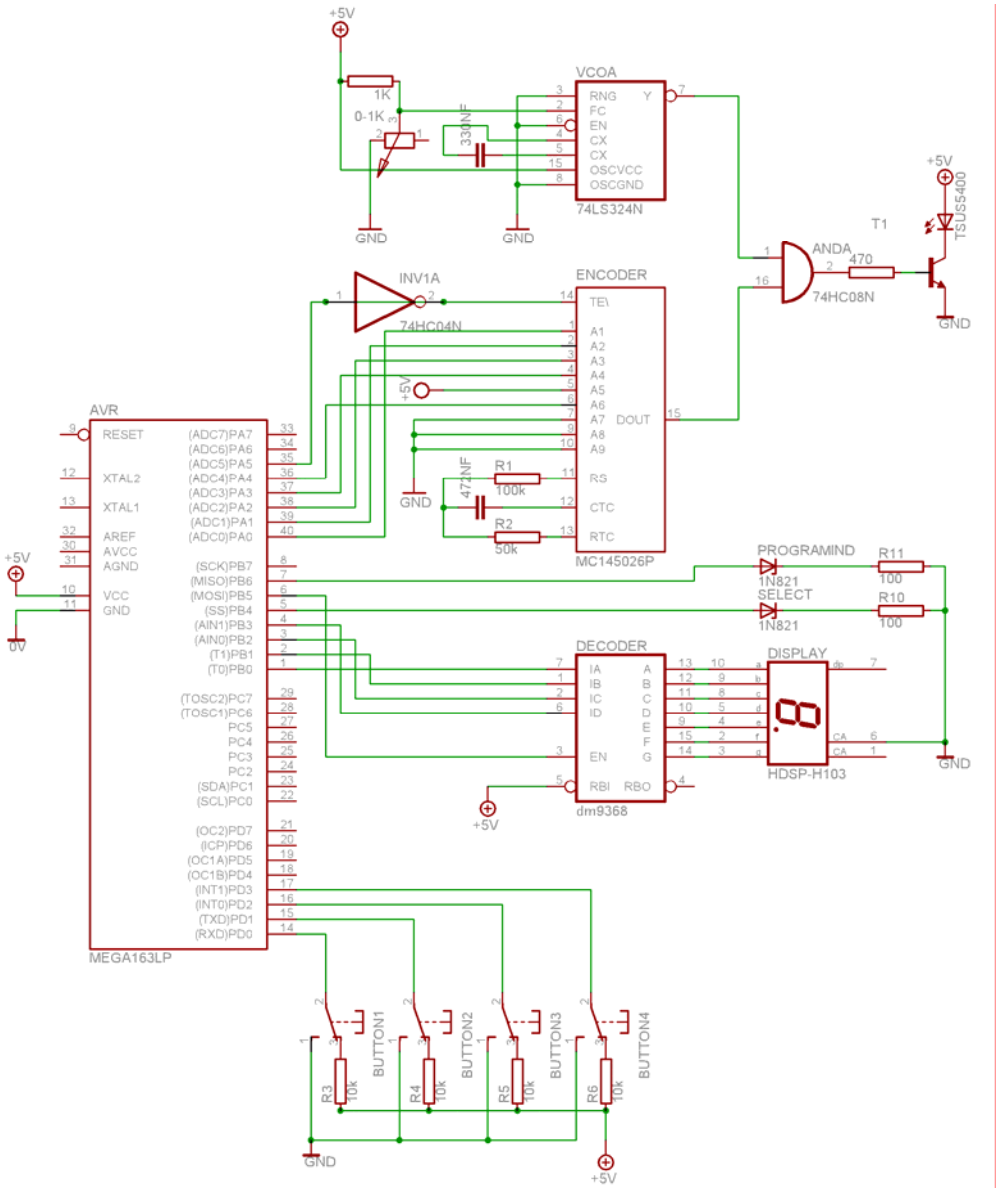
7 Referenser

[1] AVRlib [www], Tillgängligt på (hämtat 2 mars 2004) <http://www.procyonengineering.com/avr/avrlib/index.html>

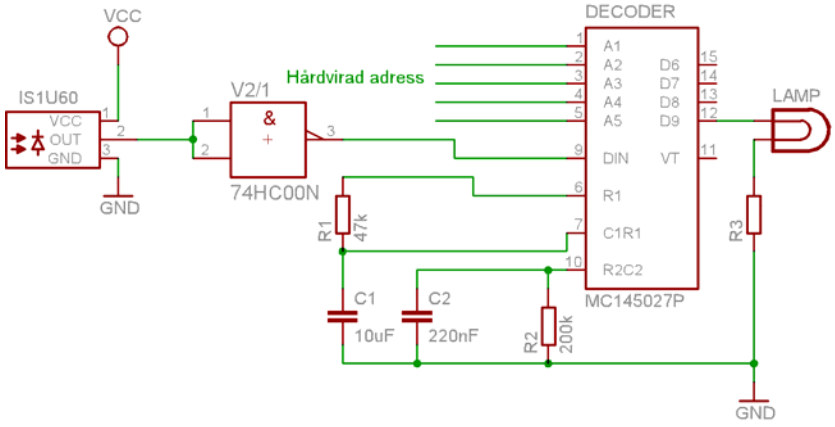
[2] Datablad för alla kretsar [www], Tillgängligt på (hämtat 2 mars 2004) <http://www.it.lth.se/courses/digp/infobank.html>

Appendix A: Kretsschema

Kontrollpanelen



Mottagaren


VCC
IS1U60
VCC
OUT
GND
GND
V2/1
&
74HC00N
Hårdvirad adress
DECODER
A1
A2
A3
A4
A5
D6
D7
D8
D9
DIN
VT
R1
C1R1
R2C2
MC145027P
LAMP
R1
47k
C1
10uF
C2
220nF
R2
200k
R3
GND

Appendix B: Programmet

alc_v3.c

```
//*****************************************************************************
// File Name   : alc_v3.c
//
// Title       : Software to drive the ATmega16 and the controller functions
// Revision    : 3.0
// Notes       : Created by Per Hyttfors, February 2004
// Target MCU  : Atmel AVR series
//*****************************************************************************

//----- Include Files -----
#include <avr/io.h>           // include I/O definitions (port names, pin names, etc)
#include <avr/signal.h>       // include "signal" names (interrupt names)

#include "global.h"           // include our global settings
#include <timer.h>            // include timer function library

//----- Defines -----
#define ON 1
#define OFF 0

#define HIGH 1
#define LOW 0

#define START_ADDRESS 25      // Start address for the EEPROM data.

#define DISPLAY0 0            // Port Pin 0 for digit display signal binary
#define DISPLAY1 1            // Port Pin 1 for digit display signal binary
#define DISPLAY2 2            // Port Pin 2 for digit display signal binary
#define DISPLAY3 3            // Port Pin 3 for digit display signal binary
#define DISPLAY_DOT 4         // Port Pin 4 for digit dot on/off
#define LE 5                  // Port Pin 5 for Latch Enable, LE on/off
#define DISPLAY_PROGDOT 6     // Port Pin 6 for programming dot on/off

#define BUTTON0 0     // Port Pin 0 for button.
#define BUTTON1 1     // Port Pin 1 for button.
#define BUTTON2 2     // Port Pin 2 for button.
#define BUTTON3 3     // Port Pin 3 for button.

#define DATABIT 4     // port pin for data bit, on/off lamp
#define SENDBIT 5     // port pin for enable transfer

#define B_DOWN 1      // arrow button down dec code
#define B_UP 2        // arrow button up dec code
#define B_SELECT 4    // select button dec code
#define B_PROG 8      // programming button dec code

#define STATE_PROG 1  // The programming state.
#define STATE_MAIN 0  // The main state, and also starting state.

//----- VARIABLES -----

// "unsigned char" is an 8-bit variable with a range of 0-255
// using "static" on a global variable means that the variable cannot be accessed or used
// outside of this file

// holds the current value of the display, the lamp and the current program
static unsigned char value, lamp, current_program;
// holds the hardware address for up to 9 different lamps.
unsigned char address_list[9];


//----- Functions -----

// Return EEPROM address for a given program and lamp
unsigned int EEPROM_address(unsigned char programet, unsigned char lampa)
{

        unsigned int eeprom_address = 0;
```

```
        // calculate the current address in memory
        eeprom_address = START_ADDRESS + (9*programet) + lampa;

        return eeprom_address;
}

// Write EEPROM data to address
void EEPROM_write(unsigned int uiAddress, unsigned char ucData)
{
        // Wait for completion of previous write
        while(EECR & (1<<EEWE))
                ;

        // set up address and data registers, EEPROM Address, EEAR. EEPROM Data, EEDR.
        EEAR = uiAddress;
        EEDR = ucData;

        // write logical one to EEMWE, EEPROM Master Write Enable
        EECR |= (1<<EEMWE);

        // start eeprom write by setting EEWE, EEPROM Write Enable
        EECR |= (1<<EEWE);
}

// Read EEPROM data, return data given an address
unsigned char EEPROM_read(unsigned int uiAddress)
{
        // wait for completion of previous write
        while(EECR & (1<<EEWE))
                ;

        // set up address register, EEPROM Address, EEAR.
        EEAR = uiAddress;

        // start eeprom read by writing EERE, EEPROM Read Enable
        EECR |= (1<<EERE);

        // return data from register, EEPROM DATA, EEDR.
        return EEDR;
}


// Display function, show given digit on the display
void display(unsigned char digit)
{
        // logic low to Latch Enable, open display for new data
        cbi(PORTB,LE);

        if (digit == 0)
        {
                // sbi(x,y) takes the CPU register x and sets bit y to 1, high
                // cbi(x,y) takes the CPU register x and clears bit y to 0, low

                // 0000
                cbi(PORTB,DISPLAY0);
                cbi(PORTB,DISPLAY1);
                cbi(PORTB,DISPLAY2);
                cbi(PORTB,DISPLAY3);
        }

        if (digit == 1)
        {
                // 0001
                sbi(PORTB,DISPLAY0);
                cbi(PORTB,DISPLAY1);
                cbi(PORTB,DISPLAY2);
                cbi(PORTB,DISPLAY3);
        }

        if (digit == 2)
        {
                // 0010
                cbi(PORTB,DISPLAY0);
                sbi(PORTB,DISPLAY1);
                cbi(PORTB,DISPLAY2);
                cbi(PORTB,DISPLAY3);
        }

        if (digit == 3)
        {
```

```
                // 0011
                sbi(PORTB,DISPLAY0);
                sbi(PORTB,DISPLAY1);
                cbi(PORTB,DISPLAY2);
                cbi(PORTB,DISPLAY3);
        }

        if (digit == 4)
        {
                // 0100
                cbi(PORTB,DISPLAY0);
                cbi(PORTB,DISPLAY1);
                sbi(PORTB,DISPLAY2);
                cbi(PORTB,DISPLAY3);
        }

        if (digit == 5)
        {
                // 0101
                sbi(PORTB,DISPLAY0);
                cbi(PORTB,DISPLAY1);
                sbi(PORTB,DISPLAY2);
                cbi(PORTB,DISPLAY3);
        }

        if (digit == 6)
        {
                // 0110
                cbi(PORTB,DISPLAY0);
                sbi(PORTB,DISPLAY1);
                sbi(PORTB,DISPLAY2);
                cbi(PORTB,DISPLAY3);
        }

        if (digit == 7)
        {
                // 0111
                sbi(PORTB,DISPLAY0);
                sbi(PORTB,DISPLAY1);
                sbi(PORTB,DISPLAY2);
                cbi(PORTB,DISPLAY3);
        }

        if (digit == 8)
        {
                // 1000
                cbi(PORTB,DISPLAY0);
                cbi(PORTB,DISPLAY1);
                cbi(PORTB,DISPLAY2);
                sbi(PORTB,DISPLAY3);
        }

        // logic high to Latch Enable, close display
        sbi(PORTB,LE);
}

// Update display, move the value on the display up or down
void display_update(unsigned char move)
{
        if(move == HIGH)
        {
                value++;
                if(value == 9)
                {
                        value = 0;
                }
                display(value);
        }else{
                if(value == 0)
                {
                        value = 9;
                }
                value--;
                display(value);
        }
}

// Update the lamp variable up or down
void lamp_update(unsigned char move)
{
        if(move == HIGH)
```

```
        {
                lamp++;
                if(lamp == 9)
                {
                        lamp = 0;
                }
                display(lamp);
        }else{
                if(lamp == 0)
                {
                        lamp = 9;
                }
                lamp--;
                display(lamp);
        }
}

// Update dot display
void display_dot(unsigned char displaydot)
{
        // display dot on/off
        if (displaydot == ON)
        {
                sbi(PORTB,DISPLAY_DOT);
        }
        else
        {
                cbi(PORTB,DISPLAY_DOT);
        }
}

void display_progdot(unsigned char displayprogdot)
{
        // display programming dot on/off
        if (displayprogdot == ON)
        {
                sbi(PORTB,DISPLAY_PROGDOT);
        }
        else
        {
                cbi(PORTB,DISPLAY_PROGDOT);
        }
}

// Poll the port for changes made by a pressed button and return value for the pressed
button
unsigned char button_poll(void)
{
        unsigned char button = 0, buttonAwake, buttonData;
        buttonAwake = OFF;

        // Poll PINB for changes!
        while(buttonAwake == OFF)
        {
                // read portD value
                buttonData = PIND;
                // pause the polling for a while, to ignore button bounces
                timerPause(2);
                // check if any button has been pressed, ignore 0x00 value.
                if((buttonData & 0xFF) != 0xFF && (buttonData & 0xFF) != 0x00)
                {
                        // change occured! Which button has been pressed? Invert value to
get 1,2,4,8 to represent button, 1,2,3,4.
                        button = ~(buttonData);

                        // check so the button is released
                        while((buttonData & 0xFF) != 0xFF)
                        {
                                // read portD value
                                buttonData = PIND;
                                // pause the polling for a while, to ignore button bounce
                                timerPause(2);
                        }
                        buttonAwake = ON;
                }
        }

        return button;
}
// send the program to all the lamps twice.
void send_program(unsigned char program)
```

```
{
        unsigned char lampNR;
        static unsigned int send_location = 0;
        unsigned char e_data;

        for (lampNR = 0; lampNR < 9; lampNR++)
        {
                // get eeprom address
                send_location = EEPROM_address(program, lampNR);
                // get eeprom data
                e_data = EEPROM_read(send_location);
                // set current hardware address
                PORTA = address_list[lampNR];

                // set data bit on port
                if(e_data == 0)
                {
                        cbi(PORTA,DATABIT);
                }else{
                        sbi(PORTA,DATABIT);
                }
                sbi(PORTA,SENDBIT);
                // timerPause pauses for the number of milliseconds
                timerPause(10);
                cbi(PORTA,SENDBIT);
                // wait and send again...
                timerPause(5);
                sbi(PORTA,SENDBIT);
                // timerPause pauses for the number of milliseconds
                timerPause(10);
                cbi(PORTA,SENDBIT);
        }
}

// Init function
void init(void)
{
        // initialize timers
        timerInit();

        // set port B pins to all output used for digit display signals
        outb(DDRB, 0xFF);

        // set port B pins to a value of all 0 (logic low, or 0 volts)
        outb(PORTB, 0x00);

        // set port D pins to all input used for buttons
        outb(DDRD, 0x00);

        // set port D pins to use pull-up resistors (logic high)
        outb(PORTD, 0xFF);

        // set port A pins to all output used for transmission signals
        outb(DDRA, 0xFF);

        // set port A pins to a value of all 0 (logic low, or 0 volts)
        outb(PORTA, 0x00);

        // set hardware addresses to all the lamps
        address_list[0] = 0x00;
        address_list[1] = 0x01;
        address_list[2] = 0x02;
        address_list[3] = 0x03;
        address_list[4] = 0x04;
        address_list[5] = 0x05;
        address_list[6] = 0x06;
        address_list[7] = 0x07;
        address_list[8] = 0x08;
}


//----- Main -----
void main(void)
{
        init();
        unsigned char key, state, data;

        value = 1;
        current_program = value;
        lamp = 1;
        display(current_program);
```

```
	send_program(current_program);  // skicka första programmet vid start.
	display_dot(ON);
	state = STATE_MAIN;

	// enter endless loop.
	while(1)
	{
		key = button_poll();

		if(key == B_DOWN)
		{
			if(state == STATE_MAIN)
			{
				display_update(LOW);
				display_dot(OFF);
			}
			if(state == STATE_PROG)
			{
				lamp_update(LOW);
			}
		}

		if(key == B_UP)
		{
			if(state == STATE_MAIN)
			{
				display_update(HIGH);
				display_dot(OFF);
			}
			if(state == STATE_PROG)
			{
				lamp_update(HIGH);
			}
		}

		if(key == B_SELECT)
		{
			if(state == STATE_MAIN)
			{
				send_program(value);
				current_program = value;
				display_dot(ON);
			}
			if(state == STATE_PROG)
			{
				unsigned int location;
				unsigned char eeprom_data;
				// read current data in the eeprom
				location = EEPROM_address(current_program, lamp);
				eeprom_data = EEPROM_read(location);

				if(	eeprom_data == 0)
				{
					data = 1;
				}else{
					data = 0;
				}
				// write changes to the eeprom
				EEPROM_write(location, data);
				// send current program
				send_program(current_program);
				// blink display dot to indicate transmission of program
				display_dot(ON);
				// timerPause pauses for a while
				timerPause(25);
				display_dot(OFF);
			}
		}

		if(key == B_PROG)
		{
			if(state == STATE_MAIN)
			{
				state = STATE_PROG;
				display_progdot(ON);
				display_dot(OFF);
				lamp = 1;
				display(lamp); // start with showing lamp 1 on the
display.. always...
			}else{
				state = STATE_MAIN;
```

```
                        send_program(current_program);
                        display_progdot(OFF);
                        display_dot(ON);
                        display(current_program);
                    }
                }
            }
}
```

global.h

```
//***************************************************************************
//
// File Name   : 'global.h'
// Title       : AVR project global include
// Author      : Pascal Stang
// Created     : 7/12/2001
// Revised     : 9/30/2002
// Version     : 1.1
// Target MCU  : Atmel AVR series
// Editor Tabs : 4
//
//     Description : This include file is designed to contain items useful to all
//                                  code files and projects.
//
// This code is distributed under the GNU Public License
//             which can be found at http://www.gnu.org/licenses/gpl.txt
//
//***************************************************************************

#ifndef GLOBAL_H
#define GLOBAL_H

// global AVRLIB defines
#include "avrlibdefs.h"
// global AVRLIB types definitions
#include "avrlibtypes.h"

// project/system dependent defines

// CPU clock speed
//#define F_CPU        16000000                        // 16MHz processor
//#define F_CPU        14745000                        // 14.745MHz processor
//#define F_CPU        8000000                         // 8MHz processor
#define F_CPU        7372800                       // 7.37MHz processor
//#define F_CPU        4000000                         // 4MHz processor
//#define F_CPU        3686400                         // 3.69MHz processor
#define CYCLES_PER_US ((F_CPU+500000)/1000000)      // cpu cycles per microsecond

#endif
```

timer.h/timer.c

Aktuell version av timer.h/timer.c finns att finna i AVRlib, se referenser.