EDI021

Digitala Projekt

Lazerdoom monkey

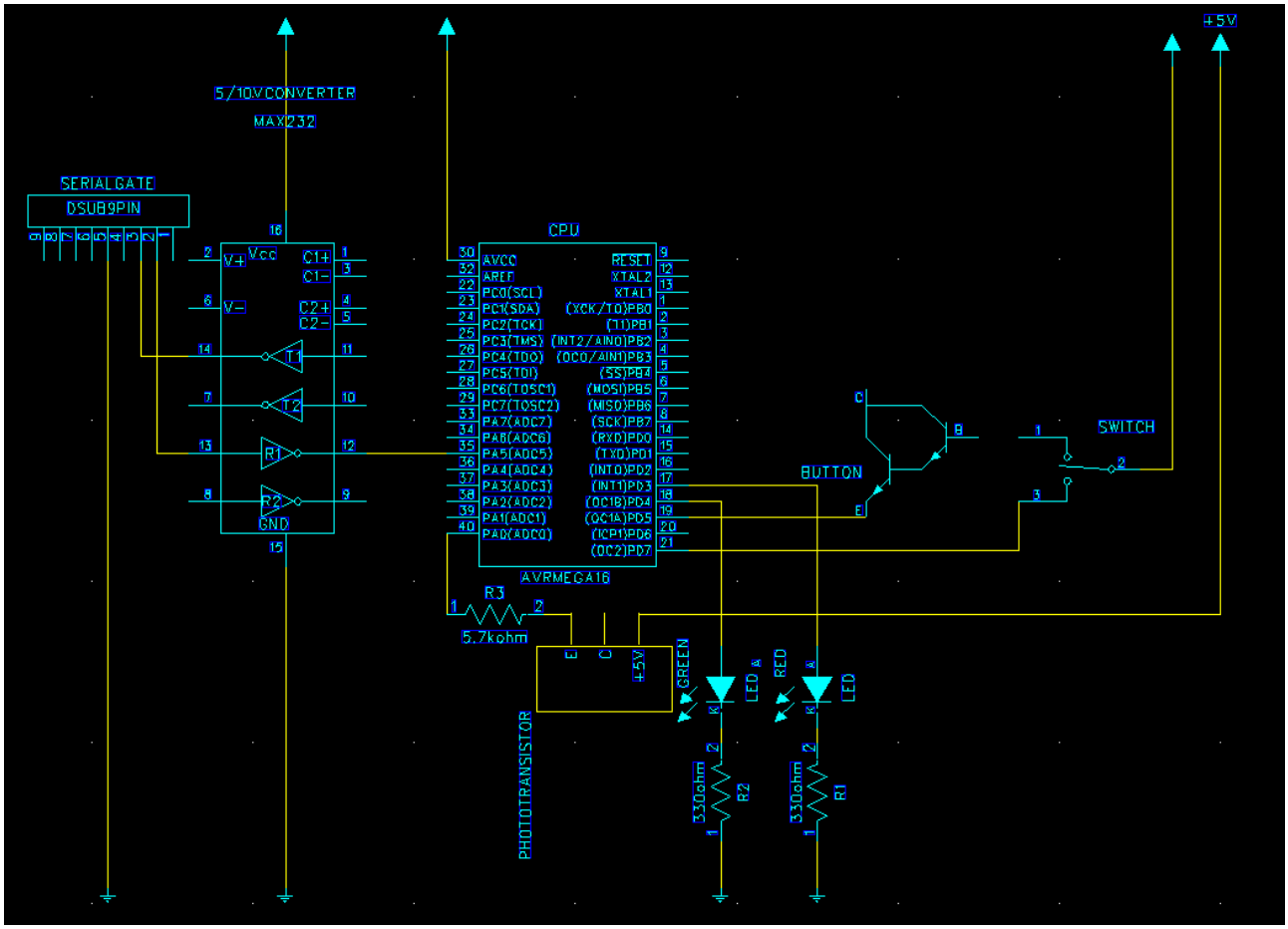
Henrik Andersson – dt05ha1
Olof Svensson – dt06os4

2009-05-18

Innehållsförteckning

1. Inledning

Projektet vi har jobbat med har gått ut på att bygga en prototyp till en laserdomevästsensor. Dvs. den delen på västen som registrerar träffen från en laser. För er som inte känner till laserdome så är det en krigssimulation där deltagarna får en väst med monterade sensorer samt ett vapen med en ofarlig laser. I vårat fall använder vi oss utav en laserpenna.
Grundtanken är att skillnaden mellan bakgrundsljuset, detta förutsätts vara mörkare än ljuset från laserpennan , och ljuset från laserpennan skall konverteras i en A/D omvandlare och sedan registreras som en träff.
Rent praktiskt har vi löst detta med en microprocessor (ATMega16) samt en fottransistor, verksam inom 450-1150nm. Som skjutattrapp använde vi oss , som tidigare nämnt, utav en helt vanlig laserpenna (~650nm).
Projektet har sträckt sig över 8 veckor under det första halvåret 2009.



Figur 1 Bild på kretsen beskriven i Power Locig

2. Specifikation

Målen vi vill uppnå är att vår prototyp skall kunna:

- Registrera en träff från en laser.
- Spara statistik om antalet träffar.
- Föra över denna statistik till en pc med rätt programvara.
- Prototypen skall kunna fungera under de flesta ljusförhållanden. Dvs. läsa av omgivningsljuset och ställa ett referensvärde beroende på detta.

I mån av tid vill vi (att):

- Lägga till fler sensorer.
- Spara mer utförlig information om träffar, såsom vilken sensor som blev träffad.
- Prototypen själv ska anpassa sig till omgivningens ljusstyrka. Det vill säga inte registrera en träff bara för att du gick in i ett ljusare rum.

Målen strävar till att göra prototypen så lik originalet som möjligt. Samt att ge den förutsättningar att faktiskt användas.

3. Lösningsförslag

Lösningen kopplar vi direkt till målsättningen. Stycket är indelat efter delmålen. En del av dessa kommer väva in varandra, här kommer de behandlas separat och kanske mer än en gång. Detta för att det ska bli lättare för utomstående att följa våra tankar för varje vital komponent.

3.1 Primär målsättning

Dessa är målen som vi ska uppfylla för att kunna anse oss ha klarat av uppgiften.

3.1.1 Registrera träff

Komponenter som används för detta är processor och fottransistor. När en ljusstråle träffar transistorn minskar resistansen över den och orsakar ett spänningsfall. Spänningsfallet tolkas sedan i processorns analog/digital omvandlare till ett värde. Detta värde i sin tur jämföras med ett referensvärde. Är detta värde högre än referensvärdet registreras en träff och en grön diod på kretsen tänds för att indikera träff.

3.1.2 Spara statistik

All statistik sparas i minnet som ”unsigned chars”. I nuvarande läget nöjer vi oss med en sensor. Läggs flera till bör statistik sparas i vektorer för att hålla reda vilka sensorer som har fått vilka träffar.

3.1.3 Föra över information till dator

För överföringen använder vi oss av en seriell uppkoppling. Denna aktiveras genom att en vippa slås till. När detta sker ställer processorn (ATMega16) i sänd-läge där den hela tiden försöker sända. Vid klartecken från datorn förs träff-statistiken över.
Vi har skapat ett enkelt java-program med tillhörande gui. I detta tittat vi på asciivärdet och konverterar träfffresultatet från unsigned char till interger. Med en knapptryckning kan man se informationen i ett textfält. Hela utförandet sker på följande vis:

1. Vippan slås över i sändarläge, vilket medför att processorn väntar på att få ett tecken från datorn.
2. Startar java-programmet och trycker på knappen i guit.
3. Ett tecken skickas över till processorn, vilket medför att den i sin tur skickar över ett tecken till java-programmet.
4. Java-programmet tar emot tecknet, gör om det till en integer och visar resultatet.
5. Processorn står och väntar på att vippan ska slås över. När den gör det är den redo att börja registrera träffar igen.

3.1.4 Funktion under flera ljusförhållanden

Genom avläsningar från transistor kommer bakgrundsljusets styrka registreras till ett referensvärde. Detta används för att avgöra hur ljus omgivningen är. I detta stadiet kommer uppdateringen av omgivningens ljusstyrka ske med hjälp av en knapptryckning. När knappen har tryckts ner så registreras det vilket ljus som fototransistorn utsätts för. För att det inte ska registreras träffar hela tiden om det bara sker lite skiftningar i bakgrundsljuset så sätts referensvärdet till lite högre än det värdet som fås av avläsningen utav fototransistorn.

3.2 Målsättning i mån av tid

Om det finns tid över när de primära målen är uppnådda kommer vi sträva efter att lösa följande uppgifter. Möjligt är att någon av dem blir implementerad innan primärmålen uppfylls om tillfälle och omständighet uppstår.

3.2.1 Lägga till flera sensorer

Detta kommer mer eller mindre ske genom att upprepa proceduren för att lägga till en sensor. För att detta ska fungera på ett lite mer sofistikerat plan bör man spara information om vilken sensor det är som har blivit träffad. Det medför att det blir lite ändringar i överföringen då man i så fall behöver skicka över information om hur många träffar varje sensor har.

3.2.2 Spara mer noggrann information om träffar

För att klara av detta så kommer vi inte längre spara informationen enskilt i minnet utan gå över till att spara den i vektorer. I dessa kommer vi lägga den information vill vi spara. Vilken sensor som träffas samt hur många gånger den träffats.

3.2.3 Automatisk uppdatering av bakgrundsljus

I koden lägger vi in en loop som inom ett visst intervall läser av bakgrundljuset och uppdaterar referensen. Man kan tänka sig att vi skiljer på en ljusimpuls och långvarigt bibehållen ljusstyrka. En impuls borde vara en träff medan det långvariga ljuset är bakgrundljuset.

4. Implementering

Vad har de viktigare komponenterna för funktion? Hur har vi implementerat dessa?

4.1 Mega16

Funktion
Detta är hjärtat i prototypen. Vi gör så lite som möjligt mekaniskt/elektroniskt utan försöker sköta allt från processorn. Konkret betyder detta att alla komponenter är kopplade till processorn. Så om knappen ger signal så bestämmer vi vad som skall ske via processorn. Koden som vi har använt oss utav finns beskriven i Appendix A.

Implementering
I stora drag så sker finns det tre saker som kan ske. Vippan andras, knappen trycks ner eller släpps samt att processorn registrerar något på den analoga ingången som fototransistorn sitter. För att sköta detta så har vi använt oss utav avbrott. Detta betyder att om det sker ett avbrott varje gång något utav det ovanstående sker, vilket sedan tas emot och olika saker sker. Det finns ett tillfälle då avbrotten stängs av, och det är när vippan står i överföringsläge. Detta för att vi inte ville att den skulle registrera träffar om vi är i det läget.

4.2 Seriell port

Funktion
Via den seriella porten kommunicerar vår prototyp med java-programmet. Koden för java-programet finns beskriven i Appendix B.

Implementering
Porten kräver 10V för att fungera, från processorn får vi 5V. Detta löste vi genom att koppla in en konverterare 5V till 10V (max232).

4.3 Fototransistor

Funktion
Transistorns resistans varierar beroende på hur mycket ljus som träffar den. Ju mer ljus desto större blir resistansen och på så vis orsakas ett spänningsfall. Eftersom den hela tiden tar in ljus och beroende på styrkan hos detta ger olika stor signal, måste den kopplas till en analog/digital omvandlare för att vi ska ha nytta av informationen den förmedlar.

Implementering
Fototransistorn är kopplad till processorn i serie med ett motstånd. Detta för att när motståndet minskar när fototransistorn blir utsatt för ljus, så kan motståndet tillslut bli 0, vilket medför att processorn i så fall kan få för mycket ström, och på så sätt gå sönder. Därför har vi ett motstånd där.

5. Utförande

5.1 Tillvägagångsätt

Vi började med att leta reda på information om de komponenter vi visste att vi skulle behöva, processor, fototransistor, överföringsport. Därefter började vi skapa en grov ritning i Powerlogic. Denna byggdes på efter hand som vi räknade fram att vi behövde fler komponenter så som resistorer, dioder, spänningskonverterare kablage mm. När vi var nöjda med resultatet började virningen.

Vi började med att få en av dioderna att lysa. På så vis bekantade vi oss med materialet och arbetsmiljön. Därefter byggdes konstruktion och kod på med tillägg av först en knapp som styrde dioden och därefter fototransistorn. Efter detta var det dax för den seriella porten och vippan som skulle ställa över konstruktionen i överföringsläge. När vi fått även detta att fungera lade vi till så att kretsen använde sig av avbrott för att hantera de olika input som behöver tas omhand.

5.2 Svårigheter

Under resans gång stötte vi på en hel del svårigheter och fick en hel del aha-upplevelser. Det första stora problemet var att upptäcka att j-tagen använde alla c-portar på vår mega16. En annan tidig sak vi upptäckte var att om man kopplade insignaler på vissa pinnar så fick andra inte användas. Att sedan hitta informationen om vilka man fick använda med varandra var inte övertydligt i databladet.
Nästa stora problem vi stötte på var att får den seriella uppkoppling att fungera. Vi spenderade många timmar med detta bara för att finna att lösningen var väldigt simpel. Vi hade förväxlat pinnarna som sköter in/ut signaler. När vi bytte på dessa fungerade överföringen.
Konverteringen av unsigned char till interger tog också lite tid att klura ut. Men även detta övervanns.
När det var klart och programmet i princip fungerade så hade vi inte implementerat det med avbrott. Detta blev nästa svårighet, att skriva om programmet så att den använde sig av avbrott. Det tog dock inte så lång tid som vi trodde, utan snart hade vi ett program som fungerade som vi ville.

6. Slutsatser

Vi anser oss nöjda med resultatet, de flesta av målen är avklarade. Det vi inte hann med var att lägga till fler receptorer samt att referensvärdet inte uppdateras automatiskt. Som prototyp fungerar konstruktionen precis så som vi kunde hoppas på.
Lite problem uppstår om man försöker registrera mer än lite drygt 65'500 träffar eller om den testas direkt under en skarp lampa.
Anledningen till att vi inte la till fler receptorer eller fixade med den automatiska uppdateringen var att tiden tin räckte till. Man måste ha i åtanke att detta projekt inte var det ända vi kunde koncentrera oss på.
Överlag är vi mycket nöjda med arbetet och arbetssättet. Friheten i uppgiften är den stora vinsten. Samt erfarenheten att få jobba med ett projekt från idéstadie till fungerande konceptmodell.
Bilder på den färdiga kretsen finns i Appendix C.

7. Referenser

Hemsidor

http://imakeprojects.com/Projects/avr-tutorial/
http://www.eit.lth.se/fileadmin/eit/courses/edi021/avr-libc-user-manual/index.html
http://www.avrfreaks.net/index.php?name=PNphpBB2&file=viewtopic&t=45341
https://www1.elfa.se/elfa~se_sv/b2b/start.do
http://pinouts.ru/connector/9_pin_D-SUB_female_connector.shtml

Datablad

http://www.datasheetcatalog.com/datasheets_pdf/M/A/X/2/MAX232.shtml
AVR JTag ICE

Appendix A

Programmet på processorn

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

#define USART_BAUDRATE 9600
#define BAUD_PRESCALE (((F_CPU / (USART_BAUDRATE * 16UL))) - 1)

int  set_PORTD_bit(int value);                  // Function prototype
void setup_analog();
void transmit();
void readLight();

unsigned int buttonStatus = 0;
unsigned int switchStatus = 0;
unsigned int level = 1024;
unsigned int lowLevel = 0;
unsigned int highLevel = 0;
unsigned int redLevel = 0;
unsigned char hits = 0;
unsigned char start;

int main(void){

        UBRRL = 12;                     // Set baudrate to 9600.
        UCSRA = _BV(U2X);               // Double the USART transmission speed.
        UCSRB = _BV(RXEN)               // Reciever enable.
        | _BV(TXEN);                    // Transmitter enable.

        UCSRC = _BV(URSEL)              // Register select.
        | _BV(UCSZ0)                    // 8-bit character size.
         | _BV(UCSZ1);

        DDRD = 0b111111111;
        ADMUX |= (1 << REFS0);          // Set ADC reference to AVCC.

        GICR = 0b11000000 |(1 << ADC);
        MCUCR = 0b00000111;

        ADCSRA |= (1<<ADIE);            // Enable ADC conversion complete interrupt.
        ADCSRA |= (1<<ADEN);            // Enable the ADC.

        sei();
        setup_analog();
        for(;1==1;){}
        return 1;
}

ISR(SIG_INTERRUPT0) {                   // The interrupts on the switch.
        if ((PIND & 0b00000100) != 0){  // Check switch status.
                transmit();
        }
}

ISR(SIG_INTERRUPT1) {                   // The interrupts on the button
        if((PIND & 0b00001000) != 0){
                PORTD  = 0b01000000;
                level = redLevel+200;
        }
                while((PIND & 0b00001000) != 0){}
                PORTD  = 0b00000000;
                _delay_ms(10);
}
```

```
ISR(SIG_ADC){                           // The interrupt on the A/D converter.
        readLight();
        setup_analog();
}

void transmit (){                       // Function which operates the transferance of data, computer-AVR Mega16.
        SREG = 0b00000000;
        PORTD &= ~(1<< 3);
        PORTD &= ~(1<< 4);
        while( !(UCSRA & ( 1<<RXC)) );
        start = UDR;
        while (! (UCSRA & (1<<UDRE))){}
        UDR = hits;
        while (switchStatus != 0){
                switchStatus = (PIND & 0b00000100);
        }
        sei();
}

void readLight(){                       // Function which reads the phototransistor.
        lowLevel = ADCL;
        highLevel = ADCH;
        highLevel <<=8;
        redLevel = lowLevel | highLevel;
        if(redLevel > level){
                hits ++;
                PORTD = 0b10000000;
                _delay_ms(500);
                PORTD = 0b00000000;
        } else {
                PORTD = 0b00000000;
                _delay_ms(10);
        }
}

void setup_analog(){
        ADCSRA |= (1 << ADEN) | (1 << ADSC);    // Set upp the analog converter and makes it ready to start convert.

}
```

Appendix B

Java-programmet

avrGUI.java – main programmet

```
package serialio;
import java.util.Enumeration;
import javax.comm.CommPortIdentifier;

public class avrGUI {
	static Enumeration<?> portList;
	static CommPortIdentifier portId;
	private static ComControl com;

	public static void main(String[] args) {
		portList = CommPortIdentifier.getPortIdentifiers();
		while (portList.hasMoreElements()) {
			portId = (CommPortIdentifier) portList.nextElement();
			if (portId.getPortType() == CommPortIdentifier.PORT_SERIAL) {
				if (portId.getName().equals("COM2")) {
					com = new ComControl(portList, portId);
				}
			}
		}
		BasicGui myApplication = new BasicGui(com);
	}
}
```

BasicGui.java – Användargränssnittet

```
package serialio;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class BasicGui extends JFrame implements ActionListener {
	private static ComControl com;
	JButton myButton = new JButton("Hämta");
	JTextArea myText = new JTextArea("För att hämta antalet träffar,  \ntryck på knappen.");
	JPanel bottomPanel = new JPanel();
	JPanel holdAll = new JPanel();
	/**
	 * Creates a simple interface to transfer the number of hits to our java application.
	 * */
	public BasicGui(ComControl com) {
		this.com = com;
		bottomPanel.setLayout(new FlowLayout());
		bottomPanel.add(myButton);
		holdAll.setLayout(new BorderLayout());
		holdAll.add(bottomPanel, BorderLayout.SOUTH);
		holdAll.add(myText, BorderLayout.CENTER);
		getContentPane().add(holdAll, BorderLayout.CENTER);
		myButton.addActionListener(this);
		setDefaultCloseOperation(EXIT_ON_CLOSE);
		setTitle("Number of hits");
		setLocation(400, 400);
		setSize(200, 100);
		setVisible(true);
	}
	/**
	 * Listen to when the button is pressed, and then get umber of hits from the AVR Mega16
	 * */
	public void actionPerformed(ActionEvent e) {
		if (e.getSource() == myButton) {
			com.readAndWrite();
			myText.setText("Antalet träffar = " + com.getHits());
		}
		else
			myText.setText("E ...?");
	}
}
```

ComControl.java – Kontakten med com-porten

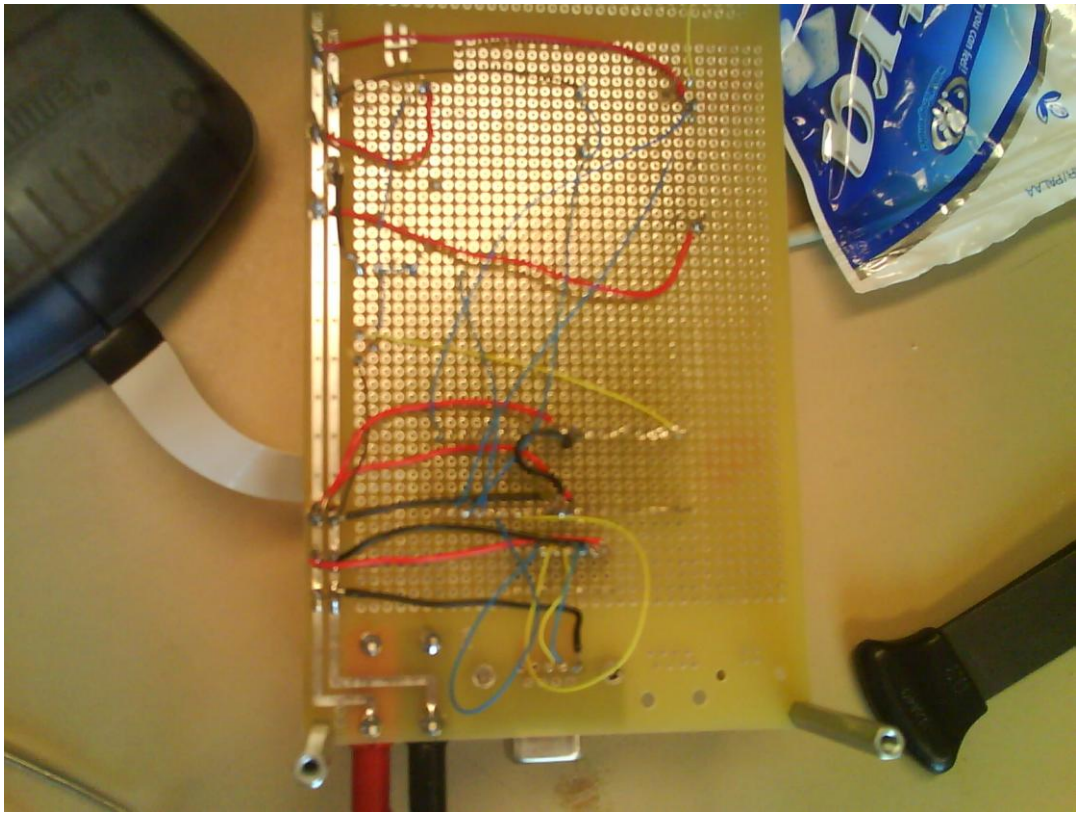
```
package serialio;
import java.io.*;
import java.util.*;
import javax.comm.*;
/**
 * A class that listen and writes on to the com-port
 * */
public class ComControl implements Runnable, SerialPortEventListener {
	static CommPortIdentifier portId;
	InputStream inputStream;
	OutputStream outputStream;
	SerialPort serialPort;
	Thread readThread;
	String previous = null;
	String[] result = null;
	private int s = 0;
	/**
	 *  The events that we listen to on the com-port are taken care of here
	 * */
	public void serialEvent(SerialPortEvent event) {
		switch(event.getEventType()) {
		case SerialPortEvent.BI:
		case SerialPortEvent.OE:
		case SerialPortEvent.FE:
		case SerialPortEvent.PE:
		case SerialPortEvent.CD:
		case SerialPortEvent.CTS:
		case SerialPortEvent.DSR:
		case SerialPortEvent.RI:
		case SerialPortEvent.OUTPUT_BUFFER_EMPTY:
			break;
		case SerialPortEvent.DATA_AVAILABLE:
			try {
				byte[] readBuffer = new byte[200];
				byte[] read = null;
				while (inputStream.available() > 0) {
					int numBytes = inputStream.read(readBuffer);
					read = new byte[numBytes];
					for (int i = 0; i < numBytes; i++) {
						read[i] = readBuffer[i];
```

```
                                }
                        }
                        char c = (char)(read[0]);
                        s = (int) c;
                } catch (IOException e) {}
                break;
        }
}
/**
 * Method used to return the number of registered hits.
 * */
public int getHits(){
        return s;
}
/**
 *  The constructor of the the class
 * */
public ComControl(Enumeration<?> enu, CommPortIdentifier portId) {
        try {
                serialPort = (SerialPort) portId.open("SimpleReadApp", 2000);
        } catch (PortInUseException e) {
                e.printStackTrace();
        }
        try {
                inputStream = serialPort.getInputStream();
                outputStream = serialPort.getOutputStream();
        } catch (IOException e) {
                e.printStackTrace();
        }
        try {
                serialPort.setSerialPortParams(9600, SerialPort.DATABITS_8,
                                SerialPort.STOPBITS_1, SerialPort.PARITY_NONE);
        } catch (UnsupportedCommOperationException e) {
        }
        try {
                serialPort.addEventListener(this);
        } catch (TooManyListenersException e) {}
        serialPort.notifyOnDataAvailable(true);
        readThread = new Thread(this);
        readThread.start();
        readAndWrite();
}
```

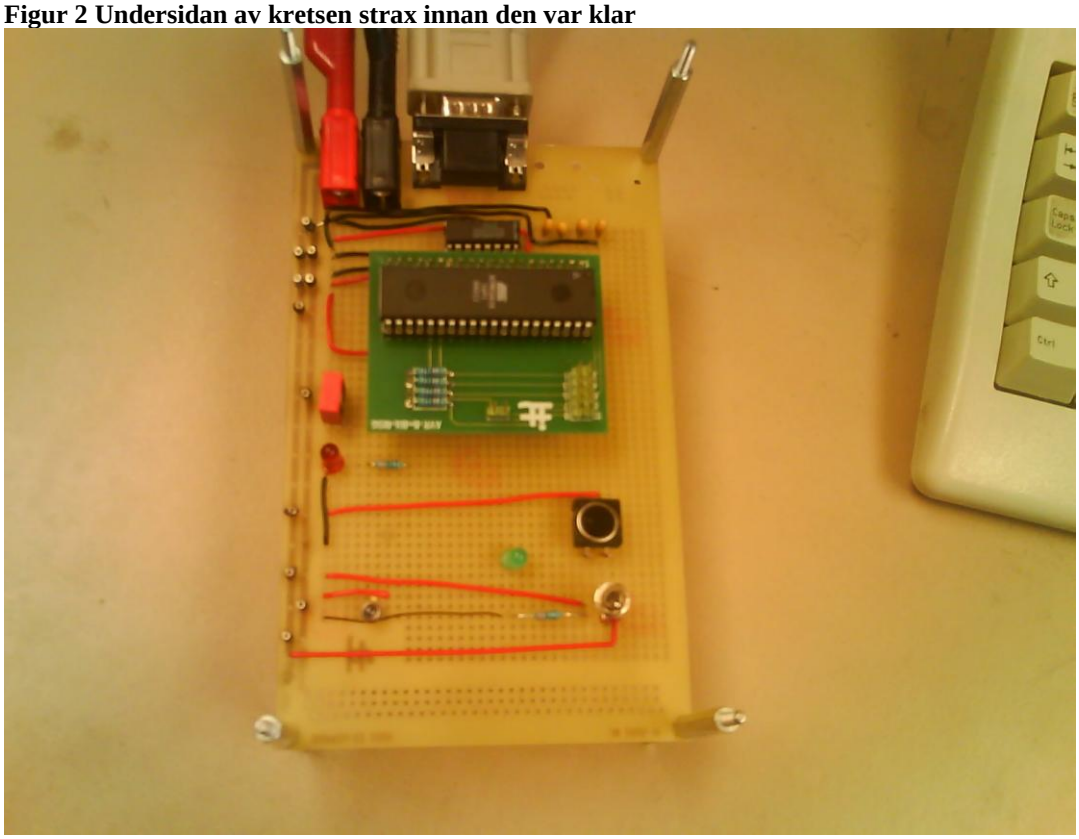
```
	/**
	 *  Starts the thread that listen for events on the com-port
	 * */
	public void run() {
		try {
			Thread.sleep(10000);
		} catch (InterruptedException e) {}
	}
	/**
	 * Wries a int to the com-port
	 * */
	public void readAndWrite() {
		int c = 's';
		try {
			outputStream.write(c);
		} catch (IOException e) {
			e.printStackTrace();
		}
	}
}
```

Appendix C

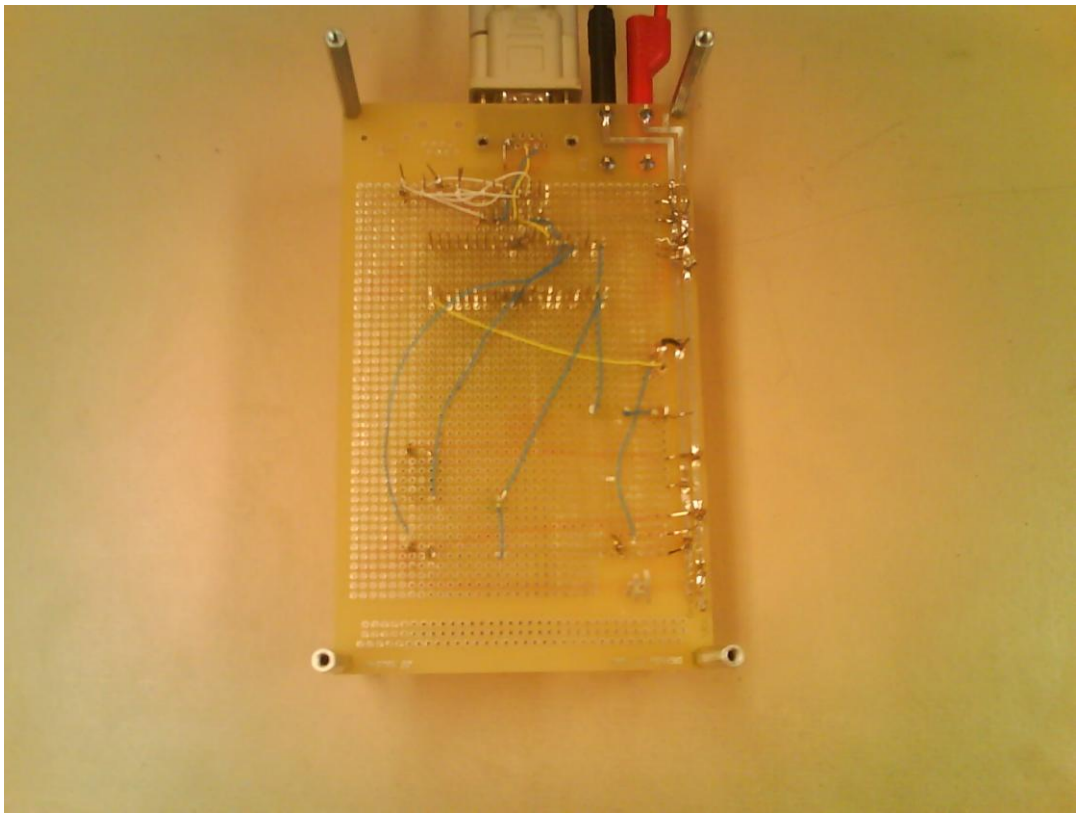
Bilder på den färdiga kretsen



Figur 2 Undersidan av kretsen strax innan den var klar



Figur 3 Den färdiga kretsen, ovansidan



Figur 4 Den färdiga kretsen, undersidan