LUNDS TEKNISKA HÖGSKOLA

Voltmeter

EITF11

2014-05-19

Grupp 3
Timmy Andersson
Jesper Tempel
Christoffer Lundgren

Project report on how to build a basic voltmeter using an AVR ATmega16.

Innehållsförteckning

1. Inledning

Följande rapport utgör en del av examinationen i kursen EITF11 ”Digitala Projekt”, vid vilken deltagarna till projektet har fått en inblick i praktiskt konstruktionsarbete. Tillsammans med en handledare har projektdeltagarna, under kursens gång, arbetat med att ta fram en prototyp av en voltmeter med stöd för enkla matematiska operationer. I denna rapport kommer den nämnda prototypen, samt projektet i sin helhet, att beskrivas och analyseras.

1.1 Bakgrund

En voltmeter används för att mäta spänning, och är idag oftast integrerad i en multimeter. Vår ursprungliga idé var att konstruera en voltmeter som kontinuerligt sparade mätvärden till ett minneskort för senare databehandling på en persondator, men detta reviderades under projektets gång. Istället kommer produkten kunna mäta 1-3 kanaler parallellt och efter en specificerad tid visa max-, min- och medelvärde av dessa, vilket är mycket användbart i många sammanhang där det momentanta mätvärdet inte ger så mycket information.

1.2 Syfte

Syftet med projektet var att få tillämpa teoretiska konstruktionskunskaper i ett praktiskt fall och därmed få en vidare förståelse för samspelet mellan hårdvara och mjukvara, samt få en inblick i hur utvecklingsprocessen för en prototyp kan se ut och vilka problem som kan uppstå.

2. Målsättning

När projektet inleddes var målsättningen att skapa en apparat som mätte och loggade spänningen på 1-4 kanaler, 0-30V. De loggade värdena skulle sparas i en textfil på ett SD-kort, vilket direkt skulle kunna läsas av en Windows/Linux dator för vidare behandling av mätvärdena i MATLAB eller motsvarande. Parametrar för mätserien, t.ex. samplingsfrekvens och aktiva kanaler, skulle finnas i en konfigurationsfil på SD-kortet och vara editerbart med godtycklig texteditor på datorn.

Ursprungliga krav för voltmetern
Kunna mäta och logga spänningen på 1-4 kanaler
Använda ett SD-kort för lagring av mätvärden, samt konfigurationsfil.
En knapp för att starta/stoppa aktuell mätserie och spara ner den på kortet
En display som visar att en mätserie pågår, anger vilka ingångar som används samt aktuella mätvärden.

Denna ursprungliga målsättning förändrades då det uppdagades att våra kunskaper för att kommunicera med SD-kortet inte var tillräckliga. I samband med detta lades möjligheten till enklare matematiska operationer till. Idén om att läsa konfiguration från SD-kortet övergavs till förmån för fler knappar och ett menysystem.

3. Produktbeskrivning

3.1 Hårdvara

Upp till tre mätsignaler kan anslutas via labbkontakterna på experimentskortets övre del. Apparaten drivs via 5V som ansluts i kortets nedre del, varpå den startar automatiskt. Med hjälp av de tre röda knapparna, vars aktuella funktionalitet visas på displayen, programmeras apparaten.

3.2 Mjukvara

Användaren blir vid start av voltmetern informerad om vilken mjukvaruversion som körs, se Figur 2. Därefter fås en fråga om användaren vill använda ”custom settings” eller ej, se Figur 3. Om användaren trycker ”No” kommer denne vidare till att välja vilka ingångar som skall vara aktiva, varpå mätningen startar. Om användaren istället väljer att använda ”custom settings” kommer frågor om samplingsfrekvens samt körtid för mätningen upp, se Figur 4 och Figur 5. Därefter specificeras antalet ingångar som skall vara aktiva, se Figur 6 innan mätningen slutligen startar. När mätningen körs visas aktuellt mätvärde för respektive ingång, samt avverkad tid i %, se Figur 7. När mätningen är slutförd visas minimum, maximum och medelvärde för respektive kanal. Genom att trycka ner samtliga knappar samtidigt återgår mjukvaran till första steget.

4. Metod

4.1 Planering

Vid genomförandet av detta projekt har projektgruppen arbetat sig igenom tre huvudsakliga faser: planering, hårdvarukonstruktion och mjukvarukonstruktion, där planeringsfasen var den första. I detta stadium togs beslut kring vilken slags prototyp som skulle tillverkas, varvid voltmeter kom upp som förslag och godkändes av gruppen. Därefter utformades en plan för vilka funktioner denne skulle innehålla och en kravspecifikation utformades. När detta var gjort utarbetades en tidsplanering där gruppen planerade hur lång tid varje moment skulle ta samt när arbetet skulle utföras.

3.1 Hårdvarukonstruktion

Efter planeringsfasen övergick projektet i konstruerandet av hårdvaran. Denna fas inleddes med att gruppen de ingående komponenternas datablad. Ett kopplingsschema upprättades som blev grunden för hur komponenterna skulle sammankopplas, se Figur 1. Därefter påbörjades själva byggandet som, tack vare all information som hade inhämtas, gick relativt smidigt.

Projektets centrala del utgjordes av en AVR ATmega16, en 8-bitars RISC enchipsdator som bygger på modifierad Harvard-arkitektur. Processorn var vid mottagandet monterad på ett riser-kort med anslutning för JTAG. På ett experimentkort monterades sedan detta riser-kort, labbkontakter för strömförsörjning, labbkontakter för mätingångarna, återfjädrande tryckknappar samt en tvåraders alfanumerisk punktmatrisdisplay. I den mån det var möjligt anslöts komponenterna genom virning, övrigt lödades. Inget externt skyddsnät byggdes med handledarens motivering att detta fanns inbyggt i processorn, vilket även databladet angav. Ansträngningarna på EMC-och avstörningsområdet sträckte sig till att trådar tvinnades, men i övrigt ansågs inget behov föreligga. Avsikten var att använda ATmegans inbyggda 10-bitars AD-omvandlare för att göra spänningsmätningen.

Avsikten var att kommunicera via SPI-buss med minneskortet. Ett separat kretskort som kunde anslutas till det stora kretskortet byggdes på vilket en SD-kortshållare monterades tillsammans med en 3.3V-regulator och spänningsdelare. Detta gjordes eftersom SD-kortet arbetar på 3,3V och inte 5V som processorn och displayen. Processorn kan tolka 3,3V som en logisk 1:a, varför det inte behövdes en step-up från 3,3V till 5V. Detta kort togs senare bort när det visade sig att vi inte kunde klara av att skriva mjukvara som kunde kommunicera med SD-kortet.

3.1 Mjukvarukonstruktion

När det väl var tid för mjukvarukonstruktionen krävdes det först en aning fördjupning i själva programmeringstekniken då projektdeltagarna inte hade så mycket erfarenheter av C med sig sedan tidigare. Parallellt med detta studerades databladen för de ingående komponenterna för att förstå hur kommunikation skall ske.

Mjukvaran i detta projekt har utvecklats i Atmel Studios 6, i en utvecklingsmiljö för ATMega16. Utöver att hantera kod innehåller programvaran även möjligheter för kompilering, debugging, transfer till mikroprocessorn samt hårdvaruprogrammering (exempelvis genom att låsa pinnars funktion eller byta XTAL). För att kommunicera med hårdvaran som tidigare diskuterats användes JTAG; som möjliggjort direkt debugging av programvaran parallellt med körning i hårdvaran. Koden har uteslutande utvecklats i C, med grund i två filer - en för att hantera displayen, och en för det faktiska programmet.

I mjukvaran har författarna försökt att optimera minnesbehov parallellt med struktur på koden, varför endast få färdiga bibliotek importeras och koden är strukturerad kring ett enkelt, tydligt huvudprogram. Biblioteken som nämns berör AVR:s färdiga paket för att kunna hantera den mest primitiva kommunikation med hårdvara (io) samt bibliotek för att hantera "polling" (vilket ungefärligt kan beskrivas som att skapa interrupts uteslutande med hjälp av mjukvara). Detta sista behövde vi göra av två anledningar; 1. Att processorns interrupts (INT0 och INT1) redan var upptagna för displayen och 2. För att

kunna skapa (/hantera) en intern klocka för tidräkning, något vår programvara krävde för att sampla under vissa frekvenser och vidare spara samplingsprov med tidsangivelser.

Koden, som finns i Appendix, utgår som nämnts från två filer. Den första, LCDHandler.c, hanterar konfiguration och användande av den alfanummeriska displayen som använts. Filen utgår från funktionsbehov att initiera skärmen, sända till skärmen (och då kolla busyflags för att undvika den dåliga vanan av delays, vilket även omöjliggör debugging), tömma skärmen, gå till nästa rad samt skriva faktiska tecken och strängar (strängar genom en vektor av karaktärer). För användarvänlighetens skull har även kod lagts till för att hantera svenska tecken (å, ä och ö). Metoderna kan sedan externt kallas från andra filer på formatet "LCD_metod", där metoder på annat format endast ämnas användas internt i filen.

Programmets huvudsakliga fil, VoltMeter.c, utgår från ett mycket enkelt huvudprogram samt en pollingmetod (mjukvaruinterrupt). Huvudprogrammet (main) initierar först pinnar som behöver nyttjas, LCD samt mjukvaruinterrupten. Därefter består programmet endast av en switch-case-sats, där ett meny-attribut (vid namn menu) bestämmer vad som sker. I flera steg tillåts användaren då göra lämpliga inställningar för att sedan starta ett sample (som senare avslutas). Interruptens funktion sker parallelt med huvudprogrammet, där menyn istället (även här via switch-case) används för att bestämma knapparnas funktion (generellt). Genom att hantera var i menyn programmet befinner sig, kan knapparna användas till olika funktioner i varje steg. Utöver de mest tydliga operationerna (som att öka ett atribut så som samplingstid), används även knapparna för konfirmation, varpå menyn stegas. Interrupten får även i samplingen ytterligare en funktion; nämligen att agera klocka. Genom att veta hur ofta avbrotten sker, kan ett attribut (internalCounter) stegas och omvandlas till en tidsangivelse. Beroende på hur ofta användaren valt att sampla använda därför denna stegning för att ange hur ofta avläsning och sparande av spänningarna skall ske.

I huvudfilen även en metod itoa, som används för att omvandla ett tal till en sträng (i C en vektor av karaktärer); vilket tillåter användning av LCD:ns printmetod för strängar.

Konfiguration av ADC-pinnar (konvertering från analog till digital signal) samt SD-kort kan även vara värt att kort beröra; där konfigurering först sker i det menysteg där användning är aktuell. På så sätt körs inte SD-kort eller pinnar när programmet inte behöver funktionen. Samma proncip gäller även för stegattributet som nyttjas för klocka, som endast hanteras under sampling.

5. Resultat

Projektet slutade inte med att samtliga krav från den ursprungliga kravspecifikationen implementerats, utan stödet för SD-kort togs bort till förmån för ett menysystem och enkla beräkningar. För ursprungliga krav och om de uppnåddes, se Tabell 1. För tillkomna funktioner, se Tabell 2.

Tabell 1: Ursprungliga krav.

Ursprungliga krav	Uppnått?	Varför inte?
Kunna mäta och logga spänningen på 1-4 kanaler	Delvis	4 kanaler var valt högst godtyckligt. Det visade sig att det enbart fanns fysisk plats för 3 ingångar.
Använda ett SD-kort för lagring av mätvärden, samt konfigurationsfil.	Nej	Gruppmedlemmarna saknade nödvändig kunskap, och processen för att inhämta denna bedömdes orealistisk för detta projekt.
En knapp för att starta/stoppa aktuell mätserie och spara ner den på kortet	Ja	
En display som visar att en mätserie pågår, anger vilka ingångar som används samt aktuella mätvärden.	Ja	

Tabell 2: Tillkomna funktioner.

Tillkomna funktioner	Varför?
Menysystem	Bättre användarvänlighet, samt det enda sätt att programmera apparaten då läsning av konfigurationsfil från SD-kort var ej möjlig.
Fler knappar	För att kunna hantera ovanstående menysystem.
Beräkning och presentaiton för min/max/medel	Då ingen möjlighet fanns för att läsa ut mätvärdena senare presenterades istället min/max/medel då mätserien var klar.

6. Diskussion

Med facit i hand skulle ett av de föreslagna projekten valts då de enbart innehöll komponenter och lösningar som var välkända i kursen. På detta vis hade vi inte behövt tveka över vad som var möjligt eller ej inom kursens ramar, och mötas av att inte kunna implementera stöd för vårt SD-kort. Likaså hade tidsåtgången lättare kunna bestämts. Från ett projektperspektiv märkte vi snabbt att det är svårt att vara flera personer som skriver kod, även om tydliga kommentarer skrivs.

7. Slutsats

Ett ur konstruktionsperspektiv relativt enkelt projekt, men samtidigt mycket lärorikt då detta är en del av ingenjörsvärlden som vi dessvärre sällan ser som I-studenter. Vi har också ”tvingats” vidga våra vyer utanför Javas bekväma och moderna värld, och lärt oss grundläggande C vilket känns mycket bra.

8. Bilagor

8.1 Kod

```
/*
 * GccVoltMeter.c
 *
 * Created: 2014-03-21 10:29:40
 *  Author: Jesper Tempel
 *    v1.00: 2014-04-09
 */

// Includes
#include <avr/io.h>
#include <avr/signal.h>

// Defines
#define XTAL                            8000000L       // Crystal frequency in Hz
#define TIMER_CLOCK       4             // Frequency for checking buttons (8 times a second)
#define ADC_VREF_TYPE   (1<<REFS0)      // 0x40. The ADMUX register description. Use Vcc as AREF.

typedef enum { false, true } bool;      // Create a boolean type

// Volatile variables
volatile char *version = "1.00";        // Version
volatile uint8_t menu = 0;              // Menu
volatile bool entered = false;          // For showing state in while
volatile uint8_t mFreq = 1;             // Frequency for measurement in times per second (max 8)
volatile uint8_t mDur = 1;              // Time for measurement in minutes (max 999)
volatile uint8_t mPorts = 1;            // Choose the number of inputs (1-3)
volatile int internalCounter = 0;       // Used as an internal clock, 8.27 ticks per second
volatile uint8_t progressCounter = 0;   // Used for progress bar
volatile int tickCounter = 0;           // Used for summarizing the ticks used for measurement
volatile long long historyOne = 0;      // Used for summarizing the voltage of port1
volatile long long historyTwo = 0;      // Used for summarizing the voltage of port2
volatile long long historyThree = 0;    // Used for summarizing the voltage of port3
volatile int minValOne = 0;             // Keeping track of minValue for port1
volatile int minValTwo = 0;             // Keeping track of minValue for port2
volatile int minValThree = 0;           // Keeping track of minValue for port3
volatile int maxValOne = 0;             // Keeping track of maxValue for port1
volatile int maxValTwo = 0;             // Keeping track of maxValue for port2
volatile int maxValThree = 0;           // Keeping track of maxValue for port3

// Methods definitions
void sample();
int read_adc(int inputNbr);
void printResultOne();
void printResultTwo();
void printResultThree();
void buttonsFunction_init();
void welcomeUser();
void askForSettingChoice();
void askForFrequency();
void askForDuration();
void askForInput();
void runMain();
void resetAll();
void resultScreen();
char* itoa();

/*
Handler for Output Compare 1 overflow interrupt
*/
SIGNAL (SIG_OUTPUT_COMPARE1A)
{
          // Read the pins for the buttons.
          uint8_t pb5 = PINA & 0x20;
          uint8_t pb6 = PINA & 0x40;
          uint8_t pb7 = PINA & 0x80;

          // Switch button-function depending on program phase
          switch(menu)
          {
```

```
case 0 :
// Entry menu
		if (pb5 == 0x00 || pb6 == 0x00 || pb7 == 0x00)
		{
			// Move further
			menu = 1;
			entered = false;
		}
		break;
case 1 :
// Settingsprompt
		if (pb5 == 0x00)
		{
			// Move past configuration
			menu = 4;
			entered = false;
		}
		if (pb6 == 0x00)
		{
			// Useless
		}
		if (pb7 == 0x00)
		{
			// Move further
			menu = 2;
			entered = false;
		}
		break;
case 2 :
// Frequencyprompt
		if (pb5 == 0x00)
		{
			mFreq = 1;
			askForFrequency();
		}
		if (pb6 == 0x00)
		{
			if (mFreq == 1) {
				mFreq = 2;
			} else if (mFreq == 2) {
				mFreq = 4;
			} else if (mFreq == 4) {
				mFreq = 8;
			} else {
				mFreq = 1;
			}

			askForFrequency();
		}
		if (pb7 == 0x00)
		{
			// Move further
			menu = 3;
			entered = false;
		}
		break;
case 3 :
// Durationprompt
		if (pb5 == 0x00)
		{
			mDur = 1;
			askForDuration();
		}
		if (pb6 == 0x00)
		{
			mDur++;
			if (mDur > 999) {
				mDur = 1;
			}
			askForDuration();
		}
		if (pb7 == 0x00)
		{
			// Move further
```

```
                        menu = 5;
                        entered = false;
                }
                break;
        case 4 :
        // Default settings - buttons should not be used
                // Useless
                break;
        case 5 :
        // Inputprompt
                if (pb5 == 0x00)
                {
                        mPorts = 1;
                        askForInput();
                }
                if (pb6 == 0x00)
                {
                        mPorts++;
                        if (mPorts > 3) {
                                mPorts = 1;
                        }
                        askForInput();
                }
                if (pb7 == 0x00)
                {
                        // Move further
                        menu = 6;
                        entered = false;
                }
                break;
        case 6 :
        // Main program
                if (pb7 == 0x00)
                {
                        menu = 7;
                        entered = false;
                }
                break;
        case 7 :
        // Sampling
                // Increase clock
                internalCounter++;

                // Sampling
                switch (mFreq)
                {
                        case 1 :
                                if (internalCounter % 8 == 0)
                                {
                                        sample();
                                }
                                break;
                        case 2 :
                                if (internalCounter % 4 == 0)
                                {
                                        sample();
                                }
                                break;
                        case 4 :
                                if (internalCounter % 2 == 0)
                                {
                                        sample();
                                }
                                break;
                        case 8 :
                                // Should tick every second!
                                sample();
                                break;
                        break;

                }

                // Check if maximumtime
                if ((internalCounter/8.27) >= (mDur * 60))
```

```
                {
                        menu = 8;
                        entered = false;
                }

                // Forcestop
                if (pb5 == 0x00 || pb6 == 0x00 || pb7 == 0x00)
                {
                        menu = 10;
                        entered = false;
                }
                break;
        case 8 :
        // Success
                if (pb7 == 0x00)
                {
                        menu = 9;
                        entered = false;
                }
                break;
        case 9 :
        // Show result
                if (pb5 == 0x00)
                {
                        printResultOne();
                }
                if (pb6 == 0x00)
                {
                        if (mPorts == 2 || mPorts == 3)
                        {
                                printResultTwo();
                        } else {
                                LCD_clear();
                                LCD_print("No 2nd input!");
                        }
                }
                if (pb7 == 0x00)
                {
                        if (mPorts == 3)
                        {
                                printResultThree();
                        } else {
                                LCD_clear();
                                LCD_print("No 3rd input!");
                        }
                }
                if (pb5 == 0x00 && pb6 == 0x00 && pb7 == 0x00)
                {
                        menu = 0;
                        entered = false;
                }
                break;
        case 10 :
        // Break
                if (pb7 == 0x00)
                {
                        menu = 0;
                        entered = false;
                }
                break;
        break;
        }
}

void sample()
{
        // One tick
        tickCounter++;

        // Should always run for first output
        int adcReading = read_adc(1);
        // (adcReading * 5000)/256 for volt
        int voltage = adcReading * (5000/256);
```

```
historyOne += voltage;
if (voltage > maxValOne)
{
		maxValOne = voltage;
}
if (voltage < minValOne) {
		minValOne = voltage;
}

// Print input1
LCD_clear();
LCD_print("Running...");
int time = internalCounter/8.27;
int percentage = (100 * time) / (mDur*60);

char percString[5];
itoa(percentage, percString);
if (percentage == 0)
{
		LCD_print("     ");
} else if (percentage < 10) {
		LCD_print("    ");
} else {
		LCD_print("   ");
}

LCD_print(percString);
LCD_print("%");
LCD_newline();

char voltageString[8];
itoa(voltage, voltageString);
LCD_print(voltageString);

if (mPorts == 2 || mPorts == 3) {
		int adcReading = read_adc(2);
		int voltage = adcReading * (5000/256);

		historyTwo += voltage;
		if (voltage > maxValTwo)
		{
			maxValTwo = voltage;
		}
		if (voltage < minValTwo) {
			minValTwo = voltage;
		}

		// Print input2
		char voltageString[8];
		itoa(voltage, voltageString);
		LCD_print(" ");
		LCD_print(voltageString);
}

if (mPorts == 3)
{
		int adcReading = read_adc(3);
		int voltage = adcReading * (5000/256);

		historyThree += voltage;
		if (voltage > maxValThree)
		{
			maxValThree = voltage;
		}
		if (voltage < minValThree) {
			minValThree = voltage;
		}

		// Print input3
		char voltageString[8];
		itoa(voltage, voltageString);
		LCD_print(" ");
		LCD_print(voltageString);
}
```

```
}

int read_adc(int inputNbr)
{
		/*
		// Sets all registers low
		ADMUX = 0x00;
		ADCSRA = 0x00;
		*/

		// We use AREF as VRef due to letting ADMUX, REFS0 be 0

		// Clear all MUX-bits
		ADMUX = 0x00;

		// Choose which value to respond with
		if (inputNbr == 1)
		{
				// Do nothing
		} else if (inputNbr == 2) {
				ADMUX |= (1<<MUX0);
		} else {
				ADMUX |= (1<<MUX1);
		}

		// Sets conversion result left justified (needed for conversion)
		ADMUX |= (1<<ADLAR);

		// Sets clock frequency
		ADCSRA |= (1<<ADPS2) |  (1<<ADPS1) |  (1<<ADPS0);

		// Sets ADC enabled. Sets Start conversion.
		ADCSRA |= (1<<ADEN) |  (1<<ADSC);

		// Variable for value
		int temp;

		// Wait for conversion to finish
		while (ADCSRA & (1<<ADSC));

		// Read upper 8bits
		temp = ADCH;

		return temp;
}

void printResultOne()
{
		LCD_clear();
		LCD_print("min/avg/max");
		LCD_newline();

		char minString[8];
		itoa(minValOne, minString);
		LCD_print(minString);

		LCD_print("/");

		int average = historyOne / tickCounter;
		char averageString[8];
		itoa(average, averageString);
		LCD_print(averageString);

		LCD_print("/");

		char maxString[8];
		itoa(maxValOne, maxString);
		LCD_print(maxString);
}

void printResultTwo()
{
		LCD_clear();
```

```
		LCD_print("min/avg/max");
		LCD_newline();

		char minString[8];
		itoa(minValTwo, minString);
		LCD_print(minString);

		LCD_print("/");

		int average = historyTwo / tickCounter;
		char averageString[8];
		itoa(average, averageString);
		LCD_print(averageString);

		LCD_print("/");

		char maxString[8];
		itoa(maxValTwo, maxString);
		LCD_print(maxString);
}

void printResultThree()
{
		LCD_clear();
		LCD_print("min/avg/max");
		LCD_newline();

		char minString[8];
		itoa(minValThree, minString);
		LCD_print(minString);

		LCD_print("/");

		int average = historyThree / tickCounter;
		char averageString[8];
		itoa(average, averageString);
		LCD_print(averageString);

		LCD_print("/");

		char maxString[8];
		itoa(maxValThree, maxString);
		LCD_print(maxString);
}

/*
Main
*/
int main(void)
{
		// Start buttons
		buttonsFunction_init();

		// Start display
		LCD_init();

		// Main program
		while(1)
		{
			if (entered == false)
			{
				switch(menu)
				{
					case 0 :
					// Entry menu

						welcomeUser();
						break;
					case 1 :
					// Settingsprompt
						askForSettingChoice();
						break;
					case 2 :
					// Frequencyprompt
```

```
                                        askForFrequency();
                                        break;
                                case 3 :
                                // Durationprompt
                                        askForDuration();
                                        break;
                                case 4 :
                                // Set default settings, Freq = 1 /s, Dur = 60s
                                        mFreq = 1;
                                        mDur = 1;
                                        // Move on
                                        menu = 5;
                                        break;
                                case 5 :
                                // Inputprompt
                                        askForInput();
                                        break;
                                case 6 :
                                // Main program
                                        runMain();
                                        break;
                                case 7 :
                                // Sampler
                                        runSample();
                                        break;
                                case 8 :
                                // Success
                                        runSuccess();
                                        break;
                                case 9 :
                                // Show result
                                        resultScreen();
                                        break;
                                case 10 :
                                // Fail
                                        runFail();
                                        break;
                                break;
                        }
                        entered = true;

                }
        }

        return(1);
}

void buttonsFunction_init()
{
        // Set buttons to 1
        PORTA |= (1<<5);
        PORTA |= (1<<6);
        PORTA |= (1<<7);

        // Use CLK/64 prescale value, clear timer/counter on compareA match
        TCCR1B = _BV(CS10) | _BV(CS11)  | _BV(WGM12);
        // Preset timer1 high/low byte
        OCR1A = ((XTAL/2/64/TIMER_CLOCK) - 1 );
        // Enable Output Compare 1 overflow interrupt
        TIMSK = _BV(OCIE1A);
        // Enable Interrupts
        sei();
}

void welcomeUser()
{
        LCD_clear();
        LCD_print("MÄTRIGG");
        LCD_newline();
        LCD_print("v. ");
        LCD_print(version);
}

void askForSettingChoice()
```

```
    {
            LCD_clear();
            LCD_print("Custom settings?");
            LCD_newline();
            LCD_print("NO");
            LCD_print("     YES");
    }

    void askForFrequency()
    {
            LCD_clear();
            LCD_print("Frequency: ");

            // Parse frequency into a string
            char frequency[3];
            itoa(mFreq, frequency);
            LCD_print(frequency);

            LCD_newline();
            LCD_print("C");
            LCD_print("   +1");
            LCD_print("  OK");
    }

    void askForDuration()
    {
            LCD_clear();
            LCD_print("Duration: ");

            // Parse time into a string
            char duration[3];
            itoa(mDur, duration);
            LCD_print(duration);

            LCD_newline();
            LCD_print("C");
            LCD_print("   +1");
            LCD_print("  OK");
    }

    void askForInput()
    {
            LCD_clear();
            LCD_print("Inputs: ");

            // Parse number of inputs into a string
            char input[2];
            itoa(mPorts, input);
            LCD_print(input);

            LCD_newline();
            LCD_print("C");
            LCD_print("   +1");
            LCD_print("  OK");
    }

    void runMain()
    {
            LCD_clear();

            // Parse number of inputs into a string
            LCD_print("Active: #");
            char input[2];
            itoa(mPorts, input);
            LCD_print(input);

            LCD_newline();
            LCD_print("       START");
    }

    void runSample()
    {
            LCD_clear();
            LCD_print("Running...");
```

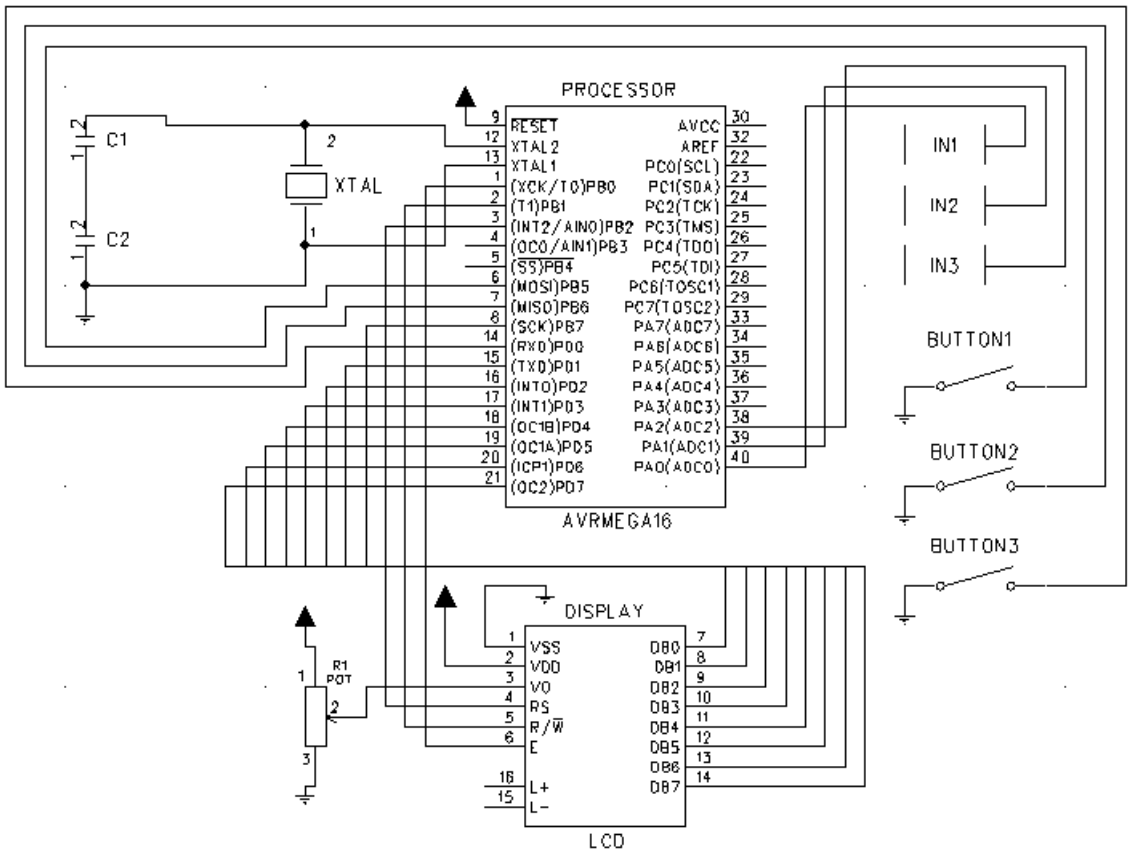
```
		LCD_newline();
		LCD_print("STOP with any");
		resetAll();
	}

	void resetAll()
	{
		internalCounter = 0;
		tickCounter = 0;

		historyOne = 0;
		historyTwo = 0;
		historyThree = 0;

		maxValOne = 0;
		maxValTwo = 0;
		maxValThree = 0;

		minValOne = 10000;
		minValTwo = 10000;
		minValThree = 10000;
	}

	void runSuccess()
	{
		LCD_clear();
		LCD_print("Sample succeeded");
		LCD_newline();
		LCD_print("      CONTINUE");
	}

	void resultScreen()
	{
		LCD_clear();
		LCD_print("ALL TO BREAK");
		LCD_newline();
		LCD_print(" 1   2   3      ");
	}

	void runFail()
	{
		LCD_clear();
		LCD_print("Sample quit");
		LCD_newline();
		LCD_print("      RESTART");
	}

	char* itoa(int i, char b[]){
		char* p = b;
		char const digit[] = "0123456789";
		if (i < 0)
		{
			*p++ = '-';
			i *= -1;
		}
		int shifter = i;
		while (shifter)
		{
			++p;
			shifter = shifter/10;
		}
		*p = '\0';
		while (i)
		{
			*--p = digit[i%10];
			i = i/10;
		}
		return b;
	}
```

```
/*
 * CDisplayHandler.c
 *
 * Created: 2014-03-21 10:29:49
 *  Author: Jesper Tempel
 *    v1.00: 2014-04-09
 */

#include <avr/io.h>

typedef enum { false, true } bool;

void ELow()
{
		// Clear E to 0
		PORTB &= ~(1<<1);
}

void EHigh()
{
		// Set E to 1
		PORTB |= (1<<1);
}

void RSLow()
{
		// RS to 0
		PORTB &= ~(1<<0);
}

void RSHigh()
{
		// RS to 1
		PORTB |= (1<<0);
}

void RWLow()
{
		// RW to 0
		PORTB &= ~(1<<2);
}

void RWHigh()
{
		// RW to 1
		PORTB |= (1<<2);
}

void send()
{
		bool RSWasHigh = false;
		if (PINB & 0x1)
		{
			RSWasHigh = true;
		}

		RSLow();
		RWHigh();			// Set read
		EHigh();			// Making sure enabled
		DDRD = 0x00;			// Set PortD to input

		while (PIND & 0x80)		// Read DB7
		{
			ELow();
			EHigh();
		}
		RWLow();			// Set write
		DDRD = 0xFF;			// Set PortD to output

		if (RSWasHigh)
		{
		RSHigh();
		}
```

```
            ELow();
            EHigh();
            return;
}

void LCD_init()
{
            // Set RS, R/W and E to output
            PINB &= ~(1<<0);
            PINB &= ~(1<<1);
            PINB &= ~(1<<2);
            DDRB |= (1<<0);
            DDRB |= (1<<1);
            DDRB |= (1<<2);

            // E to 1
            EHigh();
            RWLow();
            RSLow();

            // Set port D to output
            PIND = 0x0;
            DDRD = 0xFF;

            // Function Set
            // 0011 1000
            PORTD = 0x38;
            send();

            // Display ON/OFF
            // 0000 1110
            PORTD = 0xE;
            send();

            // Entry Mode Set
            // 0000 0110
            PORTD = 0x6;
            send();

            LCD_clear();

            return;
}

void LCD_clear()
{
            RSLow();
            // Clear screen
            // 0000 0001
            PORTD = 0x1;
            send();
            return;
}

void LCD_newline()
{
            RSLow();
            // New Line
            // 1100 0000
            PORTD = 0xC0;
            send();
            return;
}

void LCD_printChar (char ch)
{
            RSHigh();
            printChar(ch);
            send();
            return;
}

void printChar(char ch) {
            switch (ch)
```

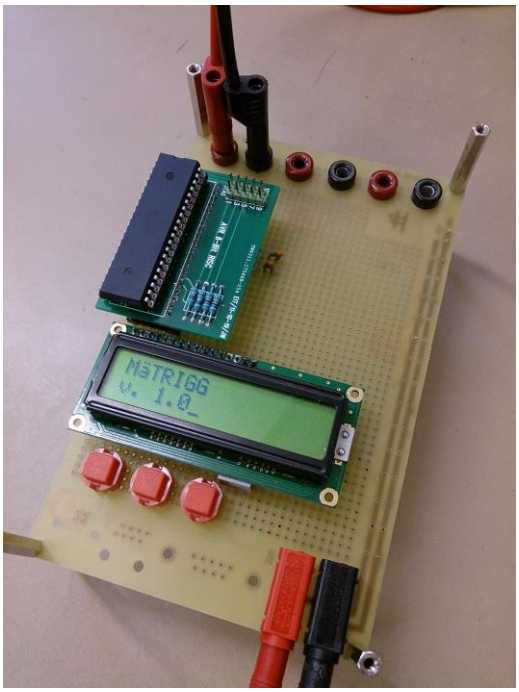
```
		{
			case 'Å' :
					PORTD = 'a';
					break;
			case 'å' :
					PORTD = 'a';
					break;
			case 'Ä' :
					PORTD = 0xE1;
					break;
			case 'ä' :
					PORTD = 0xE1;
					break;
			case 'Ö' :
					PORTD = 0xEF;
					break;
			case 'ö' :
					PORTD = 0xEF;
					break;
			default:
					PORTD = ch;
		}
}

void LCD_print (char *str)
{
		int i;
		for(i=0; str[i]!='\0'; i++)
		{
			if (i == 16)
			{
				LCD_newline();
			}
			if (i > 31)
			{
				return;
			}
			LCD_printChar(str[i]);
		}
		return;
}
```

8.2 Schema

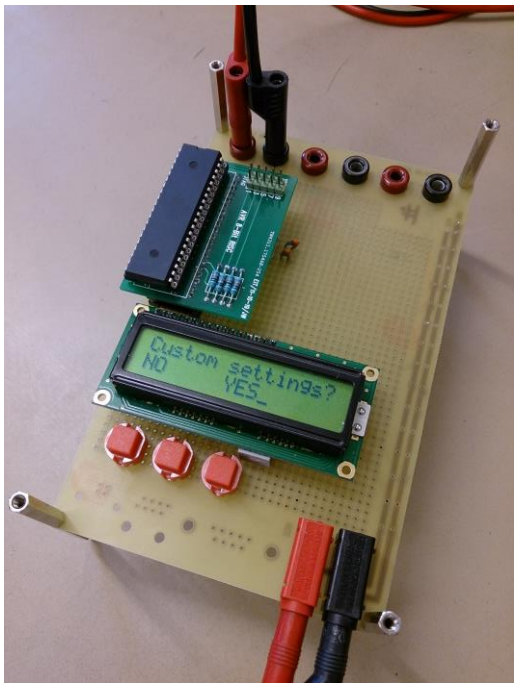


Figur 1: Kopplingschema för hela produkten.

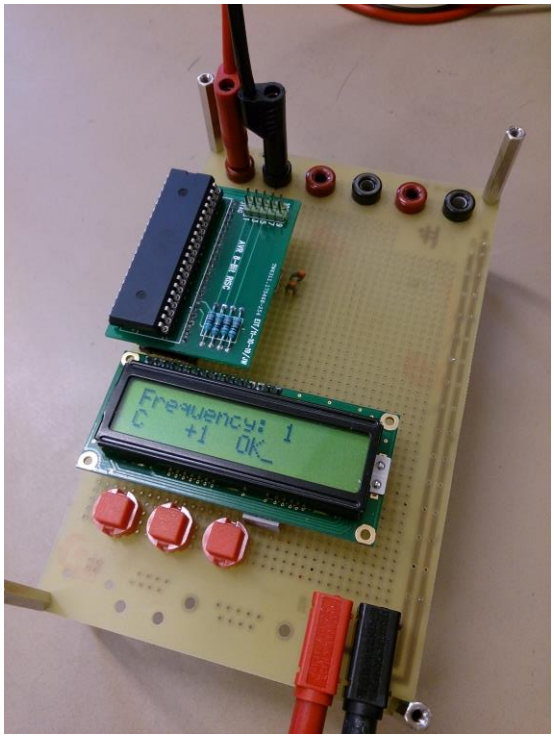
8.3 Foto



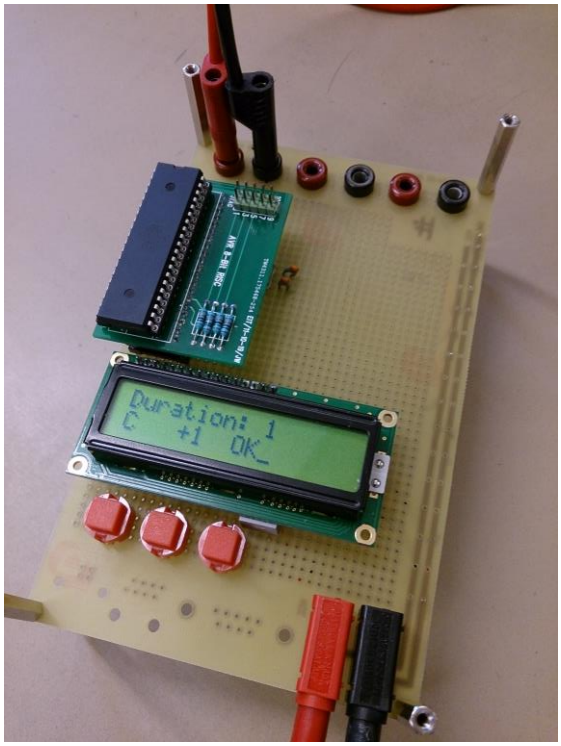
Figur 2: Uppstart. Arbetsnamnet för produkten samt mjukvaruversion skrivs ut och visas tills dess att användaren trycker på valfri knapp.



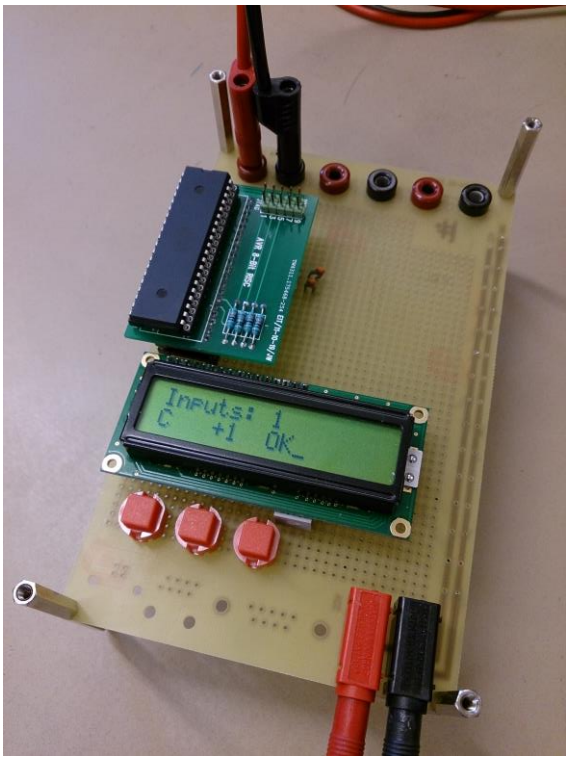
Figur 3: I det följande steget frågas användaren om denne önskar använda fördefinierade eller egna inställningar.



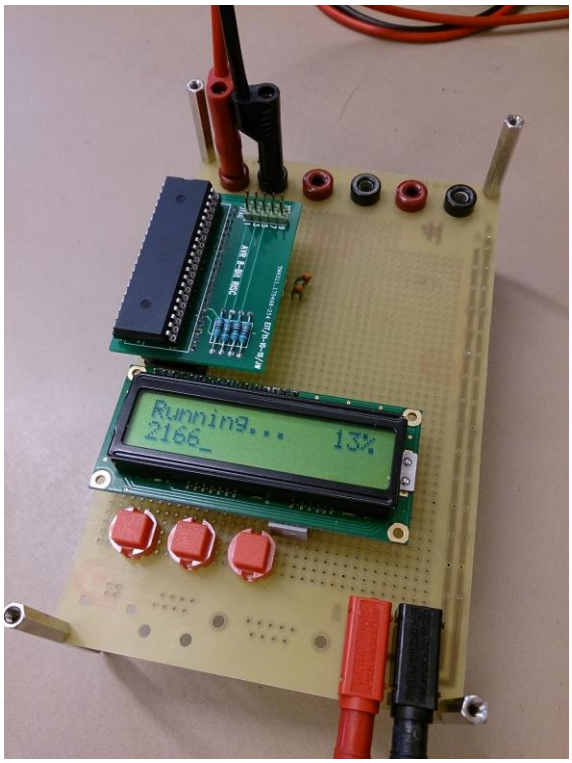
Figur 4: Samplingsfrekvensen väljs genom att stega med mittknappen, återställs till 1 vid tryck på den vänstra knappen samt bekräftas med den högra knappen.



Figur 5: Tiden som mätningen skall utföras under specificeras här i minuter enligt samma förfarande som i föregående fall.



Figur 6: Här specificeras antalet ingångar som skall användas på samma vis som i föregående fall.



Figur 7: Under pågående mätning visas aktuellt mätvärde (i mV) för respektive ingång, samt hur lång tid (i %) som avverkats.