30021 Digital Instrumenteing

Digitalt Voltmeter med PIC16F887

## Skrevet af

Kim Kofoed Nielsen s103624

Thomas Vintersborg s103632

# D. 25. februar 2013

## Introduktion

## Problem formulering

* Opbyg et digitalvoltmeter med PIC’en, analog til digital konverteren MCP 3202, spændingsreferencen MCP 1541 og LCD displayet.
* Kalibrer for reference fejl og gem kalibreringen i PIC’ens EEPROM.

## Problem begrænsning

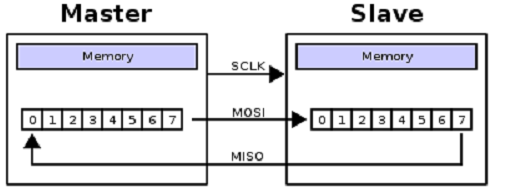
Det er ikke muligt at måle spændinger der overstiger reference spændingen.

Det er heller ikke muligt at måle AC spændninger .

## Teori

SPI BUS

Til kommunikation med ADC konverteren bruges der en SPI protekol (Serial Parallel Interface), som er en synkron dataforbindelse, der styres af en fælles klok. Kommuniaktionen fungerer ved at en SPI MASTER vælger en SPI SLAVE (ud af flere hvis de er tilgænglige) til at kommunikerer med ved brug af SLAVE select (SS) signalet. Masteren og slaven er forbundet af 2 serielle datasignaler MOSI og MISO.



I denne øvelse er MASTER’en holdets PIC16f887 mikroprocessor og slaven er A/D konverteren. Slaven aktiveres ved at MASTER’en sætter SS signalet lavt, hvor efter masteren generer 8 klok signaler. For hver klok cyklus sender MASTER’en en byte til SLAVEN bit for bit, og det samme gør SLAVEN til MASTER’en via to shift registre.

## Hardware

Microkontroller PIC16F887

Som skrevet fungerer mikrokontrolleren som MASTEREN der styrer ADC konverteren.

Relevante forbindelser:

|  |  |
| --- | --- |
| **Forbundet fra** | **Forbundet til** |
| **Mikro, ben 18: SCK** | **ADC, ben 7: CLK** |
| **Mikro, ben 24: SDO** | **ADC, ben 5: Din** |
| **Mikro, ben 23: SDI** | **ADC. Ben 6: Dout** |
| **Mikro, ben 7: CS** | **ADC, ben 1: CS/SHDN** |
| **MCP1541, Vin** | **Forsyningspænding 5 volt** |
| **MCP1541, Vout** | **ADC, ben 8: Vref** |
| **Potentiometer** | **ADC, ben 2: CH0** |

To kanals 12 bit serial Analog til Digital konverter (MCP3203)

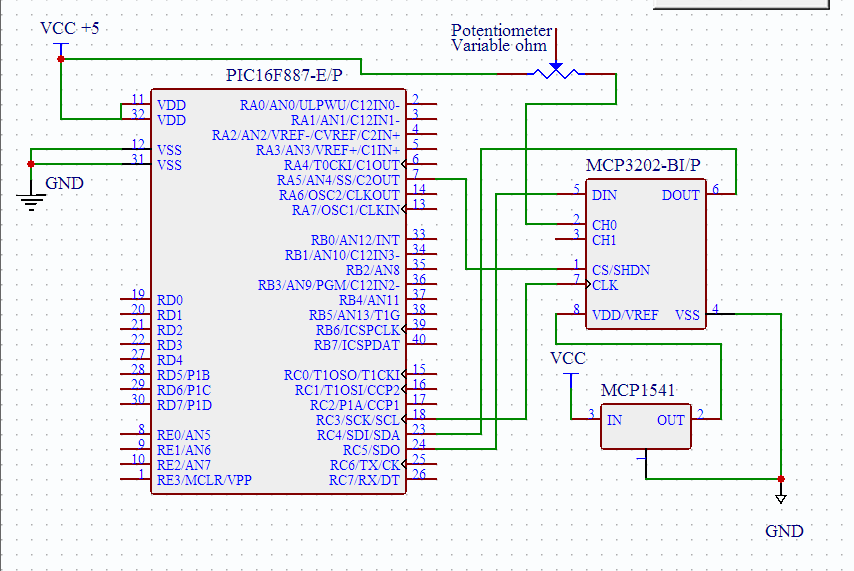
Denne analog konverter modtager en reference spænding på 4.096 volt, hvilket med dens 12bit opløsning giver 1mV pr bit.

Reference transistor MCP 1541

Reference transistoren er lavet til at forsyne 4.096 volt ud fra en forsyningspænding i området 4.3 volt til 5.5 volt. Den er bygget til at ligge meget stabilt på de 4.096 volt, hvilket er perfekt til opgaven.

Diagram

Alt hardwaren er forbundet som det ses på nedestående diagram. Forbindelserne til LCD displayet er ikke med.



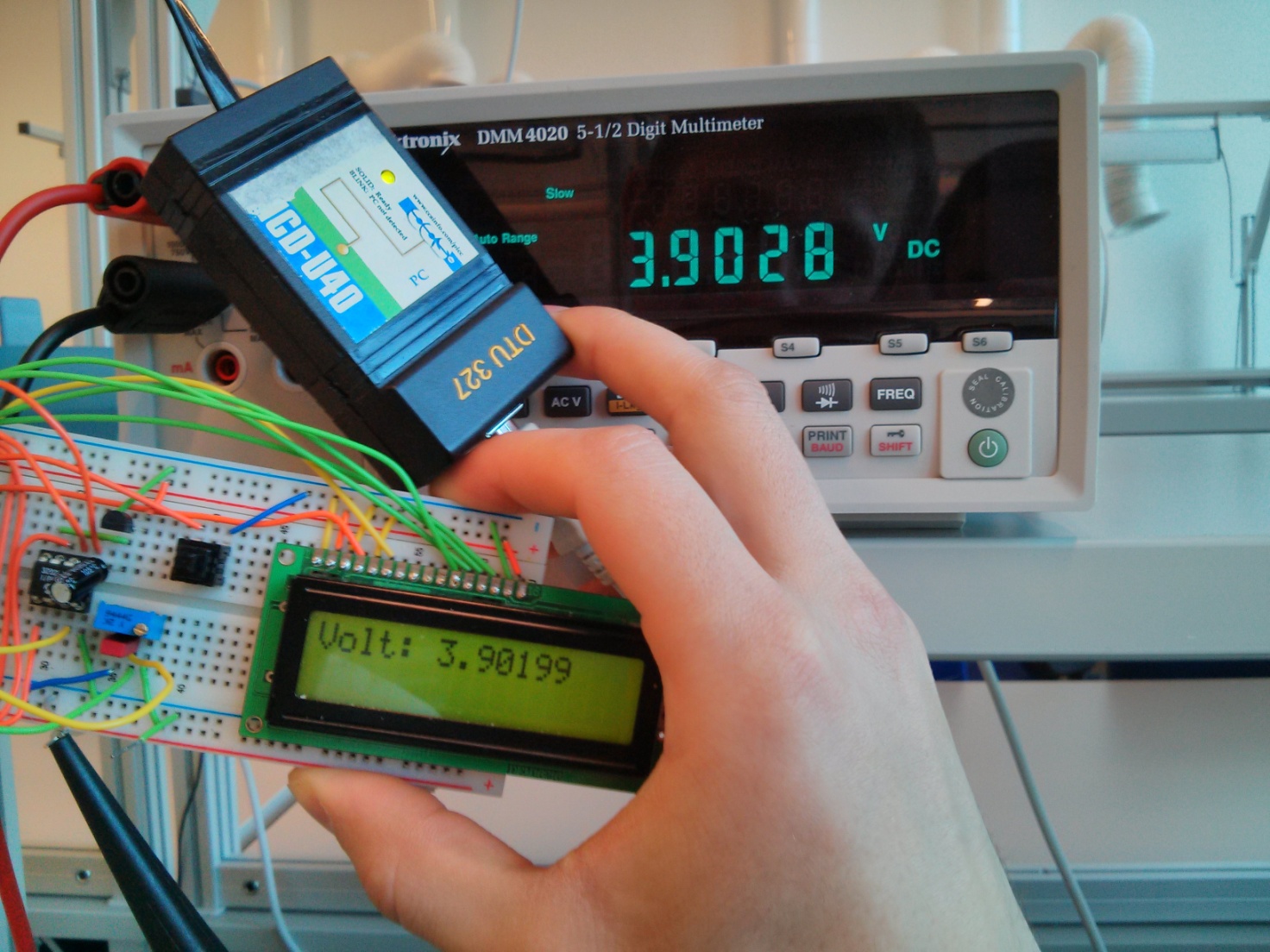
## Software

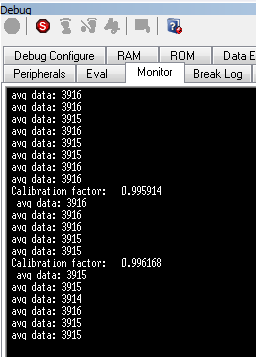
Mikrokontrolleren er programmeret således, at først skal brugeren sætte spændingen over kanal 0 på 3.9 volt for at kalibrere voltmeteret. Programmet kan nu startes og ved et knap tryk kalibrerer programmet nu voltmeteret, ved først at måle spændingen og herefter udregne en faktor, der ganges på hvis reference spændingen er forkert. Denne faktor gemmes i EEPROMMEN, så selv efter strømafbrydelse at faktoren er gemt og bruges efter genopstart. Herefter måler voltmeteret den givne spænding over enten kanal 0 eller kanal 1.

Den måde voltmeteret måler på, er ved at tage 16 målinger og adderer dem sammen og herefter lave en middel værdi. På denne måde kommer der en stabil værdi til LCD displayet. Dette mindsker nemlig støjen.

## Test of software

Softwaren er testet med screendumps af debug monitoren samt billede af LCD displayet. Multimeteret viser at spændingen er sat til 3.9 volt og at vores mikrokontroller viser det samme.





## Konklusion

Det konkluderes at vores digitale multimeter fungerer, samt at den kan kalibreres med en spænding på 3.9 volt. Det ses dog at ved det tredje decimal kommer en upræcished. Dette er grundet en lille støj fra forsyningen og komponenterne.

## Appendix

### Kilde kode

#include "ovelse51.h"

#include "lcd16216.c"

#include "internal\_eeprom.c"

#use rs232 (debugger)

#define SCK PIN\_C3 //Ben 18, forbindes til CLK (ben 7) på ADC’en

#define SDO PIN\_C5 //Ben 24, forbindes til DIN (ben 5) på ADC’en.

#define SDI PIN\_C4 //Ben 23, forbindes til DOUT (ben 6) på ADC’en

#define CS PIN\_A5 //Ben 7, forbindes til CS/SHDN (ben1) på ADC’en

//Herunder deffineres de variable der anvendes i programmet

unsigned int16 in\_data = 0;

int8 MSB\_value=0;

int8 LSB\_value=0;

int i;

char buf[16];

int int\_flag=0;

int16 tot\_data;

int16 avg\_data;

float calibrate\_fac = 0;

float calibrate\_fac\_main = 1;

int read\_flag = 0;

int calc\_flag = 0;

//Funktionerne deffineres

float read\_float\_eeprom(int16 address);

void write\_float\_eeprom(int16 address, float data);

void adc\_read();

void calibrate();

//interruptet oprettes

#int\_ext

void calibrate\_int()

{

/\*interrupts slås fra for at sikre der ikke interruptes imens interruptet er i gang

dette kunne forekomme ved bounce fra knap/kontakt\*/

disable\_interrupts(INT\_EXT);

if(int\_flag != 0) calc\_flag = 1; //if sætningen sikre at mikrokontrolleren ikke kalibrere sig selv ved startup

//calc\_flag sættes til 1 så calculate funktionen kaldes i programmet

int\_flag = 1; //sætter interrupt flaget til 1 igen

enable\_interrupts(INT\_EXT); //interrupts enables igne

}

void main()

{

setup\_adc\_ports(NO\_ANALOGS|VSS\_VDD);

setup\_adc(ADC\_OFF);

setup\_timer\_0(RTCC\_INTERNAL|RTCC\_DIV\_1);

setup\_timer\_1(T1\_DISABLED);

setup\_timer\_2(T2\_DISABLED,0,1);

setup\_comparator(NC\_NC\_NC\_NC);// This device COMP currently not supported by the PICWizard

//Setup\_Oscillator parameter not selected from Intr Oscillator Config tab

setup\_spi ( SPI\_MASTER | SPI\_L\_TO\_H | SPI\_CLK\_DIV\_16 ); //initialisering af SPI forbinnelsen

//interrupts, trigger\_egdes og LCD initialiseres

setup\_oscillator(OSC\_8MHz, 0);

port\_b\_pullups (TRUE);

enable\_interrupts(global);

enable\_interrupts(int\_ext);

ext\_int\_edge( H\_TO\_L );

int\_flag = 0;

read\_flag = 0;

lcd\_init();

lcd\_clear();

while(1)

{

delay\_ms(500);

if(calc\_flag == 1) calibrate(); //calibrate kaldes hvis interruptet har været kaldt

calc\_flag = 0; //flaget sættes lav til næste interrupt kald

/\*If sætning bestemmer om der er brug for kalibrering ved at se om eeprom'en er tom

hvis dette er tilfældet vil "read\_flag" sættes til 0 og næste if sætning vil anmode

om en kalibereing\*/

if(read\_float\_eeprom(0) != 0 && read\_float\_eeprom(0) > 0.001)

{

calibrate\_fac\_main = read\_float\_eeprom(0); //den kalibrede værdi hentes fra eeprom'en

read\_flag = 1; //flag sættes for at fortælle mikrokontrolleren er kalibreret

}

if(read\_float\_eeprom(0) == 0 )

{

read\_flag = 0;

}

lcd\_clear(); //sletter LCD displayet

lcd\_gotoxy( 1, 1); //instiller hvor der skrives på LCD displayet

//if sætning der enten anmoder om kalibereing eller tager+udskriver ADC vmålingen

if(read\_flag != 0)

{

adc\_read(); //fortager ACD måling

sprintf (buf, "Volt: %.5f ", (float)avg\_data\*calibrate\_fac\_main/1000); //ligger ACD værdien + string i buf

}

else

{

adc\_read();

sprintf(buf,"Calibrate device"); //anmoder om kalibrering af mikrokontrolleren

}

lcd\_print(buf); //udskriver "buf" til LCD displayet

delay\_ms(700); //delay så man kan se hvad der er på displayet

tot\_data = 0;

} //while slutter

} //main program slutter

/\*Denne funktion læser ADC målingen fra den eksterne ADC kreds

informationen hentes vha. en SPI forbinnelse. Der fortages 16

målinger og gennemsnittet ligges i avg\_data som anvendes til udskrivning\*/

void adc\_read()

{

for(i=0;i<=15;i++)

{

output\_low(CS); //CS sættes lav så der hentes information fra SPI forbinnelse

spi\_read(1); //De første SPI data forkastes men skal kaldes

MSB\_value = spi\_read(0XFF); //De mest betydene bit ligges is MSB\_value og der sendes FF tilbage

LSB\_value = spi\_read(0); //De mindst betydene bit ligges is LSB\_value og der sendes 0 tilbage

output\_HIGH(CS); //CS sættes høj for at stoppe SPI forbinnelsen

in\_data = ((int16)(MSB\_value & 0x0f) << 8) | LSB\_value; //de modtagede data ligges nu ned i in\_data

tot\_data += in\_data; //Alle dataen ligges ned i tot\_data

}

avg\_data = (tot\_data + 8) >> 4; //Den gennemsnitlige værdi bestemmes

printf("avg\_data: %lu\n",avg\_data); //værdien udskrives til debug monitoren

}

/\*funktionen kalibrere mikrokontrolleren ved at finde forskellen på den nuværende

inputværdi og finde en faktor der mangler for at opnå den ønskede. Denne værdi ligges

ned i eeprom'en\*/

void calibrate()

{

calibrate\_fac = (3900/(float)avg\_data)\*1; //udregner den kalibrerings faktoren

delay\_ms(50);

write\_float\_eeprom(0, calibrate\_fac); //skriver til eeprom'en

printf("Calibration factor: %10.6f\n avg\_data: %lu\n",calibrate\_fac,avg\_data); //udskriver den målte værdi og faktoren til monitoren

int\_flag = 0;

}

//denne funktion skriver til eeprom'en, skal bruge en adresse og den data der skal indlæses.

void write\_float\_eeprom(int16 address, float data)

{

int8 i;

for( i =0; i < 4; i++) write\_eeprom(address + i, \*((int8 \*)(&data)+i) );

} //Skrivning til EEPROM færdig.

//denne funktion henter data fra eeprom'en, skal bruge en adresse og retunere data'en på adressen.

float read\_float\_eeprom(int16 address)

{

int8 i;

float data;

for(i = 0; i < 4; ++i) \*((int8 \*)(&data) + i) = read\_eeprom(address + i);

return data;

}// Læsning af EEPROM stopper