

Design, Implementering og Test

Semesterprojekt 3. Semester

Gruppe 10

Vejleder: Søren Hansen

Gruppemedlemmer:

Navn	Studienummer
Tonni Nybo Follmann	201504573
Stefan Nielsen	201508282
Mikkel Espersen	201507348
Halfdan Vanderbruggen Bjerre	20091153
Ahmad Sabah	201209619
Jacob Munkholm Hansen	201404796

Indhold

Indhold	i
1 Dokumentation for brugergrænseflade	1
1.1 Indledning	1
1.2 Design og implementering	1
2 Dokumentation for SPI forbindelse imellem Devit8000 og PSoC	7
2.1 Indledning	7
2.2 Design og implementering	8
2.3 Test	13
2.4 Resultater	16
2.5 Diskussion	17
3 Motorstyring på x- og y-akser samt sensor detektering	19
3.1 Indledning	19
3.2 Design og implementering	19
3.3 Test	25
3.4 Resultater	28
3.5 Diskussion	29
4 Bipolære steppermotorer	30
4.1 Indledning	30
4.2 Teori	30
4.3 Design	32
4.4 Implementering	34
4.5 Test og resultater	36
4.6 Konklusion	36

Kapitel 1

Dokumentation for brugergrænseflade

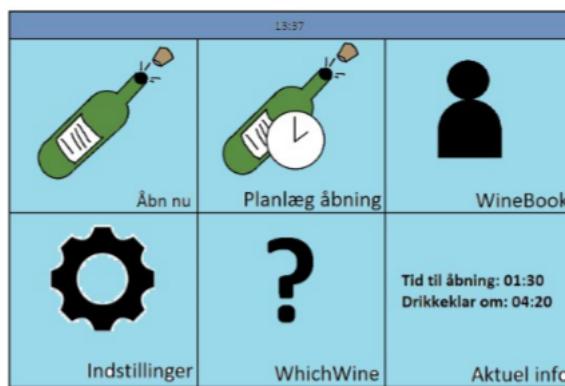
1.1 Indleding

For at en bruger kan interagere med WinePrep skal der være en brugergrænseflade. Brugergrænsefladen skal give brugeren mulighed for at kunne benytte WinePreps funktioner. Der skal være mulighed for at brugeren kan benytte sig af muligheden ”Åben nu” som åbner vinen straks efter at kommandoen er givet. Yderligere skal der være mulighed for at brugeren kan planlægge åbningen af vinen således at man kan bestemme et tidspunkt på åbning af vinen. På brugergrænsefladen skal det også være muligt at kunne tilgå WineBook, WhichtWine, Indstillinger og Aktuel info. Under Aktuel info skal der fremgå hvor lang tid der er tilbage til åbning af vinen. Der er yderligere blevet gjort nogle overvejelser over designet på brugergrænsefladen. Nogle af disse overvejelser vil blive præsenteret her.

1.2 Design og implementering

Skitsering af brugergrænseflade

Da touch-mekanismen på Devkit8000 ikke er helt optimal er det blevet besluttet at der skal benyttes store knapper således det ikke bliver svært at vælge de muligheder en bruger beslutter sig for at vælge. Den første skitse for brugergrænsefladen så således ud:



Figur 1.1: Første skitse af brugergrænsefladen

Første udgave af brugergrænseflade

På første udgave af produktet blev det dog besluttet at der ikke skal være nogen grafik på brugergrænsefladen. Yderligere blev det besluttet at det kun er funktionen Åbn nu der skal implementeres da det er denne funktion der danner grundlaget for funktionaliteten af produktet. Derfor blev Real Time Clocken (RTC) heller ikke implementeret, og derfor kunne tiden heller ikke vises, hverken klokkeslættet eller på Aktuel info. Et foreløbigt design på brugergrænsefladen ser således ud:



Figur 1.2: Første skitse af brugergrænsefladen

Her kan det ses at der ikke er blevet sat fokus på design og grafik. Når der trykkes på knappen åben nu, vil programmet sende en kommando til PSOC ved hjælp af SPI om at flasken skal registreres. Dog bruges kun kommandoerne fread og fwrite til at skrive og læse fra en character device driver. Denne kode ses her:

```

FILE *fd;
char buff[2];
fd = fopen(PSOC, "r+");
buff[0] = REGISTER_BOTTLE;
buff[1] = '\0';
fwrite(buff,1,2,fd);

```

Figur 1.3: Eksempel på brug af fwrite funktionen

For at få svar på om hvorvidt en flaske er registreret eller ej skal der læses via character device driveren. Dette gøres med funktionen fwrite(). Fwrite() ses brugt i følgende eksempel hvor der skal læses om flasken er blevet registreret:

```

buff[0]=0;
while(buff[0]!=VALID_TYPE || buff[0]!=INVALID_TYPE || buff[0]!=NO_BOTTLE)
{
    fread(buff,2 ,1, fd);
}

```

Figur 1.4: Eksempel på brug af fread funktionen

Først nulstilles bufferen som skal indeholde kommandoen, og derefter læses der i en whilelykke. Når fread() læser VALID_TYPE, INVALID_TYPE eller NO_BOTTLE vil whilelykken brydes og den læste kommando vil indsættes buff's første plads. Når værdien indsættes i buff gennemgåes en switch-case hvor der alt efter hvilken værdi der er blevet læst ind i buff reageres på forskellige måder. Et eksempel ses her:

```

switch (buff[0]) {
case VALID_TYPE:
{
    in_progress obj1;
    obj1.setModal(true);
    obj1.exec();
}
break;
}

```

Figur 1.5: Eksempel på brug af switch-case efter at have læst fra PSoC

Denne kode vil resultere i at der kommer et vindue op hvor der står ”Vinen åbnes. Vent venligst...”. Man vil få muligheden for at trykke ok for at få vinduet til at forsvinde igen. Efter dette vil vinåbningen enten lykkedes eller fejle. Devkit8000 vil derefter få besked om åbningen er lykkedes eller om den er fejlet. Der vil endnu engang benyttes en switch case til at bestemme hvilken besked brugeren får frem på brugergrænsefladen. Selve klikfunktionen på de forskellige knapper er implementeret ved at højreklikke på de forskellige knapper og trykke på goto. Herefter vil der dukke en ny menu op. Her vælges clicked(). Her vil man få mulighed for at bestemme hvilken funktion knappen skal have når den trykkes. Et eksempel ses her:

```
void MainWindow::on_pushButton_5_clicked()
{
    hideMainMenu();
    showPlanlaegMenu();
}
```

Figur 1.6: Implementering af trykknappen Planlæg åbning

Dette er implementeringen for planlæg åbning knappen. Det der sker når man trykker på knappen er, som det fremgår på kodeeksemplet, at hovedmenuen skjules, og at Planlægningsmenuen vises. Måden hvorpå menuskiftsfunktionen er blevet implementeret er ved brug af show og hide funktionerne i QT. Dette kan ses i følgende eksempel hvor tilbageknappen skjuler alle knapper på den nuværende menu, for derefter at vise alle knapper på hovedmenuen:

```
void MainWindow::on_pushButton_2_clicked()
{
    ui->label->hide();
    ui->pushButton->hide();
    ui->pushButton_3->show();
    ui->pushButton_4->show();
    ui->pushButton_5->show();
    ui->pushButton_6->show();
    ui->pushButton_7->show();
    ui->pushButton_8->show();
    ui->pushButton_2->hide();
    ui->comboBox->hide();
    ui->comboBox_2->hide();

    |
}
```

Figur 1.7: Skift af menu med tryk på tilbageknappen

Test

Som tidligere beskrevet er der i første udgave kun implementeret Åbn nu funktionen. Derfor er det kun denne funktion der kan testes. Dog kan navigationen på de forskellige menuer på brugergrænsefladen også testes. Menuerne Wine-Book, WhichWine, Indstillinger og Aktuel info er ikke blevet implementeret i denne udgave, og derfor vil test af disse menuer ikke fremgå i denne test.

Read og write funktionerne testes ved at tilkoble en PSoC til Devkit8000 og forsøge at skrive og læse til og fra den. Der er blevet konstrueret nogle meget simple forsøg, hvor funktionerne fread og fwrite bruges. Forsøget er konstrueret således at en led på PSoC'en lyser hver gang der læses fra den. Yderligere vil det der læses fra PSoC'en skrives til en fil på devkitted. Forsøget hvor der læses fra Devkit8000 er vedhæftet som en fil i bilaget, hvor man kan se at led'en på PSoC'en tænder og slukker hver gang der trykkes på Åbn nu på brugergrænsefladen. Koden der er brugt til at teste forbindelsen med er følgende:

```

FILE *fd;
int buff[2];
fd = fopen(PSoC, "r");
fread(buff, 2, 1, fd);

FILE *fd1;
fd1=fopen("/home/root/test2.txt", "w");

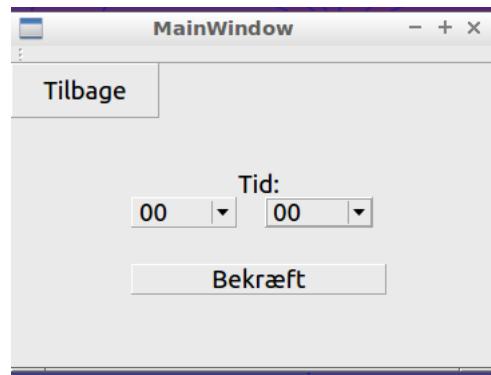
fwrite(buff, 1, 2, fd1);
fclose(fd1);

```

Figur 1.8: Kode til test af forbindelse

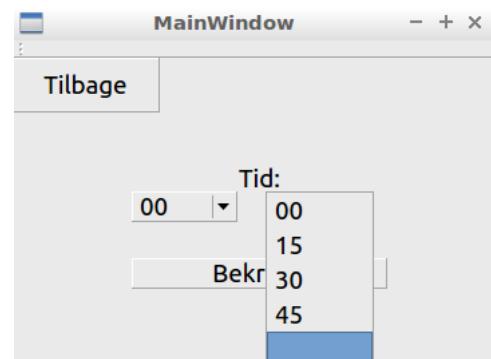
Denne kode tester blot fread og fwrite funktionerne. PSoC'en er programmeret således at dens LED lyser hver gang der bliver læst fra den. Yderligere gemmer koden det den læser i en tekstfil på devkitted.

Når man trykker på knappen Planlæg åbning dukker følgende menu op:



Figur 1.9: Menu til planlæg åbning med scrolldown menu åben for timer

For minutter ser det således ud:



Figur 1.10: Menu til planlæg åbning med scrolldown menu åben for timer

Resultater

Da hele produktet ikke er fuldt udviklet er det ikke muligt at fremvise nogle resultater. Man kan se på videoen at vi kommer til at tænde og slukke for dioden ved hjælp af brugergrænsefladen på devkittet. Videoen kan ses i bilaget.

Konklusion

Alle test gik som forventet, da dioden på PSoC'en kom til at tænde ved læsning som forventet. Yderligere er designet for brugergrænsefladen, og navigationen i den blevet som forventet. Dog blev designet ikke implementeret, da dette vil gøres i en senere itteration.

Kapitel 2

Dokumentation for SPI forbindelse imellem Devit8000 og PSoC

2.1 Indledning

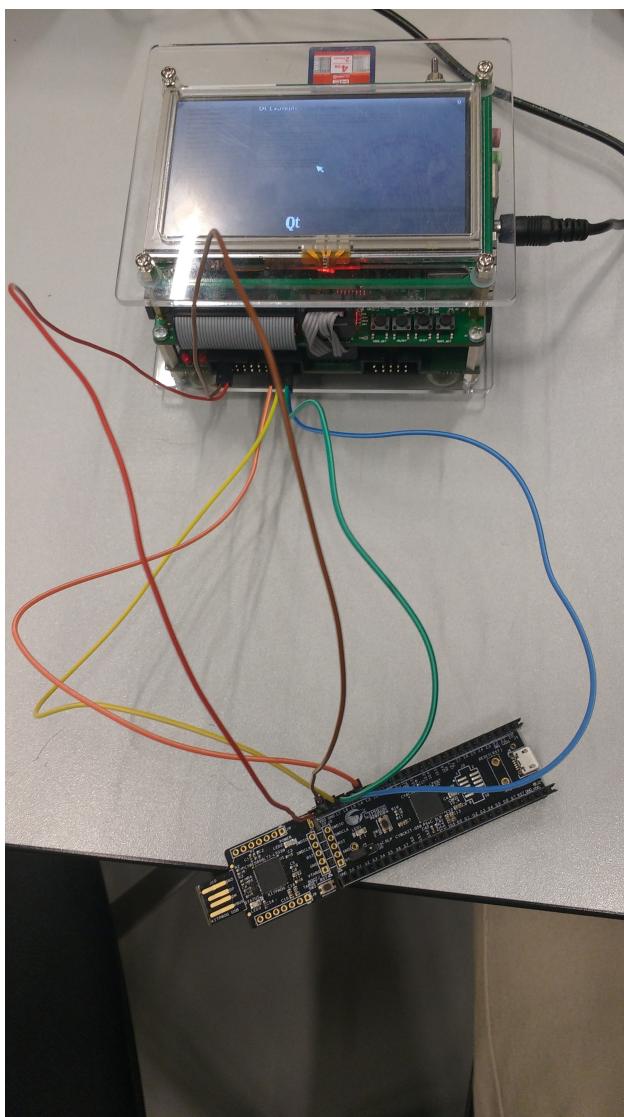
I wineprep projektet skal der bruges en linux platform, som i dette projekt består af et DevKit8000 (DK8k) hvorpå der er installeret distributionen Ångstöm. Denne skal tage imod input fra brugeren af systemet via en grafisk brugergrænseflade og omskrive disse til kommandoer, som sendes til PSoC. Desuden vil PSoC sende status beskeder tilbage til DK8k. Der vil i projektet indgå en PSoC, som står for kommunikationen med DK8k samt for nogle af systemets motorer og sensorer, men på grund af et begrænset antal GPIO-pinde og UDB'er på PSoC, vil det blive nødvendigt med to ekstra PSoCs, som kan tage sig af kommunikationen med de øvrige motorer og sensorer i systemet. Der vil derfor blive lavet en forbindelse mellem de tre PSoC-enheder, hvor PSoC'en, som tager imod kommandoer fra DK8k, vil fungere som master over de to øvrige PSoCs. Der vil i denne dokumentation blive refereret til disse PSoC-enheder som MASTER (forbindes til Devkit), TOP (slave til MASTER, som sidder i toppen af Wineprep) og BOTTOM (slave til MASTER, som sidder i bunden af Wineprep). Ovenstående taget i betragtning vil det blive nødvendigt at etablere en tovejsforbindelse imellem DK8k og MASTER, samt en tovejsforbindelse imellem MASTER og TOP, samt MASTER og BOTTOM.

Dette kan gøres på flere forskellige måder, f.eks. med en UART-protokol som på tidligere semester projekter. Der er i fagene HAL og GFV på 3. semester blevet arbejdet en del med de to serielle dataforbindelsesstandarder I2C og SPI, og det vil derfor være nærliggende at benytte en af disse to. Det er fra gruppens side og med opfordring fra vejleder blevet besluttet, at SPI vil blive benyttet i dette projekt til forbindelsen imellem DK8k og PSoC. Grunden til dette valg

er mest af alt bekvemmelighed, da der allerede er arbejdet med SPI drivere til Linux og PSoC i tidligere laboratorieøvelser.

2.2 Design og implementering

SPI er en synkron dataopførselsmetode, hvor to enheder indgår i et master/slave-forhold, og hvor data sendes og modtages mellem de to enheder på samme tid med klokken som "taktstok". For mere information omkring SPI-protokollen henvises til bilag(*reference til SPI-protokollen). DK8k vil i dette tilfælde udføre rollen som master i forbindelsen til MASTER, hvilket også betyder, at det er denne, som står for at starte dataoverførslen mellem de to enheder. Ligeledes vil MASTER fungere som master i forbindelsen til TOP og BOTTOM.



Figur 2.1: Realisering af Devkit8000-PSoC SPI forbindelse

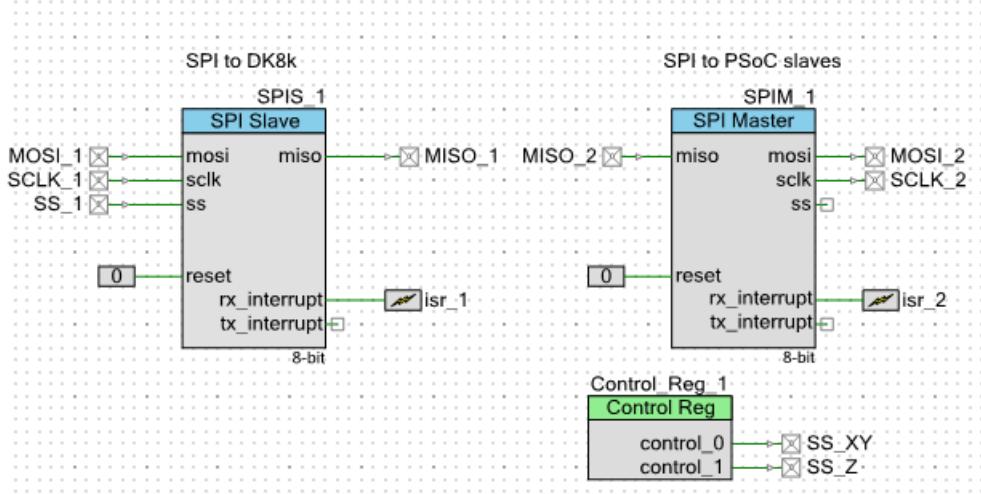
SPI undersøttes naturligvis af DK8k, men for at kunne etablere forbindelsen til MASTER skal den nødvendige SPI-driver skrives til Linux. Denne driver indeholder blandt andet den korrekte opsætning for forbindelsen samt implementeringen af SPI-metoderne til at sende/modtage data. Der skal også skrives en character device driver, der gør det muligt for et program i userspace at tilgå SPI driveren. Der laves også en hotplug-driver, der gør det muligt at indsætte PSoC i runtime. For mere information omkring implementeringen af denne driver samt eksempel på dens anvendelse henvises der til HAL øvelse 6 i bilag. Vi har i dette projekt anvendt den SPI-driver, som er blevet udleveret på redmine i faget HAL. Så SPI-driver-modulerne indsættes blot på DK8k, og

der laves de nødvendige device-noder. Disse noder er nødvendige for at kunne kommunikere mellem user- og kernelspace.

```
root@devkit8000:~# cat mk_psoc.sh
insmod hotplug_psoc_spi_device.ko
insmod psoc_mod.ko
mknod /dev/psoc_1 c 66 1
mknod /dev/psoc_2 c 66 2
mknod /dev/psoc_3 c 66 3
mknod /dev/psoc_4 c 66 4
mknod /dev/psoc_5 c 66 5
echo 0x3 > /sys/class/cplddrv/cpld/spi_route_reg
echo 0x1 > /sys/class/cplddrv/cpld/ext_serial_if_route_reg
```

Figur 2.2: Indsættelse af moduler i Linux kernen og oprettelse af device noder

De tre PSoC-enheder skal ligesom DK8k også have implementeret software, der kan håndtere SPI-forbindelsen og reagere på den ønskede måde, når der modtages data. Denne software skrives med værktøjet PSoC Creator, som via et simpelt drag-and-drop interface gør det nemt at konfigurere SPI. Selve SPI-driveren skal derfor ikke skrives, men der skal laves en source-fil, hvori det er muligt at kalde metoder til styring af SPI, håndtere Rx-interrupts og anvende de modtagne data til at udføre forskellige opgaver for systemet.



Figur 2.3: Topdesign for PSoC1

MASTER skal have både et SPI-master- (SPIM) og et SPI-slave-modul (SPIS) i PSoC Creator. Der ønskes et interrupt ved Rx på SPIS, da slaven skal reagere hver gang der modtages data fra master, hvorefter der skal skrives tilbage og udføres en opgave. Dette vil blive håndteret i en interruptroutine af en

switch implementation, der kaldes hver gang Rx-interruptflaget går højt. SPI-driveren på DK8k er opsat således, at der sendes 16 bits til MASTER. Af disse er de første 8 bit tiltænkt adresse- og kommandobits, og de næste 8 bit er selve data. Der vil i main først blive switched på adresse-bits, som fortæller hvilket device, der benyttes. Da der kun er én PSoC som slave til DK8k, vil et forsøg på at skrive til et andet device, end det, der er blevet afsat til MASTER, returnere en fejlmeddeelse. Kommandobits indikerer, om der ønskes status læst tilbage fra MASTER, eller om det blot er tilstækkeligt, at MASTER behandler den modtagne kommando og sender nogle default-værdier tilbage. Mere specifikt vil MASTER fungere som formidler af kommandoer mellem DK8k og TOP og BOTTOM. MASTER vil ikke indeholde nogen anden særegen funktionalitet, da den ikke har direkte adgang til motorer, sensorer, osv. Disse eksterne komponenter håndteres af TOP og BOTTOM på baggrund af de kommandoer, der sendes fra DK8k til MASTER. Inde i den føromtalte switch vil MASTER behandle disse kommandoer ved at påføre TOP eller BOTTOM en pågældende opgave vha. metoder fra SPIM. SPIM's Rx-interruptflag går ligeledes højt, når TOP/BOTTOM rapporterer deres status tilbage. Da vil der i en interruptroutine også være switch, som behandler disse status-meddelelse. MASTER's rolle kunne passende beskrives ved et tilstandsdiagram (*referer til applikationsmodel, når denne engang er opdateret).

```

switch (addr) {
    case 0x1:
        switch(cmd) {

            // Write status of bottle to DK8k
            case GET_STATUS:
                updateStatus(status);

                if(status != IN_PROCESS || status != VALID_TYPE) {
                    status = DEFAULT;
                }
                break;

            // Registerate bottle
            case LOCATE_XY:
                updateStatus(IN_PROCESS);

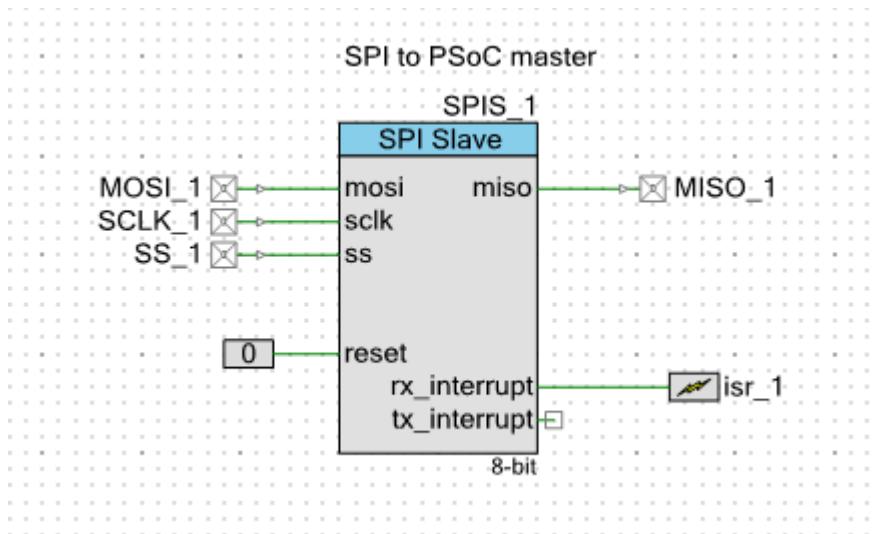
                // Validate thickness of bottle neck
                writeToXY(cmd);
                break;

            // Open bottle
            case OPEN_BOTTLE:
                updateStatus(IN_PROCESS);
        }
}

```

Figur 2.4: Eksempel på en switch statement i PSoC koden

Topdesign og koden for TOP og BOTTOM vil blive implementeret næsten på samme måde som MASTER mht. kommunikationen med denne, hvor der implementeres et Rx-interrupt og en switch som behandler den modtagne data. De to slave-PSoC-enheder vil dog kun have et SPIS modul implementeret, og derfor have færre switch-cases. Der bruges en 8bit kommando til at sende data til TOP og BOTTOM, og det er denne kommando, som bliver switched på. Når en opgave er udført, eller der er opstået en fejl, opdateres Tx-bufferen med en relevant status-meddeelse. Dette sker løbende i udørelsen af en pågældende opgave.



Figur 2.5: Topdesing for PSoC2 og PSoC3

```

CY_ISR(isr_1)
{
    uint8 cmd;

    cmd = SPIS_1_ReadRxData(); // This also clears Rx Status Register

    switch(cmd) {

        // Write status of bottle to master-PSoC
        case GET_STATUS:
            updateStatus(status);
            break;

        // Position x-sensor to just under top of bottle
        case LOCATE_Z_1:
            enabled = POSITION_Z_1;
            break;

        // Position x-sensor to just over top of bottle
        case LOCATE_Z_2:
            enabled = POSITION_Z_2;
            break;
    }
}

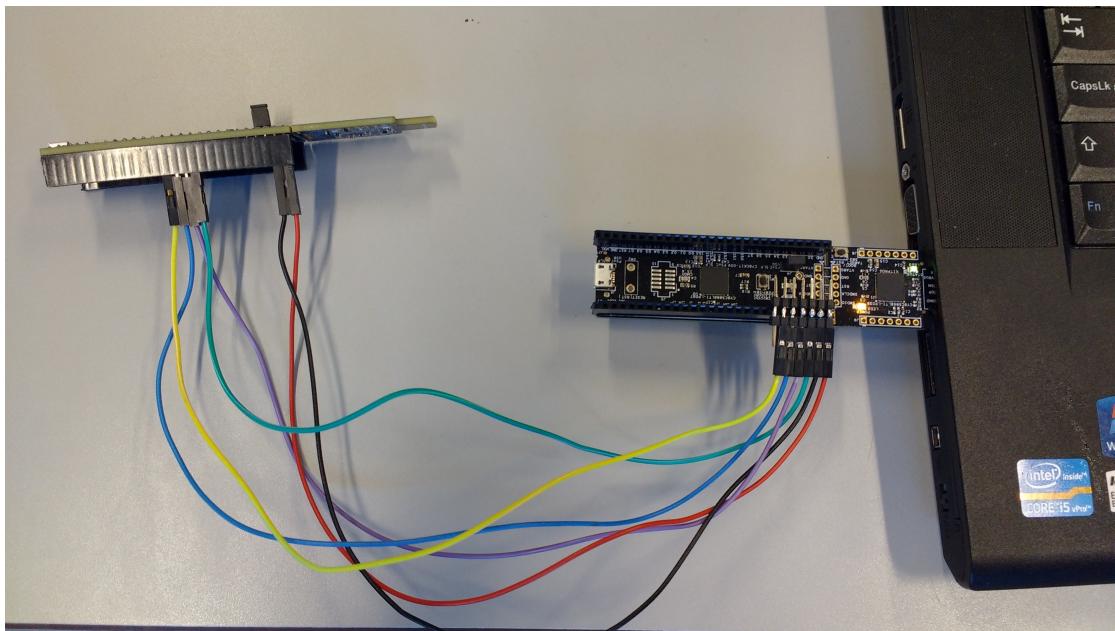
```

Figur 2.6: Eksempel på switch cases for PSoC2 og PSoC3

2.3 Test

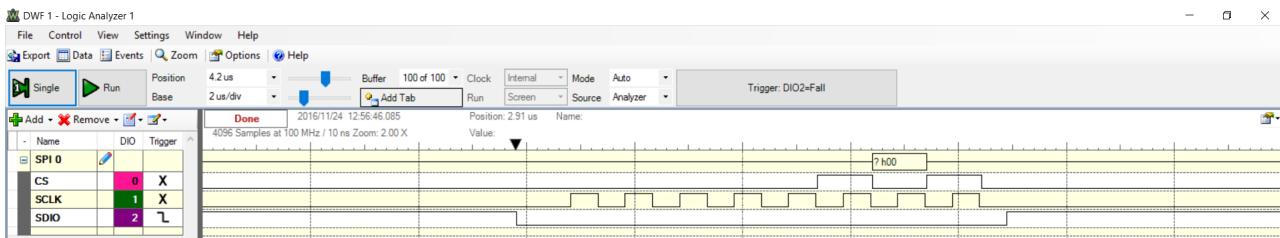
Test af SPI forbindelse mellem MASTER og PSoC-slave

Testen for forbindelsen mellem MASTER og PSoC-slave gennemføres vha. en terminal og logic analyzer på Analog Discovery. De to PSoC-enheder bliver sat til hver deres PC, og der laves en UART-forbindelse, så det kan ses på et terminalvindue, hvilke data, der bliver sendt og modtaget. Derefter forbindes de to PSoC- enheder til hinanden med SPI, og der tilføjes desuden en fælles stel- forbindelse. Hvor MASTER normalvist modtager sine kommandoer fra DK8k, vil den i denne test modtage disse fra brugeren via terminalvinduet på den ene PC. PSoC-slave er blot en test-dummy, som modtager denne kommando og udskriver den på terminalvinduet på den anden PC, og skriver den i Tx- bufferen, så MASTER kan læse den igen og udskrive den på sit terminalvindue. Derved testes tovejskommunikationen mellem PSoC'ene. På billedet ses også en VDD-forbindelse fra MASTER, denne er dog kun nødvendig, hvis slaven ikke får VDD fra PC.



Figur 2.7: PSoC forbundet til PC via UART og med anden PSoC-enhed via SPI

Desuden sættes Analog Discovery på SPI-forbindelsen, og der måles på CLCK, SS (slave select) og MOSI for at sikre, at SPI-signalet ser fornuftigt ud. Altså at SS går lav ved dataoverførsel, og den korrekte bitsekvens sendes på MOSI. Grunden til, at der ses efter, om SS går lav, er fordi, der ved SPI kan være flere slaver tilsluttet. SPI fungerer på den måde, at den trækker signalet lavt på den slave, som skal modtage signalet. CLCK er ikke særligt interessant, men det kan evt. tjekkes om frekvensen stemmer overens med det, som angives i koden for PSoC.

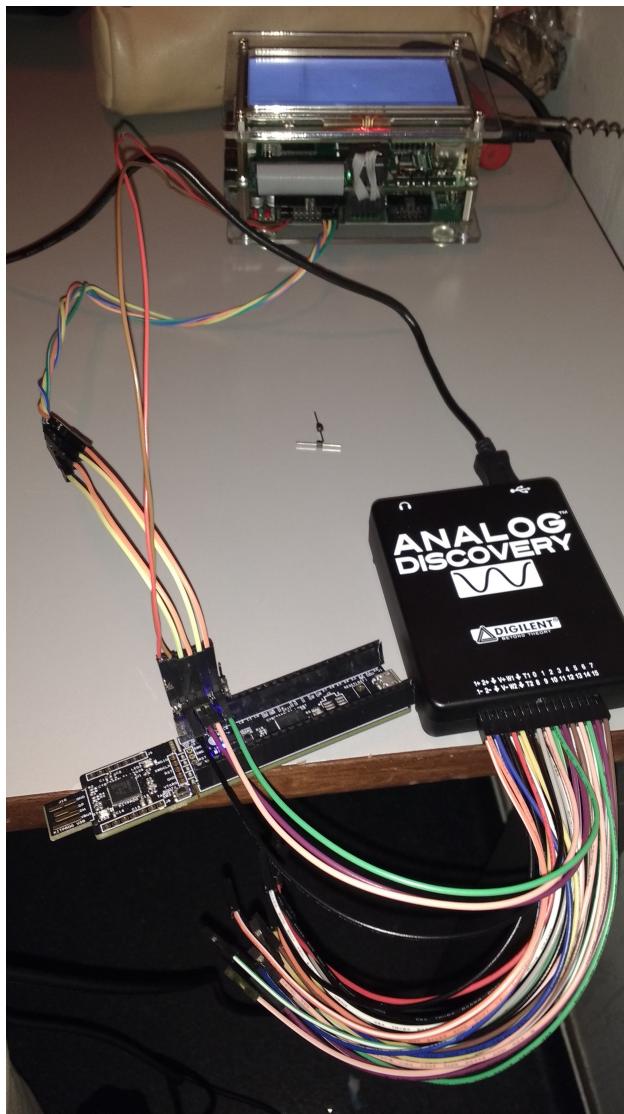


Figur 2.8: Test med Analog Discovery på SPI-forbindelsen

Der sendes data fra MASTER til slave, og der kigges på terminalen og logic analyzer for at se hvilke data, slave modtager. I denne test sendes kommandoen 0x05 til slaven at antal gange.

Test af SPI-forbindelse imellem DK8k og MASTER

Testen for SPI-forbindelsen imellem DK8k og MASTER foretages med Analog Discovery, hvor der måles på SS, CLCK, og henholdsvis MISO og MOSI.



Figur 2.9: Opstilling med Analog Discovery til test af SPI-forbindelse

Der udføres to test-scenerier. I første test sendes data fra DK8k (master) og der foretages en måling på MOSI forbindelsen med logic analyzer-funktionen på Analog Discovery. Denne test skal sikre, at vi får sendt de korrekte adresse- og data-bits over til MASTER, samt at SS og CLCK opfører sig som ønsket. Det vil sige at SS går lav ved dataoverførsel og at CLCK er stabil. I denne test

bruges Linux-terminalen på DK8k til at sende nogle forskellige værdier til MASTER med Linux-kommandoen echo, hvorefter der aflæses bit-kombinationer på logic analyzer.

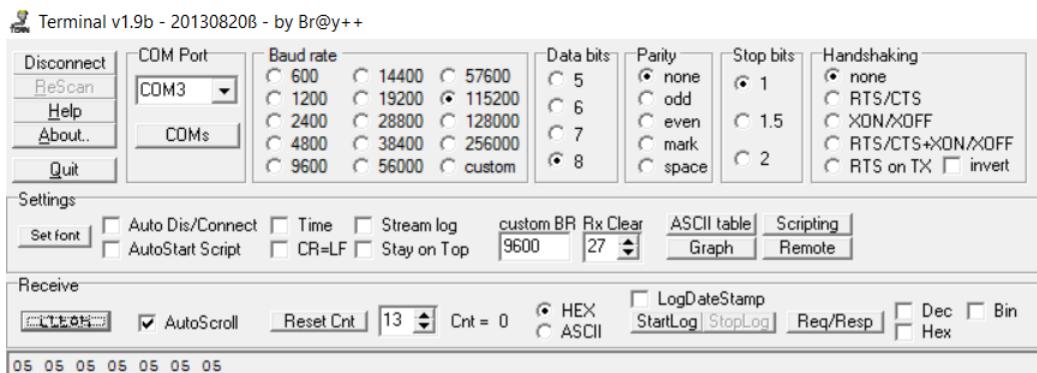
I den anden test læses der fortsat på MOSI forbindelsen, og Linux-kommandoen cat bruges til at læse fra MASTER. Her aflæses det på terminalen, hvad der bliver sendt fra MASTER. I test-programmet på MASTER er der implementeret en switch, som gør, at når der læses med kommandoen cat PSoC_5 fra Linux-terminalen, vil status på knap 2.1 på MASTER blive aflæst. Således kan det testes, at der sendes den korrekte data fra MISO.

2.4 Resultater

(

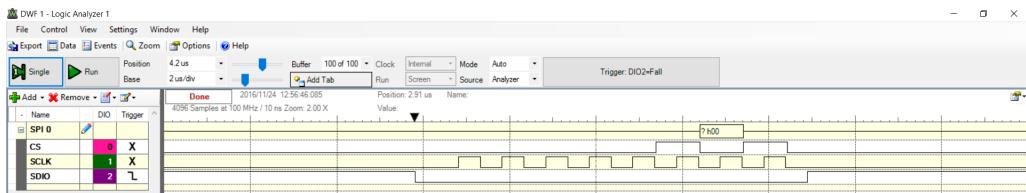
Resultater af test for PSoC-PSoC SPI)

Ved testen ses, at der udskrives 05 på terminalvindue for PSoC-slave. Dette passer fint med den kommando, der sendes fra MASTER. SPI-forbindelsen fungerer altså fint.



Figur 2.10: Terminal udskrift fra PSoC-slave-enheden

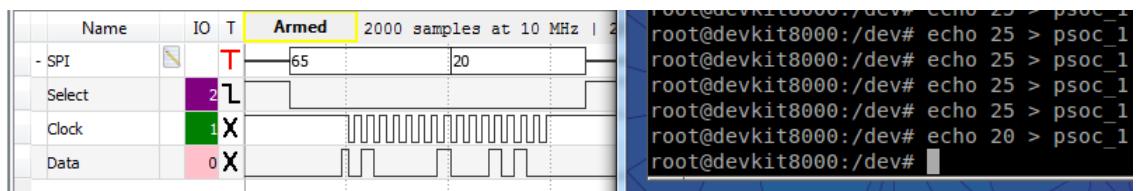
Der kigges også på logic analyzer, og det ses, at SS går lav ved dataoverførelse. Og der sendes desuden værdien 0x05 på MOSI. Ingen virker det som forventet. Det eneste problem ved denne test er, at MASTER først kan læse den sendte kommando efter at have sendt denne en tre-fem gange. Dette kunne give anledning til problemer i den endelige implementering, da MASTER skal have klar besked om PSoC'enes status, da disses opgaver skal synkroniseres. En evt. "løsning" på dette problem kunne være at indføre en "guard", f.eks en counter, som tjekker, at en kommando/meddelelse er blevet sendt/læst et vist antal gange, før der udføres yderligere opgaver.



Figur 2.11: logic analyzer målt på SPI-forbindelsen

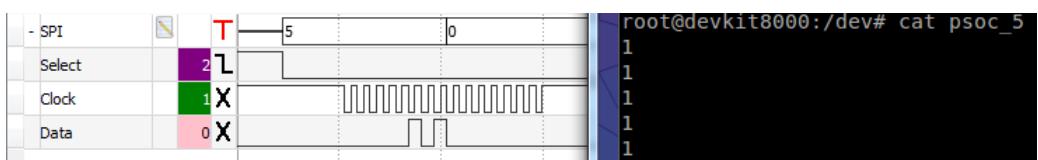
Resultater af test for Devkit-PSoC SPI

Ved første test-scenarie ses, hvordan der sendes værdien 20 fra DK8k til MASTER med Linux-kommandoen echo. På logic analyzer ses, at der sendes to gange 8 bit: først en adresse kode, som er 65, og derefter værdien 20. Det ses også at SS går lav ved dataoverførsel, og at clock er stabil. Testen er derfor tilfredsstillende.



Figur 2.12: Data bliver sendt fra DK8k til MASTER

Ved det andet test-scenarie bruges Linux-kommandoen cat til at læse fra MASTER. Her ses, at der sendes en adresse-bit 5 og 0 via MOSI, og på terminalen ses status for knappen på MASTER, som er trykket nede og derfor viser 1.



Figur 2.13: Data bliver læst fra MASTER

2.5 Diskussion

Ud fra ovenstående ses det at SPI kan bruges til at kommunikere imellem DK8k og MASTER, samt at kommunikere imellem PSoC-enhederne. Som resultaterne af testen viser, er SPI en smart måde, hvorpå der kan sendes og modtages data samtidigt. Det skal dog nævnes, at der har været mange problemer med denne kommunikation, især ifb. med at læse data fra slave til master, og der er

brugt adskillige timer på at få det til at virke. Så sammenlignet med UART, som fungerede problemfrit på 1. og 2. semester, så har SPI været meget mere udfordrende at få til at virke.

Kapitel 3

Motorstyring på x- og y-akser samt sensor detektering

3.1 Indledning

Dette afsnit har til formål at beskrive den unipolære styring af stepper motorerne der bevæger x- og y-akserne, og dermed, de to sensorer der skal detektere om der står en vinflaske i WinePrep.

Det er 2 stepper motorer af typen 28BYJ-48 der er valgt til at styre de to akser. For specifikationer af motoren henvises til databladet for 28BYJ-48 som kan findes i bilag xx.

Motoren er i forvejen kendt fra øvelsen om motorstyring i kurset Grænseflader til den fysiske verden (GFV), og det er på denne baggrund at de er valgt.

Den unipolære motorstyring er valgt frem for bipolær styring ud fra en overvejelse om moment og kompleksitet. For at være på den sikre side ift. at motoren har det nødvendige moment tændes to faser ad gangen i full step-mode.

De to sensorer er af typen GP2Y0A21YK, og er alene valgt ud fra Embedded Stocks udvalg. Sensorer med en rækkevidde fra 0 cm havde været foretrukket da designet af WinePreps ramme ville have været nemmere at implementere sensorerne i. Datablad for sensorerne kan findes i bilag xx.

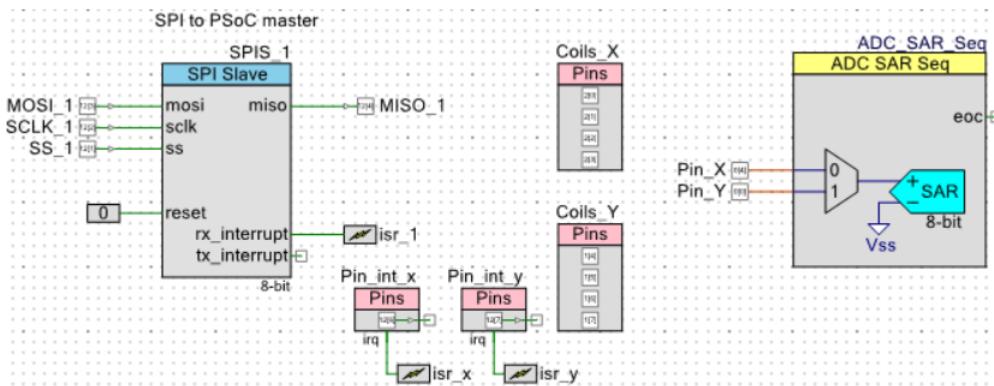
3.2 Design og implementering

Motorernes spoler styres, via et ULN2003AN board, fra en PSoC 5LP. Sensorerne aflæses direkte på PSoCen.

PSoC

Nedenfor beskrives PSoC håndteringen af motorer og sensorer, hvilke interne PSoC komponenter der har været nødvendige for dette samt pin konfigurering og kode. Kode er vedlagt som bilag xx. Bemærk at nogle af de benyttede funktioner i koden er skrevet af Cypress selv og kan kun benyttes i PSoC software.

Figur 3.1 viser TopDesign vinduet i PSoC. Det ses at SPI bruges til kommunikation, en sekventiel successive-approximation-register (SAR) analog-to-digital converter (ADC) til at aflæse sensorernes værdier, samt pins til at håndtere motorerne og trykknapperne for enderne af x- og y-akserne.



Figur 3.1: PSoC TopDesign for motorstyring og sensoraflæsning

SPI forbindelsen vil ikke forklares her. Der henvises til afsnit xx for uddybning af denne.

Når der skal detekteres for en flaske tændes ADCen og det analoge signal fra sensoren konverteres til digitale værdier som aflæses. For hvert 10. step (se linje 538 og 559 i main.c), motoren tager, læses en gang fra sensoren. Hvis sensoren detekterer begge kanter på vinflasken vil der på PSoC'en ske en kalkulation af hvor midten af flasken er. Herefter vil x- og y-akserne, og dermed åbningsmekanismen, bevæge sig ind på midten af flasken.

Sensor

Sensorerne detekterer så længe der er tilkoblet Vcc og GND til dem. I projektet betyder det at sensorerne konstant vil prøve at detekttere, men at der i koden kun aflæses fra dem når det er relevant. Dette gøres vha. den sekventielle SAR ADC fra TopDesignet.

Adspurgt værdi	Binært output
128	1
128 + 64	0
128 + 32	1
128 + 32 + 16	0
128 + 32 + 8	1
128 + 32 + 8 + 4	1
128 + 32 + 8 + 4 + 2	0
128 + 32 + 8 + 4 + 1	1
Resultat	10101101

Sekventiel SAR ADC

Sequencing SAR ADC komponenten fra Figur 3.1 består af en digital multiplexer, som udgør det sekventielle i ADCen. Yderligere findes en SAR ADC, der konverterer det analoge signal fra sensoren til en digital værdi i PSoCen. Der henvises til bilag xx for datablad for den sekventielle SAR ADC.

Den digitale multiplexer kan tage to input, 0 eller 1. Disse værdier bestemmer hvilken sensor der skal læses fra. Når multiplexeren modtager 0 læses fra x-aksen mens der læses fra y-aksen når den modtager 1. Dette styres fra main.c med funktionen "ADC_SAR_Seq_GetResult16(uint16 chan)", hvor "chan" er den port der læses fra. I main.c er 0 og 1 repræsenteret som "X" og "Y" - defineret i "enum axes" linje 63.

SAR ADC-kredsløbet indeholder en DAC, en komparator og et register til at oversætte det analoge signal fra sensoren til en digital værdi. Outputtet fra ADCen er et antal counts der med funktionen "int32 ADC_SAR_Seq_CountsTo_mVolts(int16 adcCounts)" konverteres til mVolt.

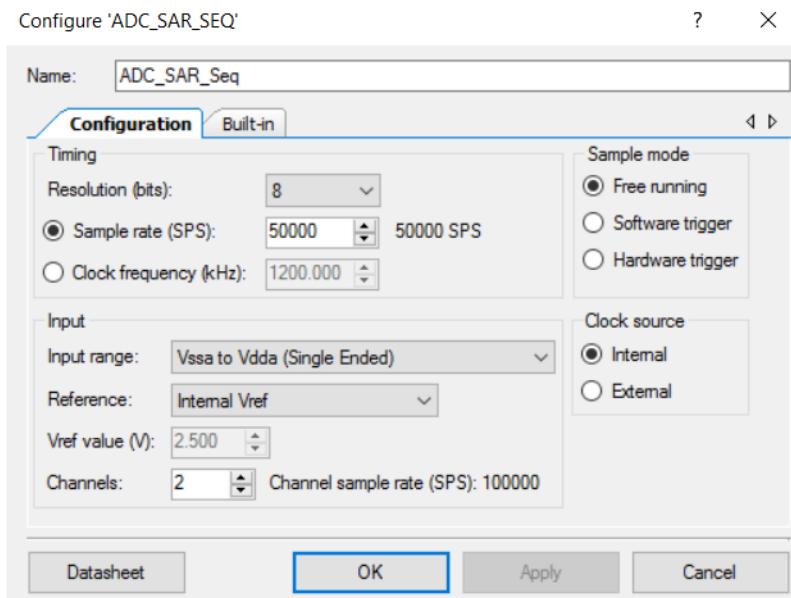
SAR er den måde som ADCen behandler dataene fra sensoren på. Successive approximation register fungerer ved at registret spørger efter en given værdi. Alt efter om den reelle værdi er højere eller lavere er outputtet henholdsvis et binært 1 eller 0. Registreret vil spørge indtil en given resolution er opnået og en tilnærmet værdi er opnået.

Nedenstående tabel viser hvordan registret behandler konverteringer. Det er et tænkt scenarie hvor resolutionen er 8 bit.

I tabellen ses det at det endelige resultatet afspejler de enkeltvise delresultater, hvor første delresultat svarer til MSB (most significant bit) og sidste til LSB (least significant bit).

Resolutionen i PSoC designet er sat til 8. Dette er gjort fordi en større nøjagtighed ikke er nødvendig. Sensorens vigtigste egenskab i WinePrep er ikke den målte afstand, men om der sker et skift i aflæst værdi, og dermed, et skift i afstand. Desuden bruges sensoren til at validere flaskens type, og der er afstanden heller ikke nødvendig. Afstanden måles mere præcist i motorernes steps. Se evt. afsnit xx for stepmotorer.

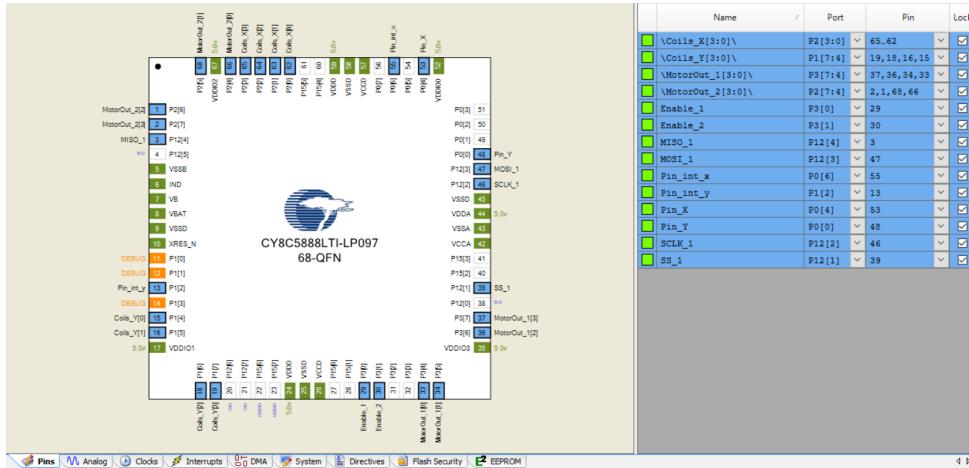
Figur 3.2 viser konfigureringen af ADCen. "Free Running" som sample mode er valgt fordi det er vigtigt at der konstant aflæses fra sensorerne og dermed sammenlignes om der er sket en ændring.



Figur 3.2: Konfigurering af sekventiel SAR ADC

Som det ses på figur 3.3 aflæses sensoren på x-aksen på port 0 pin 4 mens detekteringen på y-asken aflæses på port 0 pin 0 på PSoCen. Gennem ADCen fås et decimaltal som svarer til en afstand. Den digitale værdi for afstanden er opgivet i Volt. Sammenhængen mellem Volt og afstand i cm kan ses i databladet for sensoren på figur 4.

1. parentes	2. parentes	2. parentes • 15	Resultat
0b000000110	0b01100000	0b000000000	0b000000110
0b000000011	0b00110000	0b000000000	0b000000011
0b100000001	0b00011000	0b00001000	0b00001001
0b110000000	0b00001100	0b00001100	0b00001100



Figur 3.3: Pin diagram for hele designets komponenter

Motor

Coils_x og Coils_y fra figur 3.1 skrives til for at styre stepermotorernes spoler. Linje 3 og 4 i main.c definerer start positionen for motorerne. Hver bit er tilknyttet en bestemt spole. Når denne bit er 1 er spolen tændt, når den er 0 er spolen slukket. Der vil altid være 2 spoler tændt og 2 spoler slukket.

På linje 197 er funktionen rotateRight defineret. Dens opgave er at tænde og slukke spoler således at motoren vil dreje mod højre. Umiddelbart kan det være svært at gennemskue hvad der sker i koden. Nedenstående eksempel udpegsler 4 steps. Der tages udgangspunkt i x-aksen hvorfor det er xStep fra linje 3 der benyttes. Dennes start position er 0b1100, eller i et helt byte, 0b00001100. Første kolonne fra venstre kolonne er første (venstre) parentes fra linje 199, anden kolonne er anden parentes, tredje kolonne er resultatet af anden parentes multipliceret med 15 og fjerde kolonne er det endelige resultat der returneres fra funktionen.

Sidste række i tabellen viser at den er tilbage til udgangspunktet.

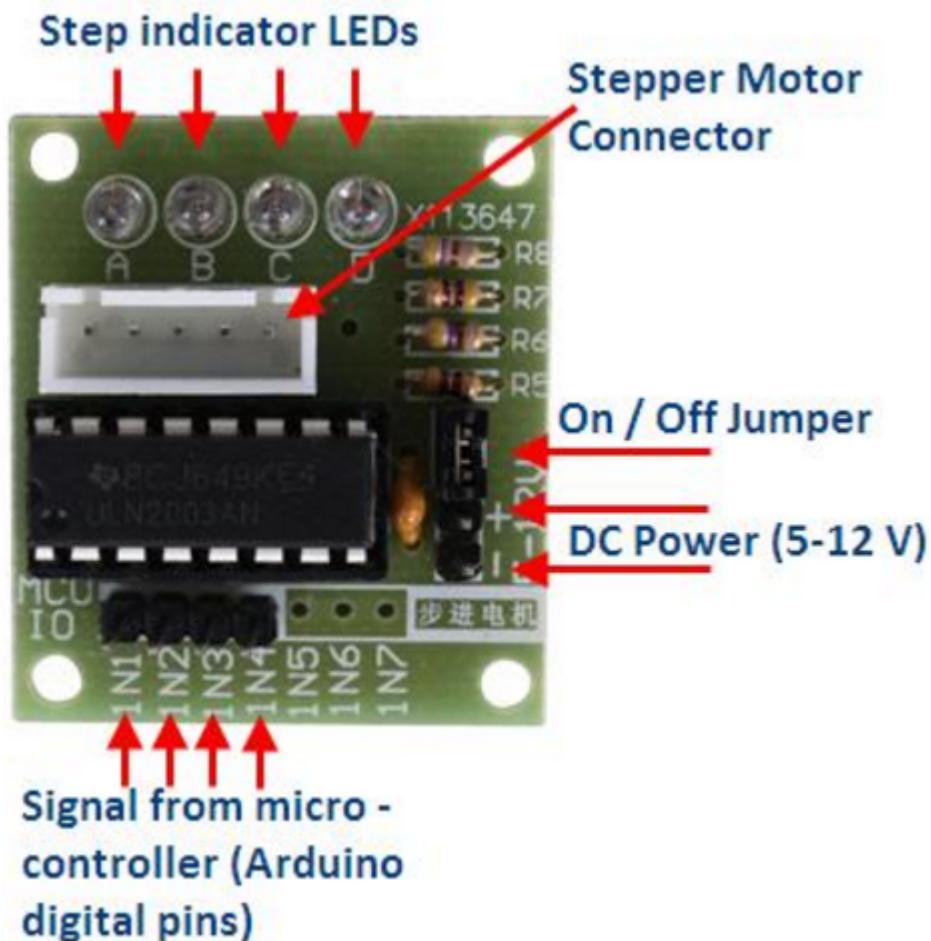
Samme forløb gælder for funktionen rotateLeft.

ULN2003AN board

Til at styre motoren fire faser anvendes fire transistorer. I stedet for fire individuelle transistorer er en chip, ULN2003AN, anvendt. Chippen indeholder 7 NPN Darlington transistor par hvor af kun 4 af parrene benyttes til motorstyringen. Værkstedet på studiet har stillet boards til rådighed hvor chippen befinder sig på. Datablad for ULN2003AN kan findes i bilag xx.

På boardet er desuden en socket til at koble motoren direkte på, en kondensator, 4 lysdioder til at indikere hvilken fase der er tændt, 4 formodstande til lysdioderne, en jumper, 2 pins til forsyningsspænding og 4 input pins fra PSoCen.

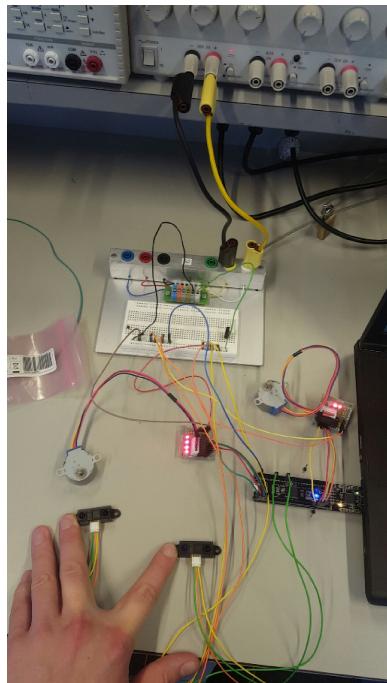
Boardet kan ses på figur 3.4.



Figur 3.4: Forside af board for styring af unipolær motor

3.3 Test

Modultesten blev gennemført ved at benytte UART forbindelse fra en PC til PSoCen. Integrationstesten blev gennemført med SPI forbindelse fra master PSoCen. Opstilling af modultest kan ses på figur 3.5 som også viser at motor og sensor for x-akse aktive.



Figur 3.5: Opstilling under modultest

Pins er konfigureret fra PSoC til ULN2003AN boardet som vist i opstillingen og på figur 3.3. Desuden er PSoCen tilsluttet Vcc og GND fælles med strømforsyning og ULN2003AN boardet. En permanent placeret strømforsyning fra E-LAB er anvendt.

For at sikre at der detekteres indenfor det relevante område, 10-20 cm fra afstandsmåleren, er tests blevet gennemført for at dokumentere hvilke værdier afstanden aflæses i. Sammenhæng mellem afstand og værdier kan ses i nedenstående tabel. Disse værdier er ikke omregnet til Volt, men afstand er målt med tommerstok.

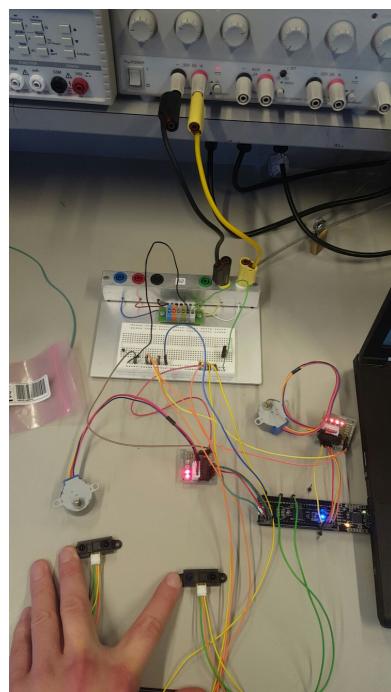
Afstand i cm	Værdi
Uden for rækkevidde	6
10	134
12	124
14	107
16	93
18	80
20	72
22	70

Afstandsmåling udenfor rækkevidde er foretaget ved at lade sensoren detektere fra et bord mod loftet af E-LAB, hvilket giver en minimumsafstand på langt over 80 cm, som sensorens rækkevidde er.

Dette kan ikke umiddelbart sammenlignes med figur 4 i databladet for sensoren fordi både værdier og graf ikke er ens. Dog kan disse værdier bruges til at definere minimums- og maksimumsafstande.

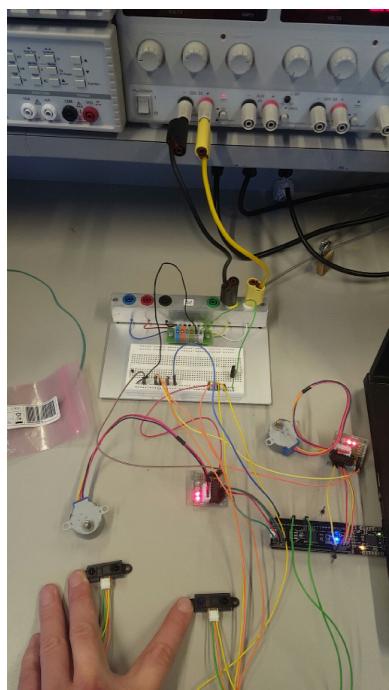
I koden er maksimumsafstand defineret som $LOW = 72$, dvs. hvis værdien fra afstandssensoren aflæses til under denne værdi, ignoreres den. I koden er der ikke taget hensyn til at en genstand kan være mindre end 10 cm fra sensoren. Dette skyldes at sensoren i WinePrep-rammen altid vil være placeret i en minimumsafstand af 10 cm fra objektet.

Først blev x-aksen aktiviteret ved at simulere "LOCATE_XY"-kommandoen fra linje 0x01 i main.c. Dette resulterede i billedet fra figur 3.5. En hånd blev anbragt foran sensoren og fjernet igen for at simulere en flaske. Y-aksen blev afsøgt for en flaske med samme procedure som for x-aksen - se figur 3.6.



Figur 3.6: Motor og sensor på y-akse aktiveret

Figur 3.7 viser endt detektering.



Figur 3.7: Endt detektering

3.4 Resultater

Følgende figurer viser værdier for målte afstande under testen. Disse kan sammenlignes med tabel 3.3. Hovedformålet med denne modultest er at undersøge om der sker en korrekt validering for detektering af en eventuel flaske.

Name	Value	Address	Type	Radix
ADCResult	6		unsigned short	decimal ▾
i	<optimized out>		unsigned short	Default

Figur 3.8: Afstandsmåling uden for rækkevidde

Denne test er i og for sig underordnet, men er taget med for at dokumentere hvilken værdi en afstandsmåling uden for rækkevidde har. Dette vil kunne bruges til at sammenligne værdier for en reel og en ureel måling.

Name	Value	Address	Type	Radix
ADCResult	115		unsigned short	decimal
i	<optimized out>		unsigned short	Default

Figur 3.9: Afstandsmåling ved 12 cm

Name	Value	Address	Type	Radix
ADCResult	69		unsigned short	decimal
i	<optimized out>		unsigned short	Default

Figur 3.10: Afstandsmåling ved 22cm

Ved testen sås det at afstandsmåling ved 22 cm resulterede i en uendelig loop i koden. Ved en afstandsmåling ved 12 cm sås det ønskede resultat, nemlig, at motoren bevægede sig indtil den aflæste værdi var mindre end LOW. Dette resulterede i at motoren roterede i modsat retning indtil en simuleret midte af flasken var nået.

For at se testen på video refereres til bilag xx.

3.5 Diskussion

Testen betragtes generelt som godkendt, dog ville en test med flere afstandsmålinger give et bedre billede. For at sikre den bedst mulige test skulle der have været testet ved grænserne, 10 og 20 cm. Dog betragtes denne test som gennemført af den årsag at den afviste en afstand som lå over den øvre grænse. Samtidig blev en test indenfor det godkendte interval, 10-20 cm, gennemført.

Kapitel 4

Bipolære steppermotorer

4.1 Indledning

Da der enkelte steder i WinePrep er behov for en større motorkraft end de unipolære stepper motorer kan leveres anvendes der her bipolære stepper motorer i stedet. Dette giver en lidt mere kompleks styring, men en noget større trækkraft.

4.2 Teori

En stepper motor er en elektromotor der kan bevæges i små trin af en bestemt vinkel, ofte 1.8 eller 0.9 grader afhængig af motoren, dette giver en meget præcis positionering til forskel fra de mere gængse DC motorer der er bedre egnet til kontinuerte bevægelser der skal foregå flydende.

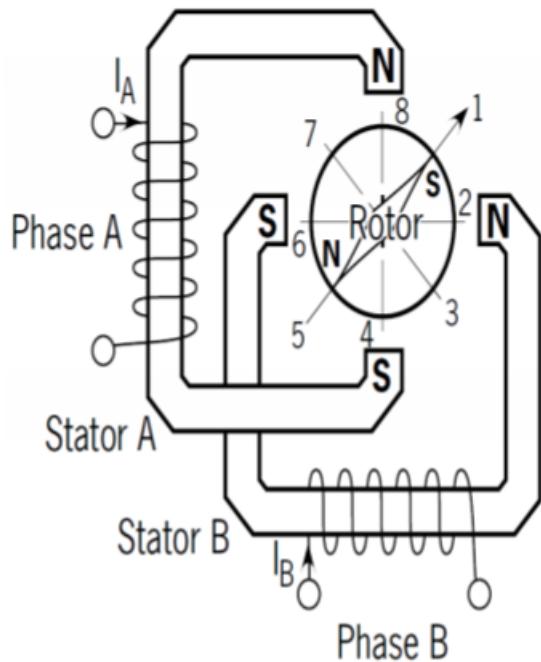
Steppermotorerne har flere fordele, i sammenhæng med WinePrep er de valgt da præcis positionering samt evnen til at gentage denne positionering er vigtig. Derudover har vi behov for en rigtig god evne til at bevare en given stillestående position på trods af ekstra belastninger, og her kommer stepper motorens rigtig gode hold torque til sin ret.

Stepper motorer kommer i flere varianter, og i WinePrep anvendes der både bipolare samt unipolare stepper motore. I dette dokument fokuseres der udelukkende på Bipolare stepper motore. Ønskes der information om brugen af unipolare stepper motore kan denne findes i dokumentet "Unipolære steppermotorer".

Bipolar Stepper Motor

Den bipolare steppermotor adskiller sig som det ses på figur 4.1 fra den unipolare stepper motor ved at have 1 vikling per fase hvor den unipolare har 1 fælles tab i midten af spolen.

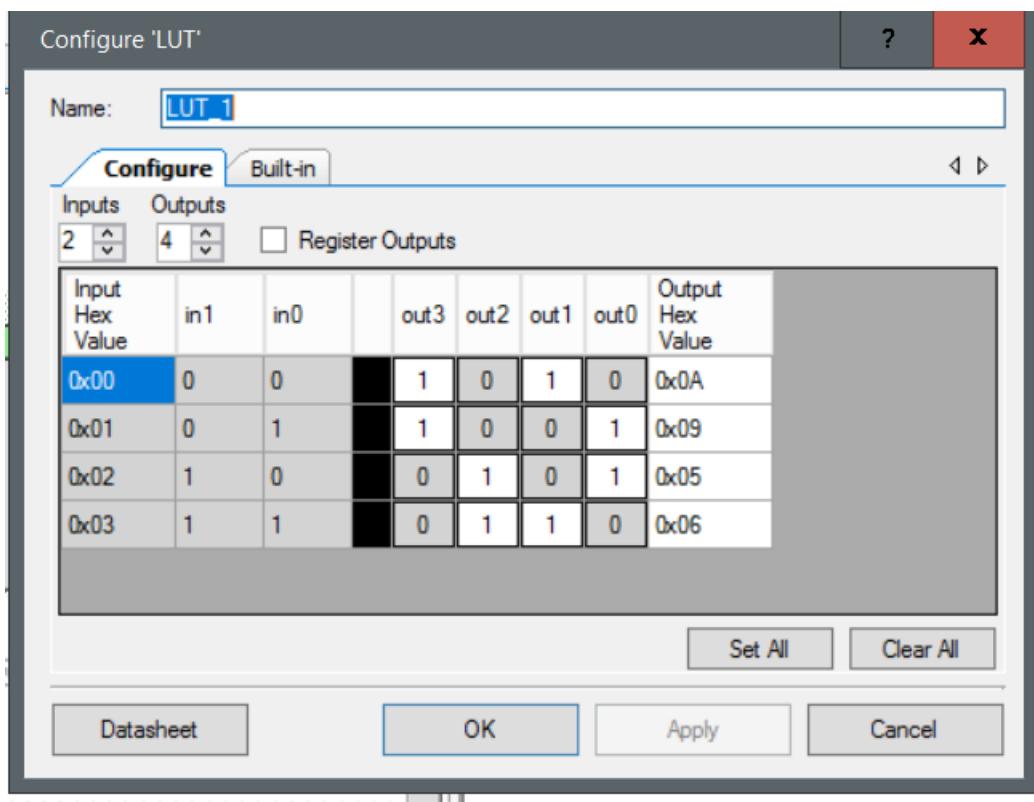
Afhængig af polariteten i den givne spole vil motoren rotere sin akse til den givne position.



Figur 4.1: Bipolar stepper motors konstruktion

Til WinePrep anvedes der kun stepper motorer med gearing, hvilket gör at vi ikke anvender microstepping, da vi allerede uden dette har en rigelig god præcision.

For at anvende en bipolar stepper til full step skal faserne påtrykkes spænding ud fra lookup tabellen på figur 4.2.



Figur 4.2: lookup tabel for bipolar stepper motor

Den bipolare stepper motor styres med 1 fuld h-bro pr fase.

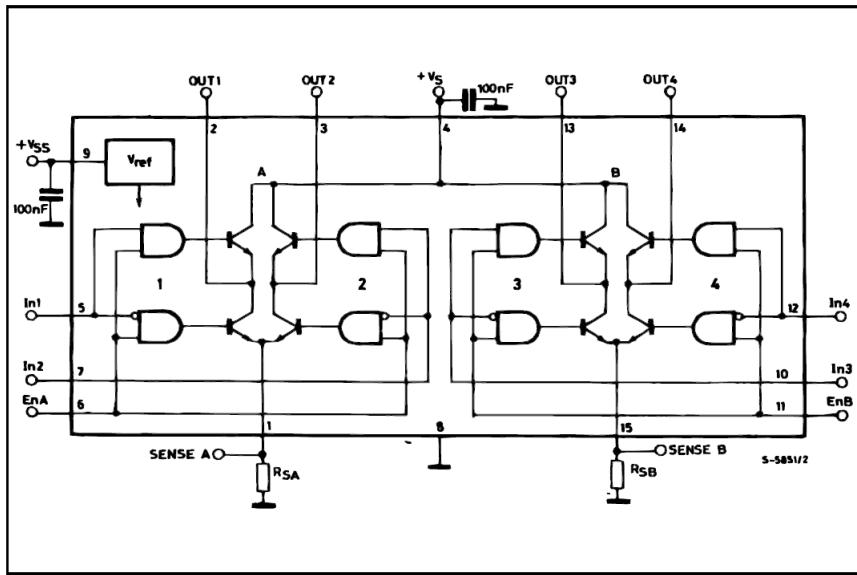
4.3 Design

Til at styre de Bipolære stepper motorer anvendes der et print designet omkring chippen L298, der er en dual full-bridge driver.

På blok diagrammet for L298 der kan ses på figur 4.3, ses det at chippen indeholder mulighed for at lave et strøm feedback loop tilbage til PSoC 5 Chippen hvor man ved hjælp af en comparator og en DAC kan sikre sig mod for store strømmme i motoren.

Derudover kan man se hvordan brugen af AND-Gates kan medføre en besparelse i antallet af pins der er nødvendige for at drive en full-bridge.

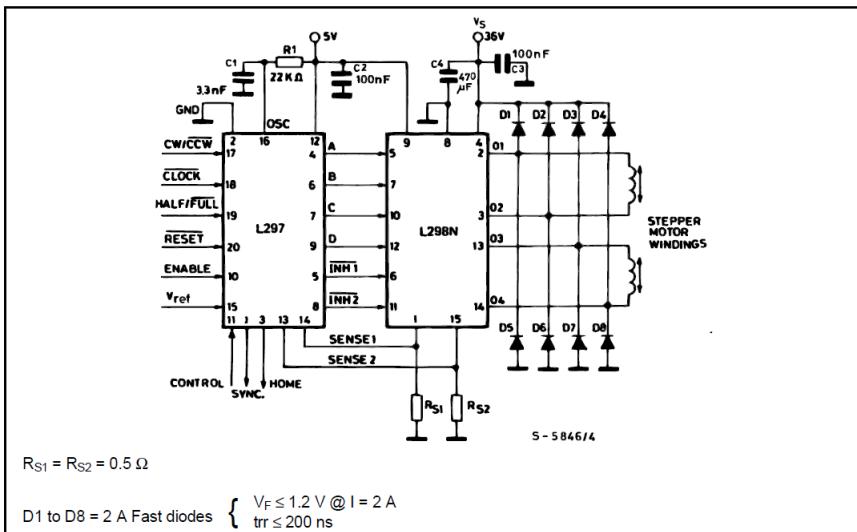
BLOCK DIAGRAM



Figur 4.3: Blokdiagram for L298, kilde: datablad for L298

Denne forsynes via 12V og 78L05 DC-DC 5V konverteren på boardet med 5V til de logiske enheder på chippen.

Derudover er der som anbefalet i databladet for 78L05 sat kondensatorer til at stabiliserer 5V forsyningen, samt på L298 ud fra anbefalinger i databladet.



Figur 4.4: kredsløb til styring a bipolar steppermotor med L298 samt L297.
Kilde: datablad L298.

Der er i designfasen taget udgangspunkt i kredsløbet der ses på Figur 4.4 , i vores kredsløb er L297 dog erstattet af en PSoC5LP og der er anvendt andre modstande til strømsfeedback delen, da vi ikke havde den samme type på lager på skolen.

Der i som RS1 og RS2 istedet anvendt en 0.1 ohm effektmodstand, da de små 1/4 watts modstande ikke ville kunne tåle de strømme der vil kunne løbe igennem motoren.

Dette giver os ud fra kirchoffs strømlov at strømmen gennem RS1 er den samme som strømmen gennem spolen på stepper motoren, derved kan vi ved hjælp af ohms lov finde frem til den spænding vi skal indstille vores DAC i PSoC5LP til, for at vi opnår den ønskede strøm i motoren.

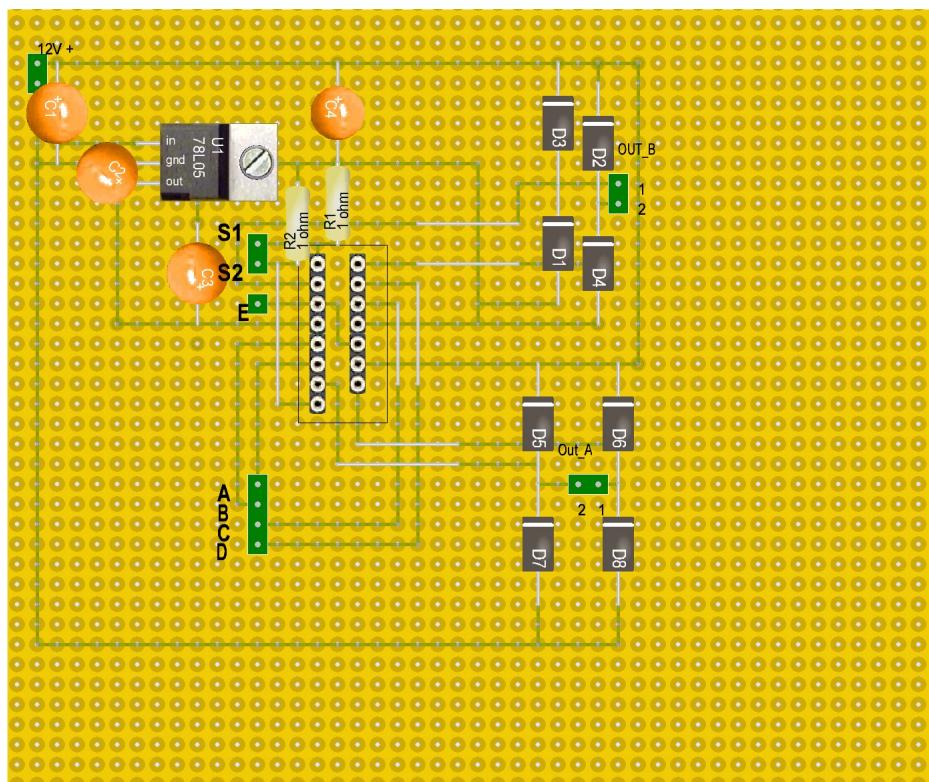
Den generelle formel findes i ligning 4.1 hvorfra en mere specifik udgave der kun vil passe på vores kredsløb kan findes på i ligning 4.2

$$\frac{I_{spole}}{R_{S1}} = V_{DAC} \quad (4.1)$$

$$\frac{I_{spole}}{0.1\Omega} = V_{DAC} \quad (4.2)$$

4.4 Implementering

Ud fra dette og de tilgængelige komponenter på vores komponentalger er vi kommet frem til et print design der ser ud som på figur 4.5.

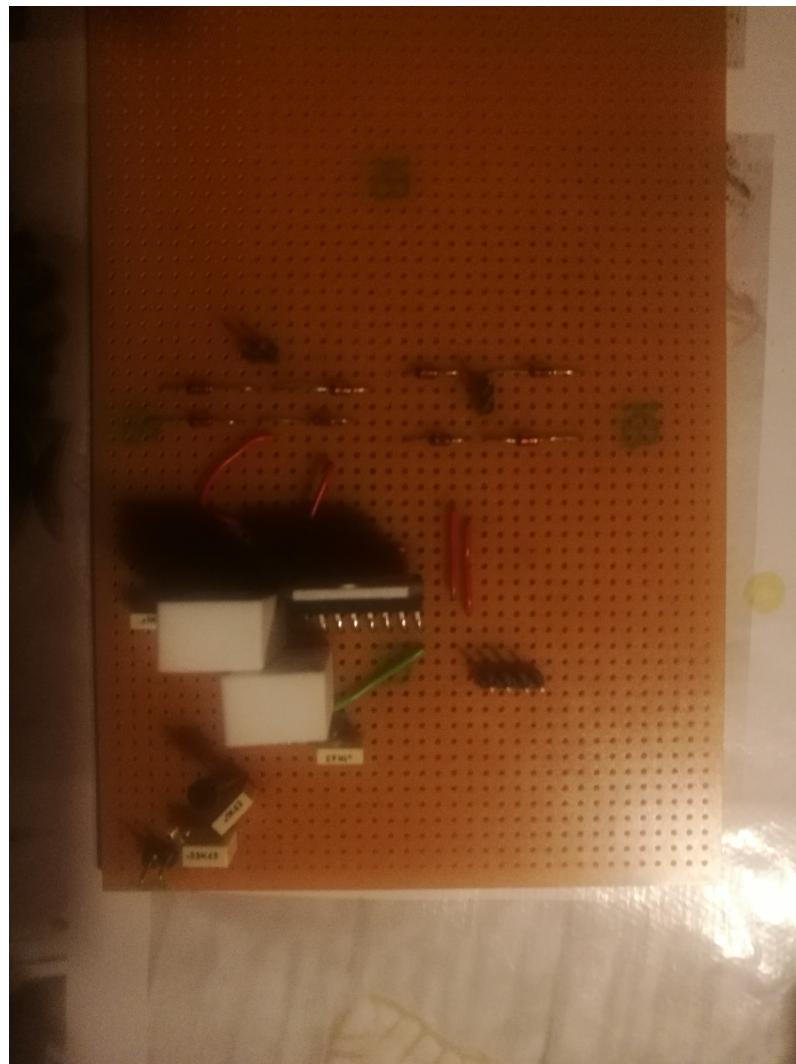


Figur 4.5: Resulterende printdesign

De anvendte dioder er af typen 1N4148 og anvendes som flybackdioder for at beskytte L298 mod de strømme der opstår når spolerne slukkes.

Kondensatorene på det endelig print er ikke af tantalum typen, men værdier er de foreskrevne i databladet for L298.

Det færdige print ses på Figur 4.6.



Figur 4.6: Resulterende print

4.5 Test og resultater

Test resultater mangler da software til styringen ikke er fuldt funktionel endnu.

Problemet med softwaren på nuværende tidspunkt at at PWM delen i PSoC generere et 14 kHz 50% dutycycle puls signal hvor den burde være lav.

4.6 Konklusion

Mangler grundet manglende test.