

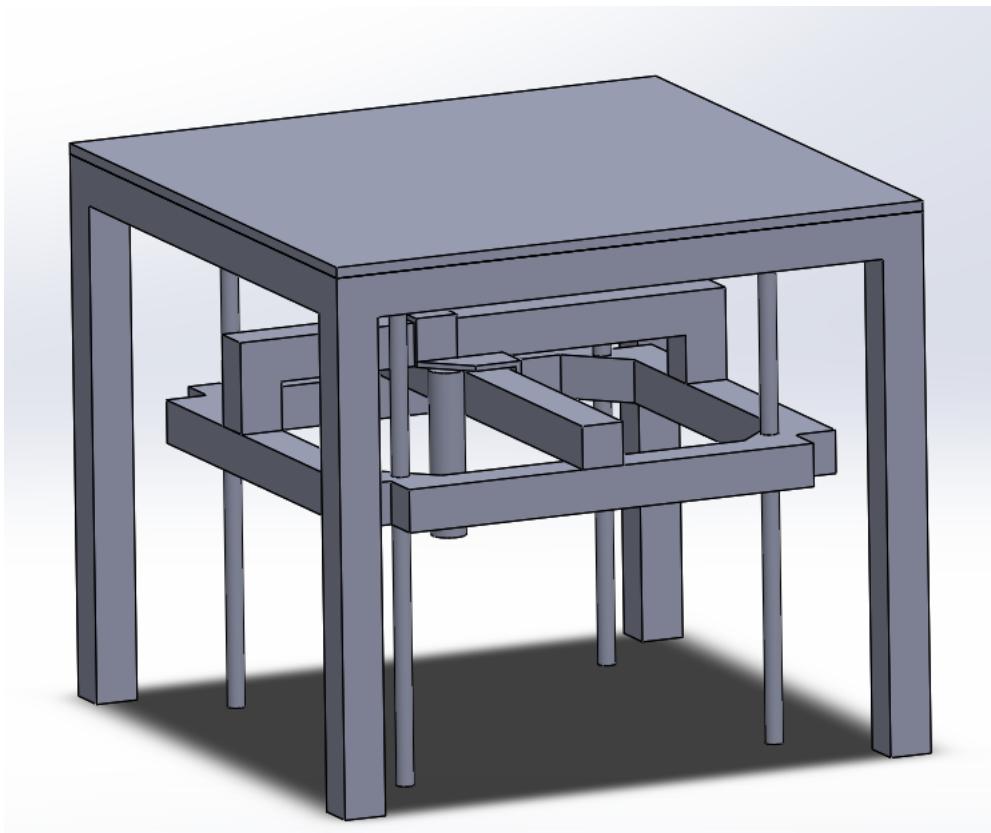
# Projektraport

## Semesterprojekt 3. Semester

### Gruppe 10

Vejleder: Søren Hansen  
Gruppemedlemmer:

Navn	Studienummer
Jacob Munkholm Hansen	201404796
Halfdan Vanderbruggen Bjerre	20091153
Mikkel Espersen	201507348
Ahmad Sabah	201209619



Figur 0.1: WinePrep

# Indhold

<b>Indhold</b>	<b>i</b>
<b>1 Forord</b>	<b>1</b>
1.1 Læsevejledning . . . . .	1
<b>2 Kapitel overskrift</b>	<b>2</b>
2.1 Sektion overskift . . . . .	2
2.2 Hovedansvarsområder . . . . .	2
<b>3 Indledning</b>	<b>4</b>
3.1 WinePrep . . . . .	4
<b>4 Krav</b>	<b>6</b>
4.1 Aktører . . . . .	7
4.2 Use-cases . . . . .	7
4.3 Ikke-funktionelle krav . . . . .	8
<b>5 Afgrænsning</b>	<b>9</b>
<b>6 Realisering</b>	<b>11</b>
6.1 Metode . . . . .	11
6.2 Systemarkitektur . . . . .	12
6.3 Design . . . . .	14
6.4 Implementering . . . . .	28
6.5 Test . . . . .	32
6.6 Resultater . . . . .	37
6.7 Diskussion af resultater . . . . .	39

# Kapitel 1

## Forord

Denne rapport er skrevet på 3. semester af gruppe 13, på retningerne IKT og EE ved Aarhus Universitet, Ingeniør højskolen. Vejleder for dette projekt er Søren Hansen. Afleveringsdatoen for denne projektrapport er den 20. December 2016, og bedømmelse er den 18. Januar 2017. Rapporten er udarbejdet på baggrund af den dokumentation, som kan findes i bilaget for projektrapporten.

### 1.1 Læsevejledning

Det er tiltænkt at rapporten skal læses i kronologisk rækkefølge, dog kan afsnittene omkring implementering og test af delsystemerne læses uafhængigt af hinanden. De forskellige dele er inddelt i kapitler. Hvert kapitel indeholder sektioner med dertil hørende undersektioner. Disse er alle nummererede.

## Kapitel 2

# Kapitel overskrift

### 2.1 Sektion overskrift

#### Undersektion overskrift

Der vil blive brugt initialer på gruppens medlemmer til angivelse af, hvem rapportens sektioner er skrevet af:

Mikkel Busk Espersen (MBS),  
Jacob Munkholm Hansen(JMH),  
Ahmad Sabah (AB),  
Halfdan Vanderbruggen Bjerre(HVB).

I de udarbejdede UML- og SysML-diagrammer og beskrivelser af disse vil der blive refereret til p- og s-motorer. Disse dækker over motorerne til styring af henholdsvis åbningsmekanismen(reference til ordliste) og skruen.

### 2.2 Hovedansvarsområder

Tabel xx viser fordelingen af hovedansvarsområder for produktet fordelt på gruppemedlemmer. Emnerne er inddelt i primær og sekundær, som informerer om medlemmers specialistviden og kernekomp petencer indenfor produktudviklingen. Enkelte sekundære felter er tomme, dette betyder at ingen har været sekundær på emnet.

Emne	Primær	Sekundær
Brugergrænseflade (GUI)	AS	HVB
SPI DevKit-PSoC	HVB	JMH
SPI PSoC-PSoC	HVB, JMH	
PSoC software sensor	JMH	MBE
PSoC software sensor	JMH	MBE
Bipolære motorer	MBE	JMH
Unipolære motorer	MBE	JMH
DC motor	MBE	
Konstruktion og mekanik	AS	HVB

Tabel 2.1: Fordeling af hovedansvarsområder

## Kapitel 3

### Indledning

Interessen for robotteknologi er steget, især indenfor hjælpemidler til ældre. Den aldrende befolkningssgruppe striger stødt, og derfor er der behov for flere intelligente løsninger, som kan hjælpe fysisk hæmmede mennesker i deres hverdag. En af ideerne bag dette projekt var konstruktionen af en robot, som kunne hjælpe svagelige mennesker med at trække proppen ud af en vinflaske.

Smart-produkter er generelt blevet mere udbredte i moderne hjem, og der bliver større krav til hvilke daglige gøremål der skal kunne løses automatisk. Her kunne en automatisk vinåbner sætte nye standarder for smart-produkter i almindelige hjem. En sådan vinåbner kunne tilbyde en ny og innoverende måde at åbne en vinflaske. Med intelligente enheder som kan detektere vinflaskens positionen og mål, skulle vinåbnernen åbne alle typer af vinflasker.

Et andet fokuspunkt for vinåbnernen er forberedelse af vinen. For at få den optimale oplevelse ud af en vin, skal den åbnes rettidigt så den iltes før indtagelse. Ilningstiden kan desuden variere fra vin til vin, og derfor kan uerfarne vindrikkere have svært ved at ilte deres vin korrekt. Dette kunne løses ved at automatisere denne ilningsprocess, hvor brugeren kan få åbnet vinen til et forudstemt tidspunkt bestemt ud fra vinens type.

Derudover kunne den automatiske vinåbner indeholde en række features som kunne forbedre vinoplevelsen. Dette kan gøre den til et tilstrækende produkt også for vinentusiaster, som ønsker et premium produkt der kan give dem en større nydelse ved vindrikning.

#### 3.1 WinePrep

Visionen for den automatiske vinåbner "WinePrep" var et system som kunne imødegå et hvert behov der måtte være indenfor drikning og forberedelse af

vin.

Udover selve åbningen af en vinflaske skal systemet:

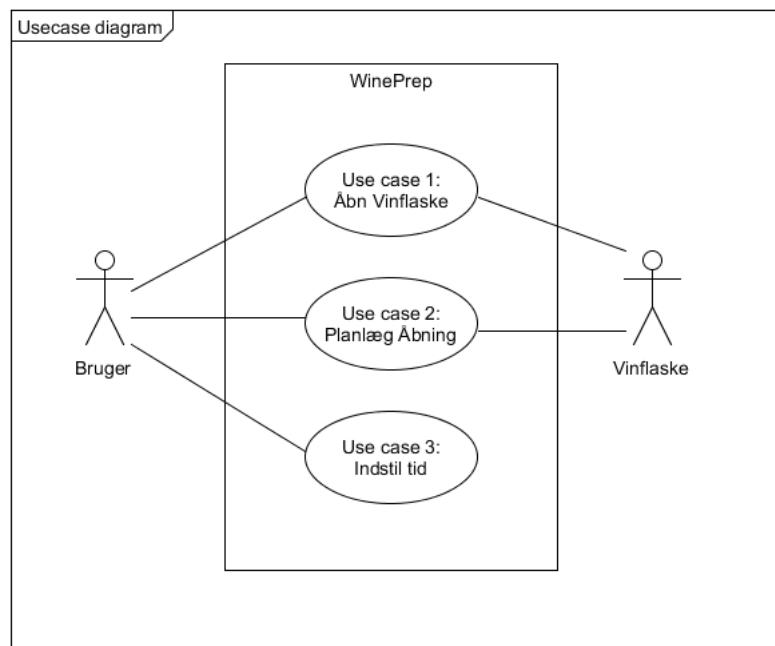
- Automatisk kunne finde vinflaskes top, så alle typer vinflasker uanset højde og øvrige mål kunne åbnes.
- Kunne åbne vinen til et forudbestemt tidspunkt og derved sikre en optimal iltning af vinen
- Kunne måle vinens temperatur, og regulere denne så vinen kan nydes ved dens optimale betingelser.
- Kunne finde information om den optimale iltningstid for en bestemt vin via tilslutning til en database
- Indeholde en social medie platform "WineBook", hvor brugere kunne anmelde vine, og interagere med andre vinelskere
- Have en mobil applikation tilsluttet, der gjorde fjernbetjening af systemet muligt
- Kunne scanne etiketten på en vinflakse og finde information om vinen via en database
- Kunne dispensere korkpropstenen for vinflasken efter vinåbningen er afsluttet.

Ved udformingen af det ideelle produkt er der ikke taget hensyn til gruppens begrænsede tid, ressourcer og kompetencer. Visionen for WinePrep fungerer i dette projekt blot som et startsted for det videre projektforløb. Ud fra det ideelle produkt vil gruppen udvælge de funktioner som vurderes til realistisk set at kunne gennemføres.

# Kapitel 4

## Krav

Med udgangspunkt i afgrænsningen for projektet er der blevet opstillet en række krav for WinePrep. De funktionelle krav er beskrevet ved tre use-cases, hvorfaf de to mest betydende for WinePrep's værdi vil blive beskrevet nøjere i dette kapitel. Disse omfatter den essentielle funktionalitet, som gør WinePrep enestående i forhold til andre produkter på markedet. Før disse beskrives er det dog påliggende at få sat nogle rammer på WinePrep i form af systemets grænseflader til dettes aktører.



Figur 4.1: usecasediagram for WinePrep

## 4.1 Aktører

Som set i figur 4.1 er der to aktører for dette system: brugeren af WinePrep; og den givne vinflaske, der skal åbnes.

### Bruger

Brugeren af WinePrep er den primære aktør, som interagerer med systemet ved at indsætte en vinflaske i WinePrep og/eller betjene systemet via dettes trykskærm, hvorpå brugeren kan benytte sig af produktets funktioner.

### Vinflaske

Vinflasken indgår som en passiv aktør i systemet, der inspiceres og åbnes af WinePrep. Denne skal være af en bestemt type og ved indsættelse i WinePrep være i en bestemt tilstand. Mere om dette findes beskrevet i bilaget(reference til detaljer om vinflaske).

## 4.2 Use-cases

De to vigtigste use-cases vil her blive beskrevet overordnet. Mere information om disse og den tredje use-case kan findes i bilaget(indsæt reference til use-cases i bilaget).

### Åbn vinflaske

Brugeren skal efter at have indsat en vinflaske i WinePrep trykke på knappen "Åbn nu" på trykskærmens. Systemet skal da foretage målinger af den indsatte vinflaske for at bekræfte, at denne er af en type, der er kompatibel med systemet. Herefter skal systemet åbne vinflasken og informere brugeren om dette. Løbende under processen vil der blive taget hånd om fejlscenarier, hvor brugeren via trykskærmens vil blive informeret om, at vinflasken ikke er indsats korrekt eller er af en ukompatibel type, hvis denne ikke godkendes af systemet(indsæt reference til udvidelser/undtagelser for UC1).

### Planlæg åbning

Brugeren skal på trykskærmens trykke på knappen "Planlæg åbning" og herefter på to scroll-down-menuer(reference til ordliste) vælge et klokkeslæt, hvor vinflasken ønskes åbnet, og vinen drikkeklar(reference til ordliste). Systemet venter da til tilnæringstidspunktet(reference til ordliste), hvor brugeren forinden skal have indsat vinflasken i WinePrep, hvorpå det påbegynder proceduren beskrevet i "Åbn vinflaske" ovenfor. Trykskærmens vil herefter vise det tidspunkt,

hvor vinen vil være drikkeklar. Kan det ønskede klokkeslæt, hvor vinen skal være drikkeklar, ikke forenes med iltningstidspunktet(reference til detaljer om iltningstidspunkt), annulleres processen, hvorefter brugeren vil blive tilbudt muligheden for at få vinflasken åbnet før det planlagte åbningstidspunkt.

### 4.3 Ikke-funktionelle krav

WinePrep skal have en trykskærm, som skal indeholde knapper med billeder på, der illustrerer hver knaps funktion(reference til billede af GUI).

Disse knapper skal ligeledes have et flademål, som gør det muligt for brugeren at kunne trykke på disse med sin finger uden at ramme en naboknap(reference til bilag: ikke-funktionelle krav/brugervenlighed).

WinePrep skal kunne behandle brugerinput indenfor et tidsinterval på 2 sekunder og løbende holde brugeren opdateret om vinflaskens status (referencer til bilag: ikke-funktionelle krav/brugervenlighed + /ydeevne).

Vinflasken skal være af en på forhånd bestemt type(reference til detaljer om vinflaske).

WinePrep skal kunne detektere vinflaskens centrum med en maksimal afvigelse på 1mm for at undgå, at vinflaskens prop knækker ifm. åbningen.

Skulle der opstå et behov for reparation eller vedligeholdelse af systemet, skal en ekspert i WinePrep's interne konstruktion(reference til bilag: ikke-funktionelle krav/vedligeholdelse) kontaktes.

## Kapitel 5

# Afgrænsning

Det er fra IHA's side opstillet følgende krav til projektet:

- Der skal indgå en aktuator og/eller sensor.
- Der skal være implementeret en brugergrænseflade.
- PSoC og Linux platform skal indgå i projektet.
- Skal indeholde faglige elementer fra semesterets andre fag.

En afgrænsning for projektet er formuleret ud fra visionen for WinePrep. Tilkobling af WinePrep til database, den mobile applikation, samt regulering af vinens temperatur, anses for spændende udfordringer, dog en anelse for tidskrævende. Desuden ligger disse funktioner uden for de faglige mål for projektet, og vil derfor ikke blive medtaget. Implementeringen af et socialt medie som "WineBook" er meget omfangsrigt, og er ikke realistisk for dette projekt.

Hovedfunktionaliteten for systemet er åbning af en vinflaske, og denne funktionalitet ønskes derfor med i projektet. Herudover bliver timing af åbningstidspunkt og detektering af vinflaskens position udvalgt som realistisk mål for dette projekt. Dispensering af korkproppen bliver også medtaget, men simplificeret så denne proces blot består i at rotere skruen den modsatte vej, og dermed lade proppen falde af.

For at få systemet til at fungere, skal der løses nogle mekaniske udfordringer som ligger uden for de faglige mål for projektet. Trods disse udfordringer bliver konstruktionen af WinePreps fysiske rammer medtaget for at kunne udføre en vinåbning.

Projektet skal gerne udmunde i en prototype med ovenstående funktionalitet, som kan bruges til videreudvikling. Prototypen vil kun være i stand til at behandle vinflasker af en bestemt type og mål, for mere information omkring kompatibilitet henvises til dokumentation (reference). Med henblik på videreudvikling af WinePrep, skal prototypen dog stadig have en positioneringsmekanisme der muliggør detekteringen af forskellige typer vinflasker.

Den endelige prototype vil ikke fremstå som et færdigt produkt, hverken i funktionalitet eller konstruktion.

## Kapitel 6

# Realisering

### 6.1 Metode

**UML- og SysML:** Er værktøjer til konstruktionen af arkitekturen for systemet. De har til formål at give et visuelt overblik over de delelementer systemet består af. UML og SysML dækker over flere forskellige typer diagrammer til beskrivelse af software og hardware komponenter.

Til beskrivelsen af hardwarekomponenter og deres interne grænseflader kan **IBD og BDD** anvendes. BDD'et er brugt til at nedbryde systemet i blokke, således at man hurtig kan danne sig et overblik over hvilke fysiske elementer systemet består af. IBD'erne er brugt til at beskrive de interne grænseflader der er i systemet. Altså ind- og udgangsportene som er på de forskellige dele af produktet.

Til beskrivelsen af softwarearkitekturen blev der udarbejdet **klasser- og sekvensdiagrammer**. Klassediagrammerne viser hvilke klasser systemet består af, mens sekvensdiagrammerne skal vise kommunikationen mellem disse klasser.

Herudover indeholder projektrapporten **state mashines (STM) og flow charts** til beskrivelse af softwares adfærd.

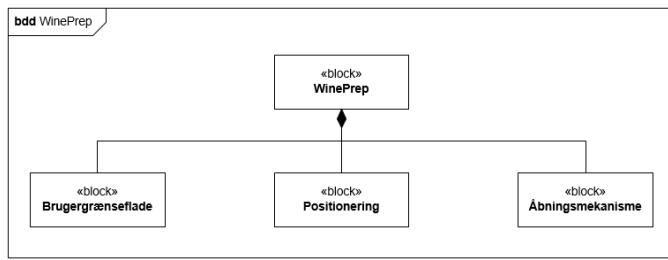
**usecases** er benyttet til at definere de funktionelle krav, mens **FURPS+** og **MoSCoW** er benyttet til at definere de ikke-funktionelle krav. Usecasene er udviklet ud fra systemniveauet.

**domænemodel** er ikke medtaget i rapporten, hvilket er en afvigelse fra ASE modellen. Da usecasene er lavet på systemniveau, er der ikke et godt fundament til udarbejdelsen af en domænemodel. Det er svært på systemniveau at udlede konceptuelle klasser for winePrep, og derfor giver domænemodellen ikke nogen værdi mht. applikationsmodellerne.

For at definere hvorledes software og hardware er allokeret, er der lavet **allokeringsdiagrammer**. Disse mapper hardware og software ned på de enkelte hardware blokke.

## 6.2 Systemarkitektur

Med udgangspunkt i usecasene er systemets funktionalitet forsøgt brudt ned i mindre logiske blokke. Disse skal dække over betjening af systemet, detektering af vinflaske og åbningen af denne. På baggrund af disse krav er BDD'et på figur 6.1 udarbejdet.



Figur 6.1: BDD over WinePrep

**Brugergrænsefladen** betjenes af bruger og initierer usecasene som beskrevet i Krav (indsæt reference hertil).

**Positionering** finder vinflaskens position og flytter **Åbningsmekanismen**, så den er klar til at åbne vinflasken.

**Åbningsmekanismen** fjerner proppen fra flasken og dispenserer proppen.

For at forstå den interne funktionalitet og kompleksitet i blokkene **Positionering** og **Åbningsmekanisme** er disse brudt yderligere ned.

### Positionering

I forhold til den specificerede afgrænsning<sup>1</sup> for projektet, kræves det, at **Positionering** skal operere på tre rumlige akser: to horizontalt vinkelrette (x og y) og én vertikal (z). Derudfra kan flaskens koordinater bestemmes. Da flaskens position er statisk, forløber detekteringen sig i bevægelse langs dennes akser. **Positionering** kan dermed opdeles i to moduler: ét modul tager sig af detektering af flasken, mens det andet tager sig af bevægelsen langs akserne.

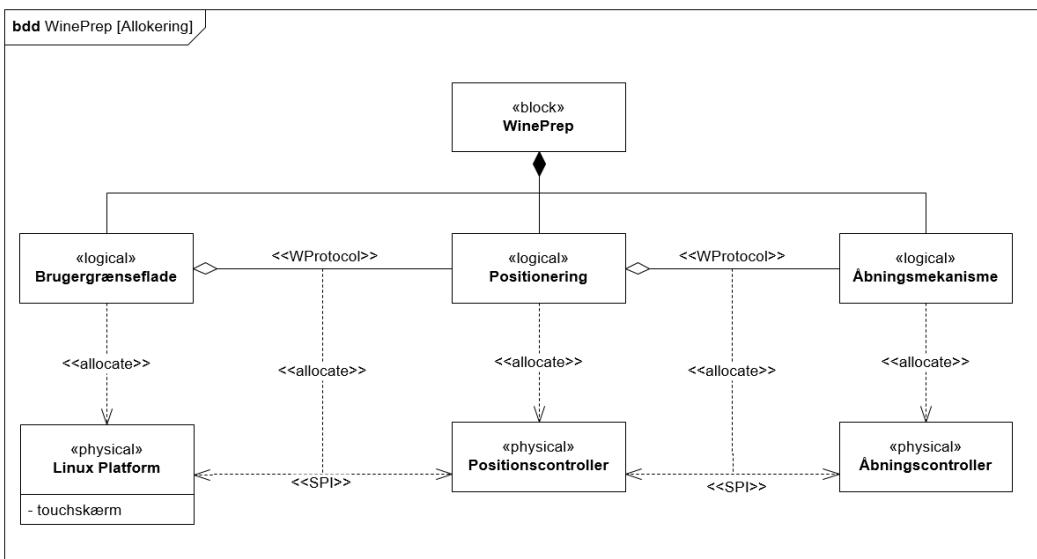
<sup>1</sup>Se kapitlet Afgrænsning

## Åbningsmekanisme

Det at åbne en flaske kan brydes ned i to funktioner: **iskruning** i flaskens prop, og **proptrækning** af proppen. Derfor kan **Åbningsmekanismen** opdeles i to moduler, som tager sig af disse funktioner.

## Allokering

De blokke, der er opstillet på figur 6.1, allokeres på en række fysiske blokke, som vist på figur 6.2.



Figur 6.2: Allokeringsdiagram over WinePrep

**Positionering** og **Åbningsmekanisme** er allokeret på hver deres microcontroller, som i dette projekt er besluttet til begge at være PSoC 5LP. **Brugergrænsefladen** er allokeret på en Linux-platform, som er DevKit8000.

Figur 6.2 viser ydermere allokeringen af det interne hierarki blandt blokkene. **Brugergrænsefladen** bruger **Positionering** vha. en seriel kommunikationsform. Det samme gør sig gældende mellem **Positionering** og **Åbningsmekanismen**.

## 6.3 Design

### Positionering

Positionering består af 2 sensorer som drives af aktuatorer på de tre akser. Én motor på henholdsvis x og y flytter disse to akser mens to motorer driver z-aksen.

Som set i allokeringsdiagrammet<sup>2</sup> benyttes en PSoC til at styre disse motorer og sensorer. Det forudsættes, at der til at kontrollere, at bestemte punkter på akserne er nået, benyttes trykknapper, som, når disse påtrykkes, igangsætter en interruptroutine på PSoC'ene.

### Hardware

#### Detektering

At lokalisere flasken kræver en type af sensor. Embedded Stocks udvalg var begrænset så valget stod mellem to typer af afstandsmålere som enten vha. lys eller ultralyd kan detektere et objekt.

Tabel 6.1 viser forskellen i præcision for afstandsmåling mellem ultralydssensoren, HC-SR04, og lasersensoren, GP2Y0A21YK. Præcisionskravet er 1 mm for at **Åbningsmekanismen** har det mest centreret punkt på proppen at åbne.

	Faktisk præcision
HC-SR04	300 mm <sup>3</sup>
GP2Y0A21YK	2,57 mm <sup>4</sup>

Tabel 6.1: De to sensores præcision

Tabellen udmundede i et fravælg af HC-SR04 sensoren fordi dennes præcision er meget unøjagtig. Da begge sensorer ikke opfylder kravet måtte en anden løsning om præcision findes. Denne kan ses under afsnittet om Aksestyring.

Sensorens ansvar er derfor kun at registrere om der er et objekt eller ej. Ved at sammenligne afstanden, målt i volt<sup>5</sup>, med et referencepunkt som er konstant<sup>6</sup> er det muligt at detektere for en flaske.

Fordi PSoC 5LP er valgt som microcontroller kan håndteringen af sensoren

---

<sup>2</sup>Figur 6.2 i kapitel 6.2

<sup>5</sup>Datablad for GP2Y0A21YK

<sup>6</sup>Dokumentation for Konstruktion

gøres i PSoC Creator. Det analoge signal fra sensoren konverteres til digitalt med SAR\_ADC komponenten<sup>7</sup>.

### Aksestyring

Til at flytte akserne bruges motoren 28BYJ-48<sup>8</sup>. Valget er taget på baggrund af en analyse af forskellige typer af motorer<sup>9</sup>, som viser at steppere er DC- og servomotorer overlegne i præcision. En bevidsthed om at sensorerne ikke opfyldte præcisionskravet på 1 mm. er også grundlaget for at vælge 28BYJ-48. Desuden udbyder Embedded Stock kun denne model, og ikke andre motorer, i et antal, som skal bruges i projektet.

I full-step mode flytter motoren aksen 0,04 mm.<sup>10</sup> per step, som er langt mere præcis end projektet kræver. Dermed er det ikke sensoren der afgør hvor flasken befinner sig, men motorerne. Dette gøres på softwaresiden som kan læses længere nede.

Valget af denne motor begrænser hastigheden hvormed akserne bliver flyttet fordi antallet af steps per rotation for skaftet er så stort.

### Unipolær motor

28BYJ-48 er bygget som unipolær og kan styres sekventielt gennem fire transistorer<sup>11</sup>, hvilket ULN2003AN boardet<sup>12</sup> kan bruges til. Et step tages ved at tilføre en transistor nok strøm til at den "åbner" og på den måde kan en strøm løbe i en af motorens spoler. I databladet for 28BYJ-48 ses en rød ledning der lader til at være sat på midterudtaget af de to spoler. Den røde ledning deler de to spoler i fire halve spoler. Ledningen fungerer som strømbærer til spolerne mens de fire andre ledninger skiftevis ledes til GND.

Momentet for motoren er dog relativt svagt, som kan ses under kapitlet Test, hvor den unipolære motor ikke har tilstrækkelig moment til at drive akserne uden små ophold. Udfordringen med motorens moment løses ved at ændre den til bipolær.

### Bipolær motor

Med baggrund i Biot-Savars lov er det udledt at B-feltet i en spole er proportional med antallet af viklinger i spolen, som ses i ligning 6.1. For uddybning af denne udledning henvises til dokumentationen for bipolære motorer.

---

<sup>7</sup>Datablad SAR\_ADC i bilagene

<sup>8</sup>Datablad for 28BYJ-48

<sup>9</sup>Dokumentation Typer af motorer

<sup>10</sup>Dokumentation 28BYJ-48

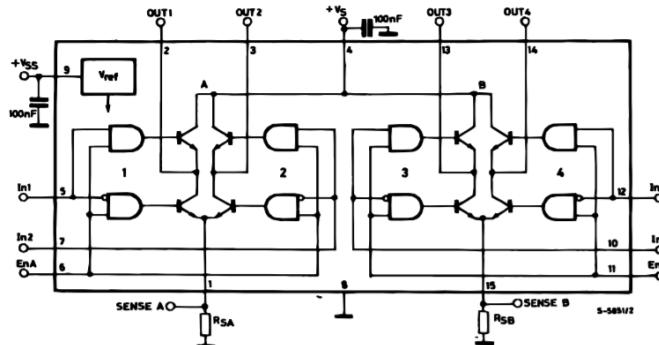
<sup>11</sup>Dokumentation Unipolær motor

<sup>12</sup>Datablad for ULN2003AN

$$B = \frac{\mu_0 \cdot i}{2 \cdot r} \cdot N \quad (6.1)$$

Hvis teorien fra Biot-Savars lov overføres til 28BYJ-48 betyder det, at motoren burde få det dobbelte moment ved, at fjerne den røde lednings forbindelse til resten af motoren således, at motoren får to hele spoler i stedet for fire halve. Motoren burde så have den dobbelte længde spole og dermed antages spolen at have dobbelt så mange viklinger. I tabel 6.4, i kapitlet Test, er det faktiske moment noteret, hvor det ses at momentet blev mere end fordoblet ved at omdanne 28BYJ-48 til bipolær.

Hver bipolær motor styres gennem to H-broer så det er muligt at vende strømmen, og dermed vende retningen på motoren. Det er først gjort gennem chippen L298<sup>13</sup> hvis datablad har været udgangspunkt for to designede og implementerede vero boards som blev udarbejdet. Figur 6.3 viser de to H-broer som L298 indeholder.



Figur 6.3: L298 indeholdende to H-broer

Da boardsene aldrig kom til at fungere blev motor drivere af typen Pololu-A4988<sup>14</sup> taget i brug. Driveren følger samme princip som L298 med to H-broer, men motoren styres gennem et PWM-signal, en enable pin og en direction pin. Se databladet for Pololu-A4988 for yderligere information.

## Software

Programmet for **Positionscontrolleren** er designet på baggrund af systemarkitekturen samt de overvejelser, der er gjort i sektion 6.3. Det er **Positionscontrollerens** opgave at vurdere, om en vinflaske er indsat i WinePrep, og i så fald, om denne er kompatibel med systemet. Den skal da positionere **Åbningsmekanismen** således, at denne låser flasken i en fast position.

<sup>13</sup>Datablad for L298

<sup>14</sup>Datablad for Pololu-A4988

**Positionscontrolleren** fungerer herved også som bindeled mellem **Brugergrænsefladen**, som den modtager kommandoer fra og giver meddelelser om flaskens status tilbage til, og **Åbningscontrolleren**.

Ovennævnte grænseflader, samt de i sektion 6.3 nævnte motorer og sensorer, kan omsættes til 5 grænsefladeklasser:

**DevKit\_SPI**, hvorigennem kommunikationen med **Brugergrænsefladen** foregår.

**PSoC\_SPI**, hvorigennem kommunikationen med **Åbningscontrolleren** foregår.

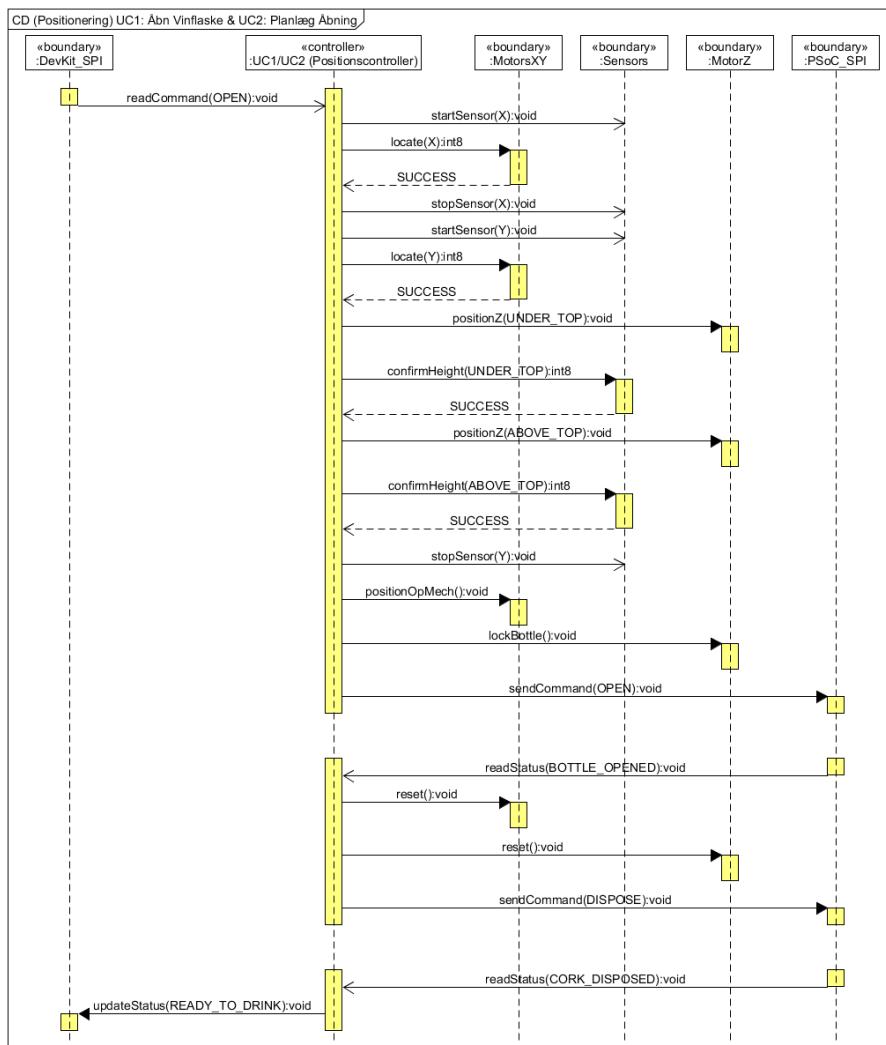
**MotorsXY**, som styrer motorerne for x- og y-akserne, hvis funktionalitet er identisk.

**MotorZ**, som styrer motoren for z-aksen.

**Sensors**, som styrer sensorerne.

### **Sekvensdiagram**

Sekvensdiagrammet i figur 6.4 viser, hvordan der kommunikeres mellem klasserne, og hvilken funktionalitet, disse indeholder.

Figur 6.4: Sekvensdiagram over **Positionering**

Detektering og åbning af vinflasken påbegyndes ved en kommando fra **Brugergrænsefladen** til **Positionscontrolleren**, som efter at have fastslået flaskens x- og y-position checker at flasken har en korrekt højde ved først at løfte sensorerne til en position, der er en vis afstand under flaskens top og få bekræftet, at flasken kan detekteres, og derefter løfte sensorerne til en vis afstand over flaskens top og få bekræftet, at flasken ikke længere kan detekteres. Herefter flyttes åbningsmekanismen til en position over flasken, hvorefter denne fastlåses. Der gives da besked til **Åbningscontrolleren** om at åbne flasken. Efter dette er gjort indstiller **Positionscontrolleren** **Åbningsmekanismen** til dennes starposition, hvorefter der gives besked til **Åbningscontrolleren**.

om at dispensere proppen. Når dette er gjort gives der besked til **Brugergrænsefladen** om, at flasken er drikkeklar.

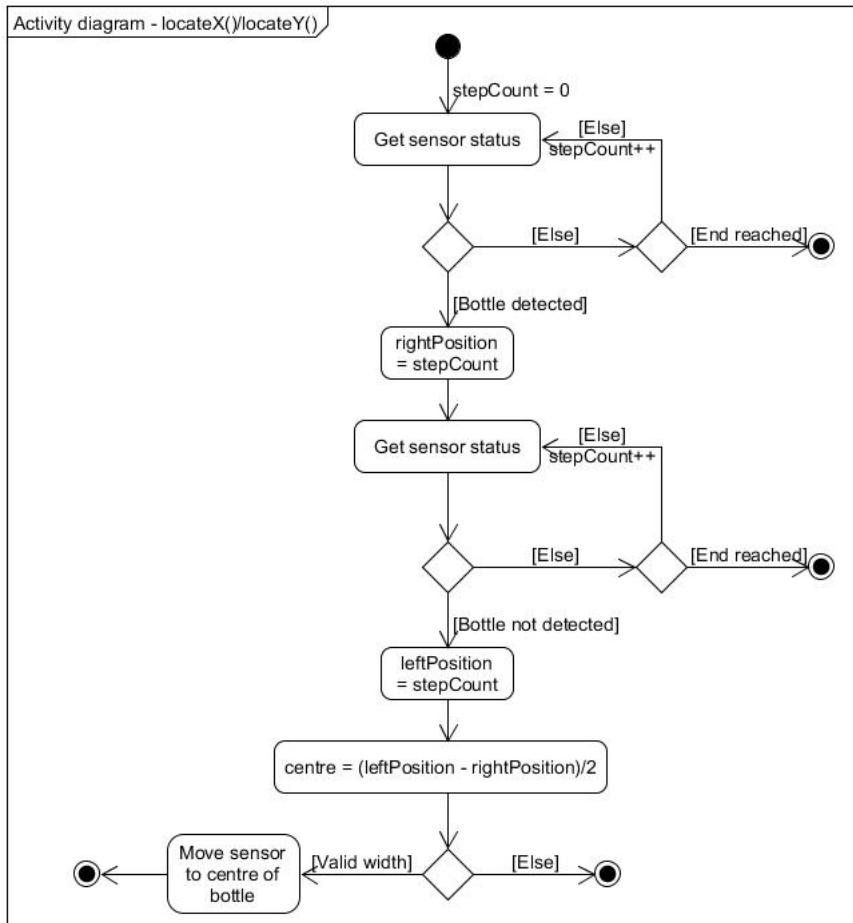
Noget, som ikke er vist i sekvensdiagrammet, er fejlscenarier. Disse findes for de metoder, hvor der i diagrammet returneres *SUCCESS*. I disse tilfælde resettes motorerne, og **Positionscontrolleren** sender en statusbesked til **Brugergrænsefladen** om den pågældende fejl.

#### Funktionsbeskrivelse

Programmets overordnede funktionalitet ikke videre kompliceret, dog kan der for især locate(X)/locate(Y) findes grund til nærmere beskrivelse.

#### locate(Axis)

For nærmere at beskrive denne metode er følgende aktivitetsdiagram blevet udarbejdet:



Figur 6.5: Aktivitetsdiagram over locateX() / locateY()

Ved indtrædelse i metoden nulstilles en tæller `stepCount`, som tæller antallet af steps taget. Derefter måles med sensor, om en flaske er registreret. Så længe dette ikke er tilfældet, skal motoren fortsat køre, og `stepCount` skal tælleres op. I så fald enden på aksen nås, skal metoden afslutte og returnere en fejlværdi. Hvis flasken registreres, gemmes `stepCount` i `rightPosition`. Der fortsættes efter samme mønster, indtil der ikke længere registreres en flaske, eller enden er nået. Hvis førstnævnte er tilfældet, gemmes `stepCount` i `leftPosition`, hvorefter afstanden til flaskens midte beregnes ud fra `rightPosition` og `leftPosition`. Denne værdi sammenlignes med en forudbestemt værdi for at sikre, at det er en kompatibel flaske, der er indsatt. Hvis ikke, afsluttes metoden og returnerer en fejlværdi. Ellers flyttes sensoren til flaskens midte, og metoden returnerer en succesværdi.

### Øvrige metoder

**positionZ(POSITION)** løfter sensorerne et vist antal steps, som er bestemt ud fra det medgivne argument, op fra startpositionen.

**confirmHeight(POSITION)** skal registrere med y-sensoren om en flaske kan registreres i den givne højde alt efter den medgivne parameter (UNDER\_TOP: flaske skal registreres, ABOVE\_TOP: flaske skal ej registreres).

**positionXY()** skal ud fra forudbestemte værdier køre motorerne et vis antal steps mod åbningsmekanismens centrum.

**reset()** kører motorerne indtil deres respektive trykknap ved startpositionen er påtrykt.

## Åbningsmekanisme

### Hardware

#### Iskruning

Ved at teste hvor stor en kraft der skal til for, at skrue proptrækkeren i proppen, kunne en aktuator vælges ud fra testen. Desværre for projektet har det været umuligt at finde ordentligt måleudstyr. Desuden har det ikke været muligt at sammenligne med lignende eksisterende produkter så et design af iskruning er aldrig opnået.

#### Proptrækning

Test af proptrækning med kraftmåler lånt fra Navitas giver et krav til motoren om moment på +25 kg.

### Software

**Åbningscontrolleren** er designet på baggrund af systemarkitekturen og de overvejelser, der hidtil har forløbet sig gennem sektion 6.3.

**Åbningscontrolleren** skal kunne styre en motor, som står for iskruning i proppen, og en motor, som står for at trække proppen ud. Ydermere skal den modtage kommandoer fra og give statusmeddelelser tilbage til **Positionscontrolleren**.

Følgende kan der opstilles 3 grænsefladeklasser for **Åbningscontrolleren**:

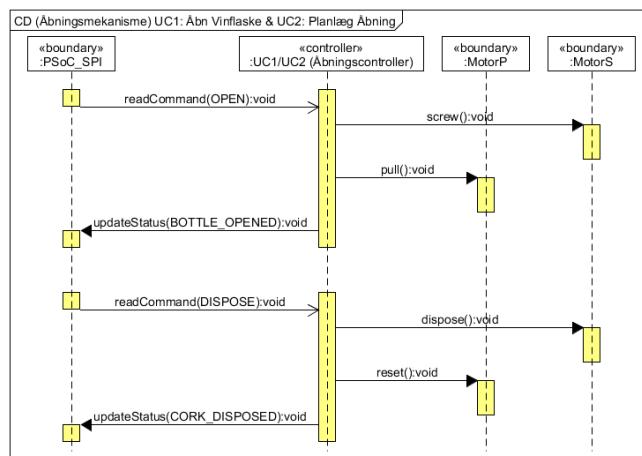
**PSoC\_SPI**, hvorigennem kommunikationen med **Positioneringscontrolleren** foregår.

**MotorS**, som styrer motoren for skruen.

**MotorP**, som styrer motoren for proptrækkeren.

### Sekvensdiagram

Sekvensdiagrammet i figur 6.6 viser, hvordan der kommunikeres mellem klasserne, og hvilken overordnet funktionalitet, disse indeholder. Det skal dog understreges, at, dersom der ikke er et færdigt hardware-design af **Åbningsmekanismen**, er følgende diagram og funktionsbeskrivelser kun et bud på et muligt software-design.



Figur 6.6: Sekvensdiagram over Åbningsmekanisme

Åbningen af vinflasken påbegyndes efter ordre fra **Positionsmekanismen**. Iskruning påbegyndes, og der skrues i proppen indtil skruen har fat i denne, hvorefter proppen trækkes ud. Herefter meddeles om dette til **Positionscontrolleren**, og efter ordre dispenseres proppen, og der meddeles om dette.

### Funktionsbeskrivelse

Metoderne kan kort beskrives:

**screw()** skal sætte skruen til at dreje et forudbestemt antal steps.

**pull()** skal få proptræksmotoren til hive i proppen, indtil en trykknap rammes.

**dispose()** skal blot dreje skruen et vis antal steps for at disponere proppen.

**reset()** kører proptræksmotoren trykknap ved startpositionen er påtrykt.

### Seriell kommunikation

SPI blev anvendt som seriell kommunikation imellem de logiske enheder pga. den kendskab gruppen allerede havde fra HAL øvelse 6 (reference). SPI er en 4 kablet forbindelse, hvor to hardware enheder indgår i et master-slave forhold.

Overførelse af databits initieres af master, og læsning/skrivning foregår samtidigt over de to forbindelser MISO (master in/slave out) og MOSI (master out/slave in). Klok frekvensen på master enheden er forbundet til slave, og det er denne som bestemmer hastigheden for dataoverførelse. Den fjerde og sidste forbindelse er slave select, i og med flere slaver kan tilsluttes samme master, bruges denne forbindelse til at udvælge den rette slave.

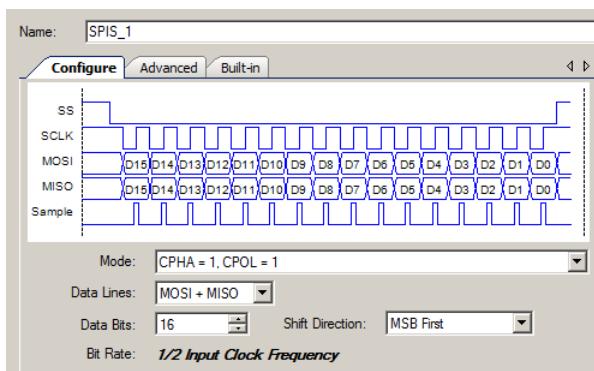
### Devkit8000-PSoC Master

For at kunne kommunikere over SPI fra en Linux platform, skal den rette driver indsættes i Linux kernen. Denne driver skal via det SPI interface som er defineret i biblioteket spi.h, tillade programmer i userspace adgang til den SPI hardware, som er lokaliseret på Devkittet. Da den grafiske brugergrænseflade netop ligger i userspace, er der derfor behov for en SPI device driver, som gør det muligt at skrive og læse data til/fra SPI forbindelsen til SPoC Master.

I driveren skal opsætningen for SPI forbindelsen naturligvis erklæret. Dette indebære bl.a. bus nummer, antal databits, maksimal overførelsels hastighed m.m.

SPI device driveren fra øvelse 6 I HAL blev benyttet som udgangspunkt til at lave en tilpasset driver, som kunne kommunikere med PSoC Master. Denne forbindelse viste sig dog at volde store problemer for gruppen, og det lykkedes ikke at få hverken sendt eller modtaget data med denne driver.

Det blev herefter besluttet, at der ikke skulle bruges mere tid på selv at lave en driver, og i stedet benytte en SPI device driver som var udleveret fra skolen. Dog var det blot den binære fil som var tilgængelig, hvilket betød at der ikke var adgang til source-koden. Dette gjorde, at gruppen ikke kunne tilpasse driveren, og alt information omkring opsætningen for SPI-forbindelse måtte udledes fra det PSoC-Creator program som medfulgte.



Figur 6.7: Opsætning for SPI forbindelse Devkit8000-PSoC Master

Ud fra figur 6.7, kan det aflæses at SPI clock mode er sat til CPHA = 1 og CPOL = 1, og antal databits sat til 16. Det PSoC program som var udleveret blev brugt som skabelon for gruppens eget program til PSoC master, dog med nogle modifikationer. Især håndteringen af de databits som blev modtaget fra devkittet blev genbrugt, da der ikke kunne ændres på disse bitkombinationer. For information omkring implementeringen af håndtering af databits for SPI kommunikation, henvises til afsnittet "Implementering".

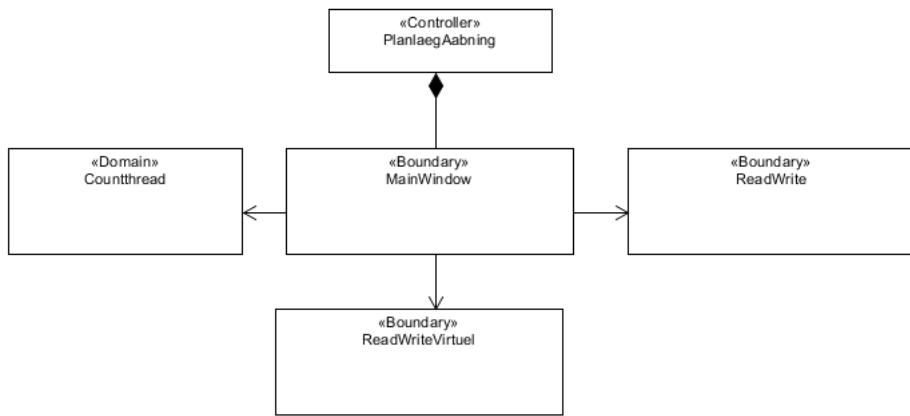
### **PSoC Master - PSoC Slave**

Til SPI forbindelsen mellem PSoC Master og PSoc slave havde gruppen frie hænder til opsætte SPI. Her blev clock mode valgt til CHPA = 0 og CPOL = 0, og antal databits til 8. Grunden til der kun bliver sendt 8 bits her, er at det er tilstrækkeligt til den simple form for kommunikation der er mellem PSoc enhederne. 8 databits ville også have været fint for Devkit-PSoC forbindelsen, men som tidligere nævnt kunne det ikke ændres da der ikke var adgang til SPI device driveren på Devkit8000. Clock mode er ændret til default værdien fra PSoC-Creator, efter inspektion af denne clock mode ligger der intet til grund for at den ikke skulle virker fint.

### **GUI**

Programmet QT blev anvendt til at designe og implementer brugergrænsefladen. Dette gave nogle udfordringer. Udfordringerne bestod primært i at kendskabet til programmet QT ikke var særligt stort. Da QT selv skaber klasserne og metoderne er det svært at beskrive disse på forhånd. Derfor blev det besluttet at klasserne først skulle udarbejdes efter at brugergrænsefladen var designet.

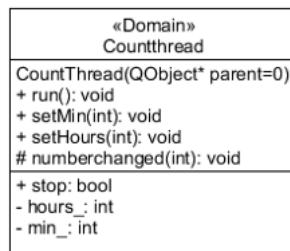
Det endelige klassediagram der blev udarbejdet kan ses illustreret på figur 6.8. Dette klassediagram indeholder 5 forskellige klasser.



Figur 6.8: Klasserne indsættes i et klassediagram

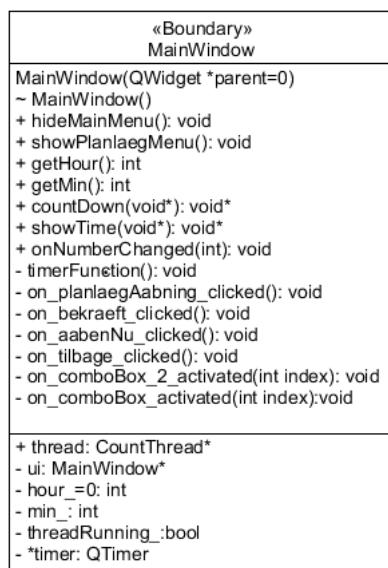
Her kan det ses at der er oprettet to forskellige klasser til read og write funktionerne, som sørger for kommunikationen med PSoC. Der er oprettet en virtuel read og write, og en reel read og write. Den virtuelle er blot oprettet for at gøre det muligt at teste systemet før det endelige system er fuldt udviklet, mens den reelle klasse er oprettet til at bruges når det endelige system udvikles.

For at holde styr på den indtastede og den resterende tid er der blev oprettet en Count klasse . Denne count klasse er implementeret som en domainklasse da det er her den resterende tid for vinåbningen gemmes. Det er denne klasse som skal sørge for at tiden tælles ned når den startes. Illustrationen af count-klassen kan ses på figur 6.9.



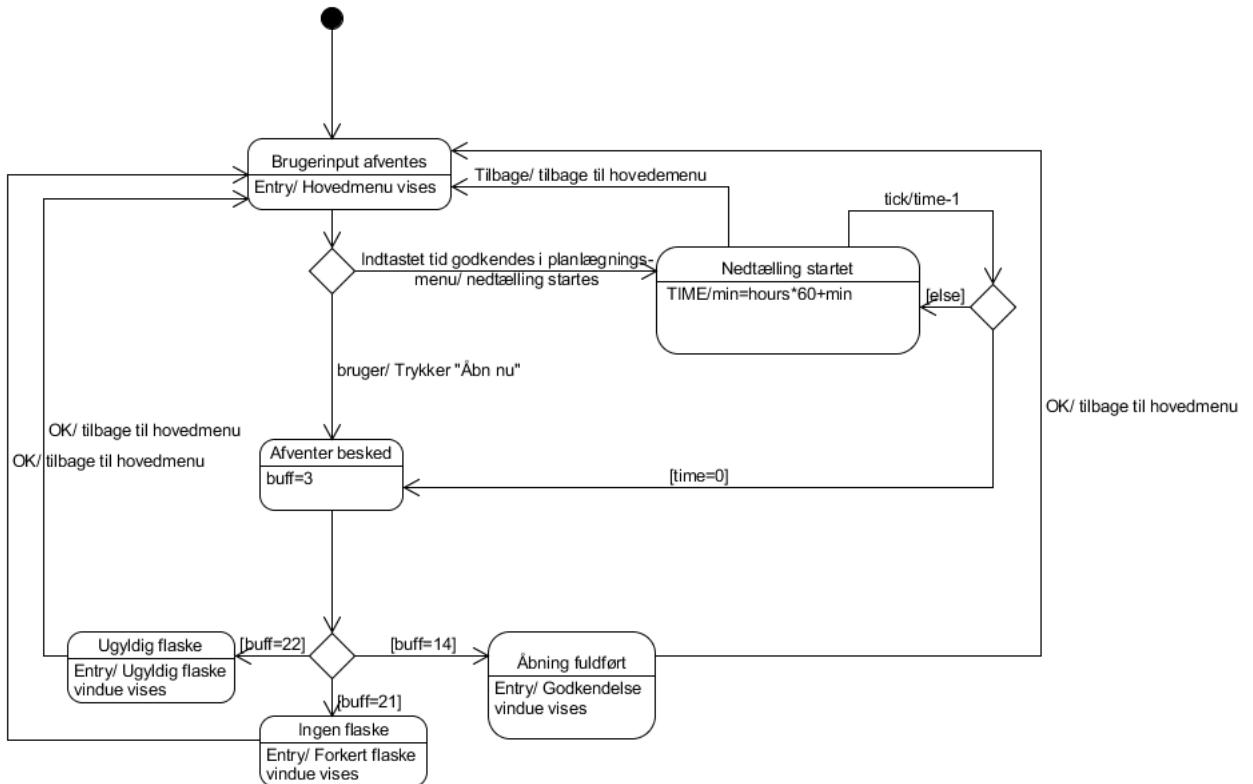
Figur 6.9: CountThread klasse illustreret

Der er i alt 2 klasser i brugergrænsefladen. Der er en klasse for MainWindow, hvor alle funktionerne er defineret. MainWindow er klassen som sørger for at vise den grafiske brugergrænseflade. Det er her alle trykknap funktionerne er defineret. Klassen ses illustreret på figur 6.10.



Figur 6.10: MainWindow klasse illustreret

For at se hele klassediagrammet med alle metoder henvises der til bilag x.  
Brugergrænsefladen er state styret. Derfor har det været nødvendigt at lave et statemachine diagram for brugergrænsefladen. Statemachinen ses på figur 6.11



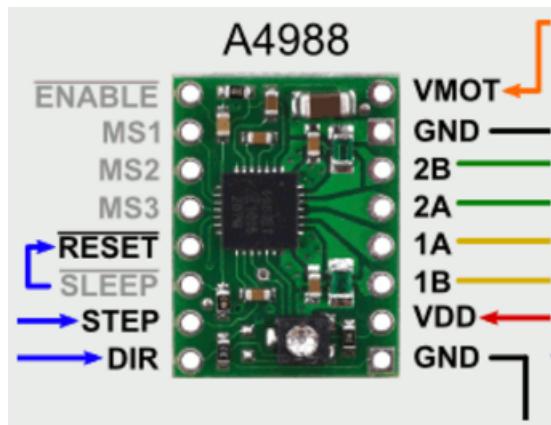
Figur 6.11: statemachine

som illustreret i figuren starter systemet i en state der hedder "Brugerinput afventes". Denne state beskriver systemet når, brugergrænsefladen er i hovedmenuen. Her kan brugeren gøre 2 ting. Enten trykke på "Åbn nu" og få vinen åbnet med det samme, eller trykke på planlæg åbning, indtaste et åbningstidspunkt, for derefter at få systemet til at tælle ned til åbningstidspunktet. Vælger brugeren planlægningsmenuen, kan brugeren selvfølgelig navigere i systemet imens der bliver talt ned. Derfor er der ført en pil tilbage til "Brugerinput afventes". Når timeren sættes i gang, begynder den at tælle et minut ned ad gangen. Når minut og time antallet til sammen går ned på 0 bryder systemet ud af denne state og ind i en ny state der hedder "Afventer besked". I denne state ventes der på en besked fra PSoC'en. PSoC'en kan komme med 3 forskellige beskeder. Den første er at der ingen flaske er i systemet. Dette vil udløse et advarselsvindue på brugergrænsefladen. Den anden besked er at den indsatte flaske er ugyldig, hvilket også vil udløse et advarselsvindue. Den sidste besked man kan få er at flasken godkendes. Alle disse states fører brugeren tilbage til hovedmenuen ved at brugeren trykker OK på vinduet.

## 6.4 Implementering

### Hardware

På figur 6.12 ses hvordan Pololu-A4988 opsættes til PSoC 5LP og en motor. I den endelige implementering er antallet af Pololu-A4988 motor drivere og motorer i forholdet 1:1. Driveren fungerer som bindeledd mellem PSoC 5LP og motoren. Tabel 6.2 viser pin konfigureringen fra Pololu-A4988 til henholdsvis PSoC 5LP og 28BYJ-48. Rækkefølgen af pins i tabellen svarer til billedet fra figur 6.12 læst fra øverste venstre hjørne. MS1, MS2 og MS3 er ikke forbundet til noget. RESET og SLEEP er forbundet til hinanden.



Figur 6.12: Pololu-A4988

15

	PSoC 5LP	Motor	VDD
ENABLE	X		
STEP	X		
DIR	X		
VMOT (10V)			X
GND	X		X
2B		X	
2A		X	
1A		X	
1B		X	
VDD (5V)			X
GND			X

Tabel 6.2: Pololu-A4988 pin konfigurering

<sup>15</sup><https://forum.pololu.com/t/a4988-issue/6518>

VDD er altså spændingsforsyning som leverer både 5V og 10V til henholdsvis driveren og motoren.

Sensorerne har tre pins som alle er sat direkte til PSoC 5LP Master, som vist i tabel 6.3.

<b>GP2Y0A21YK Sensor</b>	<b>PSoC 5LP</b>
$V_0$	P[X:X]
GND	P[X:X]
$V_{CC}$	P[X:X]

Tabel 6.3: GP2Y0A21YK sensor implementeret med PSoC 5LP

## Software

### Positionering/Åbningsmekanisme

Sekvensdiagrammerne i figur 6.4 og 6.6 og de dertil beskrevne klasser er blevet implementeret i form af hver deres headerfil. Disse klasser kunne med fordel være blevet implementeret som structs, hvor relevant information om den pågældende vinflaske kunne gemmes og benyttes, men da der i denne iteration kun kan behandles én type af flasker, er der i stedet benyttet globale konstanter. *main*-funktionen er formet som en state-machine, der vha. en switch kalder de metoder, der skal udføres på et givent tidspunkt i eksekveringen af programmet i henhold til sekvensdiagrammerne. Disse switches' cases bestemmes ud fra de kommandoer/beskeder, de enkelte PSoC's modtager fra hinanden eller DevKit8000. Denne implementering er valgt ud fra principippet om lav kobling og høj samhørighed. Der er ligeledes blevet oprettet en separat *status*-fil, som indeholder kommandoer/beskeder for SPI-kommunikationen, forudbestemte konstanter for bl.a. antallet af steps, der må tælles op for en given motor, og forkortelser, der bruges igennem programmet, for at holde koden overskuelig og øge læsbarheden af denne.

### Hardware-grænseflade

#### Sensorer

Til at måle sensorerne benyttes en SAR-ADC, som, efter hvert sample, returnerer en værdi i *counts*. For at sammenligne med databladet for GP2Y0A21YK konverteres disse værdier til mV. Dermed kan det fastslås, hvorvidt en flaske er registreret eller ej, ud fra afstanden forbundet med den målte spænding.

#### Motorer

Motorerne styres vha. et PWM-signal, som går fra en given GPIO-pen ud til STEP-inputtet på A4988-driveren, som tæller et step op for hver rising edge på PWM-signalet, samt to digitale signaler, der går til henholdsvis ENABLE-

og DIRECTION-inputtene på driveren. PWM-signalen er implementeret med en frekvens på 400 Hz (periode på 2.5 ms), da dette giver motorerne tid til at aflade mellem hvert step<sup>16</sup>.

### **Knapper**

Knapperne er implementeret som interrupts der trigger på rising edge. Heri skiftes værdien af en global variabel.

Adskillige af de metoder, der er beskrevet i design-fasen, er implementeret således, at der sendes et PWM-signal til en given motor, mens der i en while-løkke læses fra en sensor og/eller forefindes et delay, som bruges til at kontrollere antallet af tagne steps. Denne løkke fortsætter i visse tilfælde så længe, der ikke er trykket på en given knap, ved at bruge en global variabel som guard.

For nærmere inspektion af koden se bilag<sup>17</sup>.

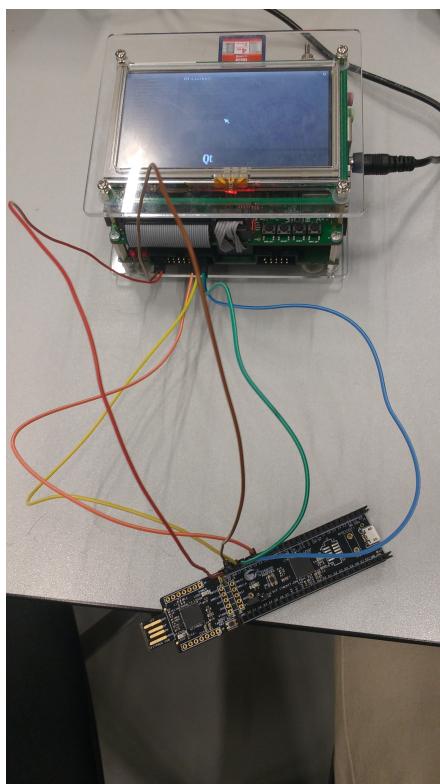
### **Seriel kommunikation**

SPI forbindelsen blev mellem de to enheder blev etableret med 6 ledninger som set på figur 6.13. En til MOSI, MISO, CLK, SS, VCC og GND (reference til SPI). Udgangen for DevKit8000 er valgt på baggrund af datasheet for denne enhed (reference til devkit datasheet). GPIO pins på PSoC er valgfrie, og konfigureres i PSoC-creator til de ønskede værdier.

---

<sup>16</sup>Se bilag

<sup>17</sup>Navn på fil i bilag



Figur 6.13: Den fysiske forbindeelse mellem Devkit800 og PSoC Master

Som nævnt i Design afsnittet for SPI (reference til design), har der ikke været adgang til koden for SPI device driveren på DevKit8000, hvorfor implementeringen af denne ikke kan beskrives nærmere.

Til implementeringen af SPI på PSoC Master og PSoC Slave er der benyttet PSoC creator, som med et drag-and-drop interface gør det nemt at opsætte SPI forbindelsen. Derudover indeholder programmet en main fil hvori der er implementeret en håndtering af de modtagende databits. Koden består dybest set at en interrupt service rutine (ISR), som kaldes hver gang, der er blevet læst en data-byte ind på Rx-bufferen. Den tilhørende interrupt-rutine vil fungere som en state-machine, hvor den pågældende data-byte læses i en switch, som, alt efter kommandoen, sætter en variabel, der læses i PSoC'ens tilhørende *main*-funktion, til en bestemt værdi. I *main*-funktionen skal der da påkaldes de relevante metoder, som skal følge den modtagne kommando. Grunden til denne implementering er, at holde så meget af programmets funktionalitet så opdelt som muligt for at opretholde principippet om høj samhørighed - lav kobling. For mere information omkring koden for PSoC master og PSoC Slave henvises til bilag(navn på bilag).

## GUI

Programmet QT blev valgt da der tidligere har været arbejdet med QT i forbindelse med andre semesterprojekter.

Sproget som brugergrænsefladen er skrevet i er C++ da det er dette sprog som teamet har haft størst erfaring med.

For at kommunikere med SPI driveren som ligger på Devkit8000 er funktionerne fread() og fwrite() brugt.

PSOC er tidligere i koden blevet defineret som path'en på PSoC driven. I koden for read og write funktioner kan man se at OPEN\_BOTTLE, skrives til PSoC driveren ved hjælp af fwrite(). OPEN\_BOTTLE er tidligere blevet defineret som 5, hvilket i den anvendte protokol betyder at vinen skal åbnes. Det er samme kode der bruges til funktionen "Planlæg åbning".

For læsning fra PSoC'en er funktionen fread() benyttet. Da der kan gå en del tid inden systemet får noget tilbage fra PSoC'en foregår læsningen i en tom for lykke som der kun kan brydes ud af hvis en af de 3 tideligere definerede svar modtages. Til visning af besked på brugergrænsefladen er QT's MessageBox klasse benyttet.

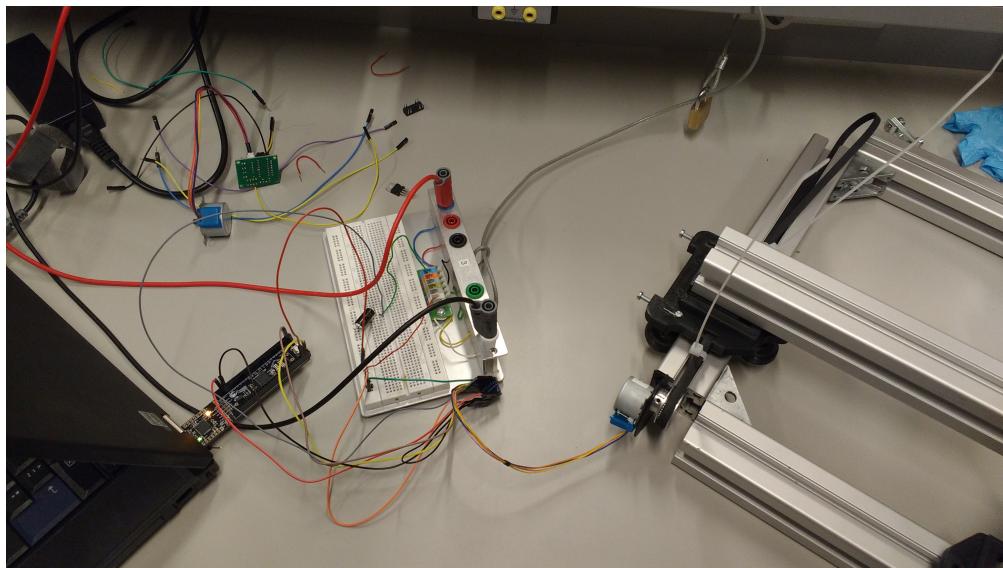
For at få tiden talt ned og samtidigt displayet på skærmen er der blevet benyttet threads. Implementeringen af dette kan ses i dokumentationsbilaget x.

## 6.5 Test

### Motorer og sensorer

Test af motorer og sensorer er foretaget med både uni- og bipolare motorer som er foregået efter samme metode, hvor komponenterne er testet enkeltvis og i moduler. Der er ikke foretaget modultest af motorer for iskruning af proptrækker eller proptræk fordi åbningsmekanismen aldrig blev færdig. Der er derfor kun foretaget modultest af akserne, dog i 2 omgange, hvor unipolare motorer senere blev udskiftet med bipolare.

Figur 6.14 viser test opstillingen hvor 28BYJ-48 testes på rammens akse.



Figur 6.14: Test af 28BYJ-48 på rammens akse

Der er bygget et stativ til at teste forskellen i moment for uni- og bipolær som ses på figur 6.15.



Sensorerne blev testet ved at lade forskellige materialer blive detekteret på afstand af varierende størrelser.

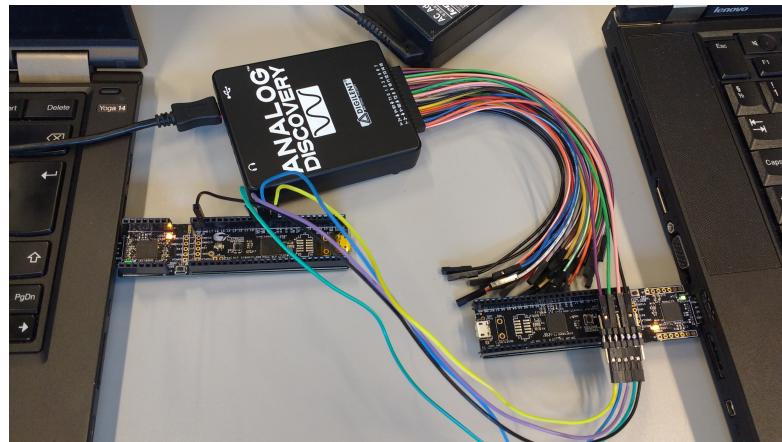
## Seriel kommunikation

### DevKit8000-PSoC Master

Til test af denne forbindelsen blev der sendt data fra DevKit8000, ved at skrive ud til den fil i /dev som var forbundet til SPI hardwaren. Herefter blev der med Logic analyser målt MOSI (udgangen på DevKit8000), CLK(klokken), SS(slave select). Der blev kigget på om de rigtige databits blev skrevet ud til MOSI, om SS gik lav ved dataoverførelse, og om CLK havde den rigtige clockmode. Efterfølgende blev der også læst fra den samme fil i /dev, og der blev målt på MISO(Indgangen på DevKit8000).

### PSoC Master - PSoC Slave

Til test af PSoC-PSoC forbindelsen, blev der tilføjet et UART modul i PSoC creator. Dette gjorde det muligt at skrive de resultater som blev send til PSoC enhederne til en terminal på en PC, hvor disse nemt kunne aflæses. Der blev også målt med Logic Analyzer for at teste om de fire forbindelser MISO, MOSI, CLK og SS og rigtig ud ligesom i testen mellem DevKit8000 og PSoC Master. På figur 6.16 ses testopstilling for PSoC-PSoC

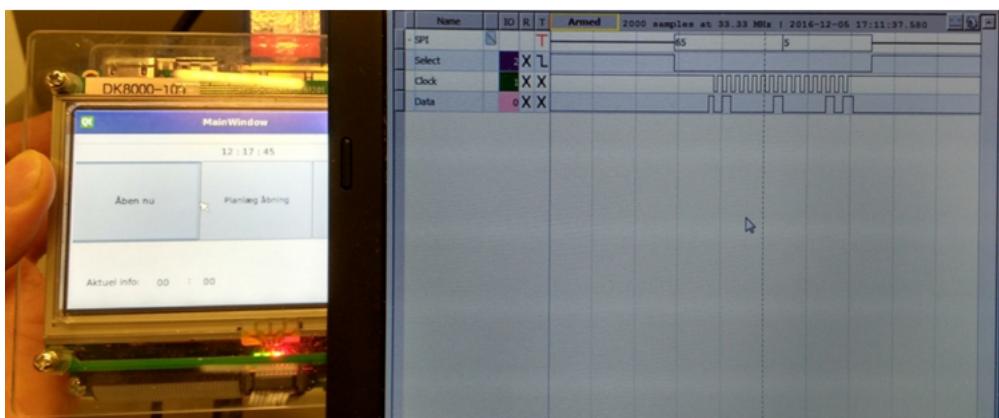


Figur 6.16: testopstilling for PSoC Master/PSoC Slave forbindelsen

## GUI

Det vigtigste der skulle testes ved brugergrænsefladen var at den kunne sende en hvilken som helst kommando ved hjælp af SPI driveren. Dette blev te-

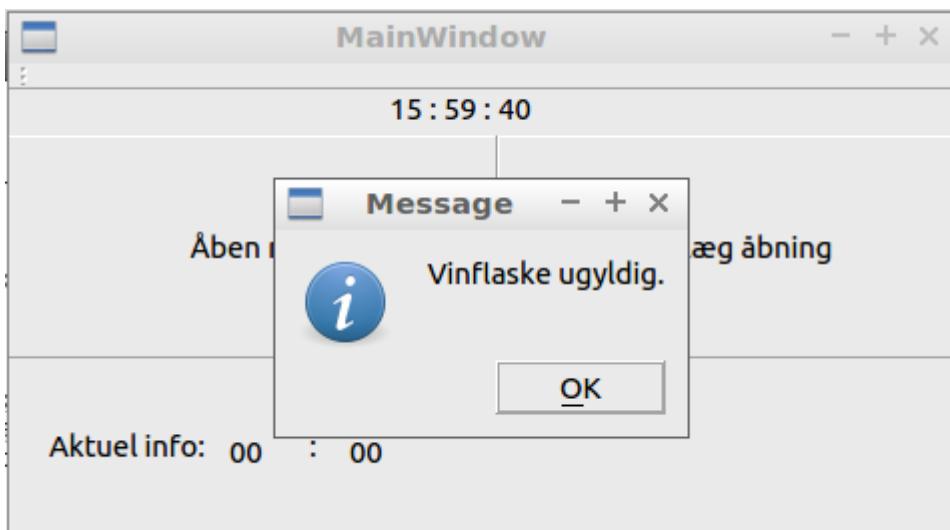
stet ved at forbinde Analog Discovery til Devkit8000's SPI ben. Derefter blev funktionen Logic Analyzer benyttet til at måle på outputtet. Det var vigtigt at vide hvornår kommandoen blev sendt ud. Da touchfunktionen ikke har været optimal på Devkit8000 blev funktion "Planlæg åbning" brugt til at teste hvad outputtet fra Devkit8000 var efter nedtællingen. Grunden til at funktionen "Åbn nu" ikke blev brugt, var fordi at man skulle trykke mange gange på touchskærmen for at Devkittet ville reagere. Dette bragte en uønsket usikkerhed i testen. Derfor var det mere hensigtsmæssigt at teste med funktionen "Planlæg åbning" da man her kan se hvornår tiden udløber, og dermed hvornår der bør sendes en kommando ud. På figur 6.18, kan det ses hvordan, det var muligt at sende kommandoen 5 ved at bruge "Planlæg åbning" funktionen.



Figur 6.17: Åben nu funktionen implementerer

### Test med virtuel klasse

Der var problemer med modtagelse af kommandoer fra SPI. For at teste om GUI var korrekt implementeret, blev der lavet en virtuel klasse som fungerede som en stub. Denne klasse skulle simulere skrivning/læsning af data til/fra SPI device driveren. Istedet for at læse fra et SPI device i /dev, blev der læst fra en txt fil istedet. Da devices bliver behandlet som filer i Linux, ville functionalitet i den virtuelle klasse ligge meget tæt på den egenlige implementering. Dermed kunne det testes om GUI kunne give status beskeder tilbage til brugeren, selvom SPI forbindelsen ikke var tilgængelig. På figur xx ses hvordan GUI meddeler bruger statusbeskeder på baggrund af den kommando som blev indlæst fra txt filen.



Figur 6.18: Test af virtuel klasse

## 6.6 Resultater

### Motorer og sensorer

Ved testen som set på figur 6.14 formåede den bipolære motor at drive aksen med små ophold.

Tabel 6.4 viser forskellen i moment for henholdsvis uni- og bipolær motor. Den bipolære motors moment er 118% mere end den unipolæres. Resultaterne er udregnet ud fra testen med stativet fra figur 6.15. Se dokumentationen for test af bipolære motorer i bilagene.

28BYJ-48	Unipolær	Bipolær
Moment i full-step mode	363 gcm	792 gcm

Tabel 6.4: Moment for 28BYJ-48<sup>18</sup>

Figur 6.19 viser detektering af et objekt ved 10 cm afstand. Værdien 2241 er i mV. Figur 4 i databladet for sensoren viser ca. 2300 mV ved samme afstand hvilket giver en afvigelse på 2,57% eller en unøjagtighed på 2,57 mm.

<sup>18</sup>Dokumentation for test af motorer

Locals				
Name	Value	Address	Type	Radix
ADCResultX	2241		unsigned long	decimal

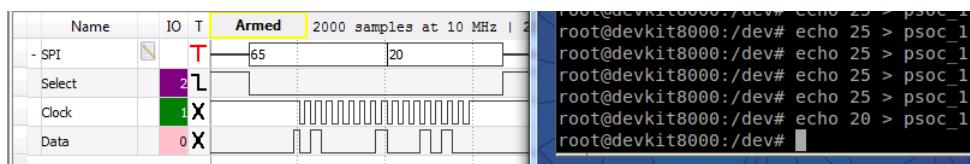
Figur 6.19: Resultat af detektering fra sensor ved 10 cm

## Seriel kommunikation

De dette afsnit er der kun medtaget de mest nødvendige test, for alle øvrige tests henvises til dokumentation (reference til test under seriel kommunikation.

### DevKit8000-PSoC Master

På figur 6.20 ses resulater på Logic Analyzer når der skrives til PSoC master fra terminalen på Devkit8000.

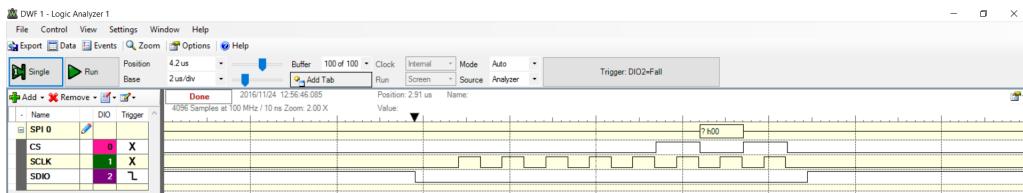


Figur 6.20: Udskrift fra Logic Analyzer fra test af SPI forbindelse mellem Devkit og PSoC Master

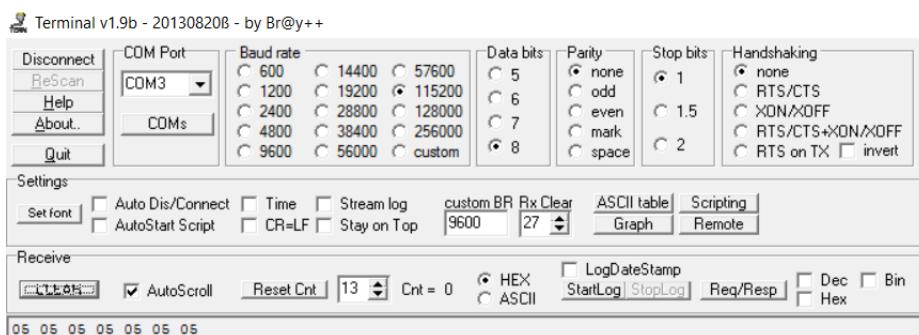
Resultater for læsningen af PSoC Master gav kun succesfuldt resultater ved en enkelt test. Derudover kunne der ikke læses fra PSoC master. Hvilket betyder at der ikke er dokumentet en succesfuld aflæsning af MISO forbindelsen. Selv efter mange forsøg, og med hjælp fra undervisere i HAL, blev dette problem aldrig løst.

### PSoC Master - PSoC Slave

På figur 6.22 ses udskrift Logic Analyzer og PC Terminal. Der er i denne test blevet skrevet værdien 5 til PSoC slave. Billeder for test af MISO er udeladt, men gav dog samme resultat.



Figur 6.21: Udskrift fra Logic Analyzer fra test af SPI forbindelse mellem PSoC Slave og PSoC Master



Figur 6.22: Udskrift fra terminal fra test af SPI forbindelse mellem PSoC Slave og PSoC Master

## 6.7 Diskussion af resultater

### Diskussion af resultater

#### Motorer og sensorer

Den interessante test for motorer er den der viser forskellen i moment for uni- og bipolær motor. Ifølge Biot-Savarts lov burde den bipolære motor have forøget momentet med to gange, men resultatet viser sig at være betydeligt bedre. Det er svært at argumentere for om der er en eller flere årsager til resultatet, og hvad grunden til resultatet skyldes. Det kan dog konkluderes at der er flere faktorer der spiller ind ift. motorenens moment, som ikke blev taget med i overvejelserne fordi databladet for 28BYJ-48 er mangelfuld<sup>19</sup>. Overvejelser omkring faktorerne kan ses i dokumentationen for bipolære motorer.

<sup>19</sup>Datablad 28BYJ-48

## Seriel kommunikation

### DevKit8000-PSoC Master

Som det ses på figur 6.21, blev der succesfuldt sendt de korrekte databits ud på SPI MOSI, SS gik også lav ved dataoverførelse hvilket er meget vigtigt for at sikre, at den korrekte slave i vores tilfælde "PSoC Master" modtager kommandoerne. CLK ser rigtig ud med den clock mode som er specificeret. Dette ses ved at CLK starter høj og skifter bits på nedadgående flanke, og læser på opadgående flanke. Dette svarer til CPHA = 1 og CPOL = 1, hvilket også var den opsænninger der er lavet for denne SPI forbindelse.

Problemer med MISO forbindelsen betød at projektets prototype ikke kunne sende status beskeder til brugerden, hvilket naturligvis var en stor skuffelse for gruppen. Problemet ligger højst sandsynlig på selve PSoC, da der ikke læses noget data på MISO forbindelsen, og det derfor ikke har noget med DevKit8000 af gøre. I test af GUI blev dette problem dog løst med en virtuel klasse, som fungerede som en stub og dermed simulerede SPI forbindelsen.

### PSoC Master - PSoC Slave

Her ses igen er det er de rigtige databits som bliver sendt, og SS går ligeledes lav som den burde. Clock mode passer også fint med CPHA = 0 og CPOL = 0, som den var sat til. CLK starter lav som den skal, og skifter på nedadgående flanke og læser på opadgående. Dette er helt som forventet. Og som det ses på figur 6.22, så udskrives de korrekte bits til terminalen på PC, hvilket igen betyder at PSoC Slave modtager de korrekte bits, og derfor burde denne forbindelse virke fint.

## GUI

Der er mange funktioner i brugergrænsefladen som til at starte med var tiltænkt, som ikke er blevet implementeret. Dette skyldes hovedsageligt 2 ting. Den første er at teamet ikke har haft den nødvendige erfaring til at kunne estimere et projekts omfang. Der var rigtig mange ting som blev planlagt som aldrig blev udført på grund af mangel på tid. Den anden store grund til at alle funktioner ikke kom med var at gruppen blev nedskåret til en 4 personers gruppe frem for en 8 personers grupper som projektet oprindeligt var tiltænkt for. Derfor er det naturligt at gruppen ikke kan nå lige så meget som en gruppe på 8 personer.

De tests som blev udført på brugergrænsefladen var yderst succesfulde, da det ønskede resultat blev opnået. Under testen blev der forsøgt at sende kommandoen 5 ud igennem SPI, og dette lykkedes som det også fremgår af afsnittet Test. Det var dog tænkt, og i første omgang implementeret således at brugergrænsefladen kan meddele brugerden meddelelse. Dette skulle ske, ved

at den fik respons fra PSoC'en, og alt efter hvilken respons den fik, ville den frembringe en dialogboks. Dog virkede dette ikke da SPI driveren var ustabil, og ikke ville læse. Istedet blev testen udførst med en virtuel klasse, som kunne simulere modtagelsen af data fra SPI driveren. Denne test var succesfuld, og derfor kan det konkluderes at GUI implementeringen for statusbeskeder virker som forventet.