

# 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

# Indhold

<b>Indhold</b>	<b>i</b>
<b>1 Forord</b>	<b>1</b>
1.1 Læsevejledning . . . . .	1
<b>2 Indledning</b>	<b>2</b>
2.1 Projektformulering . . . . .	2
2.2 Det realistiske system . . . . .	3
2.3 Hovedansvarsområder . . . . .	3
<b>3 Krav</b>	<b>4</b>
3.1 Aktører . . . . .	4
3.2 Use-cases . . . . .	5
3.3 Ikke-funktionelle krav . . . . .	5
<b>4 Afgrænsning</b>	<b>7</b>
<b>5 Realisering</b>	<b>9</b>
5.1 Metode . . . . .	9
5.2 Analyse . . . . .	10
5.3 Systemarkitektur . . . . .	12
5.4 Design . . . . .	20
5.5 Implementering . . . . .	25
5.6 Test . . . . .	28
5.7 Resultater . . . . .	30
5.8 Diskussion af resultater . . . . .	33

# 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. 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.

## Kapitel 2

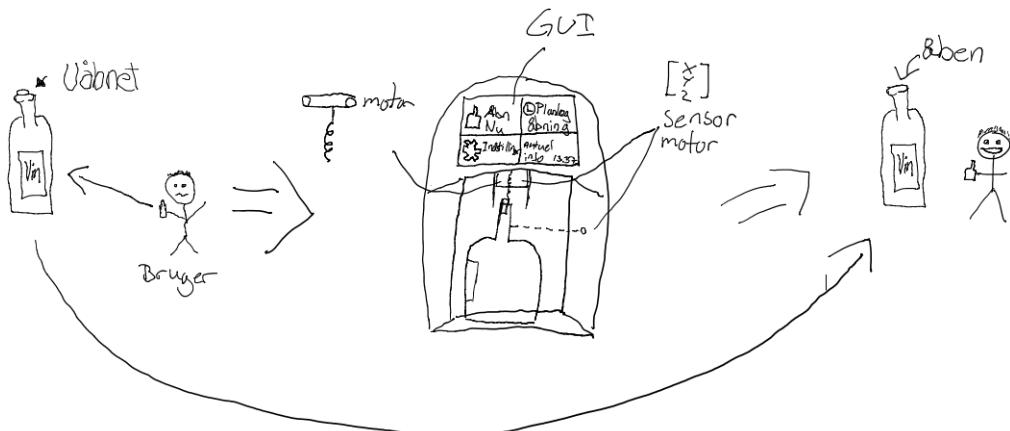
# Indledning

### 2.1 Projektformulering

Mange ældre har i dag svært ved at åbne deres vinflaske, da de ikke har den fornødne styrke til selv at trække korkproppen ud af vinflasken. Derfor vil det være ideelt for dem, at have en løsning hvor åbningen af vinflaskerne bliver automatiseret.

For at få den optimale oplevelse ud af en vin, skal den åbnes rettidigt så den iltes før indtagelse. Iltningstiden kan variere fra vin til vin, og derfor kan mange uerfarne vindrakkere have svært ved at ilte deres vin korrekt. Mange glemmer at åbne vinen i god tid, og opnår derfor ikke den optimale oplevelse. Det kan derfor være ideelt, hvis denne proces også automatiseres.

Figur 2.1: Rigt billede der beskriver WinePrep



## 2.2 Det realistiske system

WinePrep er den automatiske vinåbner som er illustreret på Figur 2.1, hvilket beskriver det realistiske system i en tænkt situation. Der er siden udarbejdelsen af det rige billede ikke ændret ved tanken bag systemets funktionalitet, blot andre måder at implementere ideerne på.

Den oprindelige tanke med WinePrep gør det for brugeren muligt at åbne en bestemt type vinflaske ved at indsætte vinen i maskinen, konfigurere WinePrep til at åbne og derefter først lade systemet lokalisere flasken hvorefter en åbningsmekanisme sænker sig over vinen og trækker korkproppen op. Dette realiseres med WinePreps ramme, brugergrænseflade, sensorer, aktuatorer og proptrækker samt microcontrollere til at lade systemet kommunikere internt.

På baggrund af de tekniske komponenter og WinePreps kompleksitet, vil der til udviklere være krav om forhåndskendskab til elektronik og programmering, på et plan der gør det muligt at forstå og bruge de oplysninger der findes i bilagene til denne rapport.

WinePrep er en prototype der er mulig at udvikle og optimere på.

## 2.3 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 kernekompentence 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

# Kapitel 3

## Krav

I henhold til projektets mål er der med udgangspunkt i FURPS+(indsæt reference til beskrivelse af FURPS+) og MoSCoW(indsæt reference til beskrivelse af MoSCoW) blevet opstillet en række krav for WinePrep. De funktionelle krav er beskrevet ved tre use-cases, hvoraf 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.

### 3.1 Aktører

Der er 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).

## 3.2 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 trykskærmen. 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 trykskærmen 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/undtaqelser for UC1).

### Planlæg åbning

Brugeren skal på trykskærmen trykke på knappen "Planlæg åbning" og herefter på to scrolldown-menuer(reference til ordliste) vælge et klokkeslæt, hvor vinflasken ønskes åbnet, og vinen drikkeklaar(reference til ordliste). Systemet venter da til iltningstidspunktet(reference til ordliste), hvor brugeren forinden skal have indsat vinflasken i WinePrep, hvorpå det påbegynder prceduren beskrevet i "Åbn vinflaske" ovenfor. Trykskærmen vil herefter vise det tidspunkt, hvor vinen vil være drikkeklaar. Kan det ønskede klokkeslæt, hvor vinen skal være drikkeklaar, ikke forenes med iltningstidspunktet(reference til detaljer om iltningstidspunkt), annulleres processen, hvorefter brugeren vil blive tilbudt muligheden for at få vinflasken åbnet øjeblikkeligt.

## 3.3 Ikke-funktionelle krav

WinePrep skal have en let betjenelig 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 øjeblikkeligt og løbende holde brugeren opdateret om vinflaskens status(reference til bilag: ikke-funktionelle krav/brugervenlighed + /ydeevne).

Denne vinflaske 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 4

# Afgrænsning

Det er et krav, at projektet skal indeholde en linux-platform, en PSoC, en aktuator og/eller sensor. Gruppen har derfor fokuseret på de usecases, som indeholdte alle disse elementer, "Åbn Vinflaske" og "Planlæg Åbning". I de indledende faser af projektet blev det drøftet, hvorvidt systemet skulle måle temperaturen af vinen, have en mobil-applikationen og tilknyttes en database via internettet. Gruppen blev dog enig om, at dette lå uden for læringsmålene for projektet og var for tidskrævende til at kunne implementeres indenfor tidsrammen. Gruppen holdte dog muligheden åben for, at disse krav kunne komme på tale, hvis der var tid og overskud sidst i projektet.

Der blev også opstillet nogle krav for et ideelt produkt, dog med den klare opfattelse af, at disse ikke var mulige for gruppen at gennemføre. Disse krav indbefattede bl.a. regulering af vinens temperatur, online vinbestilling, og genkendelse af vintype ud fra et billede af vinetiketten.

Målet for dette projekt blev at lave en prototype med en grafisk brugergrænseflade, der via kommunikation med PSoC gjorde det muligt at styre positionerings- og åbningsmekanismer til åbning af en vinflaske. Denne prototype ville indeholde motorer, sensorer, linux-platform samt PSoC-enheder og dermed opfylde minimumskravene til projektet.

Det var vigtigt for gruppen at fokusere på, hvad der var nødvendigt for at opfylde IHA's krav, og ikke hvad der kunne gøre prototypen mere imponerende eller innoverende. Ud fra den ønskede prototype blev der opstillet en riskmodel for både software og hardware, der skulle klarlægge hvilke områder, gruppen skulle fokusere på.

Fokuspunkterne for projektet blev styring af motorer og sensorer til positionerings- og åbningsmekanisme, kommunikation mellem linux-platform og PSoC-enheder, samt brugergrænsefladen. Konstruktionen af de fysiske rammer for systemet

blev ikke prioriteret særligt højt, da dette ligger udenfor de faglige mål for projektet.

## Kapitel 5

# Realisering

### 5.1 Metode

Da første review nærmede sig, blev der startet på **UML-** og **SysML-diagrammer**. En foreløbig systemarkitektur blev udarbejdet, således at teamet havde samme udgangspunkt i det videre forløb. Det var her nødvendigt at definere nogle krav til projektet. Her blev **usecases** benyttet til at definere de funktionelle krav, mens **FURPS+** og **MoSCoW** blev benyttet til at definere de ikke-funktionelle krav. Usecasene blev udviklet ud fra et systemniveau, hvilket skulle vise sig at give udfordringer senere i forløbet. Yderligere blev der udarbejdet **systemsekvensdiagrammer** som skulle vise hvorledes systemet interagere. Da gruppen består af hardware og software specialister var dette en nødvendighed, således at alle havde en fælles forståelse for produktet.

Der opstod flere udfordringer da der skulle udarbejdes en **domænemodel** til produktet. Domænemodellen bør udarbejdes med udgangspunkt i usecasene, og da usecasene var lavet på systemniveau, gav det ikke et særlig godt udgangspunkt for en domænemodel. Da der allerede i startfasen var researchet en del omkring produktet, og for hvilke muligheder der var for produceringen af produktet, blev det besluttet at domænemodellen ikke var nødvendig. Derfor blev den udarbejdede domænemodel også udeladt i projektet.

For at definere hvilket software der skulle allokeres hvor, blev der lavet et **softwareallokeringsdiagram**. Denne blev brugt til at skabe bro mellem hardware og software.

Hardwaren blev beskrevet med **BDD'er** og **IBD'er**. 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 konstrueret **klasse- og sekvensdiagrammer**. Klassediagrammerne skulle vise hvilke klasser systemet består af, mens sekvensdiagrammerne skal vise hvilke metoder der skal

være i hvilke klasser. Det var dog problematisk at skulle producere klasse og sekvensdiagrammer for brugergrænsefladen, da brugergrænsefladen blev udarbejdet i programmet QT. QT opretter egne metoder, og da der ikke var opnået nok erfaring med QT, til at kunne bestemme hvilke metoder der skulle bruges, blev det besluttet at dette først skulle gøres efter, at brugergrænsefladen var færdiglavet.

## 5.2 Analyse

### Hardware

#### Motorvalg

Tre typer af motorer, DC-, stepper- (DC) og servo motor, har været overvejet til forskellige funktioner i projektet. Krav til de forskellige motorer blev indelt i 3 overordnede emner: præcision (til positionering), hastighed (rpm) og moment (torque).

Præcision på akserne er altafgørende og her er stepper motoren de andre overlegne. Der blev desuden ikke defineret et krav for hvor hurtigt vinflasken skulle åbnes, så hastigheden er af den grund blevet nedprioriteret.

#### x-, y- og z-aksen samt iskruning af proptrækker

På baggrund af viden om forskellige typer af motorer, se evt. bilag xx, blev stepper motorer af typen 28BYJ-48 valgt pga. dens nøjagtighed indenfor positionsgenkendelse. Det var nødvendigt for at kunne åbne vinen at have koordinater, der lå indenfor en milimeters nøjagtighed, og det kunne opnås med motorens mange steps per rotation. I 4-step mode har motoren en vinkel på 11,25(grader) per step, som betyder 32 steps per rotation internt i motoren. Med en gearing på 1:64 giver det 2048 steps per rotation for motorens skaft, hvilket giver meget nøjagtige koordinater. Motoren er lille i sin fysiske størrelse og var derfor også nem at implementere i rammen for WinePrep, hvilket gjorde den yderligere attraktiv.

Der blev forsøgt lavet målinger på kraften, der skulle til for at skrue proptrækkeren i. Disse målinger viste sig dog at være meget upræcise, og de blev derfor vurderet ubrugelige for beslutningsprocessen. Uden nærmere indsigt i krav til moment for motoren for iskruning af proptrækkeren blev 28BYJ-48 også valgt til denne opgave.

#### Proptræk

Målinger af proptrækket blev foretaget med en kraftmåler, som kunne måle op

til 20 kg. Under forsøgene blev det bevist, at proptrækket kræver større kraft, end hvad kraftmåleren kunne måle, og det blev herefter besluttet at anskaffe en motor med tilstrækkeligt moment (se bilag xx). Valget faldt på stepper motor af typen NEMA17 der ifølge databladet kan trække med en kraft på 48 kg.

En DC- eller servomotor kunne lige så vel have udført arbejdet, men det var NEMA17 der var til rådighed på værkstedet på ASE.

## Sensorvalg

Ud fra en betragtning om præcision, som var et krav af høj betydning, var det underordnet om en afstandsmåler af typen lys eller ultralyd blev valgt, da præcisionen stadig ville være for unøjagtig med de komponenter der kunne anskaffes. Det essentielle for sensoren var at den detekterede om der var en genstand i WinePrep. Der var to muligheder at vælge imellem i Embedded Stock og lasersensoren, SHARP GP2Y0A21YK, vandt over en ultralydssensor, pga. dens større præcision.

## Software

### Motor- og sensorstyring

Til styring af de motorer og sensorer, der indgår i systemet, blev det besluttet at bruge PSoC's, dels fordi dette var et krav til gennemførelse af projektet, men også fordi disse med PSoC-Creator tilbydød et IDE, som gjorde det let at designe det kredsløb, der udvikledes software til, via et drag-and-drop-interface.

Der benyttes to PSoC's til styring af motorer/sensorer hovedsageligt grundet et tidligere design, hvor der brugtes look-up-tables (LUT) til at skifte mellem de forskellige step-tilstande. Disse gjorde det vanskeligt at samle al funktionaliteten på en enkelt PSoC, da de optog for mange UDB's(reference til <http://www.cypress.com/file/139386/download> eller ordliste). Efter de oprindelige prints blev skiftet ud med A4988-drivers, blev disse LUT's overflødige, og softwaren kunne principielt samles på én PSoC. Det blev alligevel besluttet at benytte to, da antallet af trykknapper krævede en port afsat til hvert interrupt triggered af disse. På en enkelt PSoC er der med 6 porte til rådighed ikke nok. Visse af disse knapper kunne udskiftes med en counter i programmet, som talte antallet af steps for en given motor, men ønsket om en præcis positionering har ført til bibeholdelsen af knapperne.

### SPI

Til kommunikation mellem systemet CPU'er skulle der benyttes en serial protokol til afsendelse og modtagelse af databits. Både Devkit8000 og PSoC understøtter UART, I2C og SPI. Gruppen tænkte i første omgang på at anvende

UART, da kendskaben til denne protokol var god. Det viste sig dog at UART porten på Devkit8000 bruges af anden hardware, og derfor stod gruppen tilbage med enten I2C eller SPI. Der blev under flere laboratorie øvelser på 3 semester anvendt SPI, og derfor var der allerede en SPI linux device driver tilgængelig. Det blev derfor besluttet at systemet skulle anvende SPI til Devkit8000-PSoC forbindelsen. I2C blev holdt åben som en mulighed til PSoC-PSoC forbindelsen i tilfældet af at der skulle kobles flere PSoC enheder på hinanden. I2C tillader nemlig flere enheder at være sammenkoblet med relativt få forbindelser, hvorimod SPI kræver en ny forbindelse for hver ny enhed der tilkobles. Systemet endte dog med kun at benytte to PSoC enheder, og da kendskaben til SPI var bedre, blev SPI også brugt til PSoC-PSoC forbindelsen.

## GUI

Da brugergrænsefladen skulle designes blev der gjort mange overvejelser. Først og fremmest skulle der selvfølgelig researches omkring programmet QT hvorpå brugergrænsefladen skulle designes. For at få en fornemmelse af hvor følsom og hvor præcis touchfunktionen på Devkit8000 var skulle den selvfølgelig testes. Den blev kalibreret og derefter testet. Det viste sig at præcisionen var meget begrænset på touchskærmen. Derfor blev det besluttet at det var nødvendigt at bruge store knapper til at navigere på brugergrænsefladen. Det indledende design for brugergrænsefladen kan ses på figur x i bilag x. Her ses det at hvordan knapperne er blevet designet således at de fylder hele skærmen.

For at skifte menu er QT funktionerne show() og hide() benyttet. Dette kunne have været gjort på flere måder, men da der ikke er mange knapper i designet er denne metode blevet vurderet til at være den mest hensigtsmæssige.

Til at starte med viste ”Aktuel info:” hvilket tidspunkt på dagen vinien stod til at blive åbnet, men da der ikke er et batteri indsatt Devkit8000 kan RTC ikke benyttes, da den vil resettes, hver gang Devkitted genstartes. Derfor blev det besluttet at ”Aktuel info:” skulle vise den resterende tid som var for åbningen af vinien.

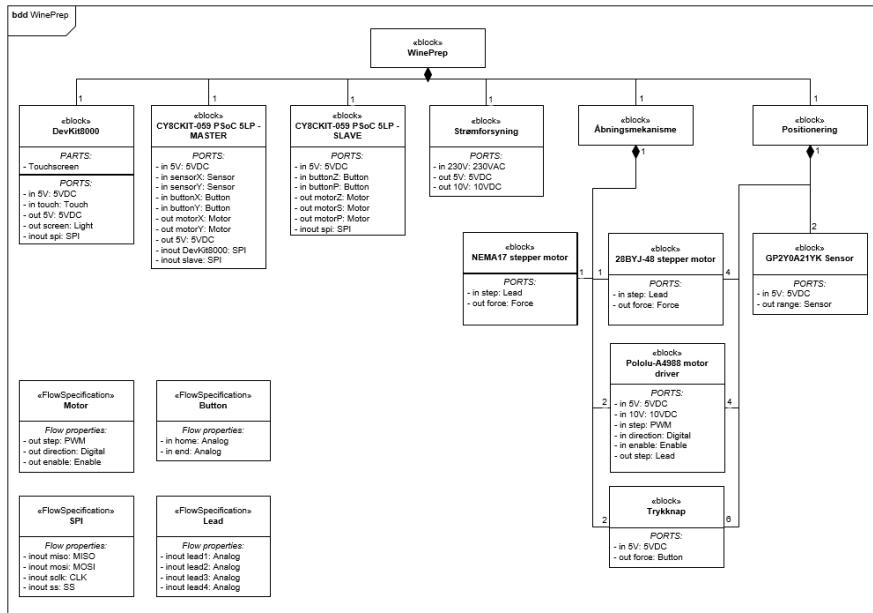
## 5.3 Systemarkitektur

I dette kapitel vil systemarkitekturen for winePrep blive beskrevet. Arkitekturen vil blive delt op i hardware og software og tager udgangspunkt i de UML-/SysML-diagrammer, der er lavet over systemet.

### Hardware

I BDD’et for winePrep ses hvilke hardwareblokke systemet består af. I dette afsnit vil disse hardwareblokke og deres funktion i systemet blive beskrevet. Systemet indeholder tre CPU’er, DevKit8000, PSoC-Master og PSoC-Slave.

Herudover er der en strømforsyning, åbningsmekanisme og positionering.



Figur 5.1: BDD for WinePrep

## Åbningsmekanisme

Til selve åbningen af en vinflaske er der konstrueret en åbningsmekanisme. Den indeholder to motorer, en til iskruning, og en til optrækning. For at kunne holde styr på skruens position, er der implementeret to trykknapper, der indikerer start og slut position. Motorene bliver styret via pololu motor drive-re(INDSÆT REFERENCE TIL POLOLU).

## Positionering

For at detektere vinflaskens position og positionere åbningsmekanismen korrekt, anvendes en positioneringsmekanisme. Herpå er monteret to motorer som finder vinflaskens x- og y-koordinater vha. sensorer. Z-koordinatet bliver reguleret af yderligere to motorer, som hæver og sænker åbningsmekanismen. Der er implementeret to trykknapper, en som indikerer at åbningsmekanismen er tilstrækkelig tæt på vinflaskens åbning, den anden indikerer startpositionen. Yderligere fire trykknapper indikerer at x-/y-motorerne har nået deres yderposition. Alle motorene bliver ligeledes styret via pololu motor drivere.

## DevKit8000

Dette er et prototypekit, hvorpå Linux distribution ångström er installeret. Det er via DevKit8000's touchskærm at interaktion med brugeren foregår. Denne enhed har dermed til opgave at tage imod input fra brugeren og sende disse videre i systemet, samt at give brugeren status beskeder. Den er forbundet til PSoC-Master via en SPI forbindelse.

## PSoC-Master

PSoC er en programmerbar CPU-enhed med GPIO pins, som har ansvaret for styring/aflæsning af hardware enheder. PSoC-Master er forbundet til positionering, hvor den styrer x-/y-motorer via pololu motor drivere, og aflæser x-/y-sensorer samt x-/y-trykknapper. Den er forbundet med PSoC-Slave via SPI.

## PSoC-Slave

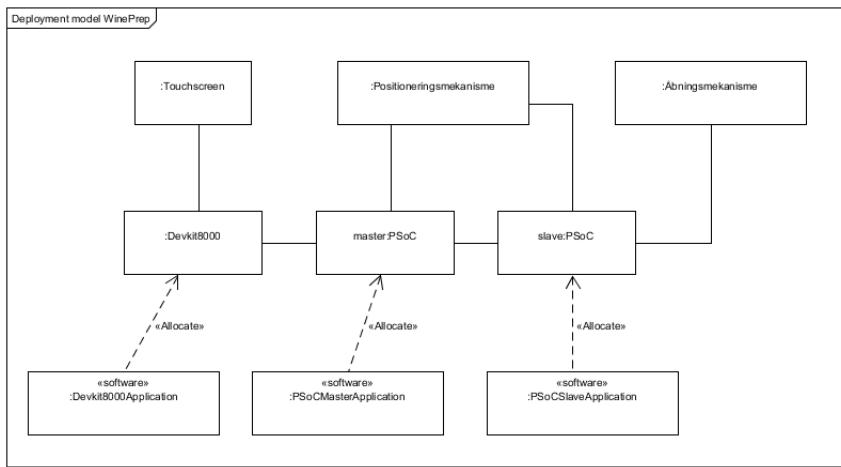
Denne enhed styrer via pololu motor drivere de to z motorer på positionering, samt motorene på åbningsmekanismen. Den aflæser også z trykknapper på positionering og trykknapper på åbningsmekanismen.

## Strømforsyning

Denne enhed leverer strøm til de enkelte hardware blokke. De tre CPU'er skal hver have 5 volt, det samme skal sensorer og trykknapper. Pololu motor driverne skal have både 5 og 10 volt.

## Software

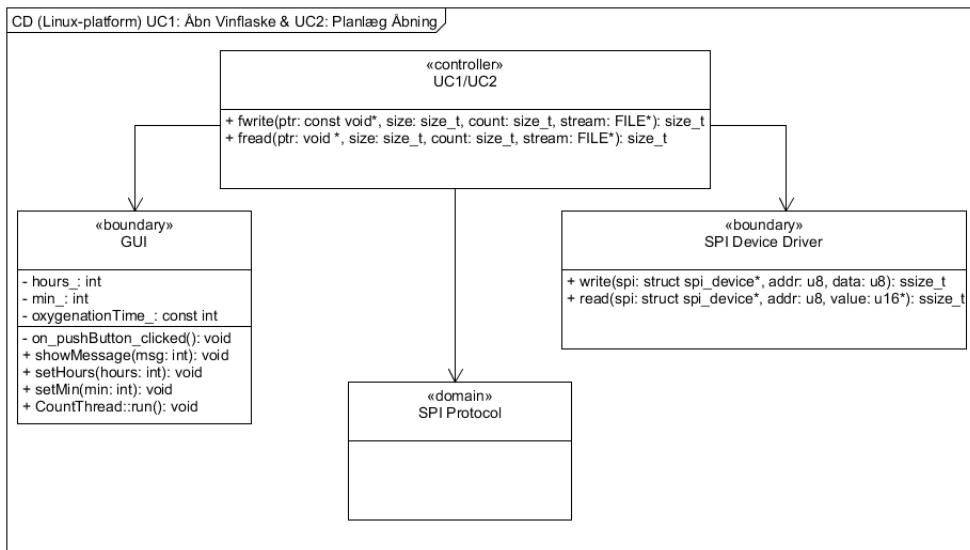
I dette afsnit vil arkitekturen for systemets software blive beskrevet. Systemet indeholder som tidligere nævnt tre CPU'er, hvorpå der er allokeret software til interaktion med brugeren samt styring/aflæsning af diverse motorer, sensorer og trykknapper. På figur 5.2 ses denne allokering af software på de respektive hardware blokke.



Figur 5.2: Software allokeringsdiagram for winePrep

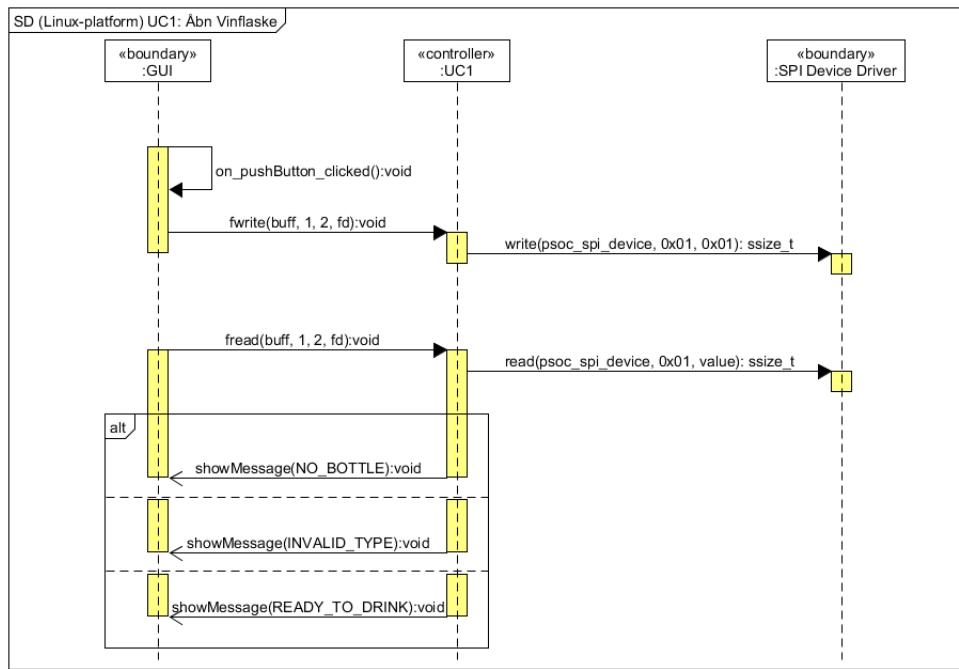
### DevKit8000 (Linux platform)

DevKit8000 har ansvaret for interaktion med brugeren via touchskærm. Derfor har systemet brug for en grafisk brugergrænseflade (GUI), hvorpå der er implementeret virtuelle knapper, som gør det muligt at oversætte de fysiske tryk til kommandoer, der kan sendes videre i systemet. Der skal også kunne vises status beskeder til brugeren, så denne er klar over systemets tilstand. Dette implementeres vha. viduer med tekstbeskeder. For at kunne sende brugerinputs videre i systemet skal DevKit8000 forbindes til PSoC-Master via SPI. Da der køres med Linux på DevKit8000 kræves det derfor at en SPI device driver bliver indsatt i kernen. DevKit8000 har altså to boundary klasser, som viser grænsefladerne for DevKit8000. I klassediagrammet ses også en protokolkasse for SPI, denne indeholder blot information til dekodning af de bits som bliver sendt over SPI.



Figur 5.3: Klassediagram DevKit8000

Med udgangspunkt i usecasen "Åbn Vinflaske", vil interaktionen mellem DevKit8000 og boundary-klasserne blive beskrevet. GUI repræsenterer her grænsefladen til brugeren, og når denne trykker på en virtuel knap, kaldes write metoden fra controllerklassen, som skriver den korrekte kommando ud til SPI device driveren. Læsning fra SPI device driveren indledes også fra GUI, og når controller-klassen har læst data, sendes status beskeder tilbage til GUI, og dermed informeres brugeren.

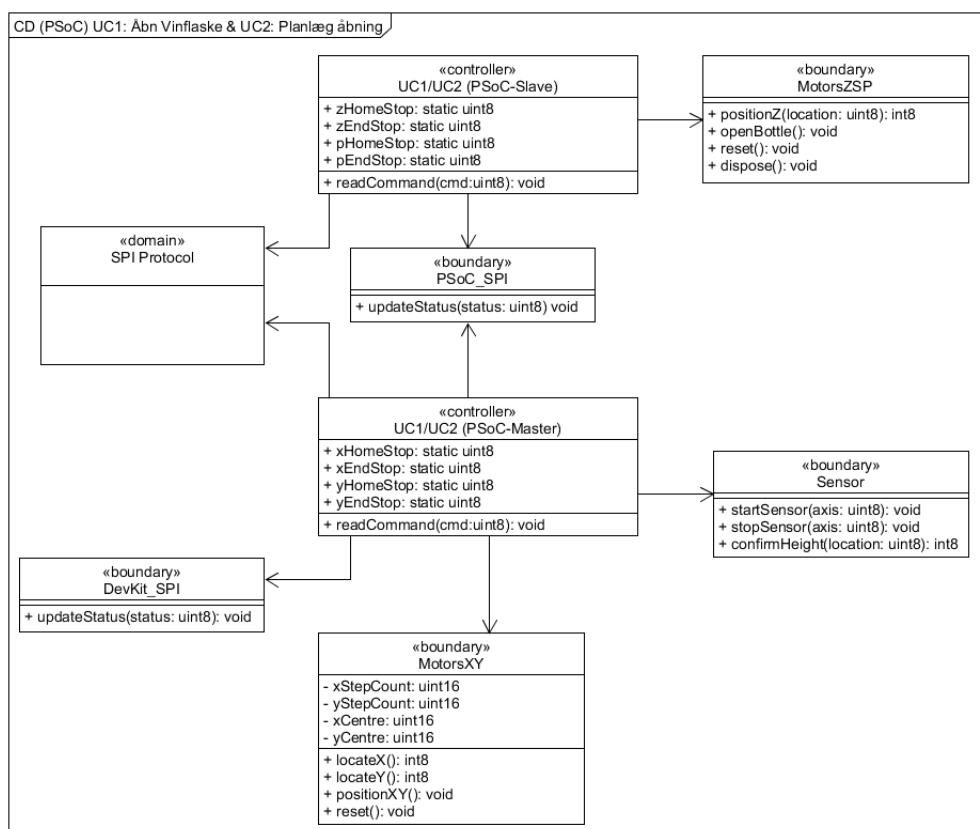


Figur 5.4: Sekvensdiagram for usecasen "Åbn Vinflaske" på DevKit8000

### PSoC-Master og PSoC-Slave

PSoC-Master og PSoC-Slave vil blive beskrevet under samme afsnit da de deler klasse- og sekvensdiagrammer. Grunden til de ikke er opdelt er for overskuelighedens skyld. Da PSoC-enhederne deler ansvaret for styring af positionering, giver det mening at de er inkluderet i samme sekvensdiagram. Det vil sige at der er to controllerklasser i klasse- og sekvensdiagrammerne.

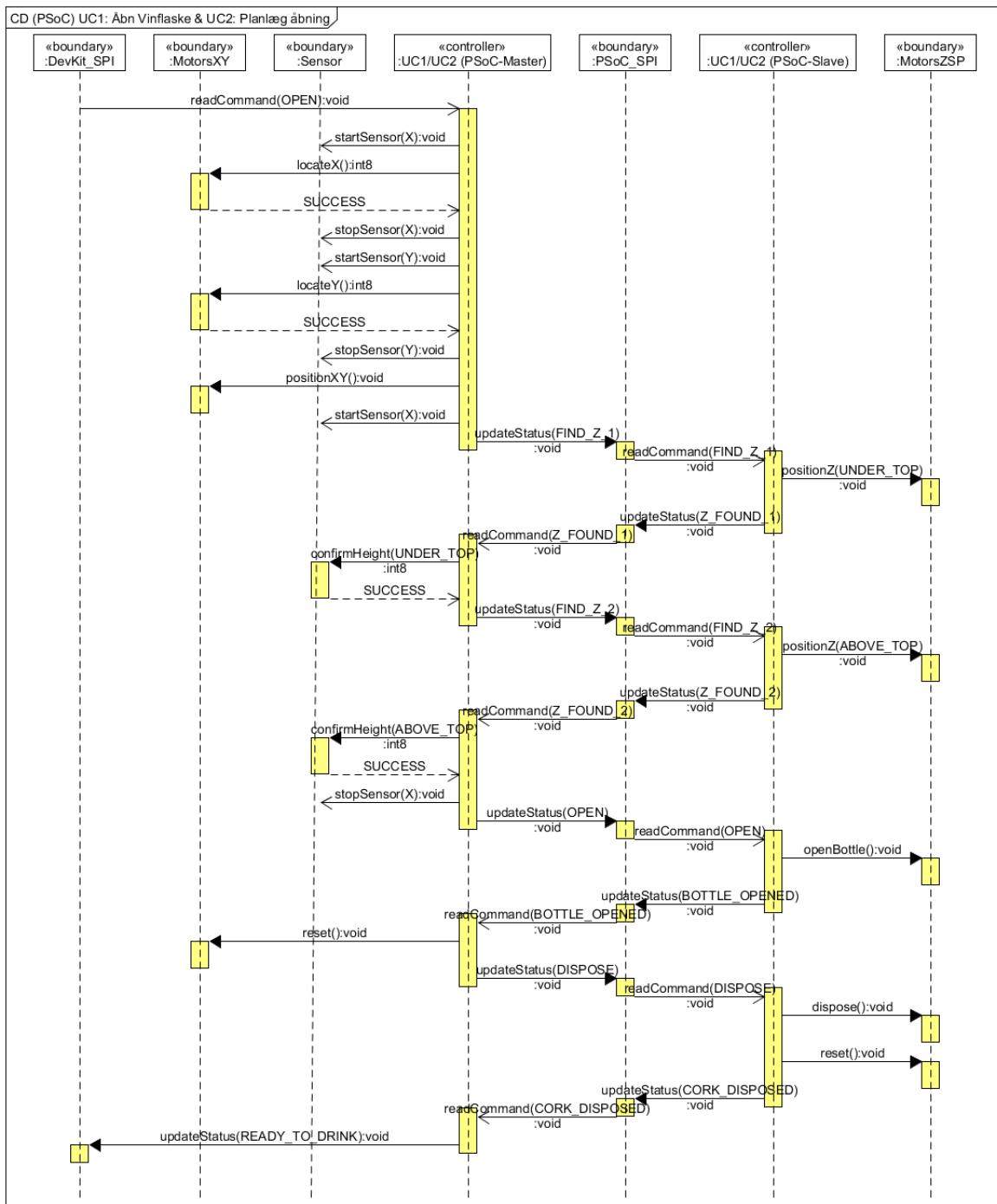
PSoC-Master har to SPI boundary klasser, en til kommunikation med DevKit8000, og en til PSoC-Slave. Herudover er der boundary klasser til x/y motorer og sensorer på positionering. PSoC-Slave har SPI boundary klasse til kommunikation med PSoC-Master og til z-motorer på positionering, og motorer på åbningsmekanismen. Begge PSoC-enheder har en SPI-protokol til dekodning af SPI-kommandoer.



Figur 5.5: Klassediagramm PSoC-Master/-Slave

Usecasene "Åbn Vinflaske" og "Planlæg Åbning" er samlet under det samme sekvensdiagram (se figur 5.6), da der fra PSoC-enhedernes synspunkt sker det samme, nemlig åbning af en vinflaske. Timing af åbningen foregår på Dev-Kit8000, og har ingen relevans for PSoC-enhederne. Der er ikke medtaget alternative scenarier i sekvensdiagrammet, da disse er triviele, og blot skaber unødvendig uoverskuelighed.

Selv åbningen initieres fra SPI-forbindelsen til DevKit8000. Herefter sætter PSoC-Master x-/y-motorer til vha. sensorerne at finde flaskens x-/y-placering. Når disse er fundet, gives der besked til PSoC-Slave om at aktivere z-motorerne og finde den rette afstand til flaskens top. Når denne er fundet påbegyndes åbningen af vinflasken, og derefter dispensering af proppen. Sekvensdiagrammet afsluttes med en return besked tilbage til DevKit8000 via SPI om succesfuld åbning.



Figur 5.6: Sekvensdiagram PSoC-Master/-Slave

## 5.4 Design

### Hardware

#### Skal medtages på en eller anden måde

Det eneste problem med motoren var dens relativt svage moment som standard, unipolær model. Det blev løst ved at omdanne motoren til bipolær og det lykkedes på denne måde at øge momentet med mere end 2 gange. Se hvordan dette lod sig gøre i bilag xx.

### Software

#### Motor- og sensorstyring

Styringen af de aktuelle motorer/sensorer sker udelukkende via 2 PSoC's: en Master- (MP) og en Slave-PSoC (SP). MP har foruden at yde statusopdateringer til DevKit8000 (DK8k) til opgave at styre motorerne/sensorerne for x-/y-akserne og at sende kommandoer til/modtage status fra SP. SP har til opgave at styre motorerne for z-aksen, skruen og åbningsmekanismen. Som set i sekvensdiagrammet (figur 5.6)) påbegyndes detektering og åbning af vinflasken ved en kommando fra DK8k til MP, som efter at have fastslået flaskens x- og y-position giver besked til SP om at løfte sensorerne til en position, der er en vis afstand under flaskens top. SP giver besked til MP om, at dette er gjort, hvorefter MP med y-sensoren detekterer, om der står en flaske eller ej. Dette gentages med en position over flaskens top. Herefter flytter MP åbningsmekanismen til en position over flasken, hvorefter der gives besked til SP om at åbne flasken. Når dette er gjort resettes alle relevante komponenter til en startposition, hvorefter proppen disponeres, og der gives besked til DK8k om, at flasken er drikkeklaar.

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 MP sender en statusbesked til DK8k om den pågældende fejl.

#### Klasser og funktioner

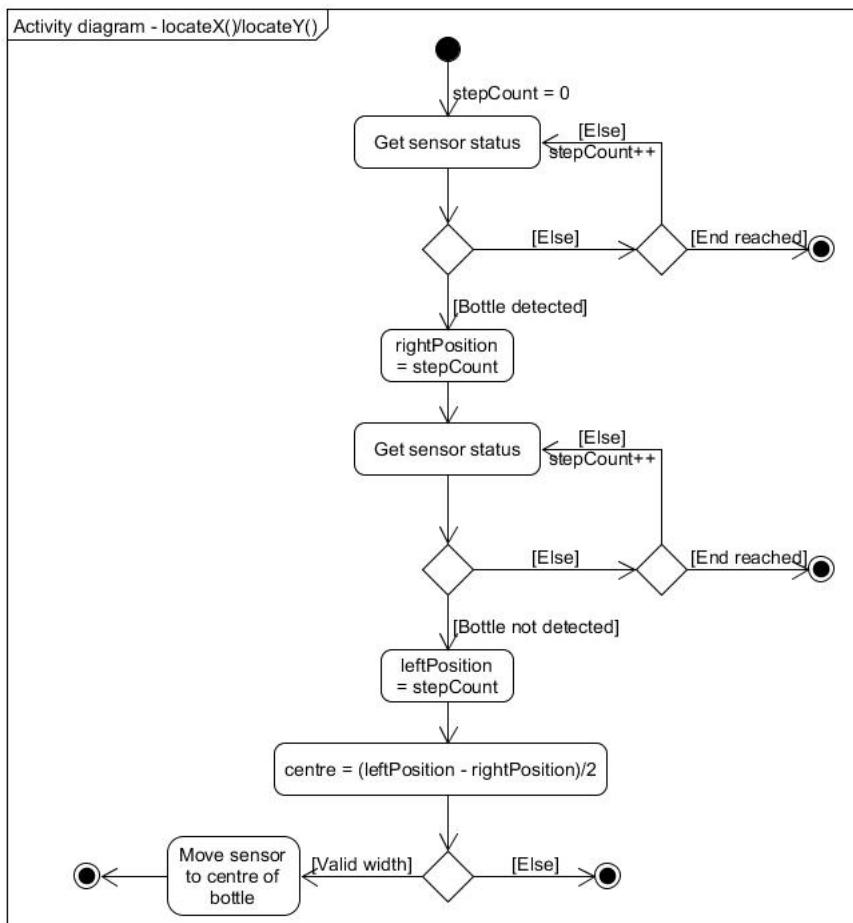
Sideløbende med sekvensdiagrammet er følgende klassediagram blevet udformet: (indsæt billede af dette)

De variable i de to Controller-klasser, der ender i *Stop*, repræsenterer trykknapperne. Disse benyttes af Motor-klasserne, når en bestemt position ønskes registreret. Heriblandt start- og slutposition på en gældende akse.

Klassernes overordnede funktionalitet burde ud fra deres titler og ovenstående beskrivelse virke forholdsvis indlysende, men visse af metoderne kræver yderligere forklaring.

### locateX() / locateY()

For nærmere at beskrive disse to metoder, hvis funktionalitet ikke afviger fra hinanden, er følgende aktivitetsdiagram blevet udarbejdet:



Figur 5.7: 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 tællses op. I så fald enden på akslen nås, skal metoden afslutte og returnere en fejlværdi. Hvis flasken registreres, gemmes `stepCount` i `rightPosition`. Der fortægtes 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 `left-`

*Position.* Denne værdi sammenlignes med en forudbestemt værdi for at sikre, at det er en kompatibel flaske, der er indsats. Hvis ikke, afsluttes metoden og returnerer en fejlværdi. Ellers flyttes sensoren til flaskens midte, og metoden returnerer en succesværdi.

### Øvrige metoder

**positionZ(uint8)** løfter sensorerne et vist antal steps, som er bestemt ud fra det medgivne argument, op fra startpositionen.

**confirmHeight(uint8)** 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.

**openBottle()** skal få z-motorerne til at presse åbningsmekanismen ned mod vinflasken ud fra et forudbestemt antal steps, hvorefter s-motoren skal sætte skruen til at dreje et bestemt antal steps, så p-motoren kan hive i propren, indtil den rammer en trykknap.

**dispose()** skal blot dreje s-motoren et vis antal steps for at disponere propren.

## Seriell kommunikation

### Devkit8000-PSoC Master

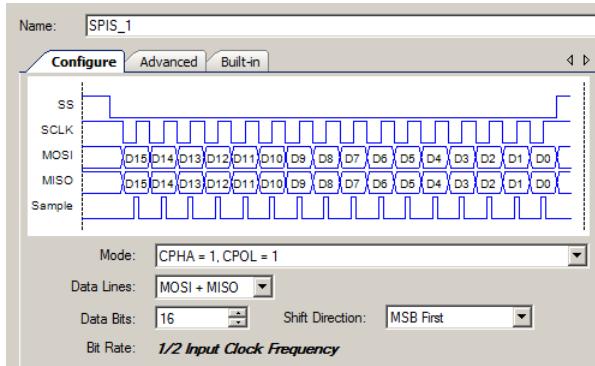
Som beskrevet i analysen for SPI (reference), blev denne protokol valgt pga. den kendskab gruppen allerede havde fra HAL øvelse 6 (reference). For at kunne kommunikere over SPI fra en Linux platform, skal den rette device driver indsættes i Linux kernen. Denne driver tillader os at tilgå den SPI hardware som er på Devkittet fra vores grafiske brugergrænseflade. Ideen er at man skriver en driver der via det SPI interface som allerede er implementeret, kan lave metoder som gør det muligt at overfører data til/fra userspace(GUI).

I driveren skal opsætningen for SPI forbindelsen naturligvis også erklæret. Dette indebære bl.a. bus nummer, antal databits, maksimal overførelseshastighed 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 besluttes at der ikke skulle bruges mere tid på selv at lave en driver, og istedet 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 5.8: Opsætning for SPI forbindelse Devkit8000-PSoC Master

Ud fra figur xx, 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 databits for SPI kommunikation, henvises til afsnittet "Implementering".

### PSoC Master - PSoC Slave

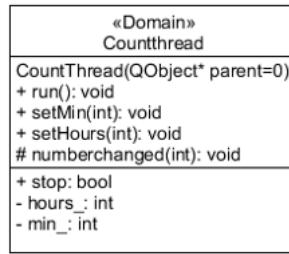
Til SPI forbindelsen mellem PSoC Master og PSoc slave havde gruppen frie hænder til opsætte SPI. Her blev clock mode valgt til CHPA = 0 og CPOL = 0, og antal databits til 8. Grunden til der kun bliver sendt 8 bits her, er at det er tilstrækkeligt til den simple form for kommunikation der er mellem PSoc 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, hvilket der ikke har været nogen yderlige designmæssige tanker omkring.

### GUI

Brugergrænsefladen gav særlige udfordringer i designfasen. Da programmet QT blev anvendt til at designe og implementere brugergrænsefladen. 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

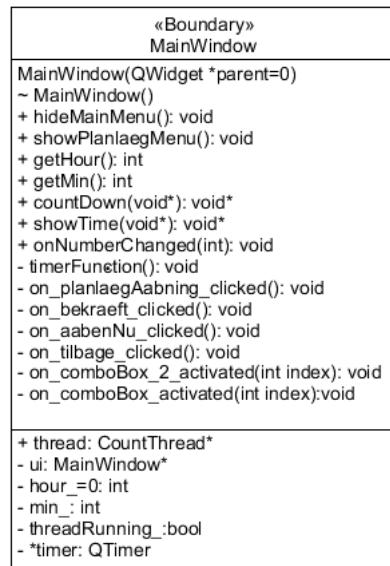
forhånd. Derfor blev det besluttet at klasserne først skulle udarbejdes efter at brugergrænsefladen var designet.

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 x.



Figur 5.9: 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 x.



Figur 5.10: MainWindow klasse illustreret

Brugergrænsefladen er state styret. Derfor har det været nødvendigt at lave et statemachine diagram for brugergrænsefladen. Der er i alt 3 overord-

nede states for hele system. De tre states er, ”Åbning”, ”Venter på åbning” og ”Åbning stoppet”.

Når vinåbneren er i gang med at åbne en vinflaske, så er den i staten ”Åbning”. Der er to ting der kan bringe systemet til denne state. Den første er at brugeren igennem brugergrænsefladen trykker på knappen ”Åbn nu”. Dette vil få systemet til at starte åbningen, og dermed bringe systemet i staten ”Åbning”. Den anden handling der kan bringe systemet i denne state, er når tiden under ”Planlæg åbning” menuen udløber og systemet dermed starter åbningen på vinflasken.

Staten ”Venter på åbning” startes ved at brugeren under ”Planlæg åbning” menuen sætter en tid og trykker på bekraeft. Når brugeren starter tiden, vil systemet begynde at tælle ned indtil åbningen påbegyndes. Den tid hvor systemet venter på at tiden udløber således at åbningen kan påbegyndes er staten ”Venter på åbning”.

Den sidste state er ”Åbning stoppet”. Det er denne state systemet starter ud med at være i. I denne state foretager systemet sig ingenting. Når åbningen er færdiggjort kommer systemet i denne state. Den fulde statemachine diagram kan ses i bilag x.

## 5.5 Implementering

### Hardware

### Software

#### Motor-/sensorstyring

Klasserne fra klassediagrammet blev implementeret i form af hver deres headerfil efter principippet om høj samhørighed - lav kobling. *main*-funktionen blev formet som en state-machine, der vha. en switch påkaldte de metoder, der skulle udføres på et givent tidspunkt i eksekveringen af programmet i henhold til sekvensdiagrammet (figur 5.6). Disse switches' cases bestemtes ud fra de kommandoer/beskeder, de enkelte PSoC's modtog fra hinanden eller DevKit8000. Der er ligeledes blevet oprettet en separat *status*-fil, som indeholder adskillige kommandoer/beskeder og forkortelser, der bruges igennem programmet, for at holde koden overskuelig og øge læsbarheden af denne.

#### Hardware-grænseflade

##### Sensorer

Til at måle sensorerne benyttedes en SAR-ADC, som, efter hvert sample, returnerede en værdi i *counts*. For at sammenligne med (GRAF FRA SENSOR-

DATABLAD) konverteredes disse værdier til mV. Derved kunne der fastslås, hvorvidt en flaske var registreret, ud fra afstanden forbundet med den målte spænding.

### **Motorer**

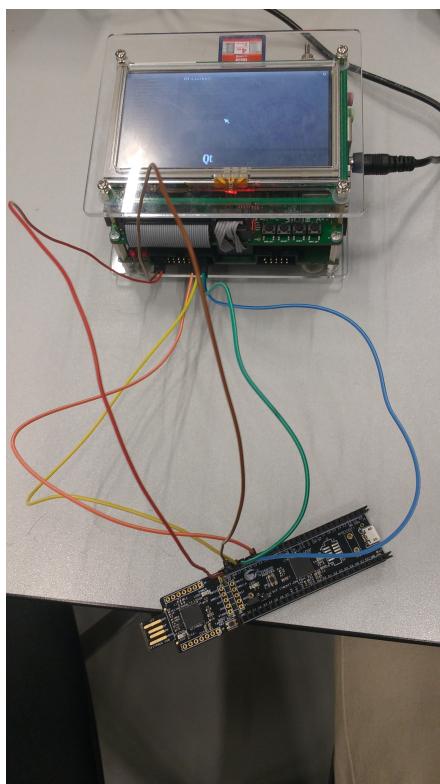
Motorerne styredes vha. et PWM-signal, som gik fra en given GPIO-pen ud til STEP-inputtet på A4988-driveren, som talte et step op for hver rising edge på PWM-signalet, samt to digitale signaler, der gik til henholdsvis ENABLE- og DIRECTION-inputtene på driveren.

### **Knapper**

Knapperne implementeredes som interrupts der trigger på rising edge. Disse var hovedårsagen til, at der skulle min. 2 PSoS's, da hvert interrupt optager en port på PSoC'en, som kun har 6 til rådighed, mens der var behov for 8.

### **Seriell kommunikation**

SPI forbindelsen blev mellem de to enheder blev etableret med 6 ledninger som set på figur xx. 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.



Figur 5.11: 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

Da brugergrænsefladen skulle laves, var der ingen tvivl om hvilket program der skulle anvendes til udarbejdelse af brugergrænsefladen. Programmet QT blev valgt da der tidligere har været arbejdet med QT i forbindelse med andre semesterprojekter. QT er et program som giver en masse muligheder som kan udnyttes. For eksempel giver QT en drag and drop mulighed, således at brugergrænsefladens udseende kan designes på en meget let og brugervenlig måde. 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. Et eksempel på hvordan funktionen fwrite() blev brugt kan ses på figur x.

```
void MainWindow::on_aabenNu_clicked()
{
FILE *fd;
char buff[2];
fd = fopen(PSOC, "r+");
buff[0] = OPEN_BOTTLE;
buff[1] = '\0';
fwrite(buff,1,2,fd);
fclose(fd);
}
```

Figur 5.12: Åben nu funktionen implementerets

På figuren ses det hvordan funktionen for trykknappen ”Åbn nu” er implementeret. PSOC er tidligere i koden blevet defineret som path'en på PSoC driven. I koden kan man se at OPEN\_BOTTLE, skrives til PSoC driveren ved hjælp af fwrite(). OPEN\_BOTTLE er tidligere blevet defineret som 5, hvilket i SPI protokollen betyder at vinen skal åbnes. Det er samme kode der bruges til funktionen ”Planlæg åbning”. For at få tiden talt ned og samtidigt displayet på skærmen er der blevet benyttet threads. Implementeringen af dette kan ses i dokumentationsbilaget x.

## 5.6 Test

### Motorer og sensorer

Test af motorer og sensorer er foretaget med både uni- og bipolære 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 unipolære motorer senere blev udskiftet med bipolære.

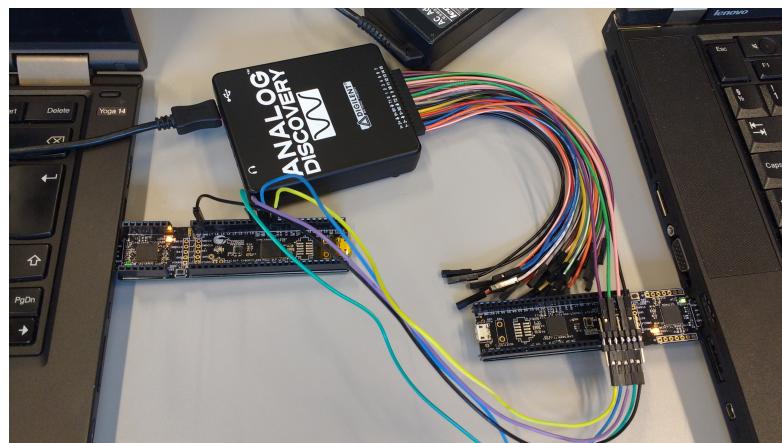
## Seriel kommunikation

### 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 ligsom i testen mellem DevKit8000 og PSoC Master. På figur xx ses testopstilling for PSoC-PSoC

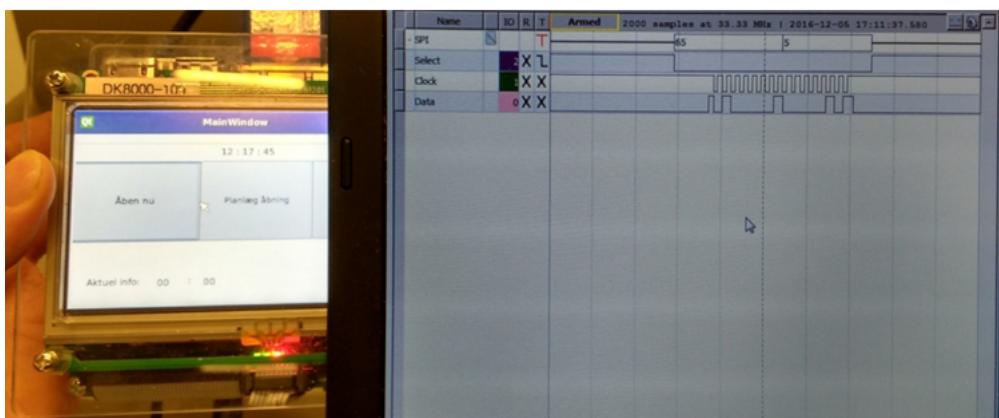


Figur 5.13: testopstilling for PSoC Master/PSoC Slave forbindelsen

## 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 testet 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 tou-chskærmen 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 5.14, kan det ses hvordan, det var muligt at sende kommandoen 5 ved at bruge ”Planlæg åbning” funktionen.



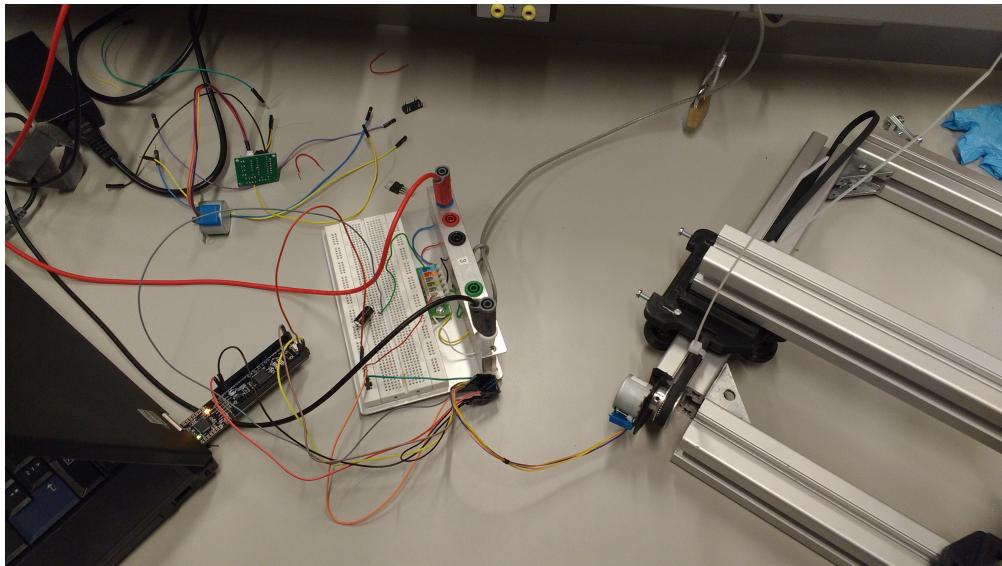
Figur 5.14: Åben nu funktionen implementerets

## 5.7 Resultater

### Motor-/sensorstyring

#### Akserne

Figur 5.15 viser test opstillingen af bipolær motor på x-/y-akse, som validerede det forventede resultat, hvor aksen blev flyttet vha. det forøgede moment fra motoren.



Figur 5.15: Test af bipolær motor på x-/y-akse

Resultaterne fra testen, og andre lignende test af z-aksen (se evt. bilag xx), betød at en udvidet modultest kunne udføres hvori akserne samt detektering foregik. Desværre lykkedes det aldrig at få glidende bevægelser på akserne, som med stor sandsynlighed skyldes konstruktionens ujævnheder.

Sensorerne blev testet ved at lade forskellige materialer blive detekteret på afstand af varierende størrelser, hvor det viste sig at sensorerne var ret pålidelige når resultaterne blev holdt op mod Figur 4 i databladet (reference) for dem. Et resultat, målt i mV, kan ses på Figur 5.16 under fanebladet Value, ellers henvises til bilag xx for flere resultater.

Locals				
Name	Value	Address	Type	Radix
ADCResultX	2241		unsigned long	decimal

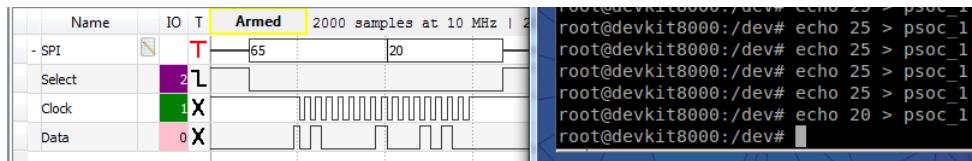
Figur 5.16: Resultat af detektering fra sensor ved 10 cm

Værdien for detektering ved 10 cm var 2241 mV som lægger sig tæt op ad de ca. 2300 mV der kan udledes af databladet, altså en afvigelse på 2,57%, eller en unøjagtighed på 2,57 mm ved denne afstand. Sensorernes unøjagtighed ville altså være for stor ift. at en åbningsmekanisme skulle lægge sig over flasken og åbne den.

## Seriel kommunikation

DevKit8000-PSoC Master

På figur 5.17 ses resulater på Logic Analyzer når der skrives til PSoC master fra terminalen på Devkit8000.

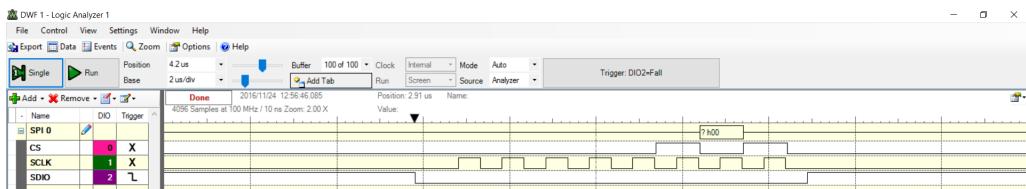


Figur 5.17: Udskrift fra Logic Analyzer fra test af SPI forbindelse mellem Devkit og PSoC Master

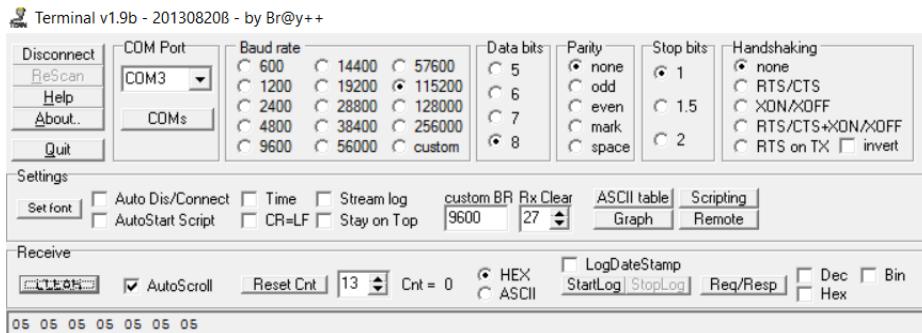
Resultater for læsningen af PSoC Master gav kun succesfuldt resultater ved en enkelt test. Derudover kunne der ikke læses fra PSoC master. Hvilket betyder at der ikke er dokumentet en succesfuld aflæsning af MISO forbindelsen. Selv efter mange forsøg, og med hjælp fra undervisere i HAL, blev dette problem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

## PSoC Master - PSoC Slave

På figur 5.19 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 5.18: Udskrift fra Logic Analyzer fra test af SPI forbindelse mellem PSoC Slave og PSoC Master



Figur 5.19: Udskrift fra terminal fra test af SPI forbindelse mellem PSoC Slave og PSoC Master

## 5.8 Diskussion af resultater

### Seriell kommunikation

#### DevKit8000-PSoC Master

Som det ses på figur 5.18, 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 nedafgående flanke, og læser på opadgående flanke. Dette svarer til CPHA = 1 og CPOL = 1, hvilket også var den opsænning der er lavet for denne SPI forbindelse.

Problemet med MISO forbindelsen betød at projektets prototype ikke kunne sende status beskeder til brugeren, hvilket naturligvis var en stor skuffelse for gruppen. Problemet ligger højest sandsynlig på selve PSoC, da der ikke læses noget data på MISO forbindelsen, og det derfor ikke har noget med DevKit8000 af gøre.

#### 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 skrifter på nedadgående flanke og læser på opadgående. Dette er helt som forventet. Og som det ses på figur 5.19, 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 brugeren 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 der ikke ville læse. Det var kun muligt at skrive med SPI driveren i lange perioder.