Dokumentation

Semesterprojekt 3. Semester

Gruppe 10

Vejleder: Søren Hansen

Gruppemedlemmer:

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

Indhold

Kapitel 1

Forord

Denne rapport er skrevet på 3. semester af gruppe 13, på retningerne IKT, E 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.

Kapitel 2

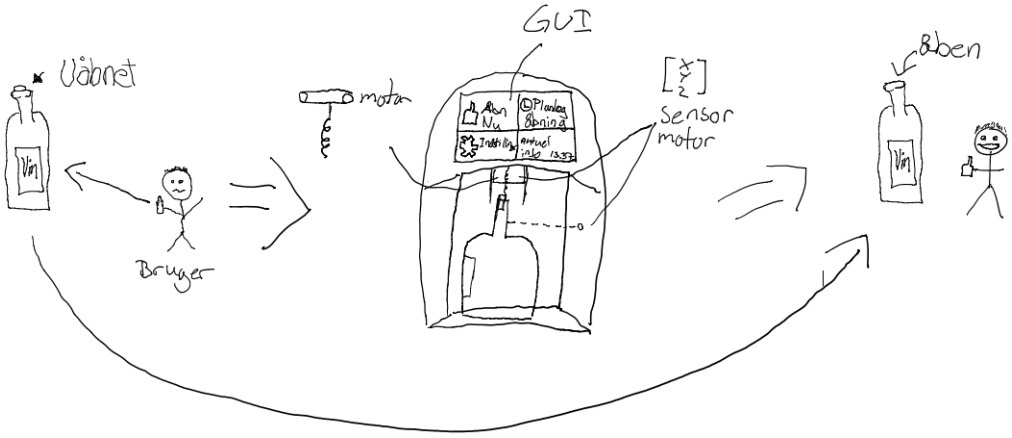
Indledning

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 vindrikkere 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 kernekompetencer 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 indsat 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 indtaste et klokkeslæt, hvor vinflasken ønskes åbnet, og vinen drikkeklar. Systemet venter da til iltningstidspunktet, 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 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 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(referencer til bilag: ikke-funktionelle krav/brugervenlighed + /ydeevne). Denne vinflaske skal være af en på forhånd bestemt type(reference til detaljer om vinflaske). 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 og aktuator. Gruppen har derfor fokuseret på de usecases som indeholdte alle disse elementer, hvilket var "åben nu"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 enige om at dette låg 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, hvilket der ikke blev grundet det mandefald gruppen havde. 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 vines 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 både motorer, sensorer, linux platform og PSoC, og dermed opfylde minimums kravene til projektet. I og med at gruppen havde en forholdsvis lille størrelse blev det prioriteret ikke at bruge ressourcer på implementering af features der låg udenfor disse minimumskrav. Det var vigtigt 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 derfor 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.

4.1

Måden hvorpå dette projekt startede, var ved at en masse viden omkring emnet blev indsamlet. Her blev der ikke taget hensyn til nogen specifik metode. Der blev allerede i løbet af de første uger udarbejdet nogle løsningsforslag til hvorledes det endelige produkt skulle se ud.

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.

Kapitel 5

Analyse

5.1 Hardware

5.2 Analyse

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 inddelt 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 opnåes 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.

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.

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 rigelig 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.

5.3 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). 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 bibeholdelssen af knapperne.

5.4 Analyse - 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 vinen stod til at blive åbnet, men da der ikke er et batteri indsat 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 vinen.

I og med SPI allerede var valgt som kommunikationsform mellem Devkit8000 og PSoC, var der ingen overvejelse over kommunikationsformen ved udarbejdelsen af brugergrænsefladen.

5.5 Analyse af SPI

Til kommuniktaion 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.

Kapitel 6

Arkitektur

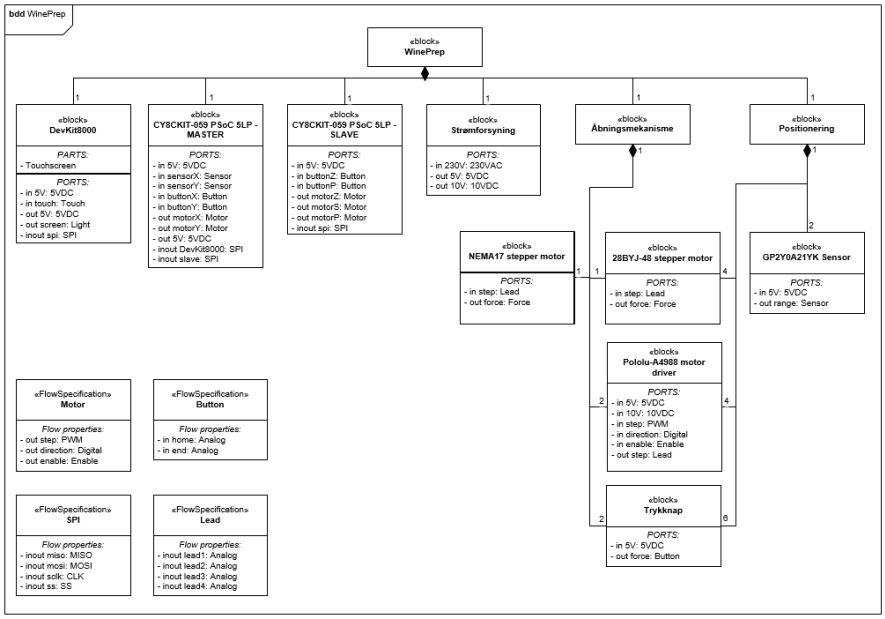
Kapitel 7

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.

7.1 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 7.1: 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 drivere.

Positionering: For at detektere vinflaskens position og positionere åbningsmekanismen korrekt, anvendes en positioneringsmekanisme. Herpå er monteret to motorer som finder vinflaskens x og y koordinator 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.

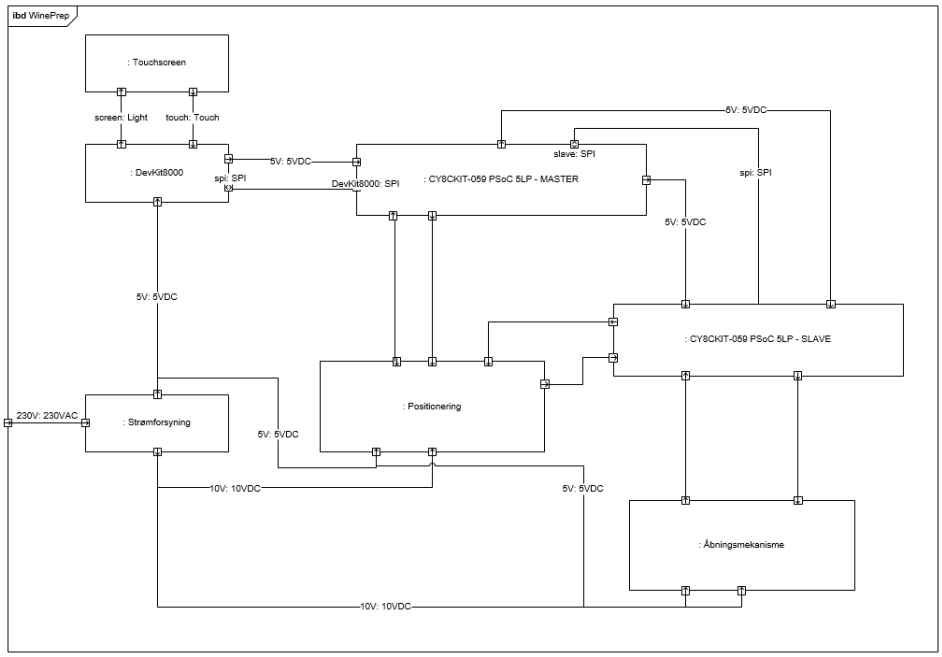
Devkit8000: Dette er et prototypekit, hvorpå Linux distribution ångström er installeret. Det er via devkittets touchskærm at interaktion med brugeren foregår. Denne enhed har dermed til opgave at tage imod input fra brugeren og sende disse videre i systemet, samt at give brugeren status beskeder. Den 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.

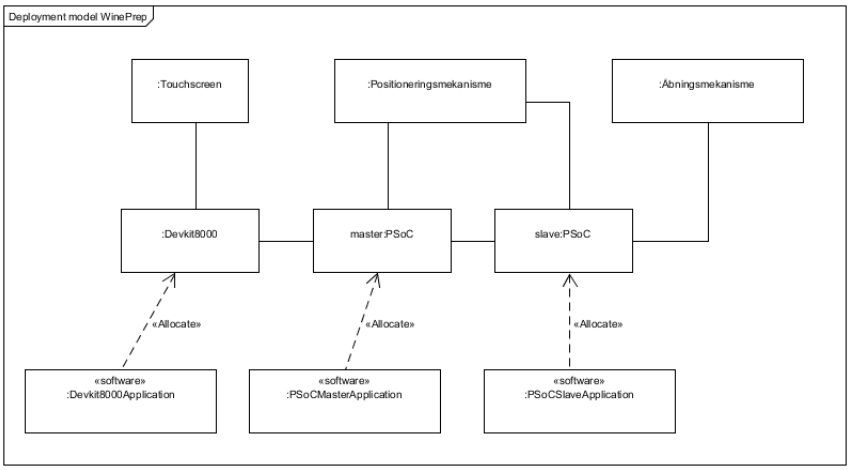
Strømforsyning: Denne enhed leverer strøm til de enkelte hardware blokke. De tre CPU'er skal hver have 5 volt, det samme skal sensorer og trykknapper. Pololu motor driverne skal have både 5 og 10 volt.



Figur 7.2: **IBD for winePrep**

7.2 Software

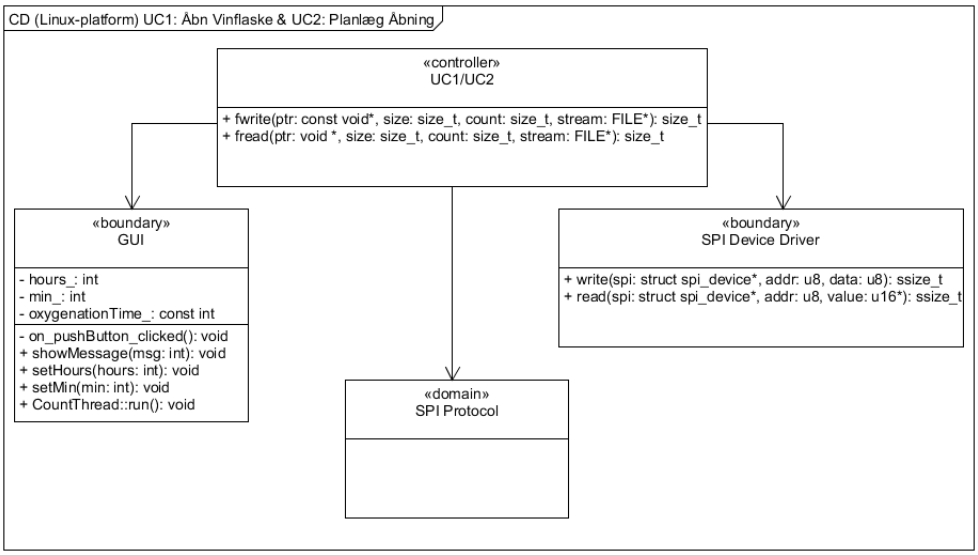
Systemet indeholder som tidligere nævnt tre CPU'er, hvorpå der er allokeret software til interaktion med brugeren samt styring/aflæsning af diverse motorer, sensorer og trykknapper. I dette afsnit vil arkitekturen for systemets software blive beskrevet.



Figur 7.3: Software allokeringsdiagram for winePrep

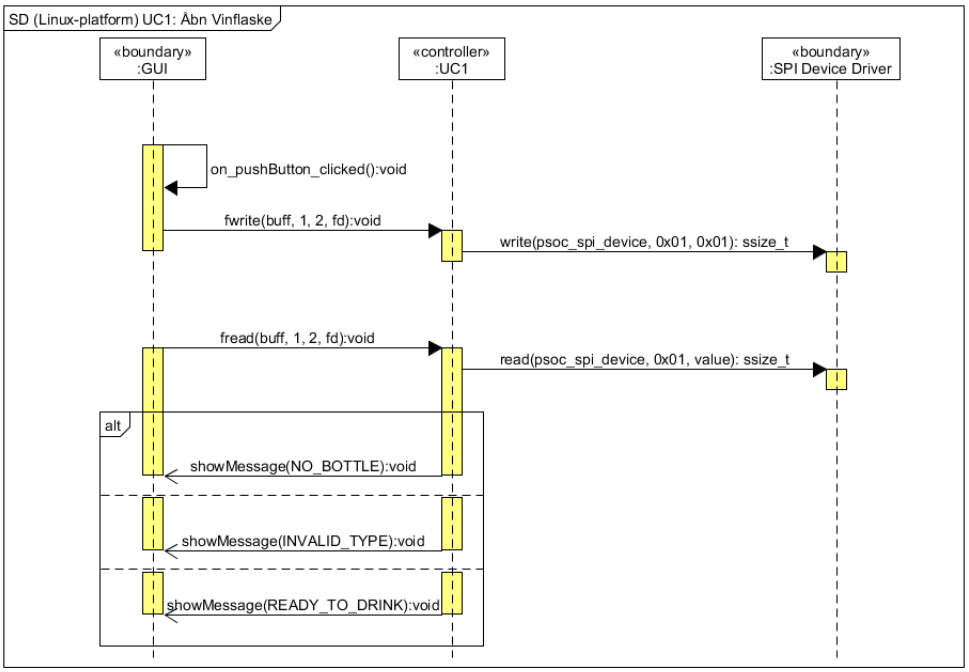
Devkit8000 (Linux platform)

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 devkittet forbindes til PSoC master via SPI. Da der køres med Linux på Devkit8000 kræves det derfor at en SPI device driver bliver indsat i kernen. Devkit8000 har altså to boundary klasser, som viser grænsefladerne for devkittet. I klassediagramet ses også en protokolklasse for SPI, denne indeholder blot information til dekodning af de bits som bliver sendt over SPI.



Figur 7.4: Klassediagram Devkit8000

Med udgangpunkt i usecasen "åbn vinflaske", vil interaktionen mellem devkittet og boundaryklasserne 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.

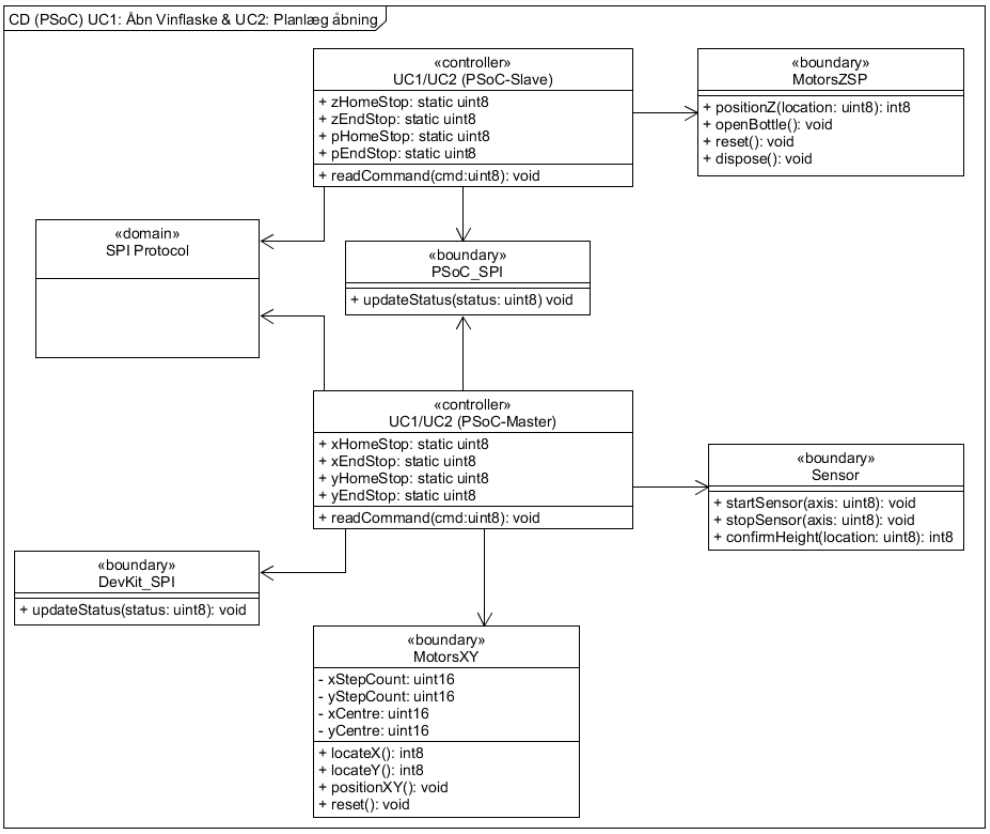


Figur 7.5: Sekvemdiagram for usecasen "Åbn vinflaske"på Devkit8000

PSoC Master og PSoC slave

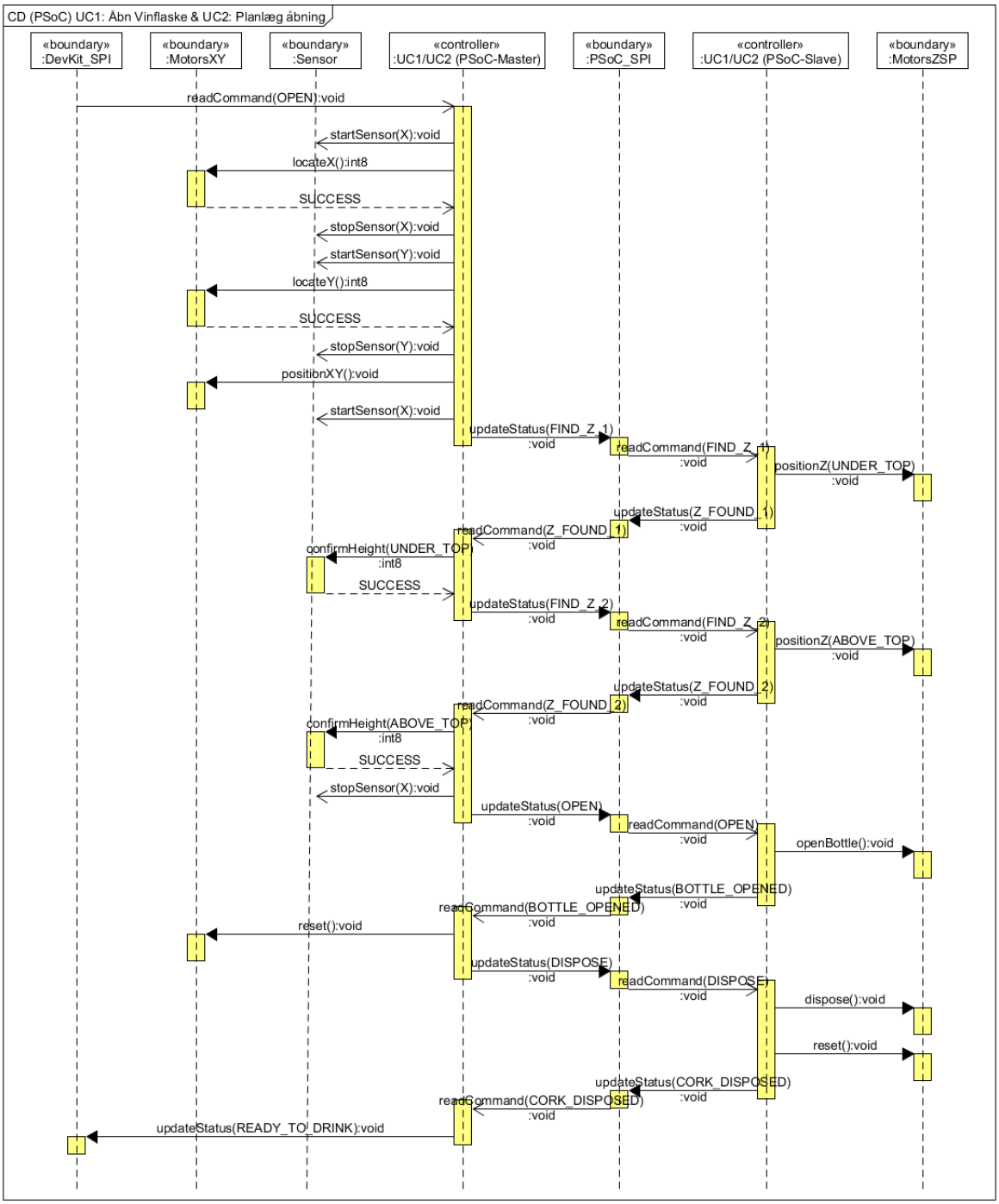
PSoC Master og PSoC Slave vil blive beskrevet under samme afsnit da de deler klasse- og sekvensdiagrammer. Grunden til de ikke er opdelt er for overskueligehedens skyld. Da PSoC enhederne deler ansvaret for styring af positionering, giver det mening at de er indkluderet i samme sekvensdiagram. Det vil 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 7.6: Klassediagram PSoC Master/Slave

Usecasene "åbn vinflaske"og "planlæg åbning"og samlet under det samme sekvensdiagram, da der fra PSoC enhedernes synspunkt sker det samme, nemlig åbning af en vinflaske. Timing af åbningen foregår på Devkit8000, og har ingen relevans for PSoC enhederne. Der er ikke medtaget alternative scenarier i sekvensdiagrammet, da disse er trivieller, og blot skaber unødvendig uoverskuelighed. Selve åbningen initieres fra SPI forbindelsen til Devkit8000. Herefter sætter PSoC master x/y motorer til vha. sensorerne at finde flaskens x/y plasering. Når disse er fundet gives der besked til PSoC slave om at aktivere z motorene 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 retur besked tilbage til Devkit8000 via SPI om succesfuld åbning.



Figur 7.7: Sekvensdiagram for usecasene "åbn vinflaske"og "planlæg åbning"på PSoC Master/Slave

Kapitel 8

Design

8.1 Hardware

8.2 Software

Motor- og sensorstyring

Indledning

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(reference til dette) 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 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 MP sender en statusbesked til DK8k om den pågældende fejl.

Klasser og funktioner

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

De variable i de to Controller-klasser, der ender i *Stop*, repræsenterer tryknapperne. 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:

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 tælles 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 indsat. Hvis ikke, afsluttes metoden og returnerer en fejlværdi. Ellers flyttes sensoren til flaskens midte, og metoden returnerer en succesværdi.

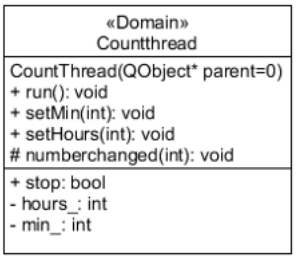
Øvrige metoder
De øvrige metoder er relativt simple: **positionZ(uint8)** løfter sensorerne et vist antal steps, som er bestemt ud fra det medgivne argument, op fra startpositionen; **confirmHeight(uint8)** skal registrere med y-sensoren om en flaske kan registreres i den givne højde alt efter den medgivne 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 proppen, indtil den rammer en

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

8.3 Implementering - GUI

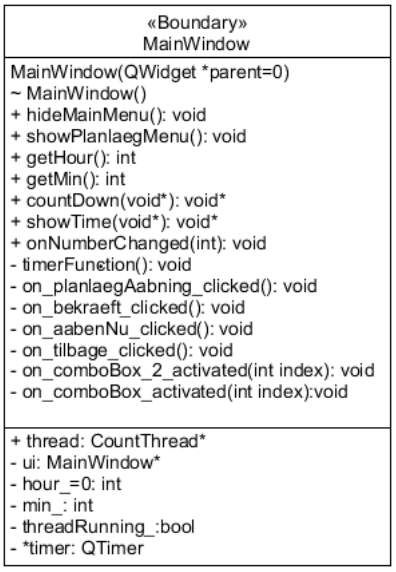
Design – GUI Brugergrænsefladen gav særlige udfordringer i designfasen. Da programmet QT blev anvendt til at designe og implementer brugergrænsefladen. Udfordringerne bestod primært i at kendskabet til programmet QT ikke var særligt stort. Da QT selv skaber klasserne og metoderne er det svært at beskrive 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 countklassen kan ses på figur x.



Figur 8.1: 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 8.2: MainWindow klasse illustreret

Brugergrænsefladen er state styret. Derfor har det været nødvendigt at lave et statemachine diagram for brugergrænsefladen. Der er i alt 3 overordnede 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 bekræft. 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. Implementering – GUI

Seriel kommunikation

Til at håndtere SPI-kommunikationen på DevKit8000 er der benyttet en af skolen udleveret driver. Det har ikke været muligt at få adgang til source-

koden for denne, men det kan formodes, at den opbygget efter den typiske struktur med en character driver, som håndterer kommunikationen mellem user-space og hardware driveren, hvormed sidstnævnte forbinder character driveren med den specifikke hardware, der skrives ud til, og håndterer specifikationer for SPI-kommunikationen. Havde det ikke været for et tilhørende design til PSoC(REFERENCE TIL BILLEDE AF SPIS-KONFIGURERING PÅ MASTER-PSOC!!!!!), ville disse specifikationer have været ukendte.

I PSoC-Creator er det muligt at benytte SPI-komponenter vha. drag-and-drop metoden og at specificere disse som master/slave. Master-PSoC'en skal håndtere både kommunikation med DevKit8000 og Slave-PSoC, så denne skal både have SPI-master- og /-slave-komponenter.

Kapitel 9

Test, implementering og resultater

9.1 Hardware

9.2 Test

Motorer og sensorer

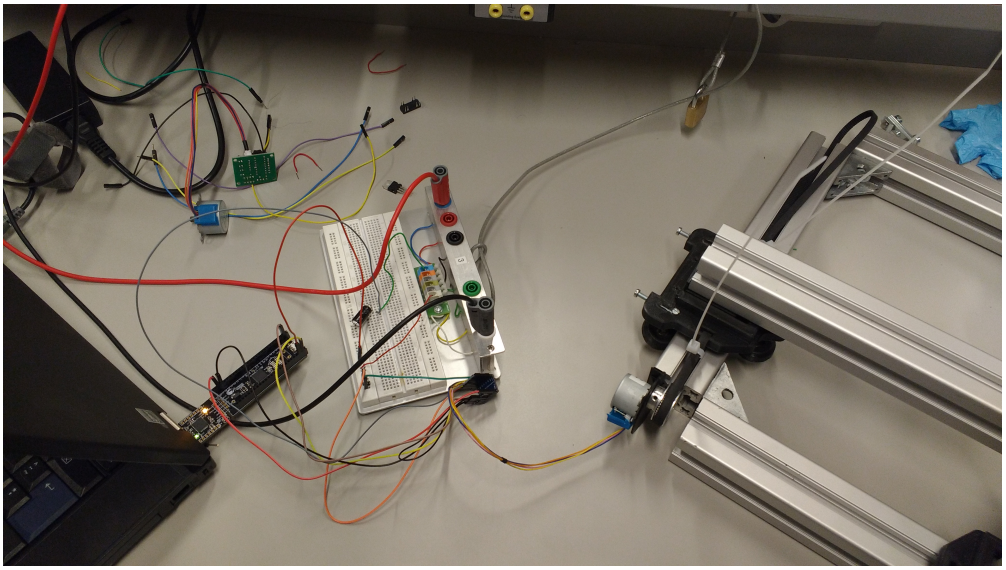
Test af motorer og sensorer er foretaget med både uni- og bipolære motorer som er foregået efter samme metode, hvor komponenterne er testet enkeltvis og i moduler. Der er ikke foretaget modultest af motorer for iskruning af proptrækker eller proptræk fordi åbningsmekanismen aldrig blev færdig. Der er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

9.3 Resultater

Akserne

Figur 9.1 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 9.1: 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 9.2 under fanebladet Value, ellers henvises til bilag xx for flere resultater.

Figur 9.2: Resultat af detektering fra sensor ved 10 cm

Locals

Name	Value	Address	Type	Radix
ADCResultX	2241		unsigned long	decimal

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.

9.4 Software

PSoC

Klasserne fra klassediagrammet blev implementeret i form af hver deres headerfil efter princippet 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(reference!!!!). Disse switches' cases bestemtes ud fra de kommandoer/beskeder, de enkelte PSoC's modtog fra hinanden eller DevKit8000. Der er ligeledes blevet oprettet en seperat *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. Dermed kunne der fastslås, hvor vidt 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.

Implementering

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.

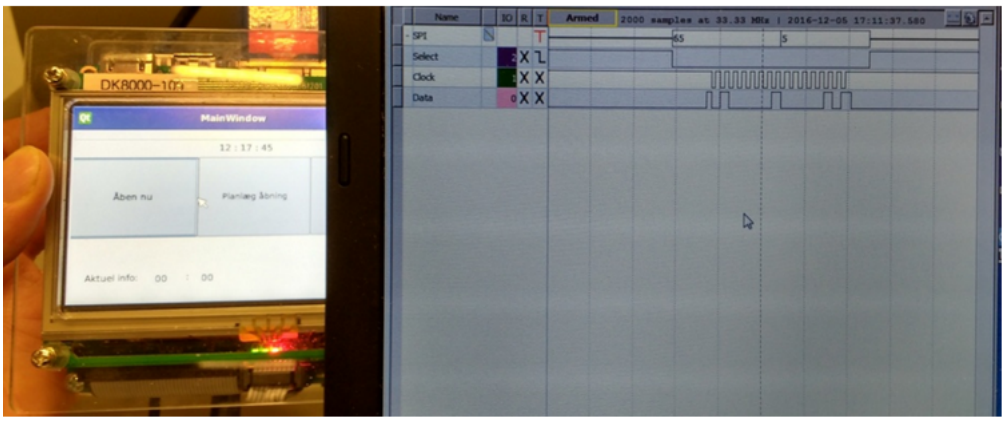
```
void MainWindow::on_aabenNu_clicked()
{
FILE *fd;
char buff[2];
fd = fopen(PSOC, "r+");
buff[0] = OPEN_BOTTLE;
buff[1] = '\0';
fwrite(buff,1,2,fd);
fclose(fd);

}
```

Figur 9.3: 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.

Test

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 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 x, kan det ses hvordan, det var muligt at sende kommandoen 5 ved at bruge "Planlæg åbning" funktionen.



Figur 9.4: Åben nu funktionen implementerets

Diskussion

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 test 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.

Seriel kommunikation

SPI Device Driver

Det nævntes tidligere i dette emnes tilhørende design-sektion(REFERENCE!!!), at en device driver for SPI-kommunikationen på DevKit8000 var blevet udleveret af skolen til brug i dette projekt. Det er denne, der er blevet brugt til dette produkt. Det var oprindeligt tænkt, at der skulle skrives en driver fra bunden af, der skulle tage udgangspunkt i HAL-øvelse 6, men grundet komplikationer med at få læst fra Master-PSoC såvel som usædvanlige men regelmæssige

bitshifts ved sending af kommandoer, overtoges den allerede færdigudviklede device driver. Dette har dog ikke været problemløst, da der, i stil med den oprindelige driver, har været problemer med at få læst fra Master-PSoC og i perioder at få skrevet til denne. Det har ligeledes været tanken, at der skulle ske et interrupt, når der gives status fra Master-PSoC til DevKit8000, men grundet den manglende adgang til driverens source-kode, er denne idé blevet forkastet.

Kommunikation mellem PSoC's

PSoC'ene er hver især implementeret med en SPI-slave-komponent (Master-PSoC'en er yderligere implementeret med en SPI-master-komponent), som benytter interrupts, der trigges 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 princippet om høj samhørighed - lav kobling.

Ønskes konfigurationen af SPI set, henvises der for DevKit8000/Master-PSoC og Master-PSoC/Slave-PSoC til henholdsvis (REFERENCE!!!) og (ARHHHHHAAHRHHRAH!!!!) i bilag.

Test af PSoC-PSoC

Test af SPI-kommunikation mellem PSoC's foregik ved at sende et tal frem og tilbage mellem disse som beskrevet i bilag(REFERENCE!!!). Værdien, der sendtes fra Master til Slave, kunne aflæses i WaveForms efter én transmission, mens det krævede 3-4 transmissioner at læse fra Slave til Master. Årsagen hertil er uvist.

Test af DevKit-PSoC

Test af SPI-kommunikation mellem DevKit8000 og PSoC foregik ved på Linux-platformen at skrive til og læse fra et SPI-device som specificeret i bilag(REREREREFEFEFEFERENC
Når der skrevet fra DevKit8000 til PSoC, sendtes de korrekte data problemfrit, men ved læsning fra PSoC modtoges et forkert resultat. Årsagen hertil er uvist.