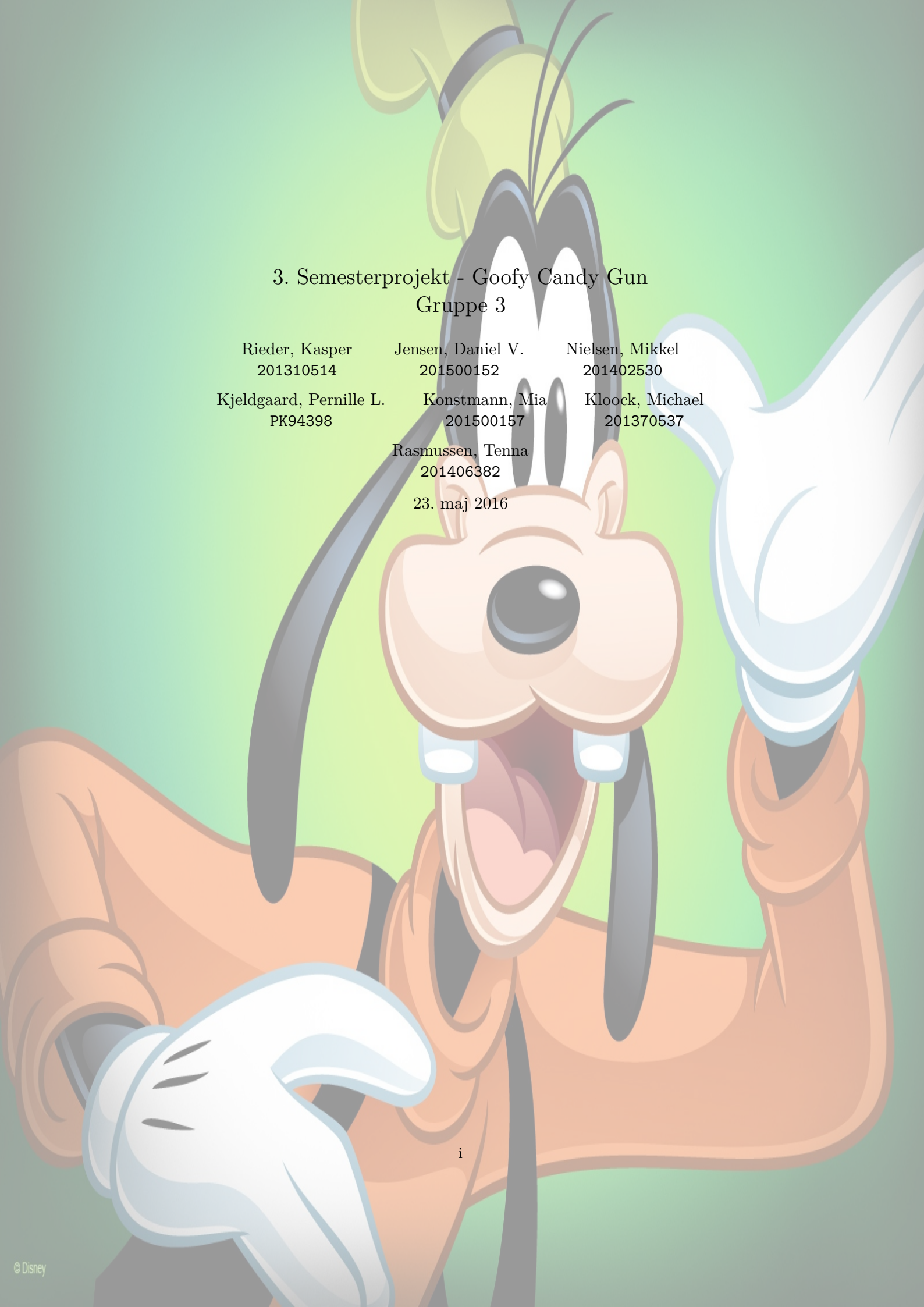


3. Semesterprojekt - Goofy Candy Gun
Gruppe 3

Rieder, Kasper
201310514

Jensen, Daniel V.
201500152

Nielsen, Mikkel
201402530

Kjeldgaard, Pernille L.
PK94398

Konstmann, Mia
201500157

Kloock, Michael
201370537

Rasmussen, Tenna
201406382

23. maj 2016

Indhold

Figurer

1 Forord

Forord I denne rapport vil produktet Goofy Candygun 3000 blive beskrevet. Goofy Candygun 3000 er udviklet i forbindelse med semesterprojektet på tredje semester på IHA.

1.1 Praktisk information

Gruppen, der har udviklet Goofy Candygun 3000, består af følgende syv personer: Kasper Rieder, Daniel Vestergaard Jensen, Mikkel Nielsen, Tenna Rasmussen, Michael Kloock, Mia Konstmann og Pernille Kjeldgaard. Gruppens vejleder er Gunvor Kirkelund. Rapporten skal afleveres fredag d. 27. maj 2016 og skal bedømmes ved en mundtlig en eksamen d. 22. juni 2016. Produktet dokumenteres foruden denne rapport med en procesrapport, et dokumentationsdokument, en prototype og diverse bilag.

1.2 Læsevejledning

2 Resumé

3 Abstract

4 Indledning

4.1 Krav til produktet

Dette projekt tager udgangspunkt i projektoplægget for 3. Semester projektet, præsenteret af *Ingeniørhøjskolen, Aarhus Universitet*. Til dette projekt er der ikke stillet krav til typen af produkt der skal udvikles, dog er der sat krav til hvad produket skal indeholde. Disse krav er som følge:

- Systemet *skal* via sensorer/aktuatorer interagere med omverdenen
- Systemet *skal* have en brugergrænseflade
- Systemet *skal* indeholde faglige elementer fra semesterets andre fag
- Systemet *skal* anvende en indlejret Linux platform og en PSoC platform

På baggrund af disse krav er der udarbejdet et system, der beskrives i afsnit 4.2.

Til dette projekt bliver systemet opbygget som en prototype. Grundet dette er der i afsnit 8 beskrevet nogle grundlæggende hardwarekomponenter til realisering af denne prototype.

4.2 Systembeskrivelse

Ønsket med dette projekt er at udvikle en mekanisk enhed der kan styres med en håndholdt controller. Med dette udgangspunkt blev forskellige muligheder undersøgt som inspirationskilde, hvor valget herefter faldt på en kanon til affyring af slik. Ideen med en kanon der affyrer slik er ikke original, som det kan ses ud fra projektet *"The Candy Cannon"* som er fundet på YouTube: **https://www.youtube.com/watch?v=VgZhQJQnnqA**. Til forskel fra *The Candy Cannon* og lignende projekter som affyrer projektiler uden et egentlig formål, vil der i dette projekt blive udviklet en kanon til brug i et spil. Kanonen affyres og styres af spillerne via en controller. Altså skal projektet ende med en kanon som indgår i et to personersspil, f.eks. til brug ved fester og andre sociale begivenheder.

I dette projektet skal der altså udvikles en slikkanon til spillet *Goofy Candygun 3000*. Denne slikkanon skal kunne skyde med slik. Dette kunne for eksempel være M&M's eller Skittle's.

Goofy Candygun 3000 er et spil til to personer, hvilket gør det velegnet i sociale sammenhænge. Spillet går ud på at opnå flest point ved at ramme et mål. Hver spiller får et bestemt antal skud. Efter skuddene er opbrugt, er vinderen spilleren med flest point.

Et typisk brugerscenarie er, at spillerne bestemmer antallet af skud for runden. Når dette er gjort, er spillet igang. Herefter går Wii-nunchucken på skift mellem spillerne for hvert skud. Dette fortsættes indtil skuddene er opbrugt. Vinderen er spilleren med flest point. Spillets statistikker vises løbende på brugergrænsefladen.

Det endelige produkt omfatter:

- En brugergrænseflade, hvor spilstatistikker fremvises til deltagerne. Dette er blandt andet:

  Pointvisning

  Kanonens vinkel

  Antal resterende skud

- En motor, der drejer kanonen om forskellige akser

  Dette styres med en Wii-nunchuck

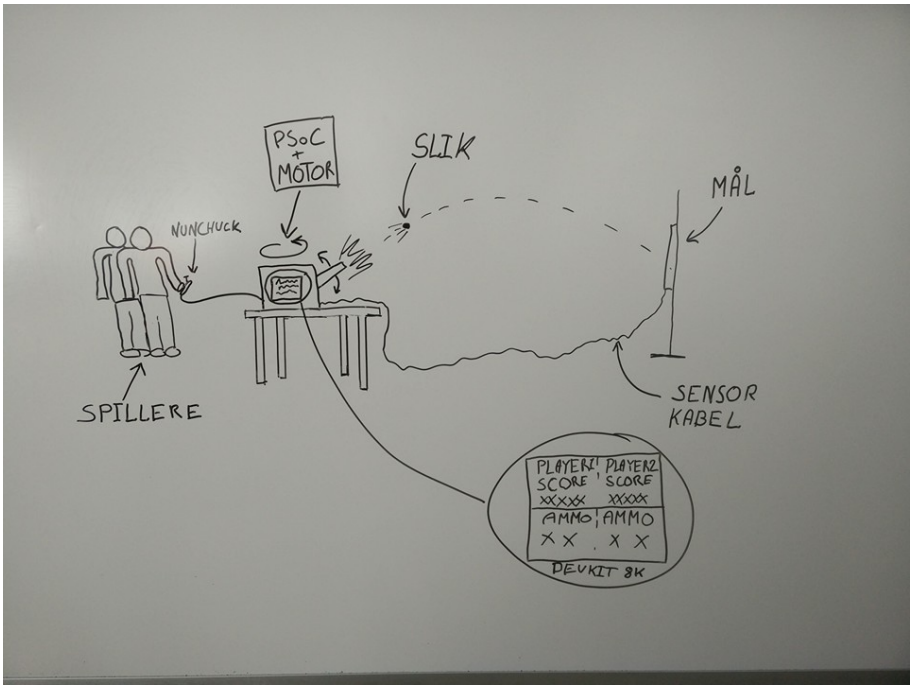
- Et mål, der kan registrere spillernes skud

Med baggrund i krav stillet af organisationen IHA, bliver produktet udviklet med følgende funktioner:

- DevKit 8000 som den indlejrede linux-platform til spillets grafiske brugergrænseflade.
- Motorer til styring af kanonen.
- Sensorer.
- PSoC 4, anvendt til styring af motorer og kommunikation med sensorer.

4.3 Rigt Billede

På figur 1 ses et rigt billede af det ønskede produkt. Billedet beskriver brugerscenariet.



Figur 1: Rigt Billede af det endelige produkt

4.4 Ansvarsområder

I løbet af projektet vil projektgruppen blive opdelt i to hovedgrupper - 'hardware' og 'software'. Softwaregruppen vil desuden stå for grænsefladeprogrammering mellem software og hardware. Disse grupper vil have til ansvar at designe og implementere hhv. hardware og software til projektet. Hardwaregruppen vil bestå af de personer, der læser til elektroingeniør (Mikkel Nielsen og Pernille Kjeldgaard). Softwaregruppen vil bestå af de personer, der læser til IKT-ingeniør (Kasper Rieder, Michael Kloock, Tenna Rasmussen, Mia Konstmann og Daniel Jensen).

På tabel **??** ses opdelingen af ansvarsområder mellem projektgruppens medlemmer. Her bruges bogstavet *P* til at angive *primært* ansvar, hvor bogstavet *S* angiver *sekundært* ansvar.

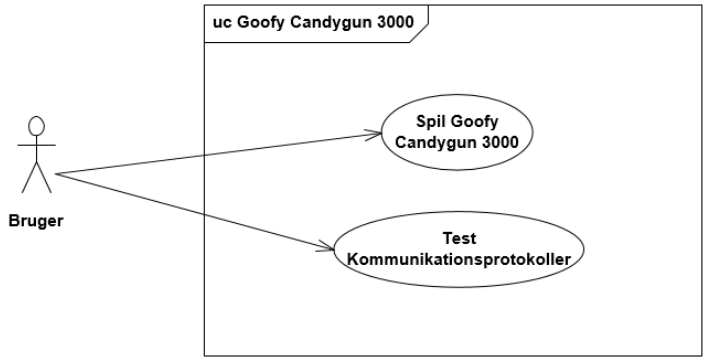
Ansvarsområder	Daniel Jensen	Mia Konstmann	Mikkel Nielsen	Kasper Rieder	Michael Kloock	Tenna Rasmussen	Pernille Kjeldgaard
I2C Kommunikationsprotokol PSoC Software	P	S		P			
Nunchuck PSoC Software	S	P		S			
SPI Kommunikationsprotokol PSoC Software	P	S		P			
SPI Driver Devkit Software		S		S		P	
Brugergrænseflade Devkit Software					P		
Motorstyring PSoC Software		P	P	S			
Use Case 2 Implementering	P	S		P			
H-bro							
Affyringsmekanisme							

Tabel 1: Ansvarsområder tabel

5 Krav

Ud fra projektformuleringen er der formuleret en række krav til projektet. Disse indebærer to use cases og et antal ikke-funktionelle krav.

På figur 2 ses Use Case diagrammet for systemet.



Figur 2: Use case diagram

5.1 Aktørbeskrivelse

På figur 2 ses det at der er én primær bruger for systemet, brugeren. Denne beskrives her.

5.1.1 Aktør - Bruger

Aktørens Navn:	Bruger
Alternativ Navn:	Spiller
Type:	Primær
Beskrivelse:	Brugeren initierer Goofy Candy Gun, ved at vælge spiltype på brugergrænsefladen. Derudover har brugeren mulighed for at stoppe spillet igennem brugergrænsefladen. Brugeren vil under spillet interagere med Goofy Candy Gun gennem Wii-Nunchucken. Brugeren starter også Goofy Candy Gun system-testen for at verificere om det er operationelt.

5.2 Use case beskrivelse

5.2.1 Use case 1

Denne Use Case indebærer at spille Goofy Candygun 3000, og den initieres af brugeren. Det første der sker i use case er, at brugeren skal vælge, hvilken type spil han/hun gerne vil spille. Det betyder, at det skal bestemmes, om det skal være et enpersonsspil, topersonersspil eller om det skal være party mode. Herefter skal der vælges, hvor mange skud et spil skal vare og disse skal puttes i magasinet. Når dette er gjort kan spillet gå i gang. Brugeren indstiller kanonen med Wii-nunchuck og affyrer den. Herefter lader systemet et nyt skud og samme procedure gentages. Til slut vises information om spillet på brugergrænsefladen, brugeren afslutter spillet ved at trykke på knappen på brugergrænsefladen og denne vender tilbage til starttilstanden. **#ref** til fully dressed use case 1.

5.2.2 Use case 2

Denne Use Case skal bruges til at teste systemets kommunikationsprotokoller, altså om der er forbindelse mellem alle hardwareblokke forbundet via systemets busser, og om disse kan fortolke kommandoer korrekt. Hvis der bliver fundet fejl, raporteres disse til brugeren via brugergrænsefladen. Use Casen initieres af brugeren, hvor der herefter bliver sendt informationer ud til de forskellige dele af systemet, som så gerne skal sende svar tilbage igen. Hvis det sker for alle dele, vil brugergrænsefladen til sidst meddele, at use casen er gennemført. **#ref** til fully dressed use case 2.

5.3 Ikke-funktionelle krav

De ikke-funktionelle krav siger noget om, hvordan systemet skal bygges, og hvilke specifikationer det skal have. I dette tilfælde er der udarbejdet syv krav, hvor der er et der siger noget om, hvordan kanonen skal kunne drejes. Nogle siger noget om kanonens størrelse, hvor der også er specificeret, hvor stort projektilet den skal affyre må være og et siger, hvor langt den skal kunne skyde. Der er et, der siger, hvor lang tid det må tage at skyde et projektil afsted og hvor hurtig den skal være til at lade kanonen igen. Endelig er der et, der specificerer, hvordan den grafiske brugergrænseflade skal se ud. Deciderede værdier for de enkelte krav kan læses i bilag. **#ref**

6 Projektafgrænsning

Ud fra MoSCoW-princippet er der udarbejdet en række krav efter prioriteringerne 'must have', 'should have', 'could have' og 'won't have'. Dette er for at gøre det tydeligt, hvad der er vigtigt, der bliver udviklet først, og hvad der godt kan vente til senere. Disse krav er som følger:

- Produktet must have:
  - En motor til styring af kanonen
  - En grafisk brugergrænseflade
  - En Wii-nunchuck til styring af motoren
  - En kanon med en affyringsmekanisme
  - En system test til diagnosering af fejl
- Produktet should have:
  - Et mål til registering af point
  - En lokal ranglistestatistik
- Produktet could have:
  - En partymode-indstilling til over to spillere
  - Trådløs Wii-nunchuckstyring
  - Afspilning af lydeffekter
- Produktet won't have:
  - Et batteri til brug uden strømforsyning
  - Online ranglistestatistik

"Must Have"kravende har højst prioritering i projektet. Det vil altså sige, at kravene under punkterne 'should have' og 'could have' har lavere prioritet. For at kravene under punktet 'must have' er opfyldt, skal use case 2, *Test Kommunikationsprotokoller* implementeres. Grunden til dette er, at use case 2 kræver at hardwareblokke er forbundet korrekt via busser, og at der er software allokeret som gør brug af kommunikationsprotokoller. Derfor blev prioriteringen i dette projekt, at use case 2 skulle implementeres til fulde, inden der kunne startes på at implementere use case 1. Det havde dog også høj prioritet at have et produkt, der fysisk kunne styres, samt skyde, hvilket betød, at selvom affyringsmekanismen ikke indgår i use case 2, blev det alligevel prioriteret højt at få implementeret denne i systemet.

7 Metode

I arbejdet med projektet er det vigtigt at anvende gode analyse- og designmetoder. Dermed er det muligt at komme fra den indledende idé til det endelige produkt med lavere risiko for misforståelser og kommunikationsfejl undervejs. Det er også en stor fordel, hvis de metoder, der anvendes, er intuitive og har nogle fastlagte standarter. Det gør det muligt for udenforstående at sætte sig ind i, hvordan projektet er udviklet og designet. Dermed bliver projektet og dets produkt i højere grad uafhængigt af enkeltpersoner, og det bliver muligt at genskabe produktet.

7.1 SysML

Til dette projekt er der anvendt *SysML* som visuelt modelleringsværktøj, til at skabe diagrammer. SysML blev valgt, da det er en anerkendt industristandard[**?**] , hvilket betyder at det er mere universalt brugbart. Specifikationen for SysML kan findes i bilag, *SysML Specification.pdf*.

SysML er et modelleringssprog, der bygger videre på det meget udbredte modelleringssprog, UML. Men hvor UML hovedsagligt er udviklet til brug i objekt orienteret software udvikling, er SysML i højere grad udviklet med fokus på beskrivelse af både software- og hardwaresystemer. Det gør det særdeles velegnet til dette projekt, som netop består af både software- og hardwaredele. Det er derfor også i store dele af arbejdet med analyse og design af produktet.

SysML specifikationen er omfattende og beskriver mange diagramtyper. Til dette projekt er der valgt diagramtyper alt efter deres nytteværdi for modellering af hardware og software, som vil summeres her.

Til modellering af hardware er der i rapport og dokumentation gjort brug af strukturdiagrammerne *Block Definition Diagrams* (BDD) samt *Internal Block Diagrams* (IDB). Disse diagrammer er brugt til at beskrive systemets hardwarekomponenter, deres signaler, og forbindelserne mellem dem.

Til modellering af software er der i rapport og dokumentation gjort brug af struktur- og adfærdsdiagrammerne *Block Definition Diagrams*, *Sequence Diagrams* (SD), og *State Machine Diagrams* (SMD). Disse diagrammer er brugt til at beskrive systemets softwarekomponenter i form af klasser, relationer mellem klasserne, samt hvordan disse klasser interagerer med hinanden og hvilket tilstande de kan være i.

7.1.1 Afvigelser fra standard brug

I SysML sekvensdiagrammer bruges beskedudveksling typisk til at repræsentere metoder på de objekter der kommunikeres med. Denne fremgangsmåde bruges i denne raport, dog er der nogle sekvensdiagrammer der afviger ved at beskedudvekslingen repræsenterer handlinger påført på objekter. Et eksempel på denne afvigelse kan ses i afsnit 9.3.2, figur 10. Denne afvigelse blev brugt for at kunne tydeliggøre interaktionen mellem systemets komponenter på en naturlig måde.

8 Analyse

Til projektets prototype er der brugt nogle grundlæggende hardwarekomponenter til realisering af systemets arkitektur. Disse hardwarekomponenter vil blive præsenteret følgende.

8.1 DevKit8000

DevKit8000 er en indlejret linux platform med et tilkoblet 4.3 tommer touch-display.

Denne indlejrede linux platform blev valgt, da den allerede fra start understøtter interfacing med et touch-display. Dette kan bruges til systemets brugergrænseflade. DevKit8000 understøtter desuden de serielle kommunikationsbusser SPI og I2C, hvilket er typiske busser der bliver brugt til kommunikation med sensorer samt aktuatorer.

DevKit8000 platformen er også brugt gennem undervisning på IHA, hvilket betyder at der er god adgang til de compilers der skal bruges til platformen.

8.2 Programmable System-on-Chip (PSoC)

PSoC er en microcontroller der kan omprogrammeres via et medfølgende *Integrated Development Environment* (IDE). PSoC'en understøtter multiple *Seriel Communication Busses* (SCB), hvilket gør denne microcontroller ideel til dette system, da der skal kommunikeres med sensorer og aktuatorer.

8.3 Wii-Nunchuck

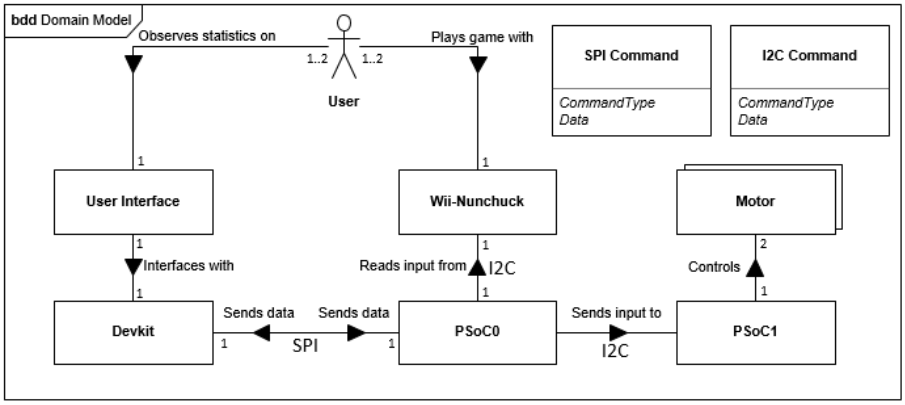
Til brugerstyring af systemets kanon er en Wii-Nunchuck controller valgt. Denne controller er valgt, da den har gode egenskaber til styring af en kanon. Wii-Nunchucken understøtter desuden en af PSoC'ens kommunikationsprotokoller, så den nemt kan kommunikere med resten af systemet.

9 Systemarkitektur

Dette afsnit præsenterer systemets arkitektur i en grad der gør det muligt at forstå sammensætningen mellem dets hardware og software komponenter.

9.1 Domænemodel

På figur 3 ses domænemodellen af systemet. Denne har til formål at præsentere forbindelserne mellem systemets komponenter, samt dets grænseflader.



Figur 3: Systemets domænemodel

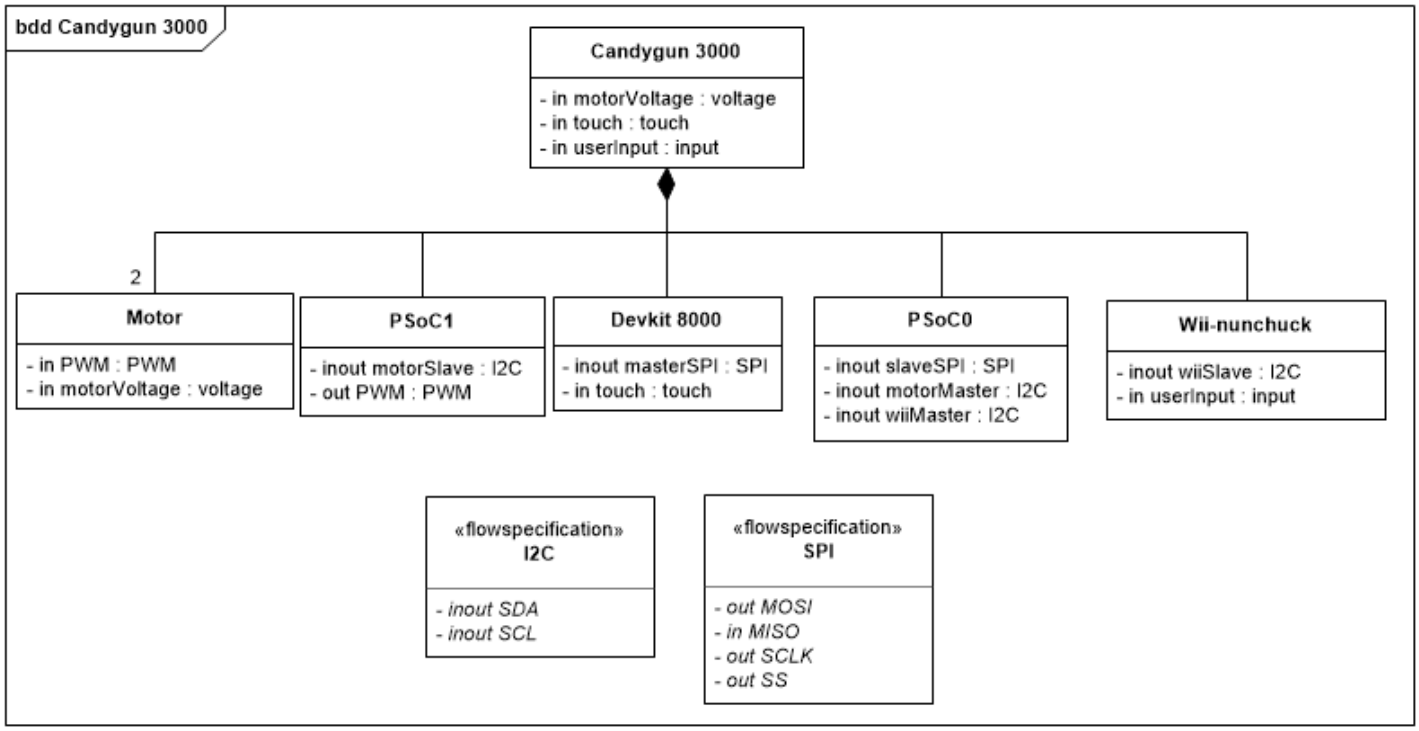
Her repræsenteres hardware som blokke forbundet med associeringer. Associeringerne viser grænsefladerne mellem de forbundne hardware komponenter (Enten *SPI* eller *I2C*), samt retningen af kommunikationen. Af modellen fremstår konceptuelle kommandoer for grænsefladerne, som beskriver deres nødvændige attributter.

Domænemodellen er brugt til at udlede grænseflader for systemet, samt potentielle hardware- og softwarekomponenter. Hvad der er udledt af domænemodellen i forhold til grænseflader og komponenter, og hvordan dette bruges omdiskuteres i de følgende arkitektur afsnit.

9.2 Hardware

9.2.1 BDD

På figur 4 ses BDD'et for systemet.



Figur 4: BDD af systemets hardware

Her vises alle hardwareblokke fra domænemodellen (figur 3) med nødvændige indgange og udgange for de fysiske signaler. Yderligere ses det at flow specifikationer er defineret for de ikke-atomare forbindelser *I2C* samt *SPI*, da disse er busser bestående af flere forbindelser. Der henvises til **IBD AFSNIT** for en detaljeret model af de fysiske forbindelser mellem hardwareblokkende.

9.2.2 Blokbeskrivelse

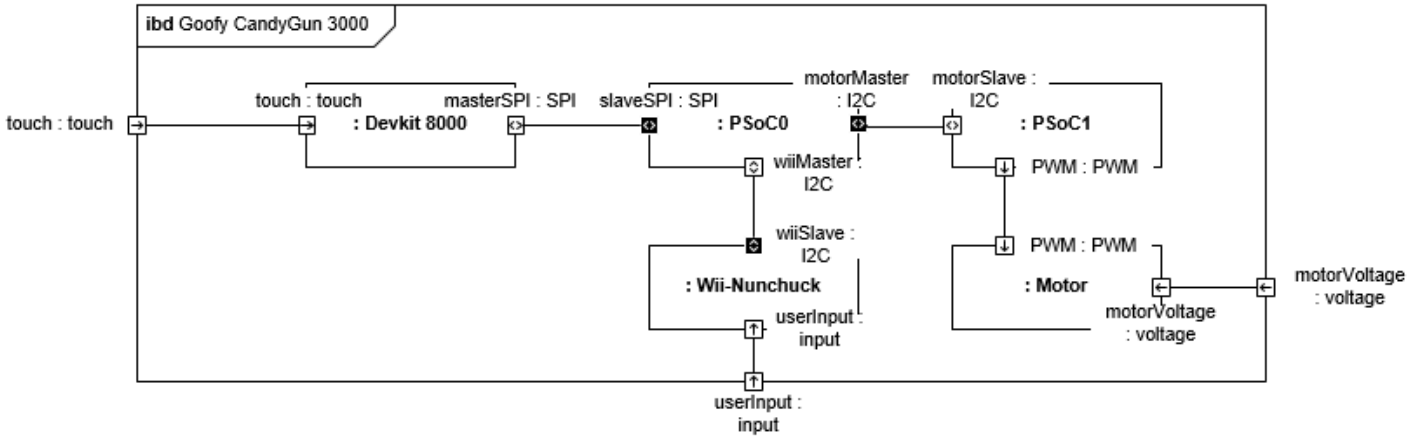
Følgende afsnit indeholder en blokbeskrivelse samt en flowspecifikation for I2C og SPI. I flowspecifikationen beskrives I2C og SPI forbindelserne mere detaljeret fra en masters synsvinkel.

Bloknavn	Beskrivelse
Devkit 8000	DevKit 8000 er en embedded Linux platform med touch-skærm, der bruges tilbrugergrænsefladen for produktet. Brugeren interagerer med systemet og ser status for spillet via Devkit 8000.
Wii-Nunchuck	Wii-Nunchuck er controlleren som brugeren styrer kanonens retning med.
PSoC0	PSoC0 er PSoC hardware der indeholder software til I2C og SPI kommunikationen og afkodning af Wii-Nunchuck data. PSoC0 fungerer som I2C masterog SPI slave. Denne PSoC er bindeleddet mellem brugergrænsefladen og restenaf systemets hardware.
Motor	Motor blokken er Candy Gun 3000's motorerer, der anvendes til at bevægekanonen i forskellige retninger.
PSoC1	PSoC1 er PSoC hardware der indeholder software til I2C kommunikation og styring af Candy Gun 3000's motorer. PSoC1 fungerer som I2C slave.
SPI (FlowSpecification)	SPI (FlowSpecification) beskriver signalerne der indgår i SPI kommunikation.
I2C (FlowSpecification)	I2C (FlowSpecification) beskriver signalerne der indgår i I2C kommunikation.

Tabel 2: Blokbeskrivelse

9.2.3 IBD

På figur 5 ses IBD'et for systemet. Figuren viser hardwareblokkene med de fysiske forbindelser beskrevet i BDD'et (figur 4).



Figur 5: IBD af systemets hardware

Her vises alle hardwareblokke med de fysiske forbindelser beskrevet i BDD'et (figur 4).

Det ses at systemet bliver påvirket af tre eksterne signaler: *touch*, *input*, samt *voltage*. *touch* er input fra brugeren når der interageres med brugergrænsefladen. *input* er brugerens interaktion med Wii-Nunchuk. *voltage* er forsyningsspænding til systemet.

9.2.4 Signalbeskrivelse

Bloknavn	**Funktionsbeskrivelse**	**Signaler**	**Signalbeskrivelse**
Devkit 8000	Fungerer som grænseflade mellem bruger og systemet samt SPI master.	masterSPI	Type: SPI Spændingsniveau: 0-5V Hastighed: ?? Beskrivelse: SPI bussen hvori der sendes og modtages data.
		touch	Type: touch Beskrivelse: Brugertryk på Devkit 8000 touchdisplay.
PSoC0	Fungerer som I2C master for PSoC1 og Wii-Nunchuck samt SPI slave til Devkit 8000.	slaveSPI	Type: SPI Spændingsniveau: 0-5V Hastighed: ?? Beskrivelse: SPI bussen hvori der sendes og modtages data.
		wiiMaster	Type: I2C Spændingsniveau: ?? Hastighed: ?? Beskrivelse: I2C bussen hvor der modtages data fra Nunchuck.
		motorMaster	Type: I2C Spændingsniveau: 0-5V Hastighed: 100kbit/sekund Beskrivelse: I2C bussen hvor der sendes afkodet Nunchuck data til PSoC1.

PSoC1	Modtager nunchuck-input fra PSoC0 og omsætter dataene til PWM signaler.	motorSlave	Type: I2C Spændingsniveau: 0-5V Hastighed: 100kbit/sekund Beskrivelse: Indeholder formatteret Wii-Nunchuck data som omsættes til PWM-signal.
		PWM	Type: PWM Frekvens: 22kHz PWM %: 0-100% Spændingsniveau: 0-5V Beskrivelse: PWM signal til styring af motorens hastighed.
Motor	Den enhed der skal bevæge kanonen	PWM	Type: PWM Frekvens: 22kHz PWM%: 0-100% Spædingsniveau: 0-5V Beskrivelse: PWM signal til styring af motorens hastighed.
		motorVoltage	Type: voltage Spændingsniveau: 12V Beskrivelse: Strømforsyning til motoren
Wii-nunchuck	Den fysiske controller som brugeren styrer kanonen med.	wiiSlave	Type: I2C Spændingsniveau: 0-5V Hastighed: 100kbit/sekund Beskrivelse: Kommunikationslinje mellem PSoC1 og Wii-Nunchuck.
		userInput	Type: input Beskrivelse: Brugerinput fra Wii-Nunchuck.

SPI	Denne blok beskriver den ikke-atomiske SPI forbindelse.	MOSI	Type: CMOS Spændingsniveau: 0-5V Hastighed: ?? Beskrivelse: Binært data der sendes fra master til slave.
		MISO	Type: CMOS Spændingsniveau: 0-5V Hastighed: ?? Beskrivelse: Binært data der sendes fra slave til master.
		SCLK	Type: CMOS Spændingsniveau: 0-5V Hastighed: ?? Beskrivelse: Clock signalet fra master til slave, som bruges til at synkronisere den serielle kommunikation.
		SS	Type: CMOS Spændingsniveau: 0-5V Hastighed: ?? Beskrivelse: Slave-Select, som bruges til at bestemme hvilken slave der skal kommunikeres med.
I2C	Denne blok beskriver den ikke-atomiske I2C forbindelse.	SDA	Type: CMOS Spændingsniveau: 0-5V Hastighed: ?? Beskrivelse: Databussen mellem I2C masteren og I2C slaver.

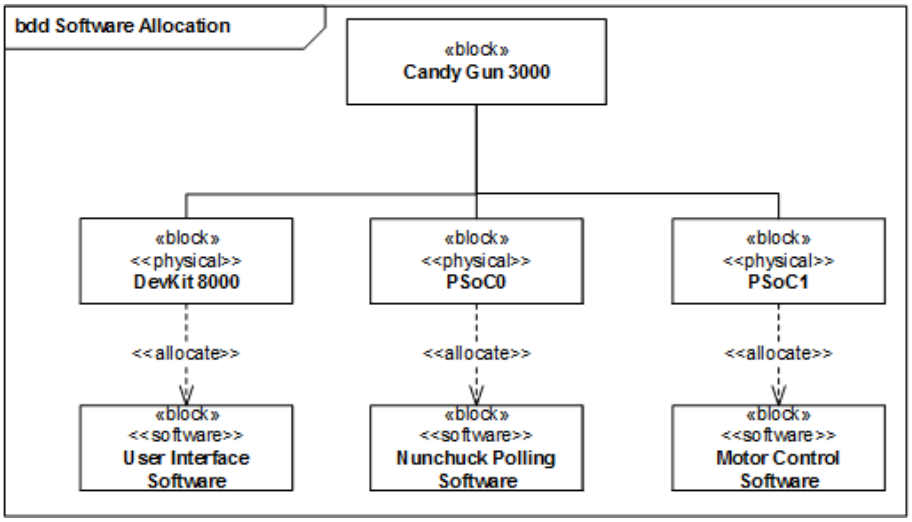
		SCL	Type: CMOS Spændingsniveau: 0-5V Hastighed: ?? Beskrivelse: Clock signalet fra master til lyttende I2C slaver, som bruges til at synkronisere den serielle kommunikation.

Tabel 3: Signalbeskrivelse

9.3 Software

9.3.1 Software Allokering

Domænemodellen i figur 3, side 12, præsenterer systemets hardwareblokke. På figur 6 ses et software allokeringsdiagram, som viser hvilke hardwareblokke der har softwaredele af systemet allokeret på sig.



Figur 6: Systemets software allokeringer

Det kan her ses at systemet består af tre primære softwaredele: *User Interface Software*, *Nunchuck Polling Software*, *Motor Control Software*. Disse er fordelt over de tre viste CPU'er.

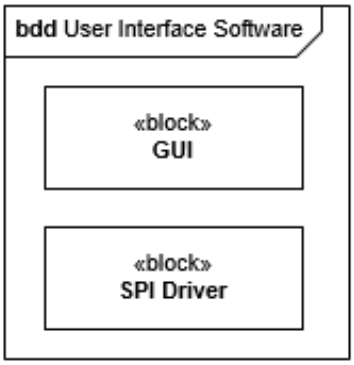
På tabel 4 er hvert allokeret software komponent beskrevet.

User Interface Software	Dette allokerede software er brugergrænsefladen som brugeren interagerer med på DevKit8000 touch-skærmen.
Nunchuck Polling Software	Dette allokerede software har til ansvar at polle Nunchuck tilstanden og videresende det til PSoC1.
Motor Control Software	Dette allokerede software har til ansvar at bruge den pollede Nunchuck data fra PSoC0 til motorstyring samt affyringsmekanismen.

Tabel 4: Beskrivelse af den allokerede software

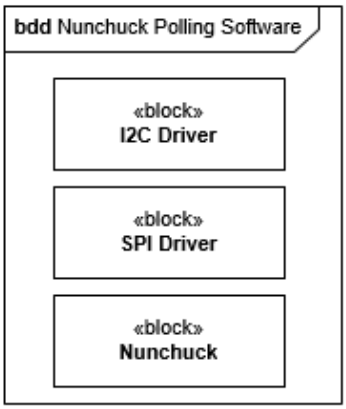
På figurerne: 7, 8, 9 ses et overblik over konceptuelle klasser der repræsenterer ansvarsområder de primære softwaredele får. For en beskrivelse af design og implementering af disse primære softwaredele henvises til **DESIGN OG IMPLEMENTERING #ref**.

På figur 7 ses det at *User Interface Software*, allokeret på DevKit 8000, har ansvar for brugergrænsefladen samt en SPI Driver til kommunikation med PSoC0.



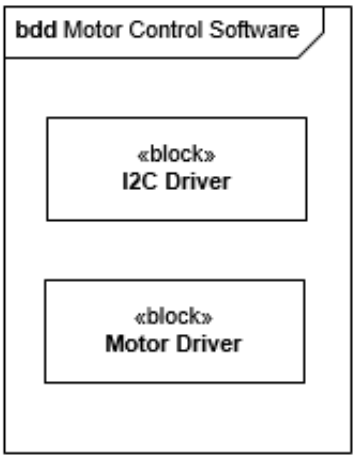
Figur 7: Systemets software allokeringer

På figur 8 ses det at *Nunchuck Polling Software*, allokeret på PSoC0, har ansvar for en I2C Driver, SPI Driver, samt en Nunchuck API. I2C Driveren skal bruges til kommunikation med den fysiske Nunchuck controller samt PSoC1. SPI Driveren skal bruges til kommunikation med DevKit 8000. Nunchuck API'en bruges for at tilgå data'en fra den fysiske Nunchuck controller.



Figur 8: Systemets software allokeringer

På figur 9 ses det at *Motor Control Software*, allokeret på PSoC1, har ansvar for en I2C Driver til kommunikation med PSoC0, samt en Motor Driver til motorstyring.



Figur 9: Systemets software allokeringer

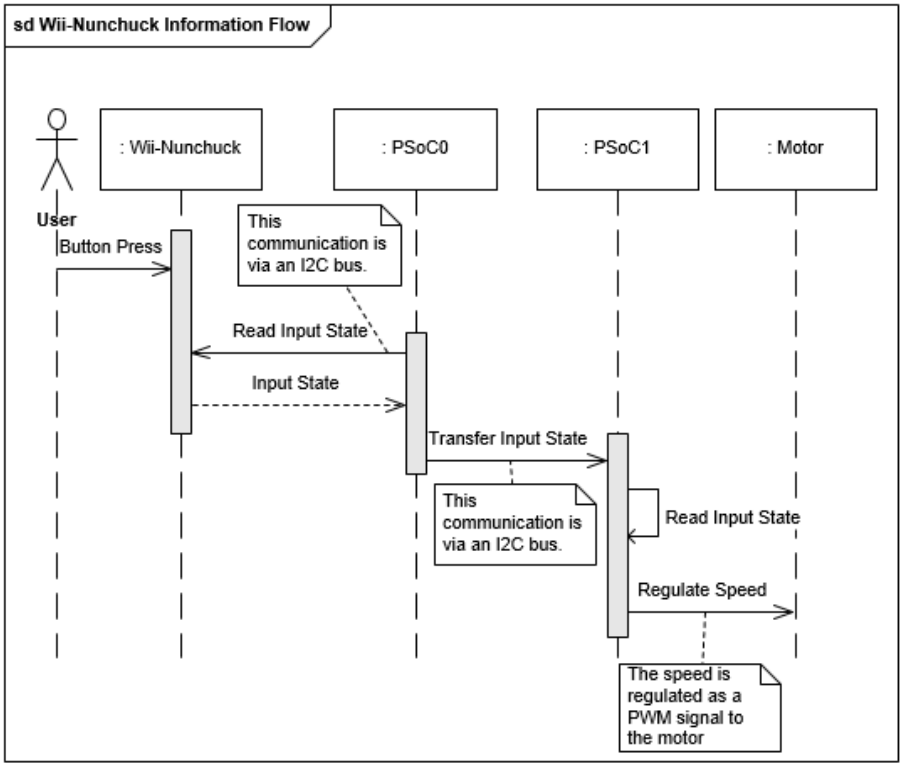
9.3.2 Informationsflow i systemet

Dette afsnit har til formål at demonstrere sammenhængen mellem softwaren på CPU'erne og resten af systemet, samt at beskrive og identificere grænsefladerne brugt til kommunikation mellem dem. Yderligere vil klasseidentifikation også blive vist, hvor disse klasser vil specificeres i Design og Implementering.

Wii-Nunchuck Information Flow

En essentiel del af systemet er at kunne styre motoren ved brug af Wii-Nunchuck controlleren. På figur 10 vises gennemløbet af Wii-Nunchuck input data fra

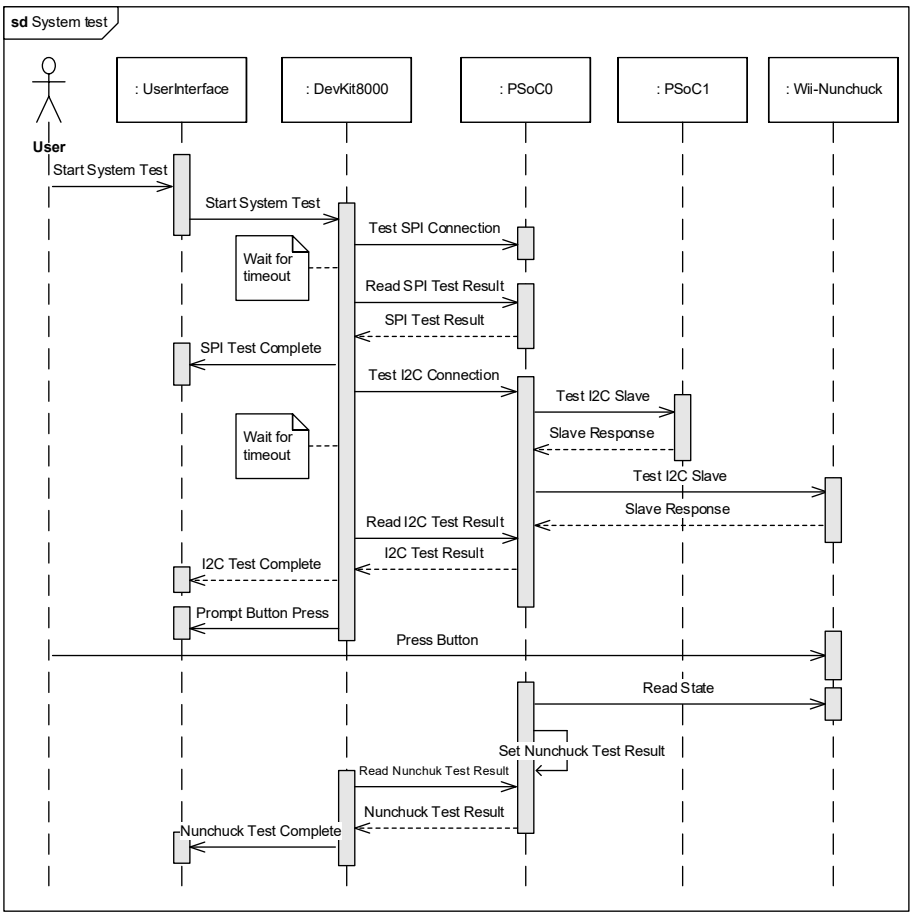
Wii-Nunchuken til motoren, med de relevante CPU'er angivet. Her ses det at input data fra Wii-Nunchuck kontinuert bliver aflæst af PSoC0. Det bemærkes her at grænsefladen mellem PSoC0 og Wii-Nunchuck er en I2C bus. Efter at PSoC0 har aflæst input data'en, overføres den til PSoC1. Grænsefladen mellem disse to PSoCs er også en I2C bus. PSoC1 kan til slut oversætte modtaget input data til PWM signaler til motorstyring samt affyring.



Figur 10: Wii-Nunchuck Input Data Forløb

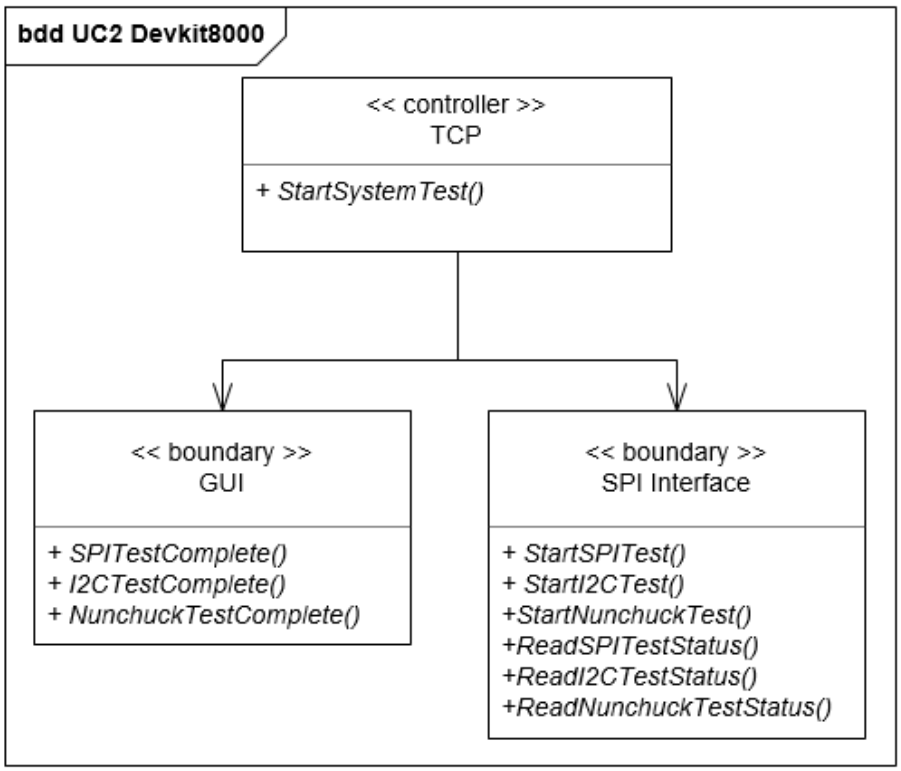
System Test

Use Case 2 beskriver test af systemet før spillet startes. På figur 11 vises test sekvensen for en vellykket test. Det ses her at system testen sker sekventielt, og tester systemets SPI og I2C busser.

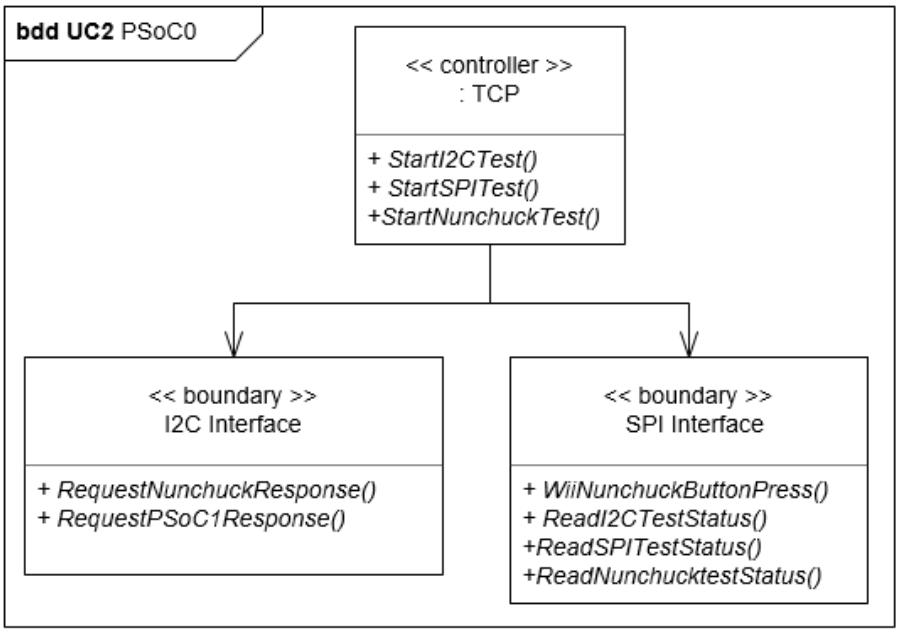


Figur 11: System Test Forløb

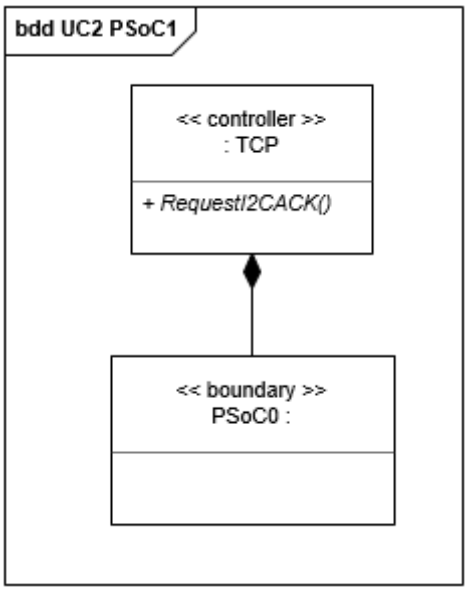
Softwaren til systemet er udarbejdet for hver CPU (se evt. software allokeringsdiagrammet på side 19 figur 6) ud fra en applikationsmodel, der kan ses i dokumentationen side (**Indsæt reference til applikationsmodel dokumentations #ref**). Ud fra disse applikationsmodeller er der udarbejdet klassediagrammer, der viser software klasserne på den givne CPU, samt klassernes relationer til hinanden. Et indledende forslag til klassediagrammerne for hver CPU i systemet kan ses på figur 12, 13 og 14.



Figur 12: Klassediagram for DevKit8000 CPU



Figur 13: Klassediagram for PSoC0 CPU



Figur 14: Klassediagram for PSoC1 CPU

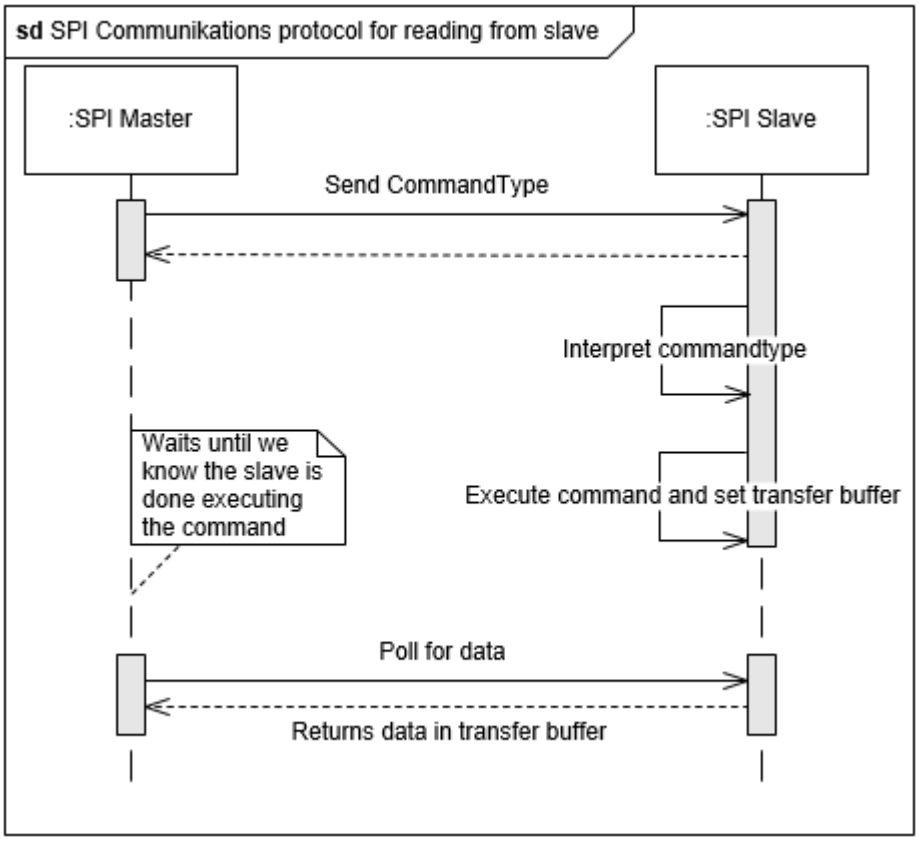
9.3.3 SPI Kommunikations Protokol

I afsnit 9.2.3 **IBD** ses på figur 5 at Devkit8000 og PSoC0 kommunikerer via en SPI bus. Kommunikationen foregår ved at der sendes kommandotyper imellem de to enheder, SPI-master og SPI-slave. På tabel 5 ses en de anvendte kommandotyper samt en kort beskrivelse for hver af disse.

Kommandotype	Beskrivelse	Binær Værdi	Hex Værdi
START_SPI_TEST	Sætter PSoC0 i 'SPI-TEST' mode	1111 0001	0xF1
START_I2C_TEST	Sætter PSoC0 i 'I2C-TEST' mode	1111 0010	0xF2
START_NUN-CHUCK_TEST	Sætter PSoC0 i 'NUNCHUCK-TEST' mode	1111 0011	0xF3
SPI_OK	Signalerer at SPI-testen blev gennemført uden fejl	1101 0001	0xD1
I2C_OK	Signalerer at I2C-testen blev gennemført uden fejl	1101 0010	0xD2
I2C_FAIL	Signalerer at I2C-testen fejlede	1100 0010	0xC2
NUNCHUCK_OK	Signalerer at NUNCHUCK-testen blev gennemført uden fejl	1101 0011	0xD3
NUNCHUCK_FAIL	Signalerer at NUNCHUCK-testen fejlede	1100 0011	0xC3

Tabel 5: SPI kommunikation kommandotyper

Kommunikation på en SPI-bus foregår ved bit-shifting. Dette betyder at indholdet af masterens transfer buffer bliver skiftet over i slavens read buffer og omvendt. Kommunikationen foregår i fuld duplex, derfor skal der foretages to transmissioner for at aflæse data fra en SPI-slave. Dette skyldes at slaven skal vide hvilken data der skal klargøres i transfer-bufferen og klargøre denne buffer før masteren kan aflæse denne. Her skal der tages højde for at en længere proces skal gennemføres før slaven har klargjort transfer-buffern. Derfor skal masteren, efter at have sendt en kommandotype, vente et bestemt stykke tid før der at aflæses fra slavens transfer-buffer. Denne sekvens er illustreret med et sekvensdiagram på figur 15.



Figur 15: Sekvensdiagram for aflæsning data fra en SPI-slave

Designvalg

SPI kommunikations protokollen er designet ud fra det grundlag at DevKit8000, som SPI master, skal læse tilstande fra SPI slaven ved brug af en timeout model. Ved timeout model menes der at DevKit8000 sender en kommandotype ud, venter et bestemt antal sekunder, og herefter læser værdien fra SPI slaven. Denne model er modelleret på figur 15

Et alternativt design ville være at generere et interrupt til SPI masteren når slaven havde data der skulle læses. Til dette system blev timeout modellen dog valgt, da det hardwaremæssigt var simplere at implementere, samt at alt nødvændig funktionalitet kan implementeres med den valgte model.

9.3.4 I2C Kommunikations Protokol

I afsnit 9.2.3 **IBD** ses på figur 5 at tre hardwareblokke kommunikerer via en I2C bus. Til denne I2C kommunikation er der defineret en protokol, som bestemmer hvordan modtaget data skal fortolkes. Denne protokol beskrives følgende.

I2C gør brug af en indbygget protokol, der anvender adressering af hardware-enheder til identificering af hvilken enhed der kommunikeres med. Derfor har

hardwareblokkene som indgår i I2C kommunikationen fået tildelt adresser. På tabel 6 ses adresserne tildelt systemets PSoCs.

I2C Adresse bits	7	6	5	4	3	2	1	0 (R/W)
PSoC0	0	0	0	1	0	0	0	0/1
PSoC1	0	0	0	1	0	0	1	0/1
Wii-Nunchuck	1	0	1	0	0	1	0	0/1

Tabel 6: I2C bus addresser

Da I2C dataudveksling sker bytevist, er kommunikations protokollen opbygget ved, at kommandoens type indikeres af den første modtagne byte. Herefter følger N-antal bytes som er kommandoens tilhørende data. N er et vilkårligt heltal og bruges i dette afsnit når der refereres til en mængde data-bytes der sendes med en kommandotype.

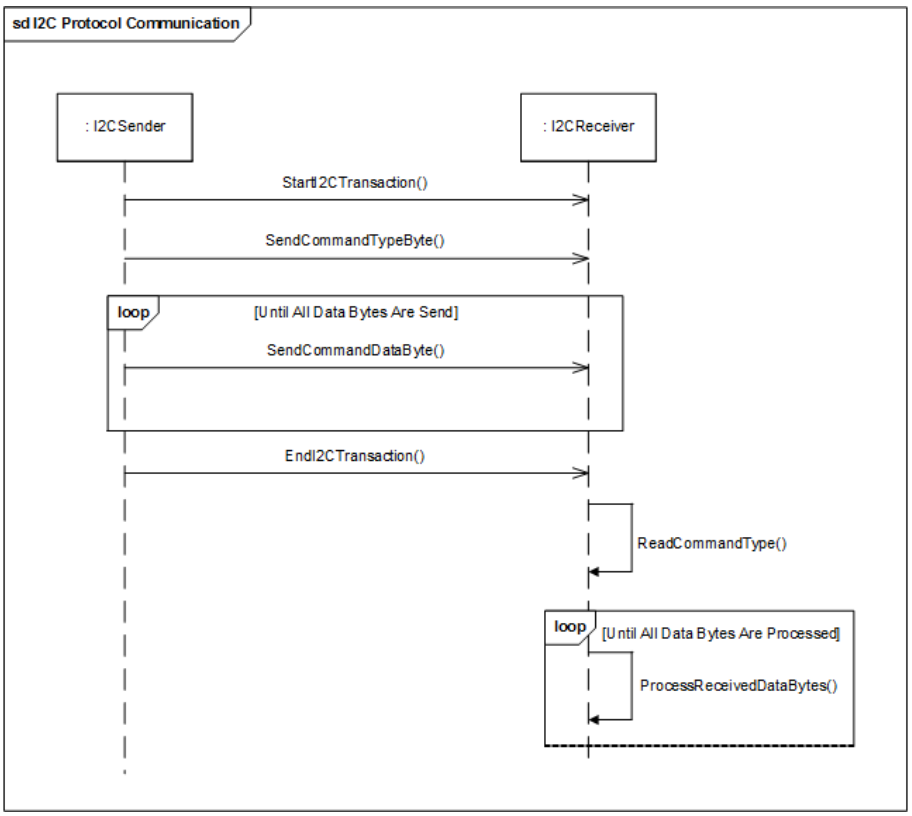
På tabel 7 ses de definerede kommandotyper og det tilsvarende antal af bytes der sendes ved dataveksling.

Kommandotype	Beskrivelse	Binær Værdi	Hex Værdi	Data Bytes
NunchchuckData	Indeholder aflæst data fra Wii Nunchuck controlleren	0010 1010	0xA2	Byte #1 Analog X-værdi Byte #2 Analog Y-værdi Byte #3 Analog Buttonstate
I2CTestRequest	Anmoder PSoC0 om at starte I2C-kommunikations test	0010 1001	0x29	Ingen databyte
I2CTestAck	Anmodning om at få en I2C OK besked fra I2C enhed	0010 1000	0x28	Ingen databyte

Tabel 7: I2C kommunikation kommandotyper

Kolonnerne *Binær Værdi* og *Hex Værdi* i tabel 7 viser kommandotypens unikke tal-ID i både binær- og hexadecimalform. Denne værdi sendes som den første byte, for at identificere kommandotypen.

Kommandoens type definerer antallet af databytes modtageren skal forvente og hvordan disse skal fortolkes. På figur 16 ses et sekvensdiagram der, med pseudo-kommandoer, demonstrerer forløbet mellem en I2C afsender og modtager ved brug af kommunikations protokollen.



Figur 16: Eksempel af I2C Protokol Forløb

På figur 16 ses at afsenderen starter en I2C transaktion, hvorefter typen af kommando sendes som den første byte. Efterfølgende sendes N antal bytes, afhængig af hvor meget data den givne kommandotype har brug for at sende. Efter en afsluttet I2C transaktion læser I2C modtageren typen af kommando, hvor den herefter tolker N antal modtagne bytes afhængig af den modtagne kommandotype.

Designvalg

Den primære idé bag valg af kommandotype metoden til I2C kommunikations protokollen er først og fremmest så modtageren kan differentiere mellem flere handlinger i systemet. En anden vigtig grund er at man, ved kommandotyper, kan associere et dynamisk antal bytes til hver kommando. Dvs, hvis en modtager får en kommandotype A, vil den vide at den efterfølgende skal læse 5 bytes, hvormiod at en kommandotype B ville betyde at der kun skulle læses 2 bytes.

En anden fordel ved kommandotype metoden er, at den er relativ simpel at implementere i kode, som vist ved pseudofunktionerne i figur 16.

10 Design og Implementering

10.1 Hardware

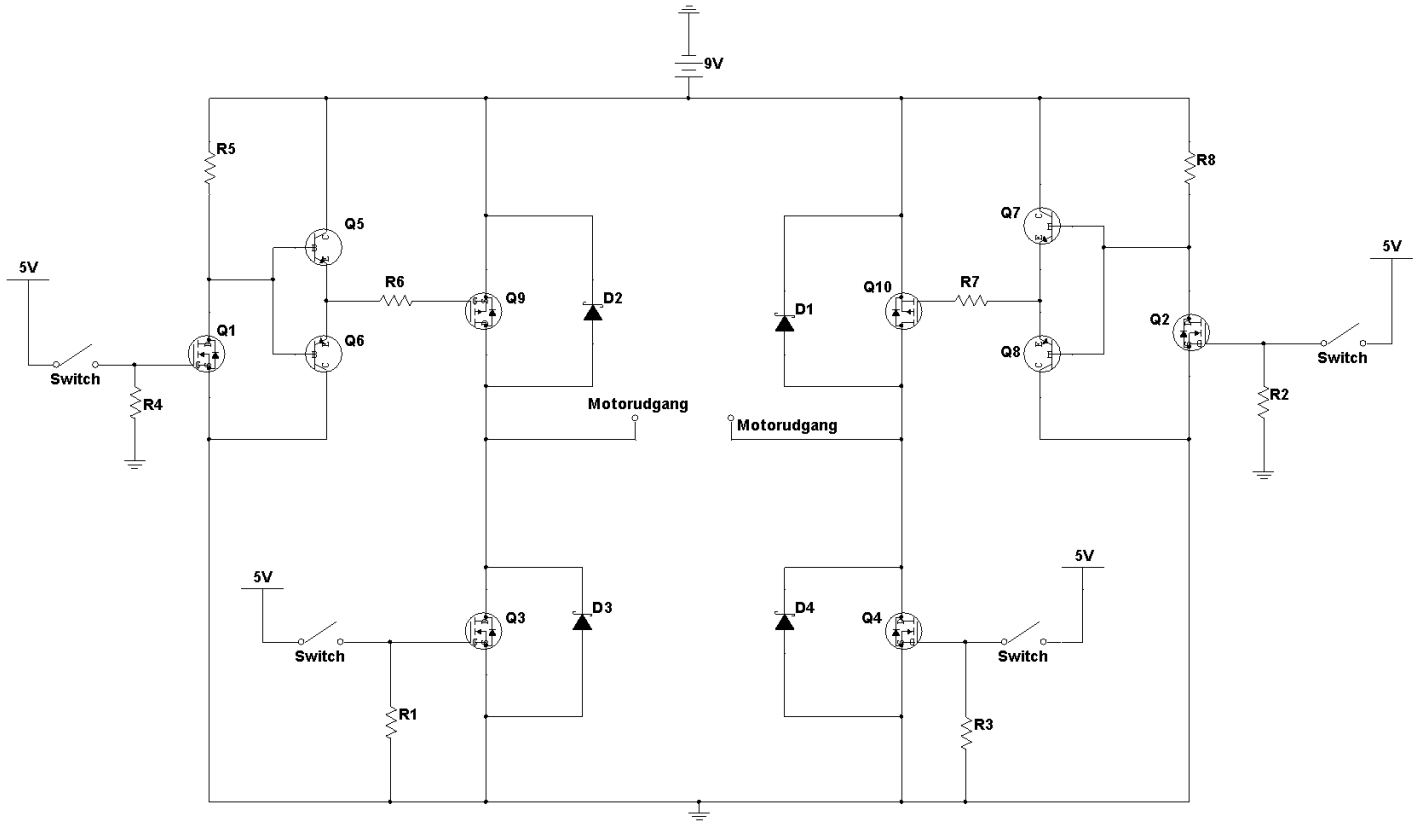
I Goofy Candygun 3000 indgår forskellige hardwaredele, som skal udgøre det færdige system. Ud fra BDD'et ses det, at de to overordnede hardwaredele, der skal udvikles, er motorstyring til den vertikale og horisontale drejeretning, og en affyringsmekanisme.

10.1.1 Motorstyring af drejeretning

For at kanonen kan sigte både vertikalt og horisontalt, skal den kunne køre i begge disse retninger. Det afhjælpes ved at bruge to DC-motorer - en til hver retning. Men for at disse så kan køre både forlæns og baglæns, skal der bruges en H-bro. For at sikre at motorerne ikke kan dreje 360 grader, er der udviklet en rotationsbegrænsning, som gør, at de kun kan dreje cirka 35 grader ud til hver side.

H-bro

For at få mulighed for, at motoren både kan køre frem og bakke, skal der bruges en H-bro. I stedet for at bruge en færdig driver er der valgt at udvikle en selv. På figur 17 ses kredsløbsdiagrammet for H-broen.



Figur 17: Diagram af H-bro

I tabel 8 ses en oversigt over komponentværdier. En fyldestgørende oversigt over disse kan ses i **??**.

Betegnelse	**Komponent**
Q1	IRLZ44(mosfet N-Channel)
Q4	IRLZ44(MOSFET N-kanal)
Q5	BC547
Q6	BC557
Q9	IRF9Z34N(MOSFET P-kanal)

Tabel 8: Komponentbetegnelser på H-bro

H-broen er bygget op af to identiske kredsløb - et til hver retning. Derfor beskrives kun den ene retning. Kredsløbet består grundlæggende af en N-MOSFET og en P-MOSFET. Når der sendes 5V ind på gate-benet til de to MOSFET's løber der en strøm gennem dem. Det betyder, at der bliver åbnet for, at motoren kan køre. I dette tilfælde skal motoren køre ved hjælp af et PWM-signal, så det betyder, at de to MOSFET's vil stå og åbne og lukke så

lang tid, dette signal sendes ind. Det betyder også, at det er muligt, at styre motorens hastighed ved hjælp af PWM-signalets dutycycle.

For at gøre det muligt, at motoren kan skifte retning hurtigt er der indsat to transistorer. Inden disse blev sat ind, blev P-MOSFET'en (Q9) opladet hurtigt men afladet meget langsomt. For at få den til at aflade hurtigt blev Q5 indsat. Det gjorde, at N-MOSFET'en (Q4) blev langsom til at oplade, så derfor blev Q6 sat ind.

Rotationsbegrænsning

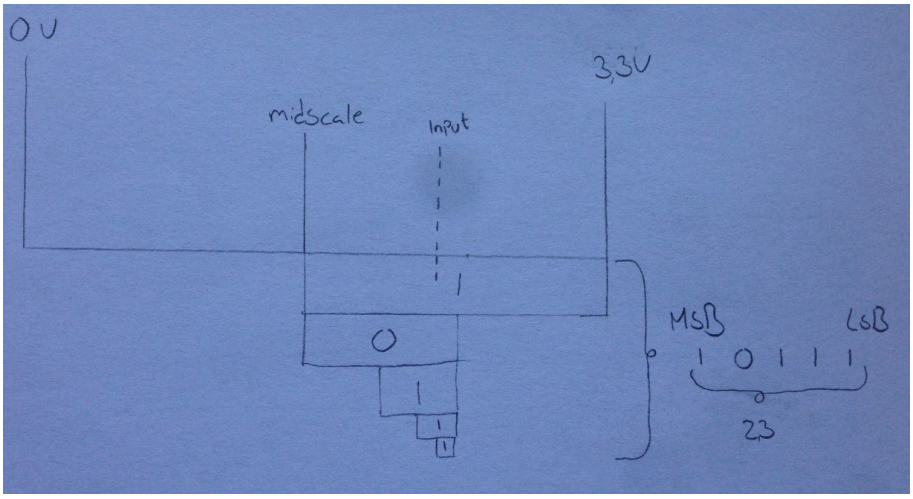
Motoren, som styrer platformen, skal kunne bevæge sig frit i intervallet, som er fastsat i kravspecifikationen **#ref Reference til kravspecikation (ikke-funktionelle krav)**. For at begrænse denne bevægelse udenfor det fastsatte interval, anvendes der et potentiometer. Når motoren bevæger sig, ændres potentiometerets modstandsværdi, og dermed ændres spændingsniveauet. Altså kan motorens position bestemmes ved at se på outputspændingen fra potentiometret. Et alternativ til denne løsning er, at der anvendes en steppermotor, der kan styres i små steps, fremfor en DC-motor. Da der var et ønske om at arbejde med flere hardwareelementer end kun motoren, blev DC-motoren med rotationsbegrænsning valgt.

Figur 18: Opstilling for rotationsbegrænsning

På figur 18 ses opstillingen for rotationsbegrænsningen. På figuren ses, at potentiometerets skaft er forbundet til motoren, som roterer når motoren bevæges. Potentiometeret bliver forsynet med 5V og stel fra PSoC1 via den røde og sorte ledning. Ouputspændingen fra potentiometeret er forbundet til PSoC1 via den gule ledning.

Potentiometeret har til formål at styre motoren i det definerede interval. Det valgte potentiometer er et lineært 47kΩ potentiometer. Det vil sige, at for hver ændring, der sker i modstanden, vil der ske en proportional ændring i outputspændingen. Outputspændingen føres ind på et ben på PSoC'en, som er forbundet til en intern AD-converter, som er en Sequencing Successive Approximation ADC. Denne anvendes til at digitalisere det analoge signal fra potentiometeret.

En sequencing SAR ADC indeholder et sample-hold kredsløb og en comparator. Kredsløbet holder på et indgangssignal, indtil det næste signal registreres på kredsløbets indgang. Dermed har converteren tid til at bestemme resultatet af konverteringen. Comparatoren sammenligner inputsignalet med midscaleværdien.



Figur 19: ADC opbygning

På figur 19 er ADC'en indstillet til midscale og bruger dette punkt til sammenligning med inputsignalet.

ADC'en består af et sampleholdkredsløb. Det vil sige, at den ser på dataene indtil der er fundet en værdi, så der går noget tid imellem hver værdi, så der er sørget for, at der ikke bliver lavet oversamples og der ikke bliver lavet aliasering af det signal, som kommer ind i ADC'en. For at finde den værdi, som bliver sendt ind i ADC'en, bliver der lavet en skala, som bliver halveret alt efter, hvor værdien ligger på skalaen, som vist på figur **??**

Der er blevet målt med en vinkelmåler for at overholde kravene i kravspecifikationen**#ref Reference til kravspecikation (ikke funktionelle krav)**. Hvor der blev fundet frem til en minimum- og maksimumværdi, hvor motoren må køre imellem.

Min=915 mV

Max = 2000mV

Disse værdier er aflæst fra PSoC'en. Da den har en referencespænding på 3.3V vil den have lavere værdier, end hvad der bliver sendt ud fra potentiometeret.

$$\frac{5V}{3.3V} = 1.515 \tag{1}$$

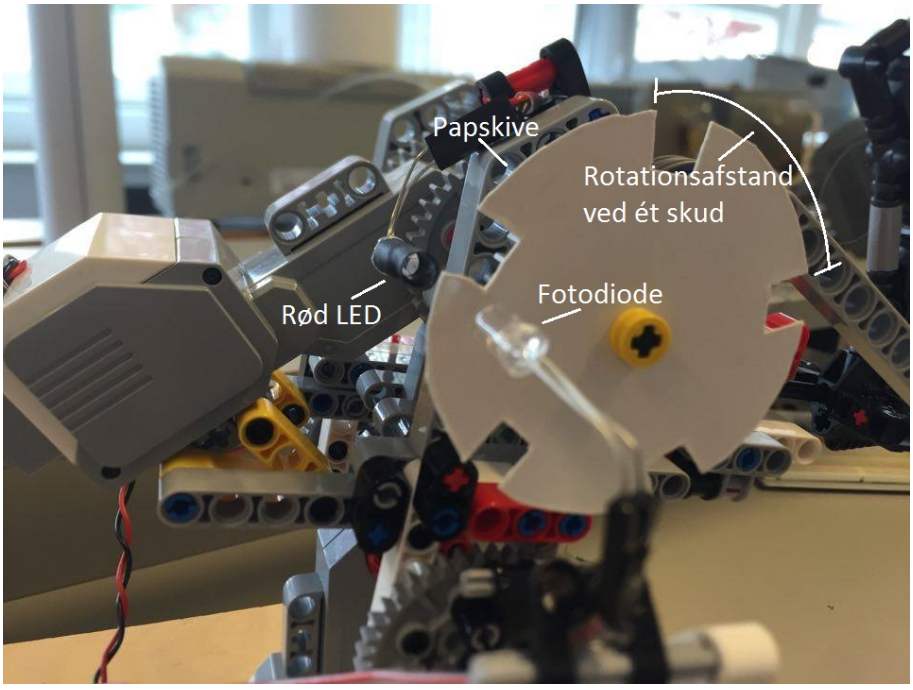
Der er en forskel på cirka 1,515 mellem potentiometer og ADC. Så ADC er cirka 1,515 mindre end hvad potentiometer sender ud.
For at forstå hvordan ADC'en og potentiometeret fungerer henvises til dokumentationen.

10.1.2 Affyringsmekanisme

Affyringsmekanismen består af en motor; et motorstyringskredsløb; et detektorkredsløb, der skal detektere, at motoren kun kører en enkelt omgang, når der skydes; og en kanon, som er bygget op af noget mekanik og LEGO.

Detektor

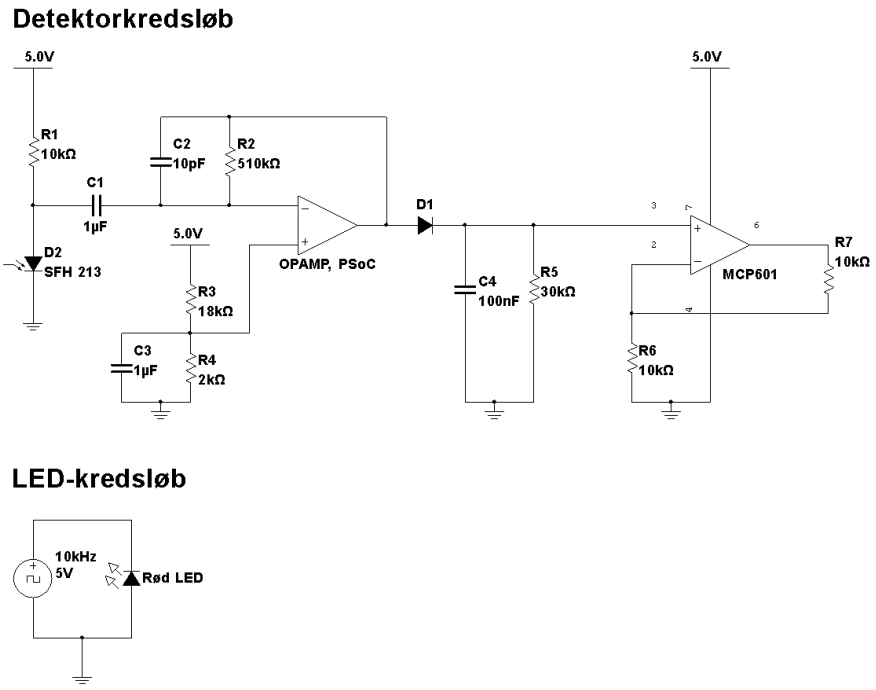
Når kanonen affyres, styres det af motoren, og som mekanikken er opbygget, er der et propertionelt forhold mellem omdrejning på motoren og antal skud, der affyres. Derfor er det væsentligt at vide, hvornår motoren har roteret en runde, så den kan stoppes, inden der igen skydes. Til det formål anvendes detektoren. Billedet på figur 20 illustrerer hvordan detektoren anvendes.



Figur 20: Detektorens placering på affyringsmekanismen

Den røde LED og fotodioden anbringes på affyringsmekanismen, som det ses på figur 20. De vender ind mod hinanden, men er adskilt af papskiven. Papskiven er forbundet til motorens rotation, og hver gang et af papskivens hakker roterer forbi dioderne, kan de se hinanden. Fotodioden sender derefter et signal, som kan bruges til at stoppe motoren. Hvert hak passer med, at der er blevet affyret et skud.

Detektoren skal kun sende et signal, når fotodioden ser lyset fra LED'en. Det er derfor vigtigt, at den ikke bliver forstyrret af dagslys og andre lyskilder. For at sikre dette, styres den røde LED af et PWM-signal, så LED'en blinker med en frekvens på 10 kHz. Detektoren opbygges tilsvarende af et båndpasfilter, med en centerfrekvens på 10 kHz, som sorterer andre frekvensområder og DC-signaler fra. 10 kHz er rigeligt højt, til at det for øjet ikke er synligt at LED'en blinker. Samtidig er det ikke for højt til, at en almindelig operationsforstærker kan håndtere det. På figur 21 ses et kredsløbsdiagram for detektoren og LED'en.



Figur 21: Kredsløbsdiagram for detektoren

Båndpasfiltret er opbygget af et højpasfilter, et lavpasfilter og en operationsforstærker. Da PSoC'en har en indbygget operationsforstærker, anvendes denne. Software design og implementering af den ses under PSoC Software!!!! Af hensyn til operationsforstærkeren, er der valgt en referencespænding på 0,5 V på den positive indgang. Det er opnået ved en spændingsdeler, for hvilken beregningen kan ses i dokumentationen !!!!. Der er negativ feedback på operationsforstærkeren, hvilket sikrer, at der opretholdes samme spænding, 0,5 V, på begge indgange i operationsforstærkeren. Når fotodioden kan se den røde

LED, genererer den en strøm, som bliver omsat til en spænding i kredsløbet. Operationsforstærkeren vil opretholde 0,5 V på den negative indgang. Den vil derfor regulere udgangen for at ophæve de ændringer, som fotodioden skaber på den negative indgang. Udgangssignalet vil dermed afspejle det PWM-signal, som den røde LED sender.

Når fotodioden kan se lyset fra den røde LED er signalet, som kommer fra udgangen af operationsforstærkeren, et firkantsignal med en frekvens på 10 kHz,. Når fotodioden ikke kan se det røde lys, er spændingen på udgangen 0,5 V. Det ønskes omdannet til et signal, der går højt, når PWM-signalet starter, og går lavt, når PWM-signalet er væk igen. For at opnå dette, blev der lavet en envelopedetector, som er opbygget af en diode, en modstand og en kondensator. Dioden sikrer, at skulle der komme negative spændinger, så vil de blive frasorteret. Kondensatoren er dimentioneret efter, at den bliver opladet på de første udsving fra firkantsignalet. Modstanden er dimentioneret, så spændingen ikke aflades mellem svingningerne på 10 kHz signalet. En simulering af dette kan ses i dokumentationen !!!!

Outputtet, fra envelopedetektoren var dog noget lavt. Det lå mellem 1,5V og 2V. Derfor blev der indsat en ikke-inverterende forstærker for at fordoble signalet. Forstærkeren består af en opamp og to modstande. Derfor blev der opstillet følgende ligning **??**:

$$U_O = (1 + \frac{R_2}{R_1}) * U_P \tag{2}$$

Det vil altså sige, at hvis de to modstande, der sættes ind er ens, vil indgangssignalet blive fordoblet. Hvis der eksempelvis sendes 2V ind vil der være 4V på udgangen:

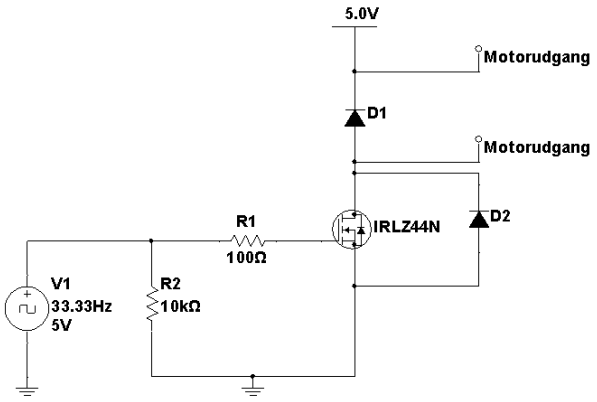
$$U_O = (1 + \frac{10k\Omega}{10k\Omega}) * 2V \tag{3}$$

Herefter var det muligt at aflæse et tydeligt firkantsignal med en peak-to-peak-værdi på 3,5V.

Den røde LED er koblet direkte til et 0-5 V, 10 kHz PWM signal fra PSoC'en. Den kan godt klare sig uden en formodstand.

Motorstyring

Til at styre affyringsmekanismens motor er der bygget et kredsløb med en MOSFET som primære komponent. MOSFET'en skal sørge for, at motoren kun kører, når der bliver sendt PWM-signal ind i den. Kredsløbsdiagrammet kan ses på figur 22.

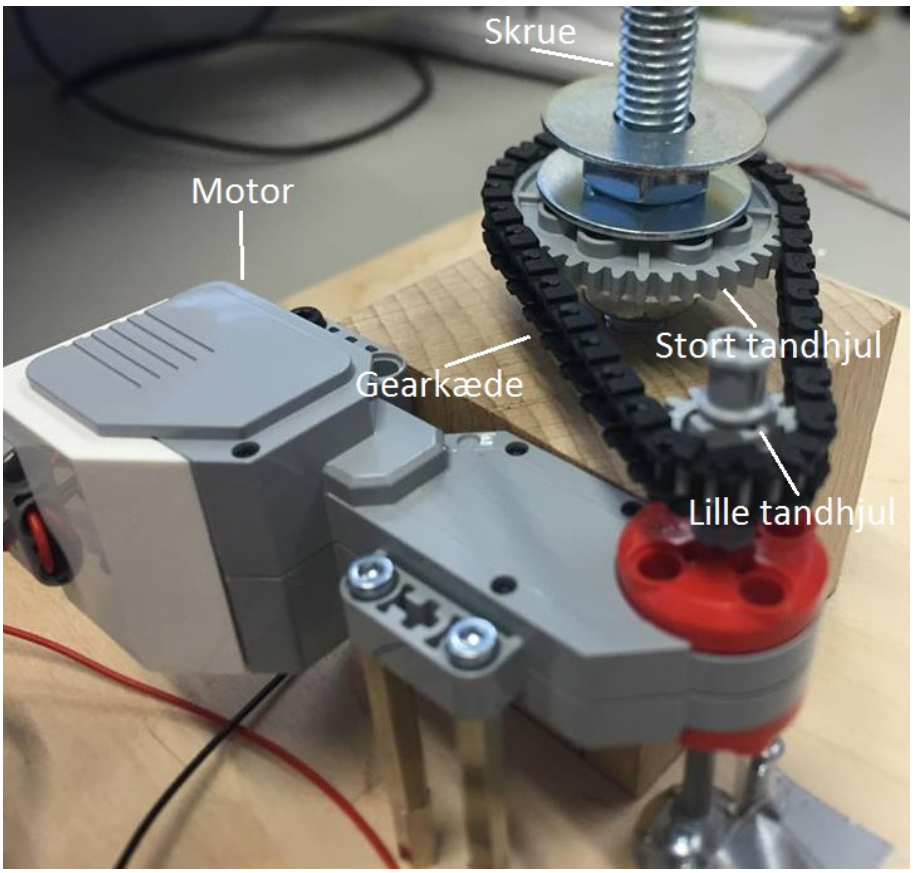


Figur 22: Diagram over motorstyring til motor på affyringsmekanisme

Dioden D1 er sat ind for at sikre motoren mod store spændingsspikes, der kan forekomme, når MOSFET'en bliver afbrudt. Dioden D2, der sidder fra source til drain, sikrer, at spikes genereret af motoren, når den slukkes, ikke brænder MOSFET'en af.

Kanon og platform

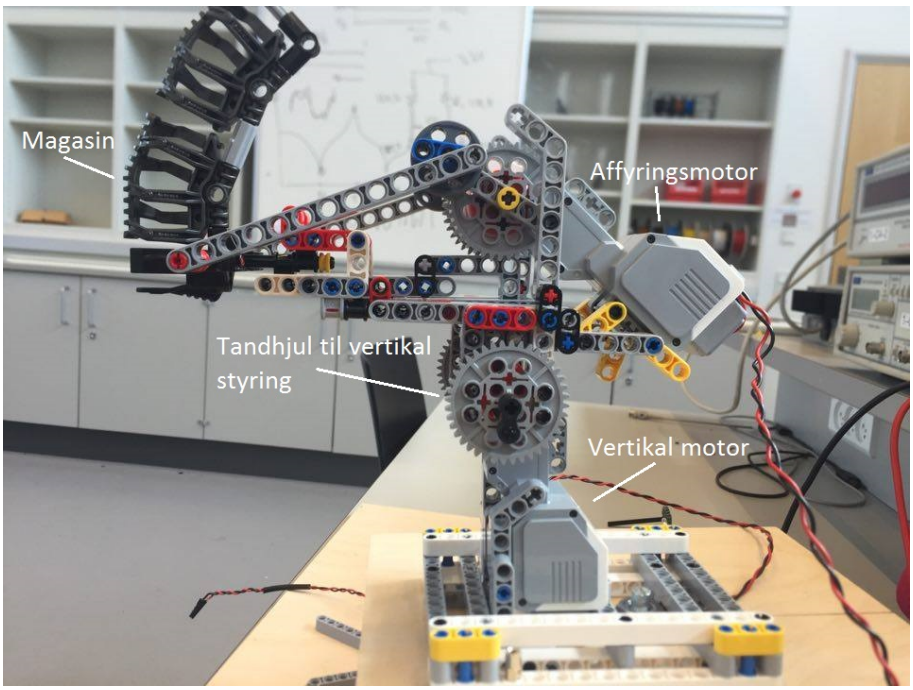
Selve kanonen og platformen den står på er bygget op af to træplader og LEGO. Den ene træplade kan dreje fra side til side, således at det er muligt at sigte i den horisontale retning. Opbygningen ses på figur 23. Træpladen placeres på den øverste metalskive omkring skruen, så den drejer med rundt, når motoren roterer. Rotationen er opnået ved, at skruen kan dreje frit, men stadig er holdt lodret. Det store tandhjul er boret ud i midten, og der er indsat en møtrik, så den kan skrues på skruen. Forholdet mellem det store og det lille tandhjul gør at rotationshastigheden bliver gearet ned. Endeligt er motoren skruet fast til den nederste træplade, men i en højde, der gør at den kan drive det lille tandhjul, som er forbundet med gearkæden til det store tandhjul.



Figur 23: Horisontal mekanik

Mekanikken for den vertikale retning er bygget af LEGO. Et billede af opbygningen ses på figur 24. Styringen af den vertikale bevægelse bliver håndteret af den vertikale motor, som ses på figur 24. Motoren er forbundet til tandhjulene til vertikal styring. Der er ét tandhjul på hver side af motoren. De er begge bygget sammen med resten af kanonen. Når motoren drejer, bliver tandhjulene og hele kanonen vippet fremover eller bagover.

Ligesom den vertikale styring er selve kanonen også bygget i LEGO. Den har et magasin, som det fremgår af figur 24. Der kan kommes slik i magasinet, som så bliver affyret. Affyringen styres af den anden motor på figur 24. Når affyringsmotoren drejer bliver to større tandhjul roteret. De to tandhjul er desuden forbundet til to små tandhjul, som er 5 gange så små. Med denne gearing roterer de små tandhjul 5 gange så hurtigt. De små tandhjul er forbundet til to mellem størrelse tandhjul, som drejer med dem rundt. Tandhjulene i mellemstørrelse styrer affyringen ved at omdanne den roterende bevægelse til en vandret bevægelse frem og tilbage, som affyrer kanonen.



Figur 24: Kanon i LEGO

10.2 Software

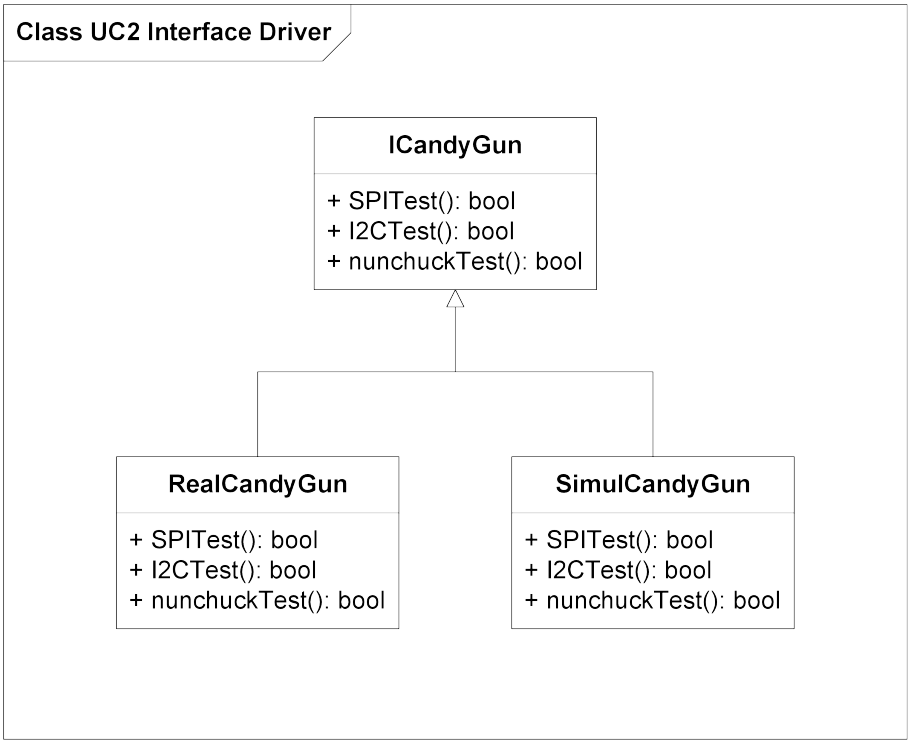
10.2.1 SPI - Devkit8000

Candydriveren sørger for SPI-kommunikationen fra Devkit8000 til PSoC0. Driveren er skrevet i c, hvilket er typisk for drivere til linuxplatforme.
SPI-kommunikationen er implementeret med SPI bus nummer 1, SPI chip-select 0 og en hastighed på 1 MHz (et godt stykke under max på 20 MHz for en sikkerhedsskyld). Desuden starter clocken højt og data ændres på falling edge og aflæses på rising edge. Dermed bliver SPI Clock Mode 3. Derudover sendes der 8 bit pr transmission, hvilket passer med SPI-protokollen for projektet.
For at kunne anvende driveren, når SPI er tilsluttet, er der oprettet et hotplug-modul, som fortæller kernen, at der er et SPI device, som matcher driveren. Det kan SPI-forbindelsen ikke selv gøre, som usb fx kan. Selve driveren er i candygun.c opbygget som en char driver. For at holde forskellige funktionaliteter adskilt er alle funktioner, der har med SPI at gøre, implementeret i filen candygun-spi-c. Så når der fx skal requestes en SPI ressource i init-funktionen i candygun.c, så anvender driveren en funktion fra candygun-spi.c til det. I probe-funktionen sættes bits_per_word til 8, da vi sender otte bit som nævnt tidligere. I exit-funktionen anvender candygun.c igen en funktion fra candygun-spi.c - denne gang til at frigive SPI ressourcen. I write-metoden gives der data med fra brugeren. I dette tilfælde udgøres brugeren af Interface driveren og dataet er en 8 bit kommando fra SPI-protokollen. Dog er dataet fra brugeren i første omgang læst ind som en charstreng. I write-metoden bliver det så lavet om til en int. For

at overføre dataet på en sikker måde anvendes funktionen copy_from_user() til at overføre data fra brugeren. Write-funktionen fra candygun.c anvender derefter en write-funktion fra candygun-spi.c, hvor den sender brugerinputtet med. I den spi-relaterede write-funktion bliver bruger inputtet lagt i transfer bufferen og der NULL bliver lagt i receive bufferen, og med spi_sync-funktionen bliver det sendt.
Ofte ville der en spi read-funktion først indeholde en write-del, som fortalte SPI-slaven, hvad der skulle læses over i bufferen. Det ville typisk efterfølges af et delay og så en read-del. Men i dette projekt skal der ofte afventes et brugerinput, som ikke kan styres af et fast delay, og der skal generelt sendes en aktiv kommando før der læses. Derfor er det besluttet at read-funktionen kun indeholder en read-del i transmissionen. Dermed skal write-funktionen altid aktivt anvendes inden der læses, da PSoC0 ellers ikke ved, hvad der skal gøres/lægges i bufferen.
Når funktionen har modtaget resultatet fra transmissionen returneres det til brugeren med funktionen copy_to_user(), som igen sørger for at overførslen af data foregår på en sikker måde.

10.2.2 Interface Driver

Interface driveren fungerer som bindeled mellem brugergrænsefladen og candy-driveren på Devkit8000. Den indeholder tre funktioner. Funktionerne anvendes i use case 2 til at teste kommunikationsforbindelserne i resten af systemet. Interface driveren er designet og implementeret i C++ og gør brug af klasserelationen arv. Et klassediagram for interface driveren se på figur 25.



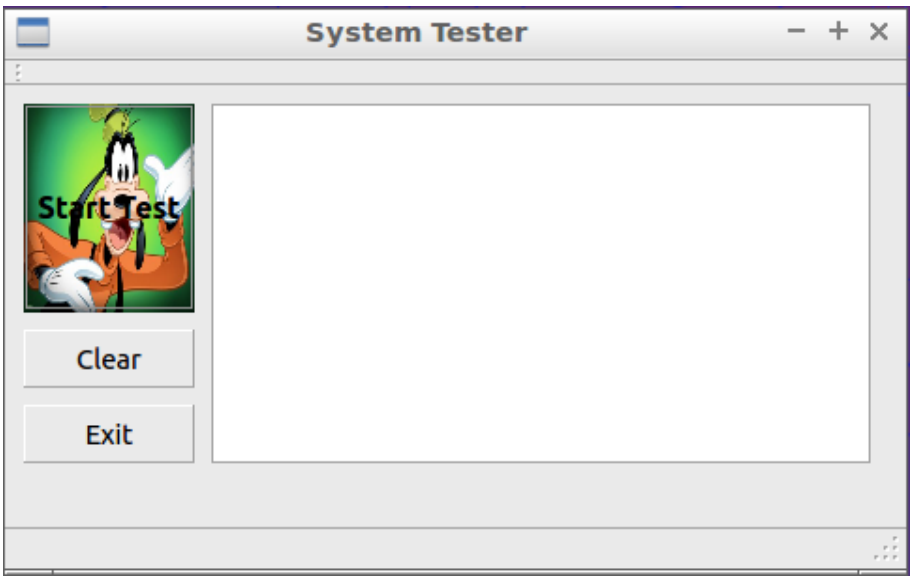
Figur 25: Interface driver for UC2

Basisklassen er ICandyGun. Det er en abstrakt klasse, da den udelukkende indeholder virtuelle metoder. Derudover er der to afledte klasser; SimulCandyGun og RealCandyGun. SimulCandyGun implementerer metoderne til at simulere respons fra Candydriveren. Dermed kan brugergrænsefladen testes uafhængigt af de resterende dele af systemet. Simuleringen er implementeret med *srand()*-funktionen fra *cstdlib*-biblioteket, som returnerer et tilfældigt tal, som her bliver mellem 0 og 1. I RealCandyGun-klassen er metoderne implementeret efter den reelle SPI-protokol og med de nødvendige funktioner til at skrive til et kernemodul. Fx open(), close(), read () og write(). Da interface driveren er implementeret med arv, skal der ikke foretages betydelige ændringer i brugergrænsefladen, når der skiftes mellem simuleringsklassen og den rigtige version. Dermed opnås lav kobling.
De tre funktioner som Interface driveren indeholder i forbindelse med use case 2 (test use casen) er: SPITest(), I2CTest(), NunchuckTest(). Hver af de tre funktioner anvendes til at starte en test af de forskellige kommunikationsforbindelser: SPI, I2C og brugerinputet fra nunchucken. Alle funktionerne returnerer en bool, som enten er true eller false, alt efter om testen var succesful eller ej. Når der skal startes en test, åbner den pågældende funktion filen *dev/candygun* og skriver SPI-kommandoen for *start test* til filen. Derefter venter funktionen ét sekund og læser så svaret fra filen. Da der i nunchucktesten ventes på et brugerinput, og brugeren skal have lidt tid til at trykke på nunchuck-knappen, er der oprettet en while-løkke, som tjekker flere gange om testen returnerer

true. Hvis testen ikke returnerer true ved første check, venter funktionen atter et sekund og tjekker igen. Det gør den op til 15 gange og melder derefter om fejl, hvis ikke den returnerer true inden.
Brugergrænsefladen anvender interface driveren ved at inkludere headerfilerne og oprette en ICandyGun pointer, der peger på en instans af én af de to afledte klasser. Ved at pakke kommunikationen til kernemodulet for candydriveren væk i funktioner kan brugergrænsefladen anvende funktionerne uden at kende til SPI-protokollen. Det sikrer igen lav kobling, og i tilfælde hvor det kunne ønskes, at SPI-kommunikationen kan erstattes af en anden kommunikationsform, kan det gøres uden, at der skal foretages ændringer i brugergrænsefladen.

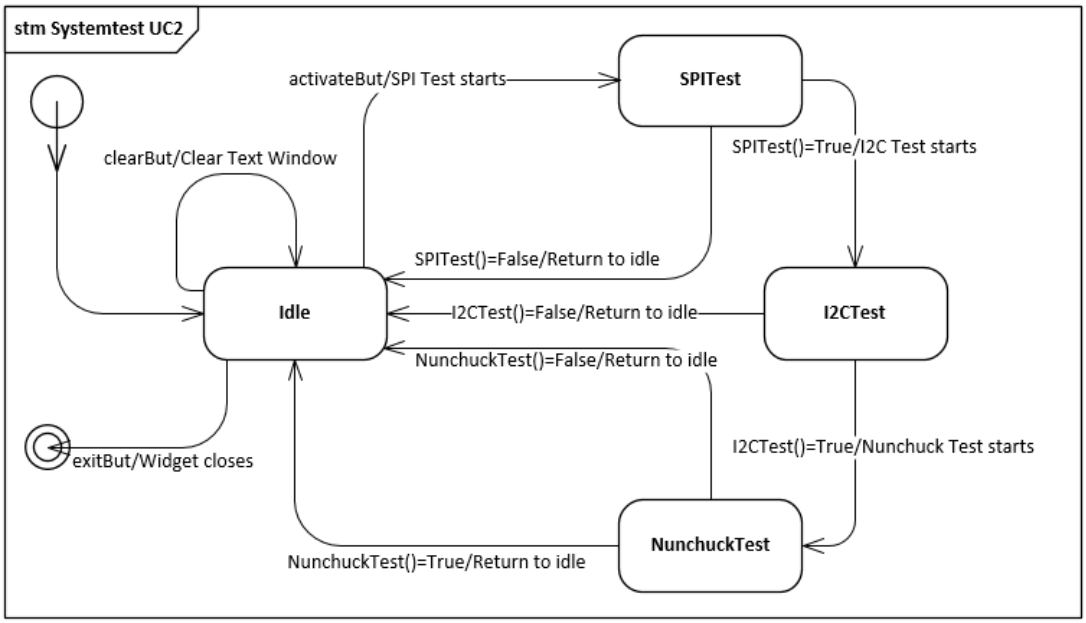
10.2.3 Brugergrænseflade

Selve Usecase 2 styres via brugergrænsfladen fra Devkit8000. Dette afsnit beskriver brugergrænsefladens design.



Figur 26: Brugergrænseflade for usecase 2

Brugergrænsefladen er lavet med det indbyggede design framework i QT Creator 5. QT frameworket opretter "hovedvinduet"i brugergrænsefladen som en klasse. Knapperne tilføjes som private slots i klassen hvilket gør dem i stand til interagere i brugergrænsefladen. Når en knap er assignet til et slot i klassen, og der trykkes på den pågældende knap, bliver det assignede signal broadcastet og slot-funktionen bliver kørt. Alle tre knapper i brugergrænsefladen er assignet signal-typen "clicked()".



Figur 27: State machine for brugergrænsefladen for usecase 2

Brugergrænsefladen for UC2 er en simpel test-konsol. Den består af 3 knapper og et tekstvindue. Brugergrænsefladen interfacer med SPI-protokollen, gennem vores interface driver. Den første knap, Start test, initierer UC2. Efterhånden som testen løbes igennem kaldes test funktionerne, og ved hjælp af if-conditions, bliver der tjekket på retur-værdierne fra interface-funktionerne. Hvis retur-værdien er true, skrives der en --test successful"besked i tekstvinduet. og widgeten kører videre. Hvis retur-værdien er false, skrives der en --test unsuccessful"i tekstvinduet og widgeten returnerer til idle tilstand. Når alle test er successful, skrives "System test successful, system is ready for use"til tekstvinduet, og widgeten returnerer til idle tilstand. Den anden knap, Clear, clearer tekstvinduet til blank tilstand. Den tredje knap, Exit, lukker widgeten.

10.2.4 Nunchuck

Til styring af kanonen bruges en Wii-nunchuck. Følgende afsnit beskriver PSoC0's håndtering af data fra Wii-nunchuck.

Afkodning af Wii-Nunchuck Data Bytes

Aflæste bytes fra Wii-Nunchuck - indeholdende tilstanden af knapperne og det analoge stick - er kodet når de oprindeligt modtages via I2C bussen. Disse bytes skal altså afkodes før deres værdier er brugbare. Afkodningen af hver byte sker ved brug af følgende formel:

AfkodetByte = (AflæstByte XOR 0x17) + 0x17

Fra formlen kan det ses at den aflæste byte skal *XOR*'s (Exclusive Or) med værdien 0x17, hvorefter dette resultat skal adderes med værdien 0x17.

Kalibrering af Wii-Nunchuck Analog Stick

De afkodede bytes for Wii-Nunchuck's analoge stick har definerede standardværdier for dets forskellige fysiske positioner. Disse værdier findes i tabel 9

X-akse helt til venstre	0x1E
X-akse helt til højre	0xE1
X-akse centreret	0x7E
Y-akse centreret	0x7B
Y-akse helt frem	0x1D
Y-akse helt tilbage	0xDF

Tabel 9: Standardværdier for fysiske positioner af Wii-Nunchuck's analoge stick

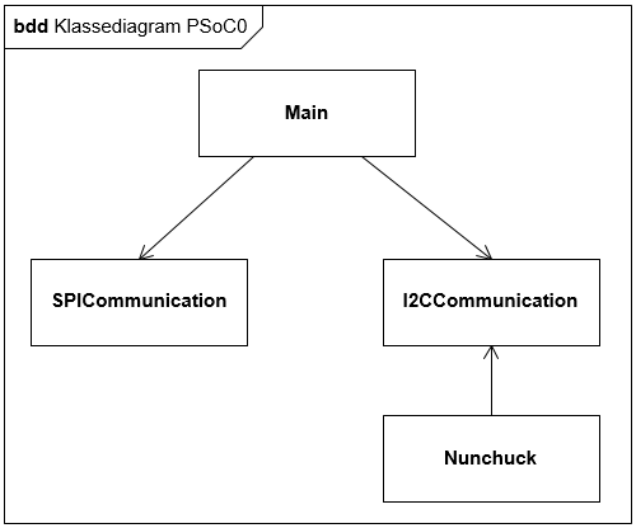
I praksis skal de afkodede værdier for det analoge stick kalibreres, da slør pga. brug gør at de ideale værdier ikke rammes.

I projektet er de afkodede værdier for det analoge stick kalibreret med værdien -15 (0x0F i hexadecimal), altså ser den endelige formel for afkodning samt kalibrering således ud:

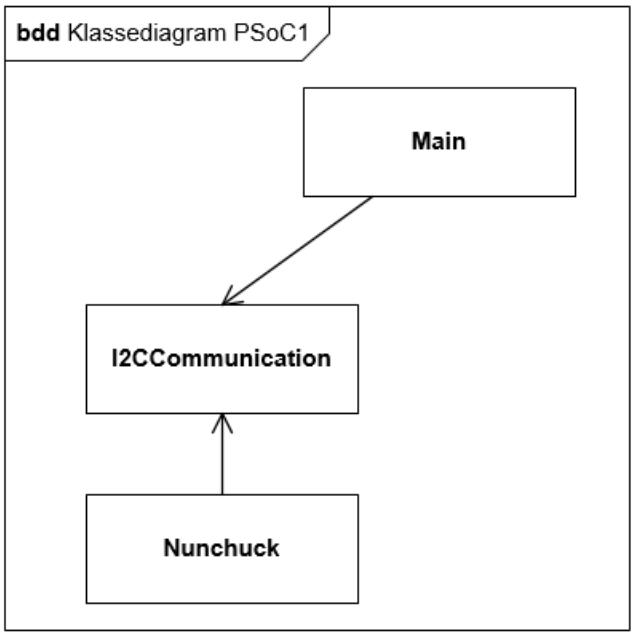
AfkodetByte = (AflæstByte XOR 0x17) + 0x17 - 0x0F

10.2.5 PSoC Software

De følgende klassediagrammer på figur 28 og 29 giver et overblik over hvilke klasser der bliver gjort brug af på PSoC0 og PSoC1. De efterfølgende afsnit vil beskrive klasserne og deres funktioner.



Figur 28: Klassediagram oversigt for PSoC0



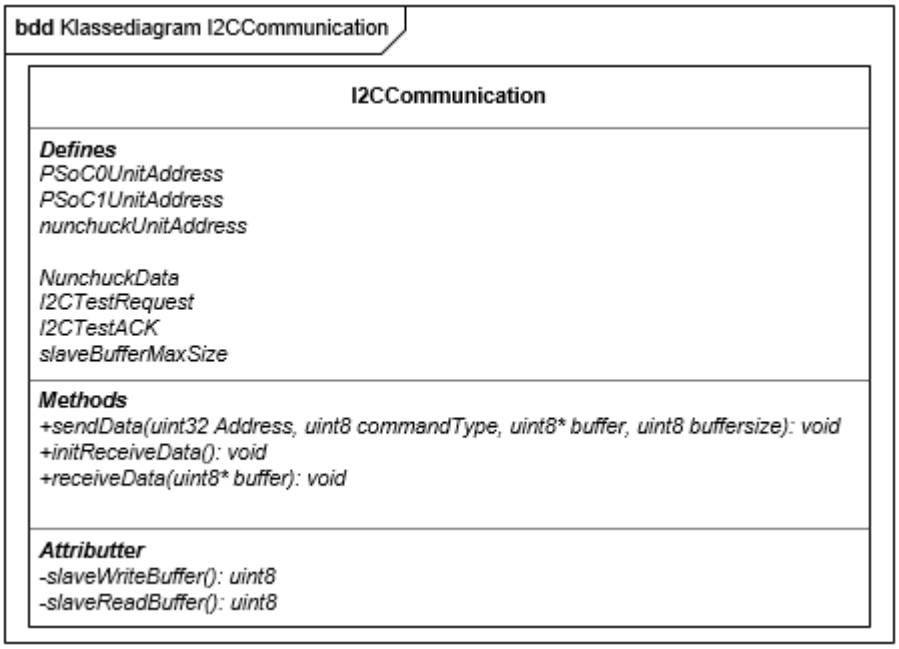
Figur 29: Klassediagram oversigt for PSoC1

10.2.6 I2CCommunication

I dette afsnit vil softwaren der omhandler I2C-kommunikation blive beskrevet. Dette inkluderer et klassediagram, samt en klassebeskrivelse.

Klassediagram

På figur 30 ses klassediagrammet for I2CCommunication.



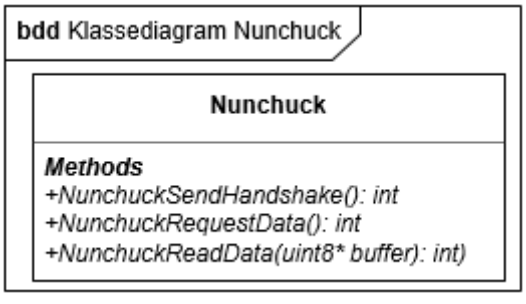
Figur 30: Klassediagram for I2CCommunication klassen

10.2.7 Nunchuck

I dette afsnit vil softwaren der specifikt omhandler kommunikationen mellem PSoC0 og Nunchucken blive beskrevet. Dette gøres vha. et klassediagram og klassebeskrivelser.

Klassediagram

På figur 31 ses klassediagrammet for Nunchuck klassen.



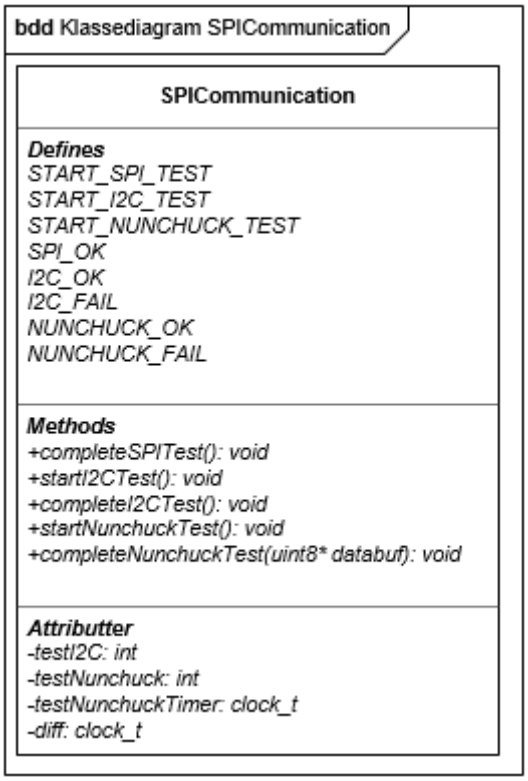
Figur 31: Klassediagram for klassen Nunchuck

10.2.8 SPI - PSoC

I dette afsnit vil softwaren der specifikt omhandler SPI-kommunikationen mellem PSoC0 og DevKit8000 blive beskrevet. Dette gøres vha. et klassediagram og klassebeskrivelser

Klassediagram

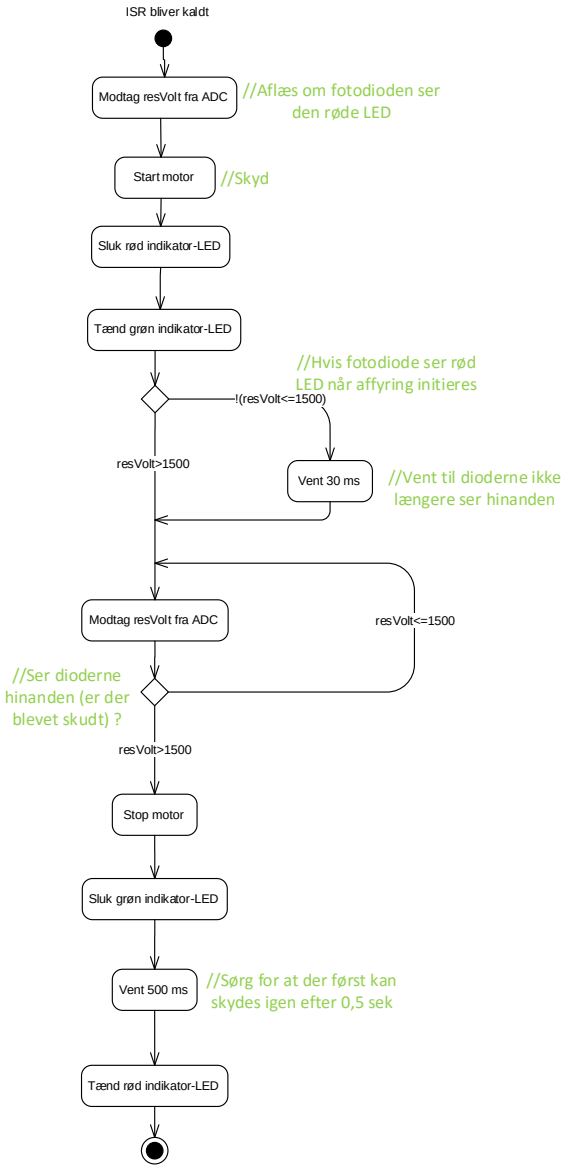
På figur 32 ses klassediagrammet over SPICommunication klassen.



Figur 32: Klassediagram over klassen SPICommunication

10.2.9 PSoC - Affyringsmekanisme

Affyringsmekanismen består udover hardware og mekanik også flere software dele, som er programmeret på PSoC2. Til rotationsdetektorens operationsforstærker er der anvendt en, der er indbygget i PSoC'en. For at se hvilke pins den er ført ud til, se dokumentationen !!!ref!!. Derudover styres bl.a. interupt og PWM signaler i forbindelse med affyringsmekanismen, ved hjælp af PSoC2. På figur 33 ses kodesekvensen som køres, når kanonen skal affyres.



Figur 33: Interface driver for UC2

For at affyringssekvensen kører, skal der modtages et interrupt. Det sker, når brugeren affyrer kanonen. PSoC????? sender et højt signal, som går ind på en inputpind på PSoC2, som så er programmeret til at trigge på rising edge og derved køre interruptrutinen. Det første der sker i interruptrutinen er, at ADC-værdien aflæses som en spænding i millivolt. Den lægges over i variablen

"resVolt". Så startes motoren, derved begynder kanonen at affyre. PSoC'ens røde LED er som udgangspunkt tændt, når der ikke affyres, når interruptet kommer, slukkes den, og den grønne PSoC-LED tændes i stedet, for at indikere at interruptrutinen køres. Dernæst anvendes spændingen fra resVolt, der blev aflæst fra ADC'en så til at vurdere om fotodioden, kan se lyset fra den røde LED. Når de ikke kan se hinanden er spændingen 500 mV, og når de kan se hinanden er spædningen mellem 4000 og 5000 mV. Normalt indikerer det, at de kan se hinanden, at affyringen er færdig, og at motoren skal stoppes, men hvis de kan se hinanden på dette tidspunkt ved interruptets start, betyder det, at de, inden affyringen er startet, allerede er placeret så de kan se hinanden. Hvis det er tilfældet køres et delay på 30 ms, for at sikre at motoren er drejet, så de igen ikke kan se hinanden. Derefter aflæses spændingen ved ADC'en igen - nu for at tjekke om affyringen er afsluttet. Aflæsningen gentages indtil fotodioden kan se den røde LED, og motoren dermed skal stoppes. Affyringen sker på langt under et sekund, og det er derfor ikke problematisk med polling. Alternativt, hvis affyringen varede længere tid, kunne det have været implementeret med et interrupt, men i dette tilfælde kan det klares ved at tjekke ADC'en flere gange.

Når motoren er stoppet, slukkes den grønne PSoC-LED, og der køres et delay på 500 ms, så der går et halvt sekund, inden der igen kan skydes. Derved undgås det, at kanonen affyres med det samme igen, hvis triggeren ved en fejl ikke er sluppet. Som det sidste tændes den røde PSoC LED, som indikator på, at kanonen igen er klar til at blive affyret.

Både PWM-signalet til affyring af motoren og til generering af 10 kHz til den røde LED styres af PSoC2. De to PWM-blokke deles om en clock, der er sat til en frekvens på 2,5 MHz. PWM til den røde LED, har en periode på 250, dermed bliver PWM-signalet 10 kHz. Derudover er dutycyclen for LED'en 10 %. PWM-signalet til motoren har en periode på 75, derved bliver frekvensen 33,33 kHz. En DC-motor skal gerne styres af et PWM-signal på over 20 kHz, for at rotere uden problemer. Dermed passer de 33,33 kHz godt.

11 Test

For at verificere systemets funktionaliteter for både hardware og software blev der udført modultests, integrationstests samt en endelig accepttest. I følgende afsnit vil fremgangsmåden for hver type test blive beskrevet, samt opsummerede resultater af udførte tests.

11.1 Modultest

Ved modultests blev én funktionalitet testet isoleret, dvs. ved så lidt påvirkning fra resten af systemet som muligt. Modultests blev udført før integrationstests, for at sikre individuel funktionalitet før sammensætning af alle komponenter. Modultests blev udført både for software-komponenter og hardware-komponenter.

11.1.1 Software

Til modultest af software er der gjort brug af white-box testing. Dette er gjort, da softwaren ligger tæt op ad hardwaren til kommunikationsbusserne, hvilket kræver teknisk viden om den interne struktur for både hardware og software.

Wii-Nunchuck

Dataoverførsel fra Wii-Nunchuck sker ved at PSoC0 først sender et *handshake*, hvilket er en enkelt byte med værdien 0x00. Herefter kan PSoC0 aflæse Wii-Nunchuck tilstanden ved kontinuert at sende en byte med værdien *??*, efterfulgt af den egentlig aflæsning. Det er altså disse to dele der skal testes på.

For flere tekniske detaljer, samt billeder af målingerne, refereres til **DOKUMENTATION #ref**

Tabel 10 og 11 præsenterer modultest resultaterne for Wii-Nunchuck.

Forventet Resultat	På I2C Bussen måles et *ACKNOWLEDGE* fra Wii-Nunchuck slaven når den får tilsendt et handshake fra PSoC0. Det skal desuden kunne ses at en byte med værdien 0x00 modtages af Wii-Nunchuck.
Egentlig Resultat	Et *ACKNOWLEDGE* blev målt som forventet, og handshake byten med værdi 0x00 blev modtaget korrekt.

Tabel 10: Modultest af Wii-Nunchuck Handshake

Forventet Resultat	På I2C bussen måles en aflæsning af bytes fra Wii-Nunchuck slaven.
Egentlig Resultat	På målingen af I2C bussen ses det at bytes bliver aflæst fra Wii-Nunchuck slaven.

Tabel 11: Modultest af Wii-Nunchuck Data Aflæsning

Det kan på tabel 10 og 11 ses at de egentlige resultaterne stemte overens med de forventede.

I2C Kommunikationsprotokol

I2C Kommunikationsprotokollen beskrevet i afsnit 9.3.4 blev modultestet ved to tests. Den første test er til for at verificere at kommandotyper bliver overført på I2C bussen i korrekt format. Den anden test er til for at verificere at modtaget I2C data fortolkes korrekt af software på PSoC0.

Forventet Resultat	På I2C Bussen måles kommandotypen NunchuckData i korrekt format.
Egentlig Resultat	Målingen af I2C bussen viste kommandotypen NunchuckData i korrekt format.

Tabel 12: Modultest af kommandotype på I2C Bussen

Forventet Resultat
Egentlig Resultat

Tabel 13: Modultest af kommandofortolkningssoftware

SPI Kommunikationsprotokol

Rotationsdetektor

Indgangssignal	**Forventet PWM**	**Målt PWM**
0V	Ja	Ja
1400mV	Ja	Ja
1500mV	Ja	Ja
1600mV	Nej	Ja
2,2V	Nej	Ja
2,3V	Nej	Nej
5V	Nej	Nej

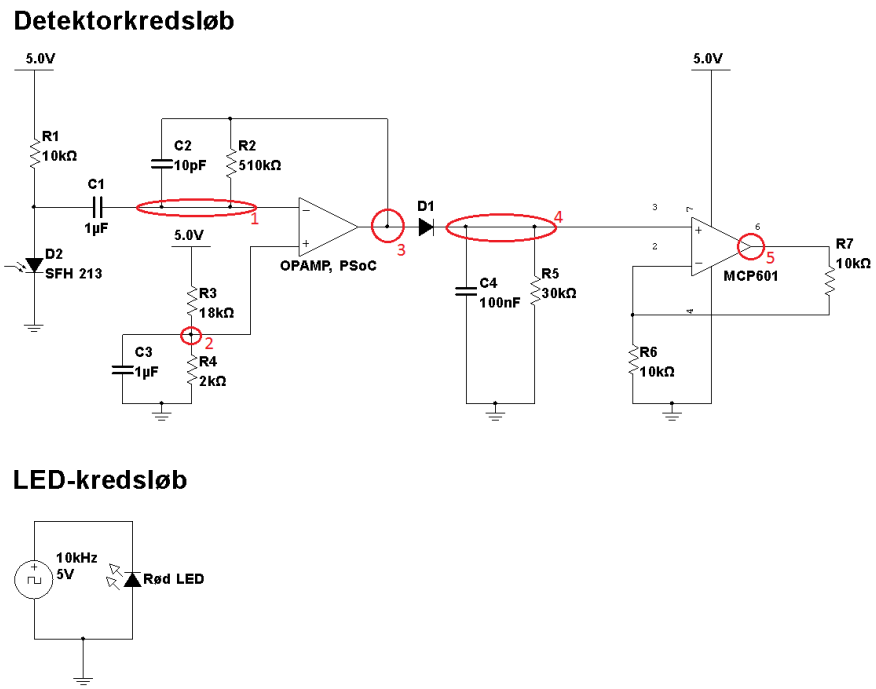
Tabel 14: Modultest af ADC

11.1.2 Hardware

Rotationsdetektor

Der er udført en række modultests af rotationsdetektoren. Først er der en beskrivelse af testen, der skulle teste de forskellige dele i rotationsdetektorkredsløbet, både mens fotodioden modtager et lyssignal fra LED'en, og når den ikke gør. Herefter følger en beskrivelse af modultesten af det båndpasfilter, der indgår i rotationsdetektoren og til sidst er der en beskrivelse af modultesten af motorens PWM-signal.

På figur 34 ses, hvor der er målt for at teste rotationsdetektoren.



Figur 34: Målepunkter på rotationsdetektor

I tabel 15 ses resultaterne fra modultesten af rotationsdetektoren, når fotodioden ikke får noget lyssignal. Det ses, at det virtuelle nul fra spændingsdeleren ligger på 0,4V, og at forstærkeren fordobler signalet, hvilket var meningen.

Knudepunkt	Målested	Forventet resultat	Målt resultat
1	Virtuelt 0	0,5V	0,4V
2	Spændingsdeler	0,5V	0,45V
3	Udgang på OpAmp på PSoC	0,5V	0,4V
4	Envelope detector	0,5V	0,4V
5	Udgang på forstærker	1V	0,85V

Tabel 15: Modultest af rotationsdetektor, når fotodioden ikke modtager lyssignal fra LED

I tabel 16 ses resultaterne fra modultesten af rotationsdetektoren, når fotodioden får lyssignal fra LED'en. Her ses det, at der igen er 0,4V, der hvor det var meningen, der skulle være 0,5V, så det er fint. Det ses også, at der er et PWM-signal på udgangen af PSoC'ens operationsforstærker og forstærkeren forstærker signalet, som ønsket.

Sted	Forventet resultat	Målt resultat
Virtuelt 0	0,5V	0,4V
Spændingsdeler	0,5V	0,45V
Udgang på OpAmp på PSoC	PWM, 0-5V	PWM, 0-4V
Envelope detector		3,7V
Udgang på forstærker		4,7V

Tabel 16: Modultest af rotationsdetektor, når fotodioden modtager lyssignal fra LED

I tabel **??** ses resultaterne fra modultesten af båndpasfilteret.

PWM-værdi	Forventet værdi	Målt værdi
40kHz		3V
29kHz		3,9V
10kHz		4,7V
10Hz		-
5V (lyser LED?)	Ja	Ja

Tabel 17: Modultest af forstærkerudgangen

Afskæringsfrekvenserne for båndpasfilteret ligger på 15,9Hz og 31,2kHz, så hvis båndpasfilteret skulle være helt optimalt skulle alt, der ligger udenfor disse frekvenser dæmpes fuldstændig. Det ses dog i tabel **??**, at det ikke helt er tilfældet. Ved 40kHz er signalet dæmpet til 3V, og ved 10kHz lader filteret alt gå igennem. Ved 10 Hz ses det, at signalet dæmpes, men der forekommer også mange spikes, som forstyrrer signalet. Derfor er der ikke indsat et tal i tabellen.

11.2 Integrationstest

11.3 Accepttest

12 Udviklingsværktøjer

12.1 QT Creator

QT Creator er blevet brugt til at designe brugergrænsefladen. QT's design widget suite/API er blevet brugt som den primære del af QT frameworket. Det fungerede godt til at designe en EDP-baseret grænseflade. Frameworket's indbyggede beskedsystem passede godt til vores design. En knap er assignet til et slot i brugergrænseflade-klassen, og når der trykkes på den pågældende knap, bliver det assignede signal broadcastet, og slot-funktionen bliver kørt. Design suiten simplificerer selve programmeringen af grænsefladen. Det skjuler mange af funktioner under kølerhjelmen.

Liste: PSoC Creator Analog discovery

13 Perspektivering

13.1 Perspektivering til semesterets kurser

14 Fremtidigt arbejde

Et problem ved produktet er, at affyringsmekanismen er meget tung i begge ender, så hvis der drejes for langt ud med Wii-nunchuck vipper den enten frem eller tilbage. Dette ville kunne have været afhjulpet med en H-bro. Denne ville nemlig så kunne have bremset affyringsmekanismen ved at være blevet sat til at køre i begge retninger i den tid, den skal holde stille. Hvis denne H-bro skulle medtages i systemet skulle der også skrives noget software, hvor H-broen blev sat til at køre i begge retninger på samme tid.

Noget andet der kan arbejdes videre med, er noget i samme stil som med bremsefunktionen af affyringsmekanismen. Problemet er, at affyringsmekanismens motor ikke kører særligt stærkt. Det skyldes, at rotationsdetektoren ikke kunne nå at stoppe motoren, inden den havde kørt en omgang mere. For at afhjælpe det problem blev motorens dutycycle sat ned, men det gjorde, at affyringsmekanismen nu ikke længere kunne skyde særligt hårdt. Derfor ville det være en god ide, at bruge en H-bro til også at styre denne motor, så den kunne få en bremsefunktion, ligesom den vertikale drejefunktions.

15 Referencer