



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

Vejleder: Gunvor Elisabeth Kirkelund


25. maj 2016

Indhold

Indhold	ii
1 Forord	1
2 Resumé	3
3 Abstract	3
4 Indledning	4
4.1 Krav til produktet	4
4.2 Systembeskrivelse	4
5 Kravspecifikation	6
5.1 Aktørbeskrivelse	6
5.1.1 Aktør - Bruger	6
5.2 Use case beskrivelse	7
5.2.1 Use case 1	7
5.2.2 Use case 2	7
5.3 Ikke-funktionelle krav	7
6 Projektafgrænsning	8
7 Metode	9
7.1 SysML	9
7.1.1 Afvigelser fra SysML Standard	9
8 Analyse	10
8.1 DevKit8000	10
8.2 Programmable System-on-Chip (PSoC)	10
8.3 Wii-Nunchuck	10
8.4 Motor	10
9 Systemarkitektur	11
9.1 Domænemodel	11
9.2 Hardware	11
9.2.1 BDD	11
9.2.2 Blokbeskrivelse	12
9.2.3 IBD	13
9.2.4 Signalbeskrivelse	15
9.3 Software	21
9.3.1 Software Allokering	21
9.3.2 Informationsflow i systemet	22
Wii-Nunchuck Information Flow	22
System Test	23
9.3.3 Samlede Klassediagrammer	24
9.3.4 SPI Kommunikations Protokol	27
Designvalg	28
9.3.5 I2C Kommunikations Protokol	29

Designvalg	30
9.3.6 Interface driver	31
9.3.7 Brugergrænseflade	31
Designvalg	31
10 Design og Implementering	32
10.1 Hardware	32
10.1.1 Motorstyring	32
H-bro	32
Rotationsbegrænsning	34
10.1.2 Affyringsmekanisme	35
Detektor	35
Motorstyring	37
Kanon og platform	38
10.2 Software	39
10.2.1 SPI - Devkit8000	39
10.2.2 Interface Driver	41
10.2.3 Brugergrænseflade	42
Demo GUI	43
10.2.4 Nunchuck	43
Afkodning af Wii-Nunchuck Data Bytes	43
Kalibrering af Wii-Nunchuck Analog Stick	44
10.2.5 PSoC Software	44
I2CCommunication	46
Nunchuck	47
SPI - PSoC	48
10.2.6 PSoC2 - Affyringsmekanisme	49
11 Test	51
12 Resultater	52
13 Fremtidigt arbejde	53
14 Konklusion	54
15 Referencer	55

1 Forord

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

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 eksamen d. 22. juni 2016. Produktet dokumenteres foruden denne rapport med en procesrapport, et dokumentationsdokument og diverse bilag.

På tabel 1 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							
Printudlægd design og Lodning		S	S	P			
PSoC Software	P	S		P			
Wii-Nunchuck							
PSoC Software	S	P		S			
SPI Kommunikationsprotokol							
Stik og Ledninger		P	P				
PSoC Software	P	S		P			
SPI Driver							
Kernemodul til Devkit 8000	S	S		S		P	
Brugergrænseflade							
Interface Driver	S			S	S		
Systemtest GUI					P		
Demo GUI					P		
Rotationsbegrænsning							
PSoC Software		P	P	S			
Use Case 2							
Implementering	P	S		P			
Motorstyring							
H-bro			P				
Ultiboard Design			P				
Lodning							
Affyringsmekanisme							

Tabel 1: Oversigt over ansvarsområder

2 Résumé


3 Abstract

This project aims to develop a candygun ment for use at parties and at leisure time. The finished product is to be controlled with a touchscreen for initiation of the system and a wii-nunchuck for adjusting and firing the gun. Furthermore, as demanded by predefined requirements, an embedded linux platform is included. For this purpose the Devkit 8000 is chosen. Further predefined requirements include use of sensors and actuators and well as the use of a PSoC is required - here a PSoC4 is chosen.

Through the analysis and specification of use cases the functional demands of the system are determined, then prioritized by the use of the MoSCoW-method. In the development of the project the agile development framework, Scrum, is chosen. By working incremental and iterative in the course of sprints the development has been well structured and reflected upon. The architecture of the system is described by use of the SysML. Through analysis of the communication between sub-systems protocols for I2C- and SPI has been developed.

The end-result of this project is a prototype whose hardware include motor control of three motors and three detectors, one which makes use of a potentiometer to limit horizontal rotation of the gun, another using a photodiode for an optical detection in the firing of the gun. The last, a Wii-nunchuck reacting to user input. The hardware is closely connected to mechanic parts, who enable rotation and firing of the gun. As far as embedded software goes a big part is the implementation of the two communication systems, SPI and I2C. An eventbased graphical user interface makes up the touchscreen functionalities. And the programming of 3 PSoC4s connect the parts to complete the system. All parts meet the requirements to test the communication protocols of the system. Playing of the game with shooting the gun meets the requirements to adjust and fire the gun, but is decoupled from the touchscreen initiation of the game, though an illustrational GUI for the game has been implemented. As a results requirements for the acceptance test, with few exceptions, has been met.

4 Indledning

Hensigten med dette projekt, er at udvikle spillet "Goofy Candygun 3000"  Spillet går ud på at 1-2 spillere dyster om, at ramme et mål med slik, affyret fra en slikkanon, styret af en Wii-Nunchuck controller. For at finde inspiration til projektet, og idéer til implementering, blev der søgt efter lignende projekter på internettet. Det viste sig, at idéen med at skyde med slik, ikke er en original idé, da lignende projekter såsom "The Candy Canon"[?] allerede findes. 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. Målet med projektet, er at bygge en funktionelt prototype, samt at dokumentere dette med en projektrapport og dens dertilhørende dokumentation. Det følgende afsnit beskriver, hvilke krav der stilles til projektet fra IHA's side.

4.1 Krav til produktet

Projektet 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 produktet skal indeholde. Disse krav er som følger:

- 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

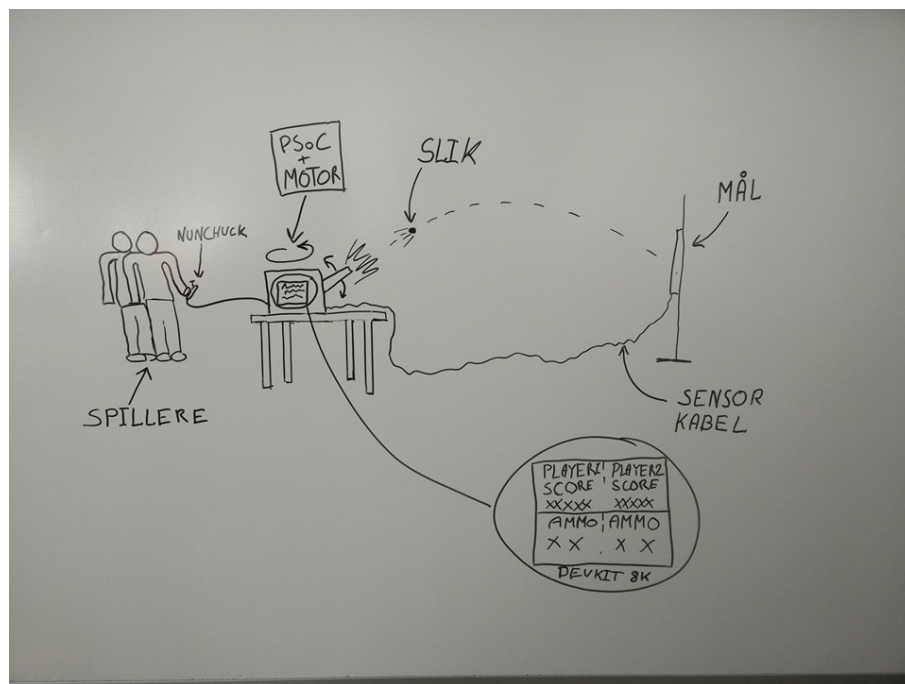
På baggrund af disse krav er der udarbejdet et produkt, der beskrives i afsnit 4.2.

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

4.2 Systembeskrivelse

I dette projekt skal der udvikles en slik kanon til spillet *Goofy Candygun 3000*. Denne slik kanon skal kunne skyde med slik, eksempelvis M&M's eller Skittles. Kanonen der afyrer slikket, skal styres af spillerne via en Wii-Nunchuck controller.

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. Spillestatistikker vises løbende på brugergrænsefladen. Dette brugerscenarie er illustreret i det rige billede på figur 1.



Figur 1: Rigt Billede af det endelige produkt

Det endelige produkt omfatter:

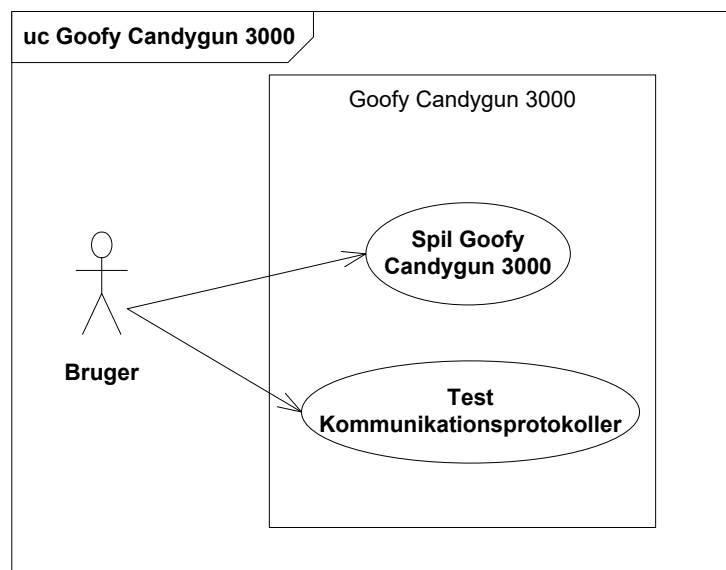
- En brugergrænseflade, hvor brugeren kan initiere både system test og selve spillet. Derudover kan brugergrænsefladen vise:
 - Point
 - Kanonens vinkel
 - Antal resterende skud
- En eller flere motorer, der drejer kanonen om forskellige akser
 - Disse skal styres med en Wii-nunchuck controller
- Et mål, der kan registrere spillernes skud

5 Kravspecifikation

Ud fra projektformuleringen er der formuleret en række krav til projektet. Disse indebærer to use cases og et antal ikke-funktionelle krav. Følgende afsnit beskriver aktøren for systemet samt de krav der er sat til systemet.

5.1 Aktørbeskrivelse

På figur 2 ses use case diagrammet for systemet. På figuren ses det at der er én primær bruger for systemet, brugeren.



Figur 2: Use case diagram

5.1.1 Aktør - Bruger

Aktørens Navn:	Bruger
Alternativ Navn:	Spiller
Type:	Primær
Beskrivelse:	Brugeren initierer Goofy Candygun 3000 samt starter systemtesten. Derudover har brugeren mulighed for at stoppe spillet igennem brugergrænsefladen. Brugeren vil under spillet interagere med Goofy Candy Gun gennem Wii-Nunchucken.

5.2 Use case beskrivelse

I dette afsnit følger en beskrivelse af de to use cases og de ikke-funktionelle krav, som er defineret i **#ref reference til kravspecifikationen i Dokumentationen**.

5.2.1 Use case 1

Brugeren initierer use casen ved at starte spillet via brugergrænsefladen og vælge spiltype; oneplayer eller partymode. Herefter vælges antallet af skud i et spil og disse puttes i magasinet. Når dette er gjort kan spillet påbegyndes. 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.

5.2.2 Use case 2

Brugeren initierer use casen ved at starte systemtesten via brugergrænsefladen. Herefter testes forbindelserne mellem hardwareblokkene forbundet via systemets busser. Hvis der sker fejl under systemtesten, rapporteres disse til brugeren via brugergrænsefladen og use casen afbrydes. Ved en successfuld systemtest, rapporteres dette til brugeren via brugergrænsefladen og systemet er klar til brug.

5.3 Ikke-funktionelle krav

Til beskrivelse af produktets specifikationer er der udarbejdet nogle ikke-funktionelle krav. Disse sætter krav til systemets og projektilers dimensioner. Derudover sættes der en begrænsning til kanonens rotation, samt krav til afvikling af kanonens udløsning.

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 diagnoserer af fejl
- Produktet should have:
 - Et mål til registrering af point
 - En lokal ranglistestatistik
- Produktet could have:
 - En multiplayer-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 allokateret 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 standarder. 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 hardware-systemer. 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 hardware-komponenter, deres signaler, og forbindelserne mellem dem.

Til modellering af software er der i rapport og dokumentation gjort brug af struktur- og adfærdsmaskinerne *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 SysML Standard

I SysML sekvensdiagrammer bruges beskedudveksling typisk til at repræsentere metoder på de objekter der kommunikerer med. Denne fremgangsmåde bruges i denne rapport, 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 7. 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 *Serial 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.

8.4 Motor

Der blev testet og stillet krav for hvad motoren skulle kunne for at den kunne bruges i dette projekt. Kravene var som følger:

- Skulle have en lave omdrejnings hastighed
- Skulle være en DC motor
- Den skulle bruge over 8V, for at h-broen ville kunne funger med motoren (for ifølge multisim kunne den ikke klar en spænding under 8V)
- Skulle gerne kunne trække noget

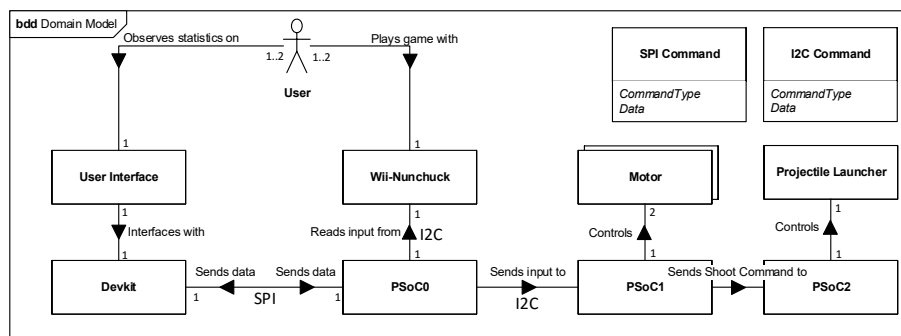
Efter nogen test og læsning i datablade, blev der fundet frem til en motor, EV3 Large motor, som overholdte alle kravene som den var blevet stillet for. Motoren bruge en spænding på 9V og den har en omdrejnings hastighed på 175rpm, læs mere om den i databladet#ref datablad, hvilket gjord den drejet pænt rundt.

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

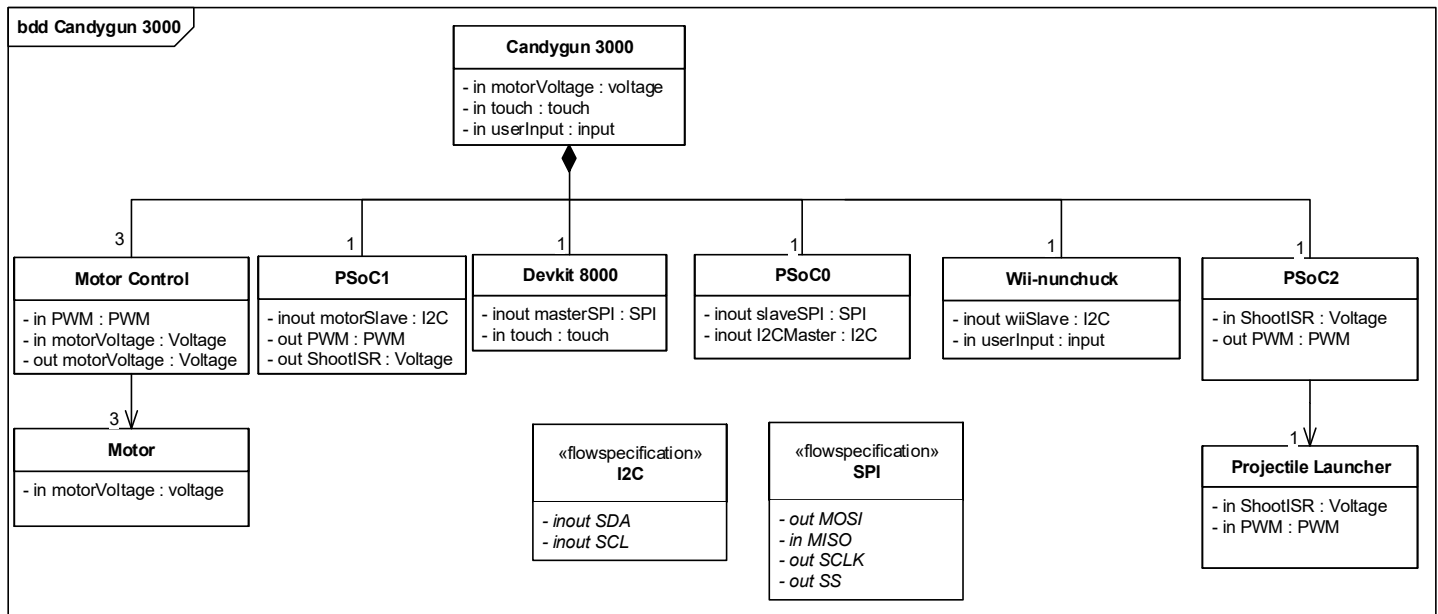
Her repræsenteres hardware som blokke forbundet med associeringer. Associeringerne viser grænsefladerne mellem de forbundne hardware komponenter (Enten *SPI* eller *I2C*), samt retningen af kommunikationen. Af modellen fremstår konceptuelle kommandoer for grænsefladerne, som beskriver deres nødvendige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

9.2.2 Blokbeskrivelse

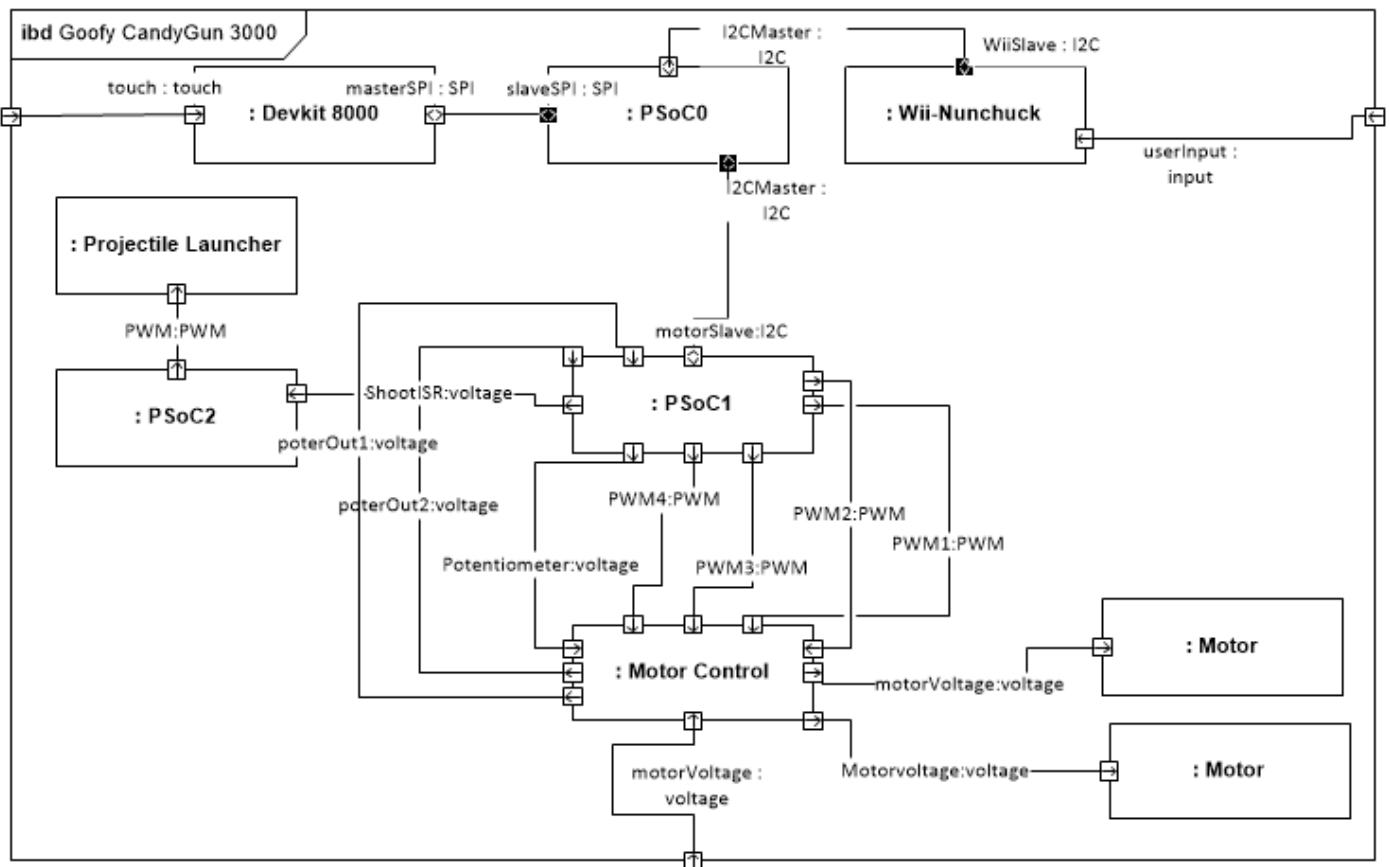
Følgende afsnit indeholder en blokbeskrivelse samt en flowspecifikation for *I2C* og *SPI*. I flowspecifikationen beskrives *I2C* og *SPI* forbindelserne mere detaljeret fra en masters synsvinkel. I blokbeskrivelsen beskrives hver enkel blok, så man har en ide om hvad hver blok består af, hvis de består af flere ting og så skal det give et overblik over hvad hver blok skal bruges til i systemet

Bloknavn	Beskrivelse
Devkit 8000	DevKit 8000 er en embedded Linux platform med touch-skærm, der bruges til brugergrænsefladen 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 master og SPI slave. Denne PSoC er bindeleddet mellem brugergrænsefladen og resten af systemets hardware.
MotorControl	MotorControl blokken er Candy Gun 3000's motorer, der anvendes til at bevæge kanonen. Denne blok består af H-bro blokken og rotationsbegrænsnings blokken.
Motor	Motor er motorene der bruges til at bevæge platform og kanon.
H-bro	H-bro bruges til at styre mototrens rotationsretning
Rotationsbegrænsning	Rotationsbegrænsning er til at begrænse platformens rotation så denne ikke kan dreje 360 grader. Den blok består af et potentiometer og en ADC'en, som sidder internt på PSoC0
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.
PSoC2	PSoC2 denne blok står for at sende et signal til affyringsmekanismen om at den skal skyde og så skal den også holde styr på at der kun bliver skydt en gang.
Projectile Launcher	Projectile Launcher denne blok er vores affyringsmekanismen

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

Blok-navn	Funktionsbeskrivelse	Signaler	Signalbeskrivelse
Devkit 8000	Fungerer som grænseflade mellem bruger og systemet samt SPI master.	masterSPI	Type: SPI Spændingsniveau: 0-5V Hastighed: 1Mbps 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: 1Mbps Beskrivelse: SPI bussen hvori der sendes og modtages data.
		wiiMaster	Type: I2C Spændingsniveau: 0-5V Hastighed: 100Kpbs 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.
		ShootISR	Type: voltage Spændingsniveau: 0-5V Beskrivelse: giv et højt signal når den skal skyde.
		PWM	Type: PWM Frekvens: 22kHz PWM %: 0-100% Spændingsniveau: 0-5V Beskrivelse: PWM signal til styring af motorens hastighed.
		PotenOut1	Type: voltage Spændingsniveau: en spænding 0V-5V alt efter hvad potentiometer står på Beskrivelse: den spænding viser hvor motoren står henne
		PotenOut2	Type: voltage Spændingsniveau: en spænding 0V-5V alt efter hvad potentiometer står på Beskrivelse: den spænding viser hvor motoren står henne
		PWM1	Type: PWM Frekvens: 3MHz PWM%: 0-100% Spændingsniveau: 0-5V Beskrivelse: PWM signal til styring af motorens hastighed.

		PWM2	Type: PWM Frekvens: 3MHz PWM%: 0-100% Spændingsniveau: 0-5V Beskrivelse: PWM signal til styring af motorens hastighed.
		PWM3	Type: PWM Frekvens: 3MHz PWM%: 0-100% Spændingsniveau: 0-5V Beskrivelse: PWM signal til styring af motorens hastighed.
		PWM4	Type: PWM Frekvens: 3MHz PWM%: 0-100% Spændingsniveau: 0-5V Beskrivelse: PWM signal til styring af motorens hastighed.
		motorVoltage	Type: voltage Spændingsniveau: 9V Beskrivelse: Strømforsyning til motoren
		potentiometer	Type: voltage Spændingsniveau: 5V Beskrivelse: giver Rotationsbegrænsning 5V
MotorControl	Den enhed der skal bevæge kanonen	PWM1	Type: PWM Frekvens: 3MHz PWM%: 0-100% Spændingsniveau: 0-5V Beskrivelse: PWM signal til styring af motorens hastighed.

		PWM2	Type: PWM Frekvens: 3MHz PWM%: 0-100% Spændingsniveau: 0-5V Beskrivelse: PWM signal til styring af motorens hastighed.
		PWM3	Type: PWM Frekvens: 3MHz PWM%: 0-100% Spændingsniveau: 0-5V Beskrivelse: PWM signal til styring af motorens hastighed.
		PWM4	Type: PWM Frekvens: 3MHz PWM%: 0-100% Spændingsniveau: 0-5V Beskrivelse: PWM signal til styring af motorens hastighed.
		motorVoltage	Type: voltage Spændingsniveau: 9V Beskrivelse: Strømforsyning til motoren
		potentiometer	Type: voltage Spændingsniveau: 5V Beskrivelse: giver Rotationsbegrænsning 5V
		PotenOut1	Type: voltage Spændingsniveau: en spænding 0V-5V alt efter hvad potentiometer står på Beskrivelse: den spænding viser hvor motoren står henne

		PotenOut2	Type: voltage Spændingsniveau: en spænding 0V-5V alt efter hvad po- tentiometer står på Beskrivelse: den spænding viser hvor motoren står henne
Motor	Denne blok beskri- ver hvad motoren får.	motorvoltage	Type: voltage Spændingsniveau: 0-5V Beskrivelse: giver spændning til motoren.(denne beskevelse glæder også for den anden motor)
Wii-nunchuck	Den fysiske control- ler som brugeren styrer kanonen med.	wiiSlave	Type: I2C Spændingsniveau: 0-5V Hastighed: 100kbit/sekund Beskrivelse: Kom- munikationslinje mellem PSoC1 og Wii-Nunchuck.
		userInput	Type: input Beskrivelse: Bru- gerinput fra Wii- Nunchuck.
SPI	Denne blok be- skriver den ikke- atomiske SPI forbindelse.	MOSI	Type: CMOS Spændingsniveau: 0- 5V Beskrivelse: Binært data der sendes fra master til slave.
		MISO	Type: CMOS Spændingsniveau: 0- 5V Beskrivelse: Binært data der sendes fra slave til master.

		SCLK	Type: CMOS Spændingsniveau: 0-5V Hastighed: 1Mbps Beskrivelse: Clock signalet fra master til slave, som bruges til at synkronisere den serielle kommunikation.
		SS	Type: CMOS Spændingsniveau: 0-5V 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 Beskrivelse: Databussen mellem I2C masteren og I2C slaver.
		SCL	Type: CMOS Spændingsniveau: 0-5V Hastighed: 100kbps Beskrivelse: Clock signalet fra master til lyttende I2C slaver, som bruges til at synkronisere den serielle kommunikation.
PSoC2	Denne blok binder ledet mellem afyrimingsmeknismen og PSoC1.	ShootISR	Type: voltage Spændingsniveau: 0-5V Beskrivelse: giv et højt signal når den skal skyde.

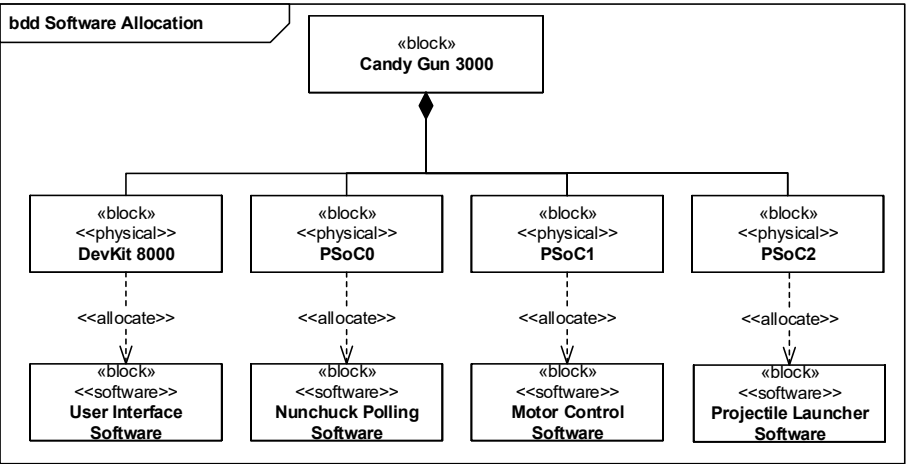
Projectile Launcher	Denne blok er vores afyrningsmeknis-men.	PWM5	Type: PWM Spændingsniveau: 0-5V Beskrivelse: giver signal til mosfet til at åbne så motoren køre en omgang.
---------------------	--	------	---

Tabel 3: Tabel med signalbeskrivelse

9.3 Software

9.3.1 Software Allokering

Domænemodellen i figur 3, side 11, 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.
Projectile Launcher Software	Dette allokerede software har til ansvar at aktivere affyringsmekanismen når et knaptryk detekteres på Nunchuck.

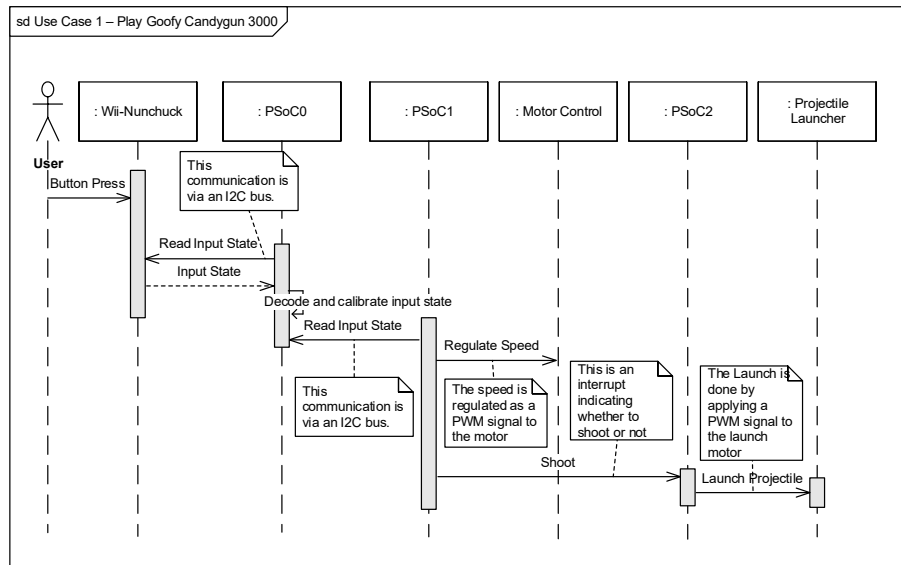
Tabel 4: Beskrivelse af den allokerede software

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

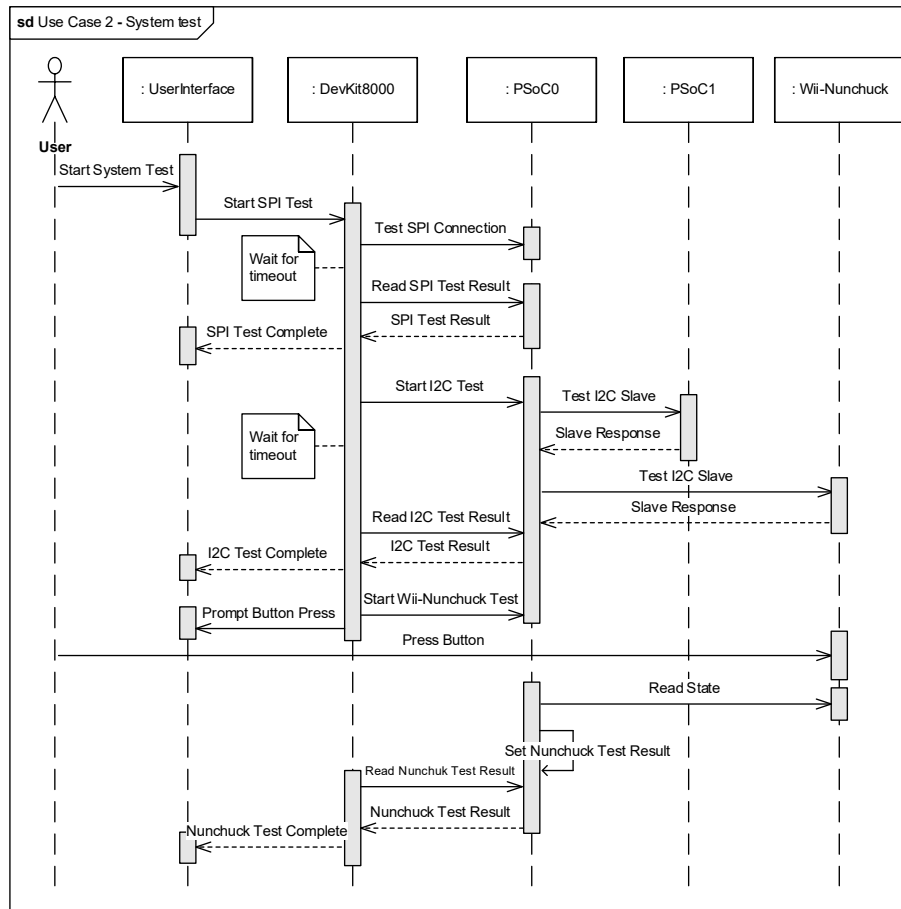
En essentiel del af systemet er at kunne styre motoren ved brug af Wii-Nunchuck controlleren. På figur 7 vises gennemløbet af Wii-Nunchuck input data fra Wii-Nunchuken til motoren, med de relevante CPU'er angivet. Her ses det at input data fra Wii-Nunchuck kontinuert bliver aflæst af PSoC0. Det bemærkes 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 7: Wii-Nunchuck Input Data Forløb

System Test

Use Case 2 beskriver test af systemet før spillet startes. På figur 8 vises test sekvensen for en vellykket test. Det ses her at system testen sker sekventielt, og tester systemets SPI og I2C busser.



Figur 8: System Test Forløb

9.3.3 Samlede Klassediagrammer

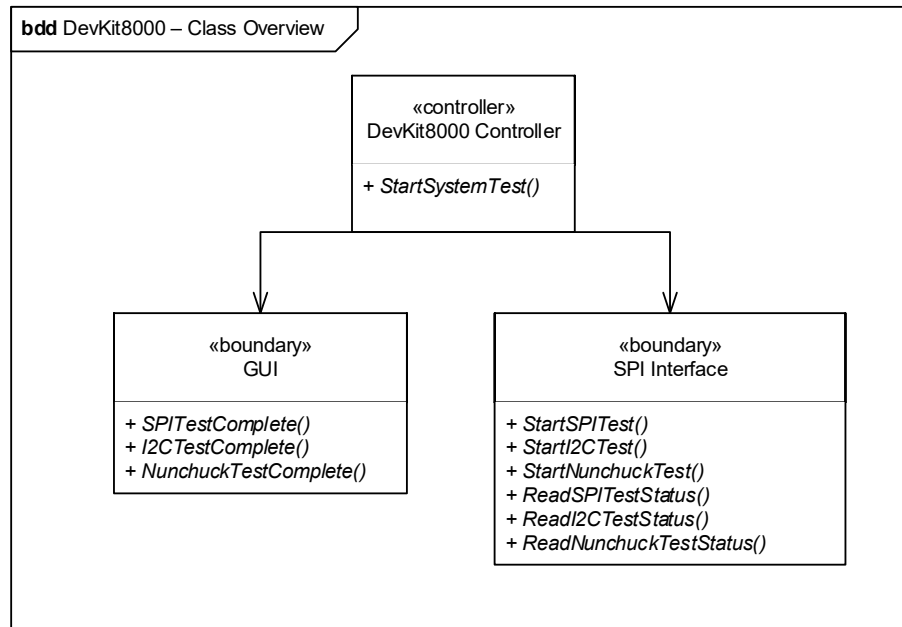
På baggrund af sekvensdiagrammerne i afsnit 9.3.2 samt de detaljerede applikationsmodeller analyseret i DOKUMENTATION #ref, er der udledt samlede klassediagrammer for hver individuel CPU i systemet. Disse er med til at identificere konceptuelle klasser og funktionaliter der skal overvejes i implementation og design af systemet, og har altså fungeret som et udgangspunkt til resten af udviklingsprocessen.

På figur 9, 10, 11, og 12 ses de samlede klassediagrammer for hver CPU.

For disse klassediagrammer er der konceptuelle klasser som gentager sig selv. Alle klassediagrammer har én klasse af typen *controller*. Disse klasse indeholder alt funktionalitet der er nødvendig for at kunne implementere systemets Use Cases. Yderligere bliver klasserne *I2C Interface* samt *SPI Interface* gentaget. Disse repræsenterer klasser for de tilsvarende bustyper, I2C og SPI, og bruges af softwaren til at sende og modtage data på disse busser.

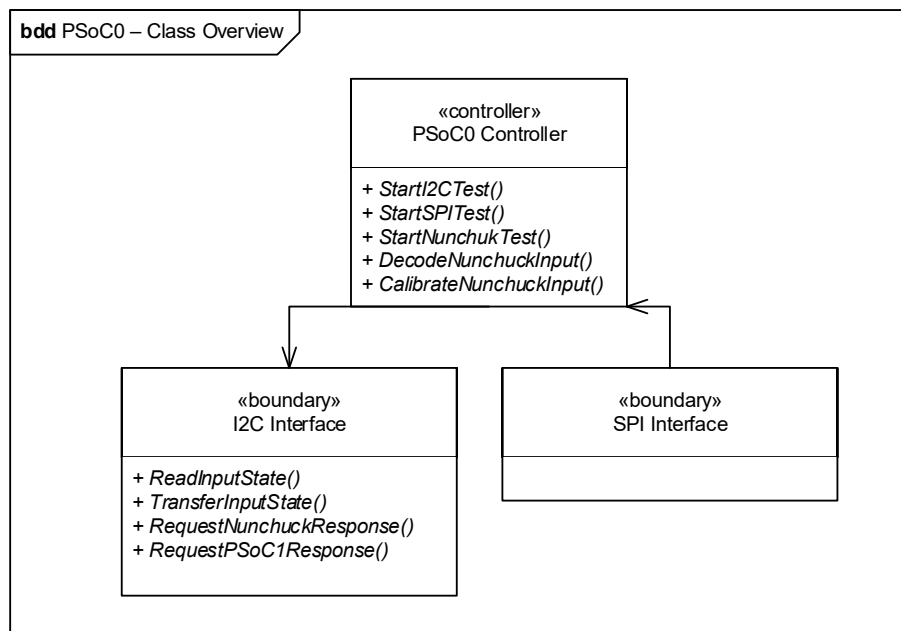
På figur 9 ses det at DevKit8000 controlleren skal have funktionalitet til start af system test, i relation til Use Case 2. For at kunne udføre dette

kommunikerer den med grænsefladen Graphical User Interface (*GUI*), for at kunne vise resultater til brugeren. Desuden skal controlleren kommunikere med grænsefladen *SPI Interface*, for at sende data ud til til resten af systemet via SPI bussen.



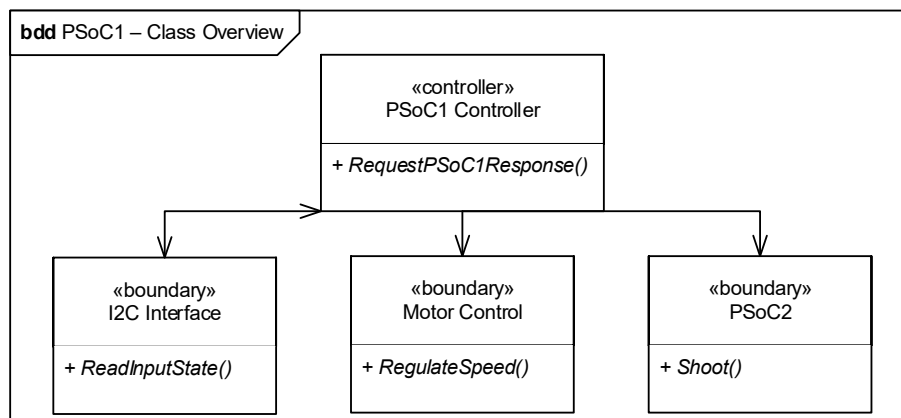
Figur 9: Samlet Klassesdiagram for DevKit8000

På figur 10 ses det at PSoC0 controlleren skal have funktionalitet til at starte tests af systemets busser. Yderligere skal den også kunne dekode og kalibrere data der kommer fra Nunchuck. I relation til disse funktionaliteter skal den kunne modtage og sende data på systemets I2C og SPI busser.



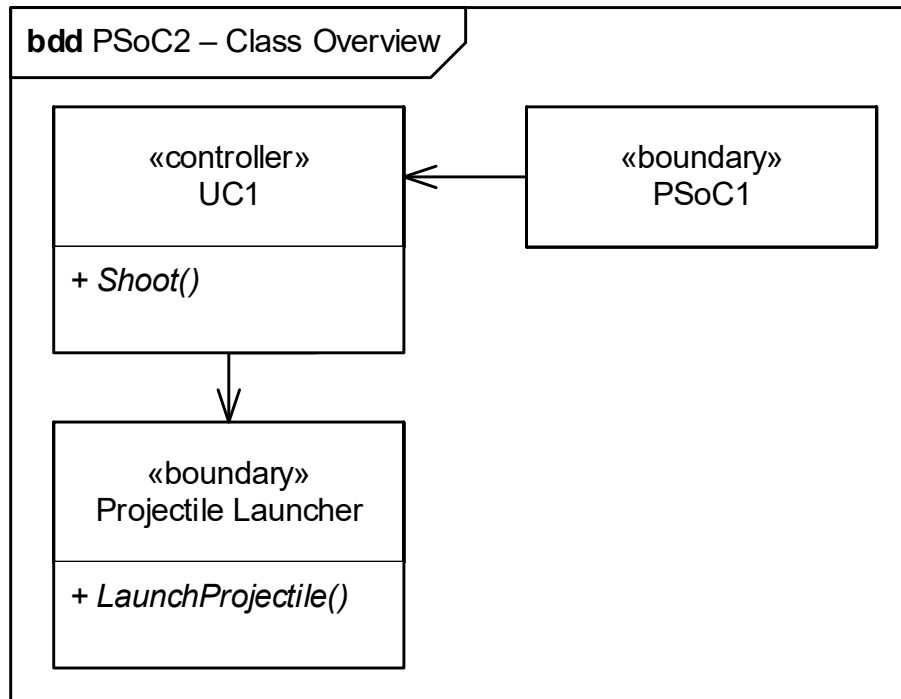
Figur 10: Samlet Klassediagram for PSoC0

På figur 11 ses det at PSoC1 controlleren kommunikerer med grænsefladerne *I2C Interface*, *Motor Control* samt *PSoC2*. Controlleren skal kunne regulere fart på systemets motorstyring, for at kunne styre kanonen. Desuden skal den kunne sende en affyringsbesked til *PSoC2*. For at regulere fart og affyre kanonen, skal controlleren kunne aflæse Nunchucken's tilstand fra *I2C Interface*.



Figur 11: Samlet Klassediagram for PSoC1

På figur 12 ses det at PSoC2 skal have funktionalitet til at aktivere afskydning, som gøres ved at kommunikere med grænsefladen *Projectile Launcher*. Afskydningen aktiveres af grænsefladen *PSoC1*.



Figur 12: Samlet Klassediagram for PSoC2

9.3.4 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.

Der er til systemet valgt en SPI bus mellem DevKit8000 og PSoC0, da der i dette tilfælde kun er brug for en bus med god support for én master og én slave. En SPI Bus skalerer ikke godt med multiple slaver i forhold til SPI, da en SPI bus skal have en fysisk forbindelse for hvert enkel slave der tilkobles. Bussen er dog simpel at implementere, og hver derfor ideel for denne grænseflade.

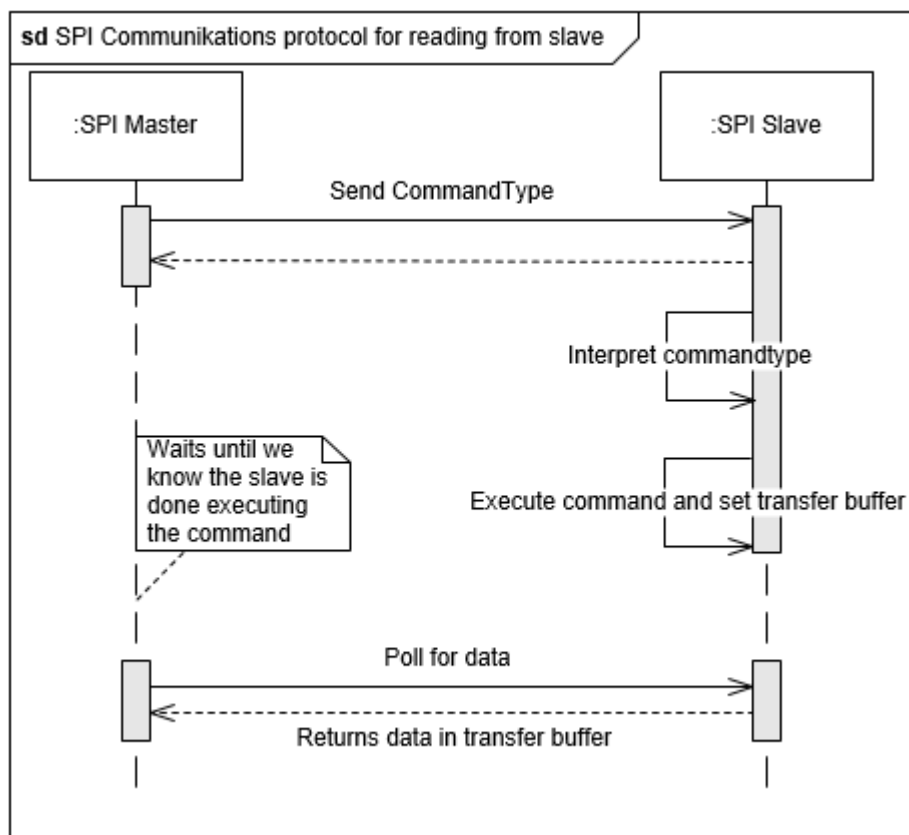
Kommandotype	Beskrivelse	Binær Værdi	Hex Værdi
START_SPI_TEST	Sætter PSoC0 i 'SPI-TEST' mode	1111 0001	0xF1
START_I2C_TEST	Sætter PSoC0 i 'I2C-TEST' mode	1111 0010	0xF2
START_NUNCHUCK_TEST	Sætter PSoC0 i 'NUNCHUCK-TEST' mode	1111 0011	0xF3
SPI_OK	Signalerer at SPI-testen blev gennemført uden fejl	1101 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

Kommandotyperne er udledt fra det samlede klassediagram for DevKit8000, figur 9. På figuren kan det ses at *DevKit8000 Controller* skal kunne starte tests

for SPI, I2C, samt Nunchuck og aflæse resultatet af disse.

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-bufferen. 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 13.



Figur 13: 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 13

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 nødvendig funktionalitet kan implementeres med den valgte model.

9.3.5 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 bussen er brugt til disse forbindelser af to primære grunde. Først og fremmest er Nunchuck controlleren implementeret som en I2C Slave fra producentens side. Derfor skulle der gøres brug af en I2C bus i alle tilfælde. Yderligere skulle Nunchuckens input sendes videre til PSoC1, og I2C er en bus der skalerer godt med multiple enheder. Derfor var det naturligt at udvide I2C bussen mellem Nunchuck og PSoC0 med PSoC1.

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 adresser

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.

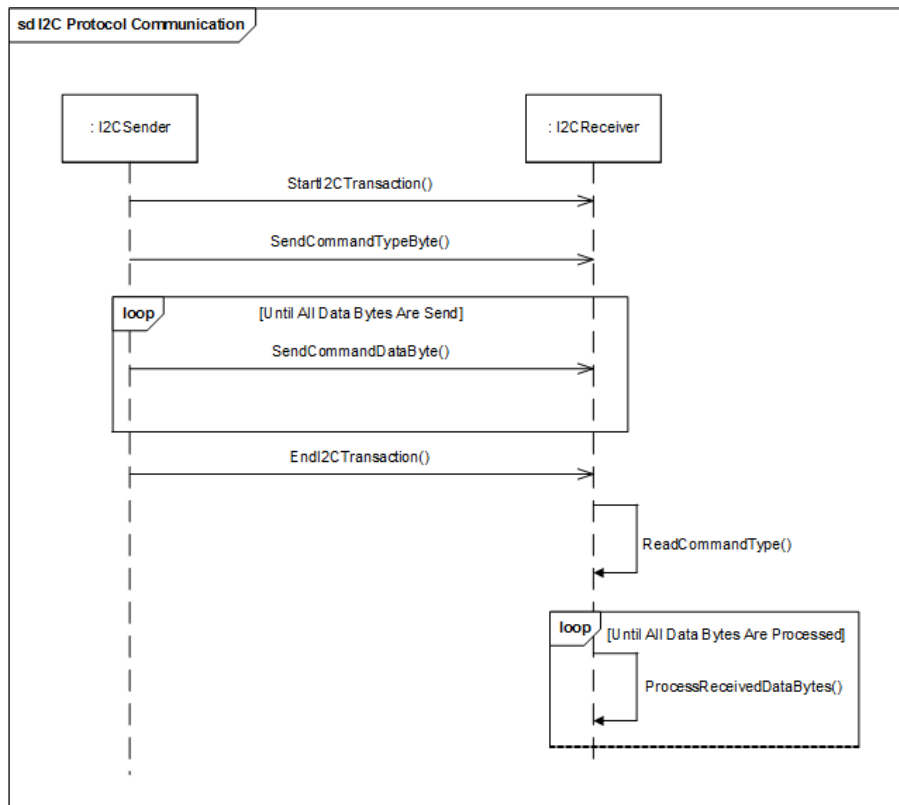
Kommandotype	Beskrivelse	Binær Værdi	Hex Værdi	Data Bytes
NunchuckData	Indeholder aflæst data fra Wii Nunchuck controlleren	0010 1010	0xA2	Byte #1 Analog X-værdi Byte #2 Analog Y-værdi Byte #3 Analog Buttonstate

Tabel 7: I2C kommunikation kommandotyper

Kommandotyperne er primært udledt fra det samlede klassediagram for PSoC1, figur 11. På figuren kan det ses at *PSoC1 Controller* skal kunne aflæse input tilstand fra I2C Interface. På det samlede klassediagram for PSoC0, figur 10, kan det også ses at PSoC0 Controller skal kunne udføre tilsvarende handling på I2C Interface. Denne funktionalitet er overført til kommandotypen *NunchuckData*.

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 14 ses et sekvensdiagram der, med pseudo-kommandoer, demonstrerer forløbet mellem en I2C afsender og modtager ved brug af kommunikations protokollen.



Figur 14: Eksempel af I2C Protokol Forløb

På figur 14 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,

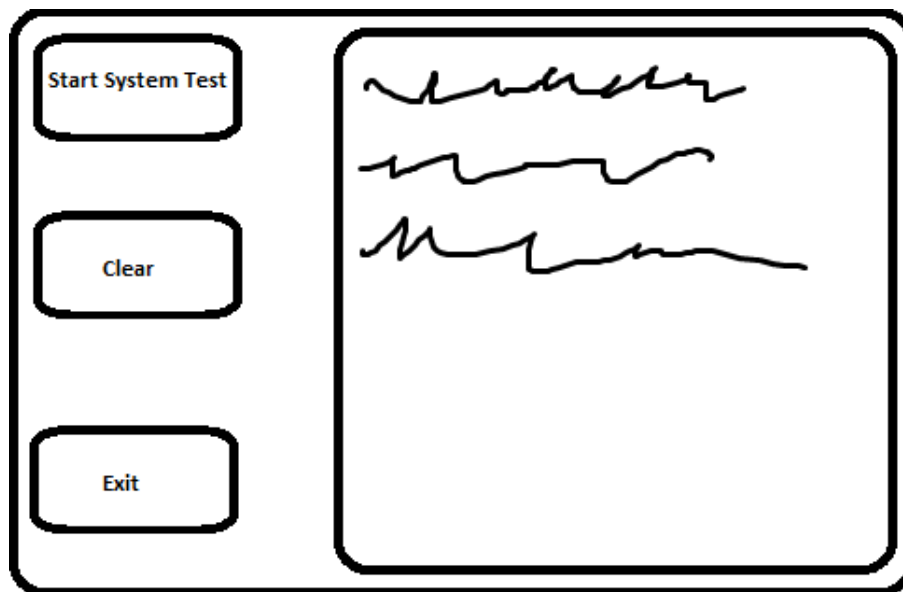
kan associere et dynamisk antal bytes til hver kommando. Dvs, hvis en modtager får en kommandotype *A*, vil den vide at den efterfølgende skal læse 5 bytes, hvormid at en kommandotype *B* ville betyde at der kun skulle læses 2 bytes.

En anden fordel ved kommandotype metoden er, at den er relativ simpel at implementere i kode, som vist ved pseudofunktionerne i figur 14.

9.3.6 Interface driver

9.3.7 Brugergrænseflade

Ved hjælp af en brugergrænseflade, kan systemtesten styres fra DevKit8000. Brugergrænsefladen er opbygget ud fra sekvensdiagrammet 8, ud fra dette kunne brugergrænsefladen skitseres som figur 15. Grænsefladen mellem DevKit8000 og PSoC0 er en SPI-bus som ses på figur 9.2.3, og brugergrænsefladen er koblet til DevKit8000 ved hjælp af interfacedriveren. Brugergrænsefladen skal sende startsekvensen til interfacedriveren, hvorefter dette sendes til PSoC0 og videre ud i systemet. Brugergrænsefladen skal aflæse svaret fra systemtesten og printe dette ud i en konsol.



Figur 15: Skitse af brugergrænseflade

Designvalg

Brugergrænsefladen er designet ud fra den sekventielle struktur i usecase 2 #Ref Dokumentation. Dette er løst ved hjælp af Event-Driven Programming. Denne model drives ved hjælp af events, som i dette tilfælde aktiveres af brugeren ved knaptryk. Knapperne vil blive tildelt forskellige funktionalteter, der faciliterer systemtesten. Et alternativ kunne være trådbaseret design. Komplexiteten i dette design ville være overvældende i forhold til den ønskede funktionalitet og blev derfor fravalgt.

10 Design og Implementering

10.1 Hardware

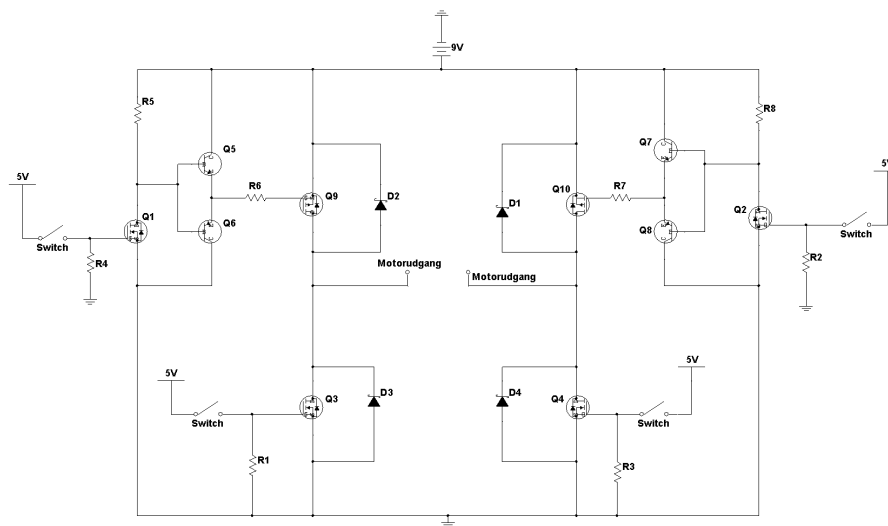
I Goofy Candygun 3000 indgår et antal hardwaredele, som skal udgøre det færdige system. Ud fra BDD'et ses det, at de to overordnede hardwaredele, der skal udvikles, er motorstyring til den vertikale og horisontale drejeretning, og en affyrimekanisme.

10.1.1 Motorstyring

Motorstyringen skal sørge for at kanonen kan styres i de vertikale og horisontale akser samt begrænse platformens rotation. Til at bevæge kanonen bruges to DC motorer, en til hver akse. Disse motorers rotationsretning styres med en H-bro. For at sikre at platformen ikke kan roteres 360 grader, er der udviklet en rotationsbegrænsning.

H-bro

H-broen er en del af motorblokken på BDD'et ???. Denne har til opgave at styre motoren, så den både kan køre til højre og venstre. Selvom der findes en komponent som indholder en H-bro, der også sagtens kunne benyttes, er der blevet designet en H-bro fra bunden, da det giver en bedre forståelse af, hvordan den fungerer. H-broen er opbygget af to identiske kredsløb, så der gives her en beskrivelse af den ene side af H-broen. På 16 ses det færdige kredsløbsdiagram over H-broen.



Figur 16: Kredsløbsdiagram for H-broen

I tabel 8 ses relevante komponentværdier for H-broen. En fyldestgørende oversigt over disse ses i ??

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

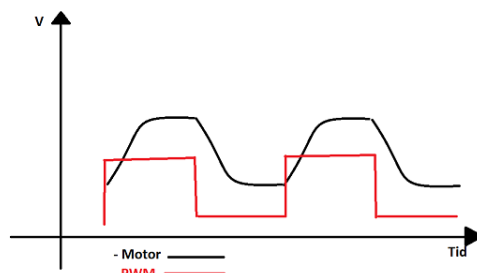
Tabel 8: Komponentbetegnelser på H-bro

Når motoren skal køre fremad, skal Q9 og Q4 åbnes. For at N-MOSFET'en (Q4) kan åbne, skal der en positiv spænding ind på gatebenet. For at P-MOSFET'en (Q9) kan åbne, skal den have en negativ spænding. For at løse dette blev der sat en N-MOSFET foran hver P-MOSFET.

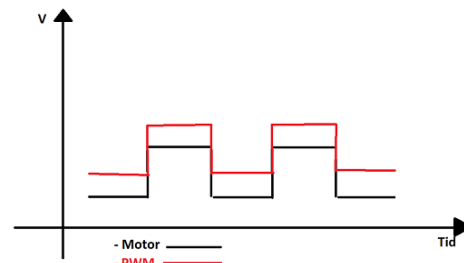
Da motoren gerne skal kunne skifte retning hurtigt og ofte, er det nødvendigt, at de to MOSFET's kan både åbne og lukke hurtigt. På figur 17 ses, hvordan Q9 åbner og lukker langsomt, hvilket skyldes kondensatoreffekten mellem benene på MOSFET'en.

For at få Q9 til åbne hurtigere blev Q5 indsat. Der var stadig det problem, at Q9 var langsom til at lukke, og derfor blev Q6 sat ind. På figur 18 ses, hvordan driveren gør, at Q9 både åbner og lukker hurtigt.

De to transistorer kunne godt have været erstattet af komponenten IRF2101, som er en driver, der gør præcis det samme, som beskrevet ovenfor. Denne er ikke anvendt, da det gav en større forståelse at udvikle driveren selv.



Figur 17: Illustration af Q9's åbne- og lukketid før transistorer blev sat ind i kredsløb



Figur 18: Illustration af Q9's åbne- og lukketid efter transistorer blev sat ind i kredsløb

De fire dioder, der er indsat i kredsløbet, skal fungere som beskyttelse af de fire MOSFET's. Det, de gør, er, at de sikrer, at den spænding, som er tilbage i motoren, når der lukkes for MOSFET'ene, ikke løber tilbage ind i mosfetene og brænder dem af.

For en fyldestgørende beskrivelse af H-broen, ses ??.

Rotationsbegrænsning

Platformen, som styres af motoren, må ikke kunne rotere 360 deg. Dette ses kravspecifikationen [#ref Reference til kravspecikation \(ikke-funktionelle krav\)](#). For at begrænse motorens bevægelse, anvendes et potentiometer samt en ADC. Når motoren bevæger sig, ændres potentiometerets modstandsværdi, og dermed ændres spændingsniveauet. På figur 19 ses den endelige opstilling af rotationsbegrænsningen.



Figur 19: Opstilling for rotationsbegrænsning

Potentiometer

Det anvendte potentiometer har en størrelse på $47\text{ K}\Omega$. Denne er lineær. Det vil sige at spændingen stiger proportionalt med modstanden. I potentiometeret findes en roterende kontakt, der danner en justerbar spændingsdeler over to conceptuelle modstande. Når skaftet på potentiometeret roteres ændres modstanden i de to variable modstande og dermed sker der en ændring i outputspændingen.

ADC

For at kunne aflæse spændingen på potentiometeret, anvendes en 12-bit AD converter af typen Sequencing Successive Approximation ADC. En sequencing SAR ADC indeholder et sample-hold kredsløb. Kredsløbet holder på et ind-

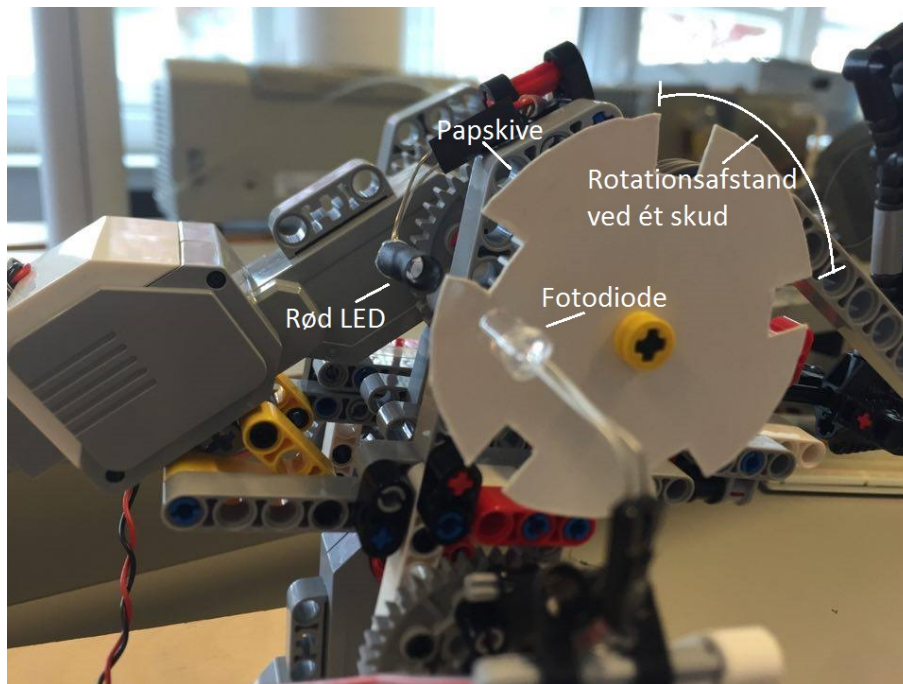
gangssignal indtil det næste signal registres på kredsløbets indgang. Dermed har converteren tid til at bestemme outputværdien.

10.1.2 Affyringsmekanisme

Affyringsmekanismen består af en motor; et motorstyringskredsløb; et detektor-kredsløb, 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 proportionelt 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.



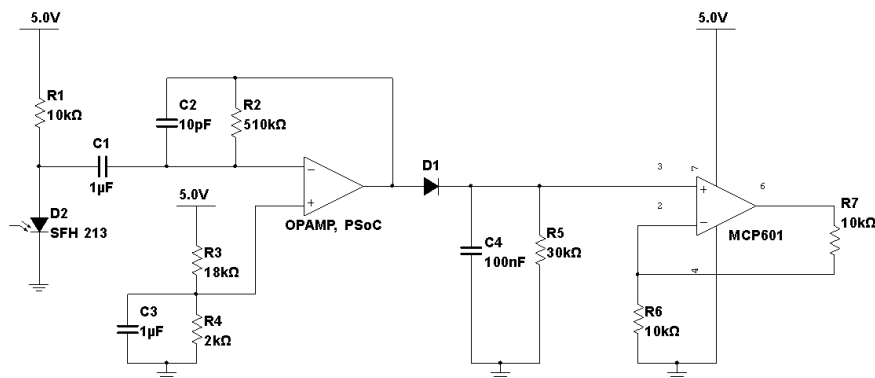
Figur 20: Detektorens placering på affyringsmekanismen

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.

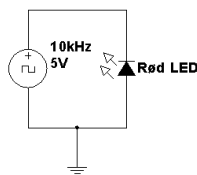
Detektoren skal kun sende et signal, når fotodioden ser lyset fra LED'en. Det er derfor vigtigt, at den ikke bliver forstyrret af dagslys og andre lyskilder. For

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.

Detektorkredsløb



LED-kredsløb



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 spændingsdelers, 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 dimensioneret efter, at den bliver opladet på de første udsving fra firkantsignalet. Modstanden er dimensioneret, 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 ??:

$$U_O = \left(1 + \frac{R_2}{R_1}\right) * U_P \quad (1)$$

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:

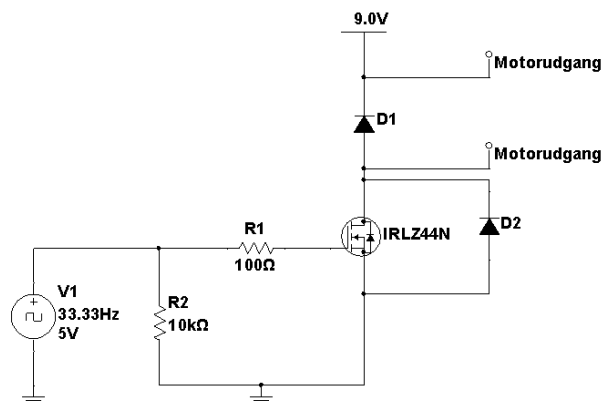
$$U_O = \left(1 + \frac{10k\Omega}{10k\Omega}\right) * 2V \quad (2)$$

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.

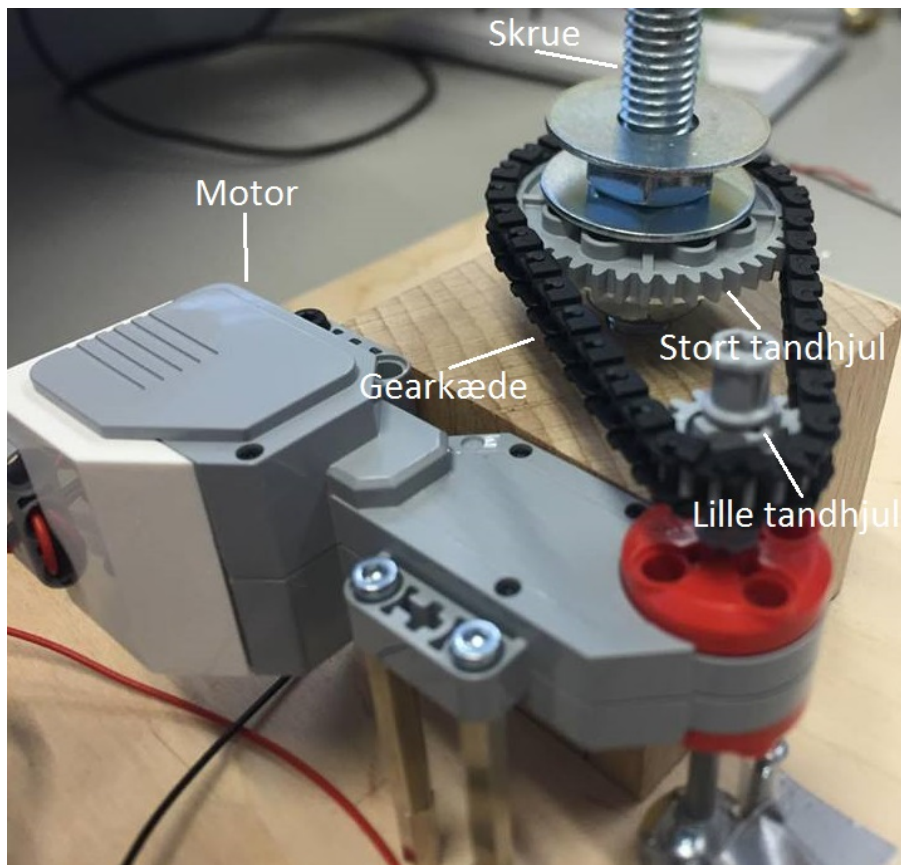


Figur 22: Diagram over motorstyring til motor på affyringsmekanisme

Dioden D1 er sat ind for at sikre motoren mod store spændingsspiques, 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 gearret 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.

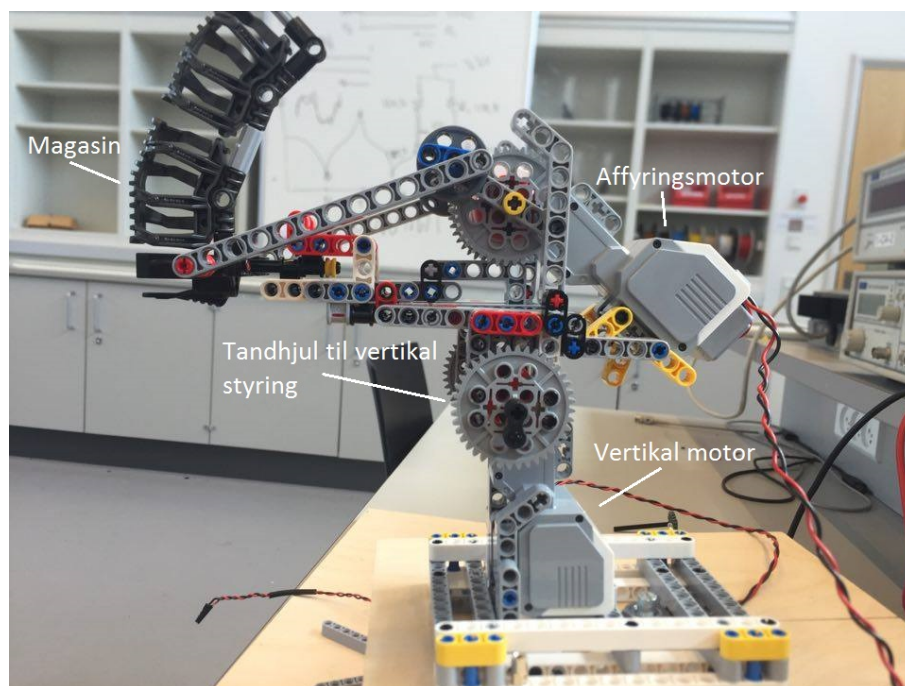


Figur 23: Horisontal mekanik

Mekanikken for den vertikale retning er bygget af LEGO. Et billede af op-

bygningen 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 mellemstørrelse 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

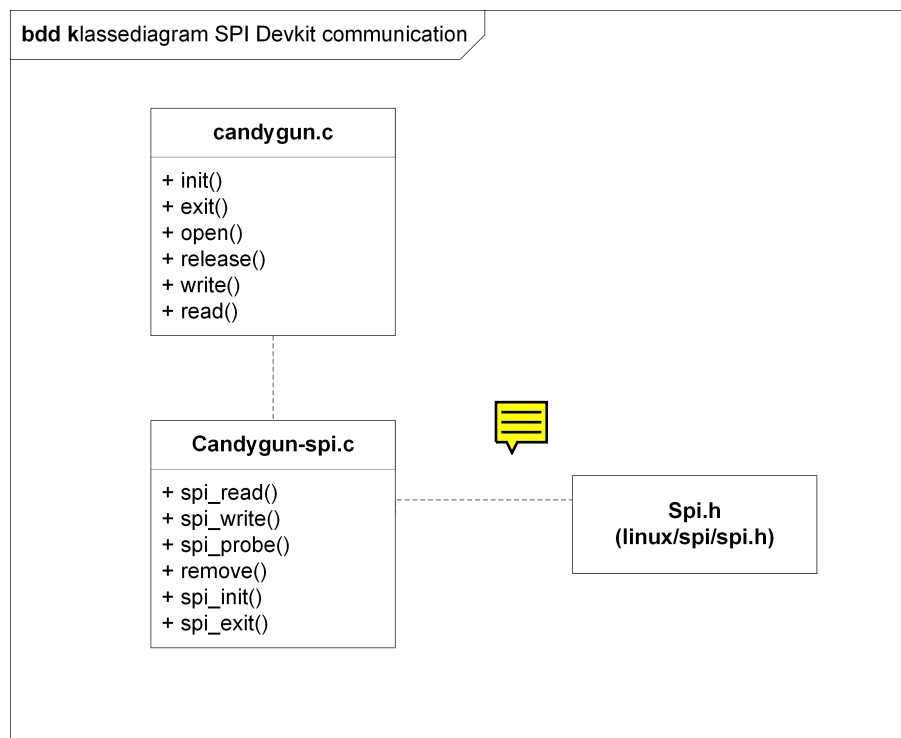
10.2.1 SPI - Devkit8000

candygun-driveren sørger for SPI-kommunikationen fra Devkit8000 til PSoC0. Driveren er skrevet i c, hvilket er typisk for drivere til linuxplatforme. I forbindelse med design og implementering af SPI, er der nogle indstillinger, der skal vælges. Disse indstillinger ses på tabel 9. Yderligere uddybning af de forskellige indstillinger kan ses i dokumentationen [#ref](#)

Tabel 9: Indstillinger for SPI

Indstillingsparameter	Værdi
SPI bus nr.	1
SPI chip-select	0
Hastighed	1 MHz
SPI Clock Mode	3
Bit per transmission	8

Selve driveren er i filen `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`. På figur 25 ses et klassediagram for opbygningen af driveren. I programmeringssproget c findes der ikke klasser, men selvom filerne i driveren ikke er opbygget som klasser, er de repræsenteret sådan i diagrammet for overskuelighedens skyld. De stiplede linjer i diagrammet indikerer at den ene klasse anvender den andens metoder, på samme måde som ved et bibliotek. `spi.h`, som også ses i diagrammet, er en indbygget del af Linux, og er derfor ikke yderligere dokumenteret her.



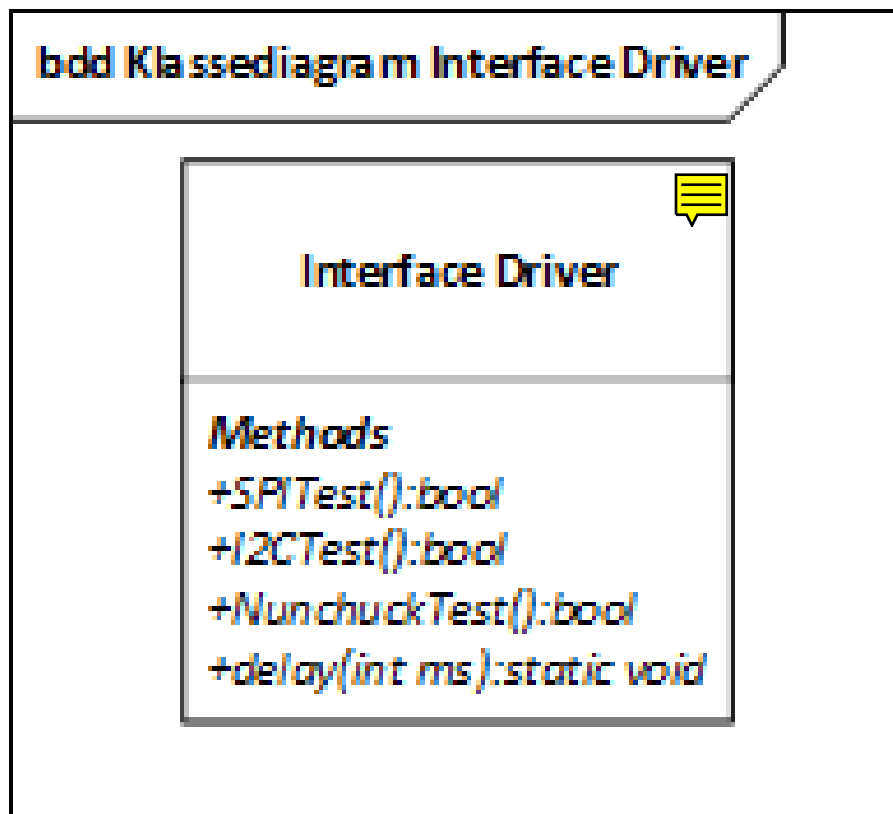
Figur 25: Klassediagram for SPI kommunikation på Devkit 8000

Når der skal skrives og læses, anvendes `read`- og `write`-funktionerne. Ofte ville 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 er generelt et behov for at sende 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. De resterende metoder, som ses i klassediagrammet på figur 25, er uddybet i dokumentationen #ref.

10.2.2 Interface Driver

Interface driveren fungerer som bindeled mellem brugergrænsefladen og candy-driveren på Devkit8000. Den indeholder tre funktioner.



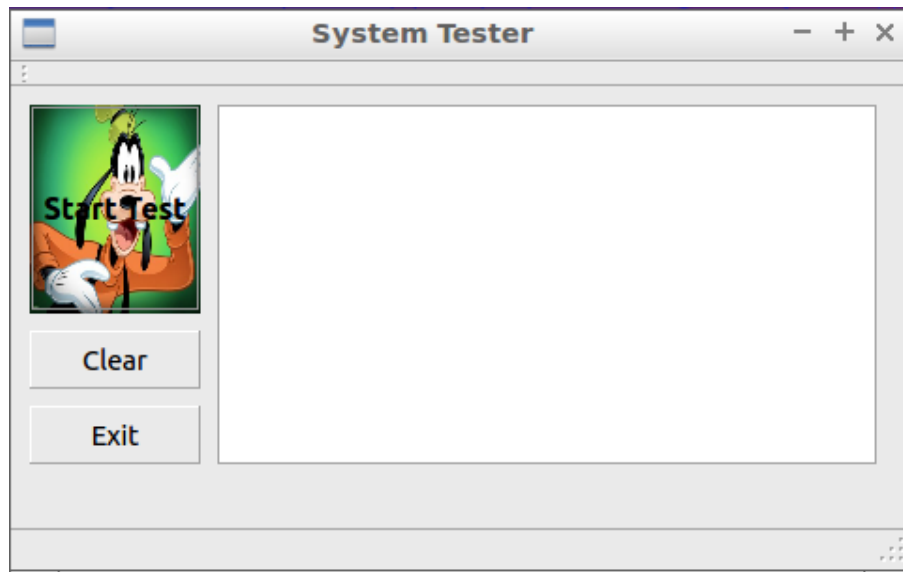
Figur 26: Klassediagram for Interfacedriver

Interface driveren indeholder fire metoder: **SPITest()**, **I2CTest()**, **NunchuckTest()** og **delay(int ms)**. De tre test metoder anvendes til at starte en test af de forskellige kommunikationsforbindelser: SPI, I2C og Nunchuck forbindelse. Hver af disse tre metoder tilgår `/dev/candygun` på DevKit8000. De skriver en unik værdi til filen, som SPI-driveren tilgår. Efter hver tilskrivning lukkes adgangen til filen. Herefter kaldes delay-metoden, så andre dele af systemet har

tid til at skrive et svar til /dev/candygun. Herefter læser metoderne fra filen. Rækkefølgen for metodekaldende kan ses på figur 8

10.2.3 Brugergrænseflade

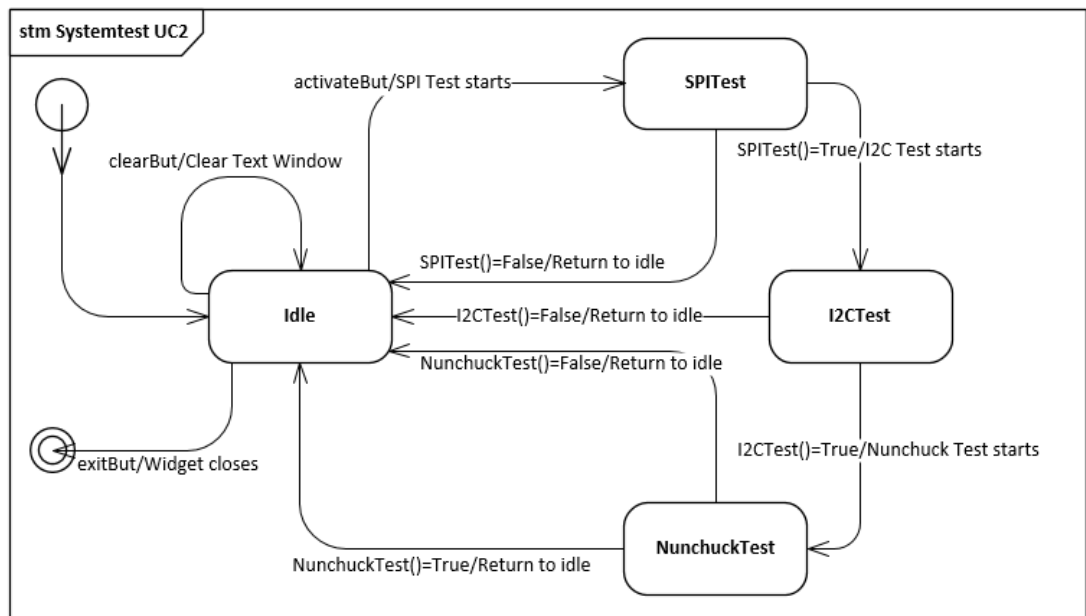
Usecase 2 styres via brugergrænsefladen fra Devkit8000. Dette afsnit beskriver brugergrænsefladens design. Figur 27 viser GUI'en for use case 2.



Figur 27: Brugergrænseflade for usecase 2 - Test kommunikationsprotokoller

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 at interagere i brugergrænsefladen. Når en knap er assignet til et slot i klassen, og der trykkes på den pågældende knap, bliver det tildelte signal broadcastet og slot-funktionen bliver kørt. Alle tre knapper i brugergrænsefladen er assignet signal-typen "clicked()".

Her vises en statemachine for brugergrænsefladens forløb igennem usecase 2.



Figur 28: State machine for brugergrænsefladen for usecase 2

Statemachinens forløb beskrives i dokumentationen, #RefDoku

Brugergrænsefladen for UC2 er en simpel test-konsol. Den består af 3 knapper og et konsol vindue. 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 konsol vinduet. 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 konsollen til blank tilstand. Den tredje knap, Exit, lukker widgeten. Koblingen i brugergrænsefladen hænger udelukkende sammen med interface-driveren. Knapperne kalder funktioner fra interfacedriveren, og ved ændring i disse funktioner, ville foresage ændringer i brugergrænsefladen.

Demo GUI

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:

$$AfkodetByte = (AflæstByte \text{ XOR } 0x17) + 0x17$$

Fra formelen 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 10

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 10: 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.

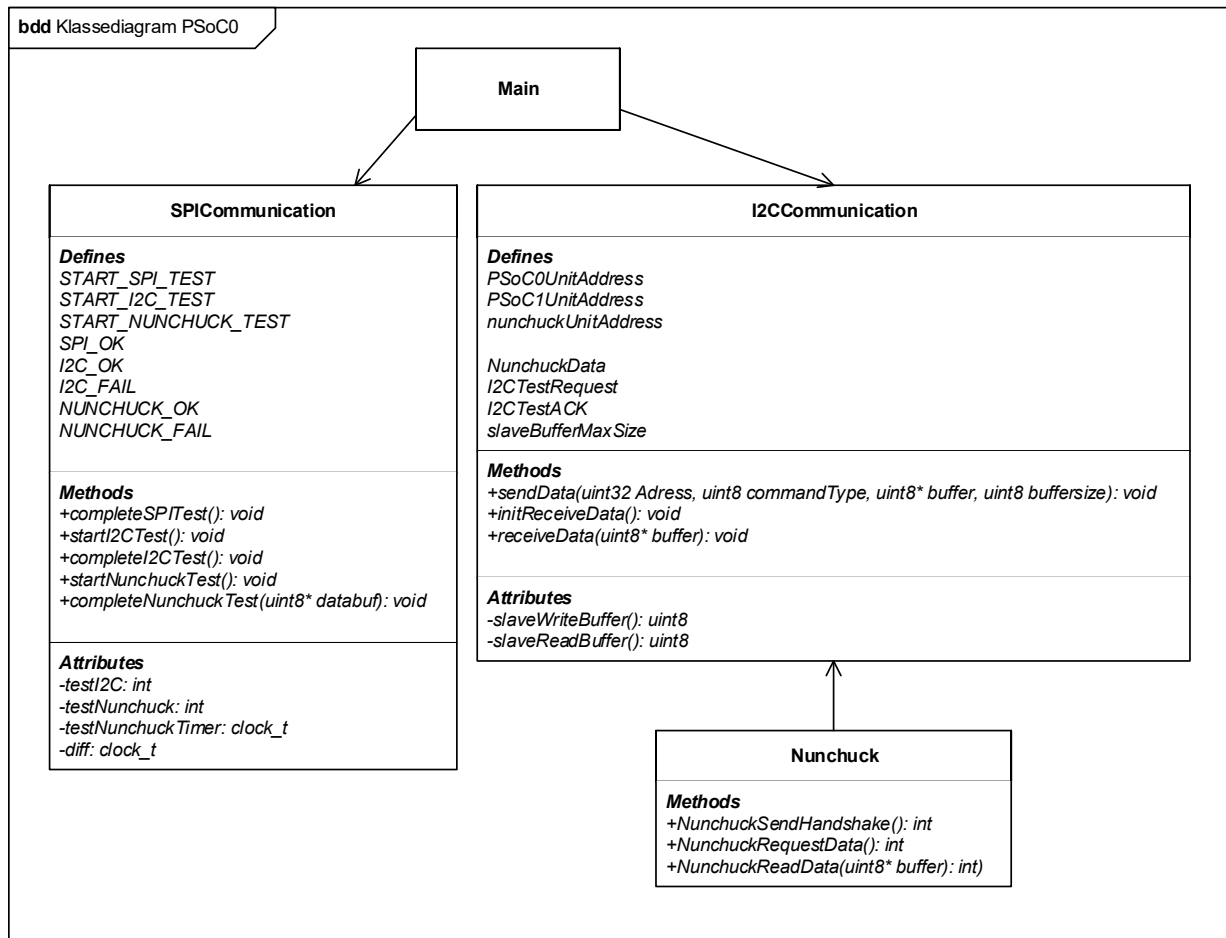
I projektet er de afkodede værdier for det analoge stick kalibreret med værdien -15 (0x0F i hexadecimal), altså ser den endelige formel for afkodning samt kalibrering således ud:

$$AfkodetByte = (AflæstByte \text{ XOR } 0x17) + 0x17 - 0x0F$$

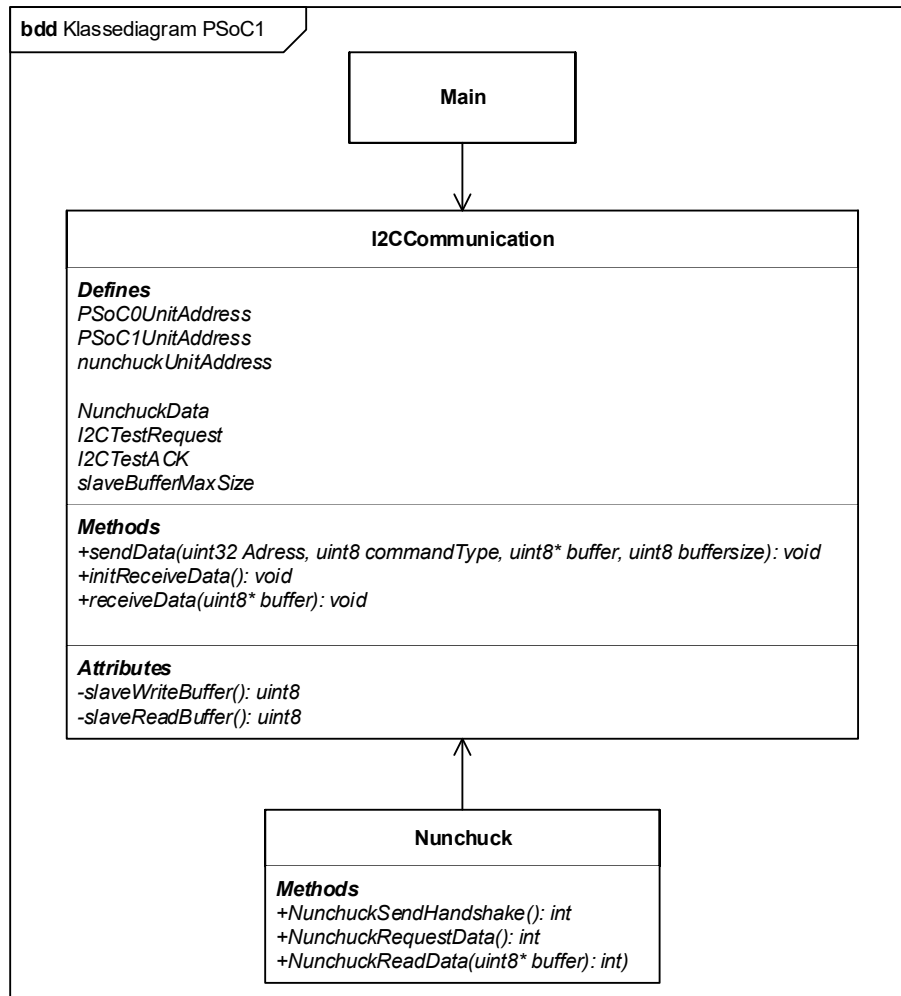
10.2.5 PSoC Software

På figur 29 og 30 ses de endelige klassediagrammer for PSoC0 og PSoC1. Disse klassediagrammer er designet ud fra applikationsmodellerne i dokumentationen **#ref Reference til applikationsmodeller i dokumentationen** og de samlede klassediagrammer for UC1- og 2 i systemarkitekturen figur 10 og 11. For at opretholde høj samhørighed, er der lavet en klasse for hver grænseflade. F.eks. indeholder "Nunchuck-klassen" al den software der skal til for at kommunikere med en nunchuck-enhed.

De efterfølgende afsnit vil beskrive klasserne og deres funktioner.



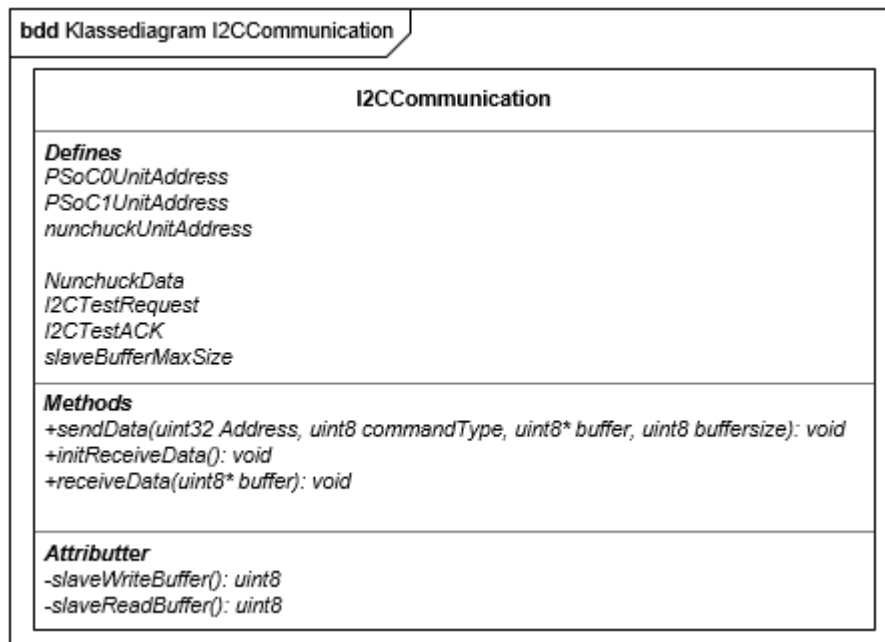
Figur 29: Klassediagram for PSoC0



Figur 30: Klassediagram for PSoC1

I2CCommunication

I systemarkitekturen på figur 10 og 11, er der udarbejdet klassediagrammer der beskriver funktionaliteterne for PSoC0- og 1 softwaren. Mht. til I2C-kommunikation klassen, er den overordnede funktionalitet, at kunne sende og modtage data via I2C-nettet. På figur 31 ses klassediagrammet for I2CCommunication klassen, der netop har metoder og attributter til at dække disse behov.

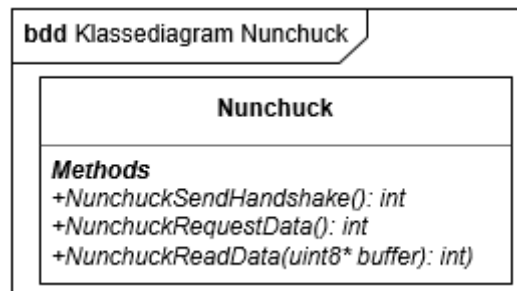


Figur 31: Klassesdiagram for I2CCommunication klassen

Den fulde klassebeskrivelse for I2CCommunication klassen kan findes i dokumentationen **#ref Indsæt reference til klassebeskrivelse for I2CCommunication i dokumentationen.**

Nunchuck

Nunchuck klassen er separat fra I2CCommunication klassen, på trods af at kommunikationen foregår over I2C-bussen, idét at Nunchucken bruger sin egen kommunikationsprotokol, som defineret i **#ref Reference til nunchuck i bilag**. Fra figur 10 og 11 i systemarkitekturen der omhandler PSoC software, er det udledt at softwaren skal have metoder til at skrive og læse fra nunchucken. På figur 32 ses klassesdiagrammet for den implementerede nunchuck klasse, der indeholder metoder der har disse funktionaliteter.

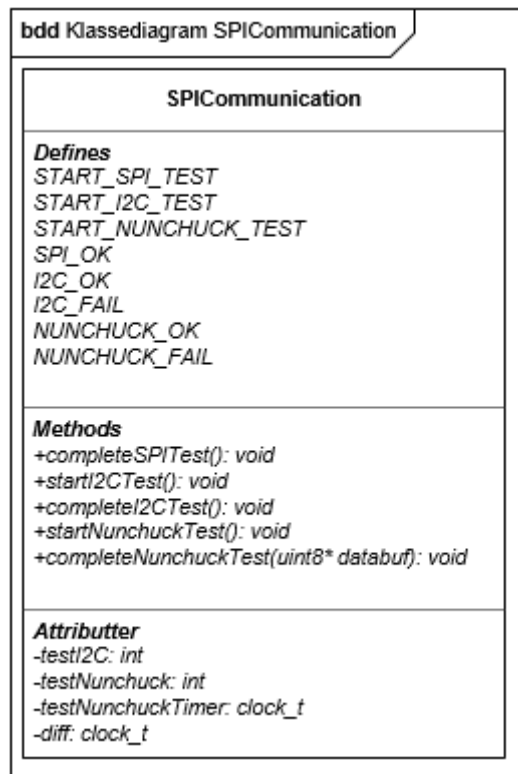


Figur 32: Klassediagram for klassen Nunchuck

Den fulde klassebeskrivelse for Nunchuck klassen og dens metoder kan ses i dokumentationen side [#ref Reference til Nunchuck klassebeskrivelser i dokumentationen](#)

SPI - PSoC

I dette afsnit vil softwaren der specifikt omhandler SPI-kommunikationen mellem PSoC0 og DevKit8000 blive beskrevet fra PSoC-CPU'ens perspektiv. På figur 33 ses den klasse der omhandler SPI-kommunikationen i systemet. Ud fra klassediagrammet figur 10 i systemarkitekturen, er det blevet udledt at SPI-klassen skal kunne starte og gennemføre forskellige test. Figur 33 viser klassediagrammet for den implementerede klasse, der afspejler netop dette.

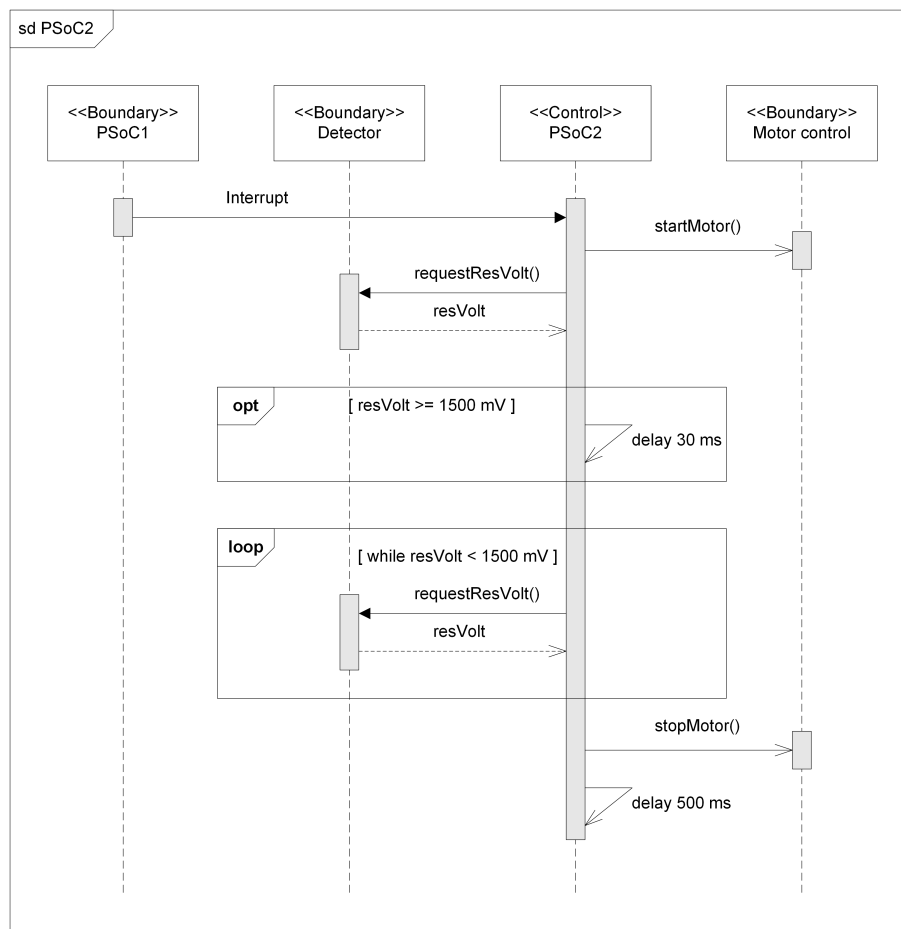


Figur 33: Klassesdiagram over klassen SPICommunication

En detaljeret klassebeskrivelse for SPICommunication klassen kan findes i dokumentationen side [#ref Referer til SPICommunication klassebeskrivelse i dokumentationen.](#)

10.2.6 PSoC2 - Affyringsmekanisme

Affyringsmekanismen består udover hardware og mekanik også flere softwaredele, som er programmeret på PSoC2. Til rotationsdetektorens operationsforstærker er der anvendt en, der er indbygget i PSoC'en. Derudover styres bl.a. interrupt og PWM-signaler i forbindelse med affyringsmekanismen ved hjælp af PSoC2. Desuden aflæses spændingen fra rotationsdetektoren af en SAR ADC, som også findes på PSoC2. For at se hvor de forskellige pins er ført ud til på PSoC2, og yderligere uddybning af indstillingerne for de forskellige blokke henvises til dokumentationen ???. I forbindelse med affyringen af kanonen køres en kodesekvens. Et sekvensdiagram for afviklingen af koden ses på figur 34.



Figur 34: Sekvensdiagram for PSoC2 software ved affyring af kanonen

Når der modtages et interrupt fra PSoC1, startes motoren, hvilket bevirker at kanonen begynder affyringen. Spændingen fra resVolt, anvendes til at vurdere, om fotodioden kan se lyset fra den røde LED. 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 det delay, der ses i "opt-boksen" på figur 34. Det sker for at sikre, at motoren er drejet, så de igen ikke kan se hinanden. Derefter aflæses spændingen fra resVolt igen - nu for at tjekke om affyringen er afsluttet. Aflæsningen gentages indtil fotodioden kan se den røde LED, og motoren dermed skal stoppes. Til sidst er der indsat 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.

11 Test

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



12 Resultater

I slutningen af produktudviklingen blev der afviklet en accepttest i samarbejde med gruppens vejleder Gunvor Elisabeth Kirkelund. Denne udfyldte accepttest kan ses i dokumentationen [#ref](#).

På tabel 11 ses en opsummering af resultaterne fra produktets udførte accepttests.

Use Cases	Resultat
Use Case 1 - Spil Goofy Candygun 3000	Denne use case er delvist implementeret.
Use Case 2 - Test kommunikationsprotokoller	Denne use case er implementeret.

Tabel 11: Opnåede resultater for systemets use cases

Use case 1 er delvist implementeret, da brugergrænsefladen er implementeret som en demo. Der er ingen bagomliggende funktionalitet til at vise spillets statistikker, samt administrering af player-modes. Der er en slik kanon som kan styres med en Nunchuck, og affyringsmekanismen kan aktiveres. Dog mangles kalibrering af motor og affyringsmekanisme, da motoren ikke kan bære kanonens vægt, og slik ikke kan affyres med tilstrækkelig kraft.

Projektførløbet resulterede i en prototype hvor use case 1 og use case 2 er implementeret som beskrevet.

13 Fremtidigt arbejde




Et problem med produktet, er at motoren der styrer kanonens vertikale rotation, ikke er kraftig nok til at holde affyringsmekanismen stabilt. Hvis kanonen vippe for langt frem eller tilbage, kan motoren ikke længere flytte affyringsmekanismen, og brugeren er nødt til at sætte kanonen tilbage til en start position. For at løse dette problem, kunne man geare motoren tilstrækkeligt, så den får en større trækraft, hvilket burde gøre den i stand til at løfte den tunge affyringsmekanisme.

Affyringsmekanismen på produktet, skubber nogle gange flere projektiler afsted, efter blot ét tryk på nunchucken. Dette skyldes at motoren der driver affyringsmekanismen stadig har momentum idét PWM signalet der driver motoren stoppes. For at afvikle dette problem, kunne der indsættes en H-bro før motoren, idét at denne, kan give motoren en "bremse" funktionalitet, der kan låse motorens rotation, og derved sørge for at kun et enkelt projektil bliver affyret.

I systemarkitekturen, og igennem implementeringen, bliver der i use case 2 kun refereret til én enkelt aktør, nemlig brugeren. Idét at use casen omhandler en system test, som den almene bruger ikke vil gøre brug af, kunne det være fordelagtigt at tilføje en "Admin" aktør til use casen, der giver adgang til system testen. Derved interagerer brugeren kun med use case 1 - Spil Goofy Candygun.

Systemets brugergrænseflade er ikke færdig implementeret, idét at der ikke gives nogen point til spilleren, for at skyde med kanonen, og derved kan der ikke føres statistik og highscores for spillerne. For at implementere denne funktionalitet, kræves det at der laves en form for målskive, der kan registrere skud fra kanonen, og at denne information bliver sendt til brugergrænsefladen. Dette kunne evt. gøres ved at sende informationen igennem PSoC0, og så videre til Devkittet via SPI-forbindelsen.

Til produktudviklingen er der blevet  brug af Devkit8000- og PSoC udviklingsboard, som har ubrugte funktionaliteter. Skulle produktet produceres til salg, ville det blive for dyrt og uhensigtsmæssigt gøre brug af udviklingsboards. I det endelige produkt skal komponenterne vælges ud fra de krav produktet har til funktionaliteter, så der ikke spildes penge på unødvendige funktionaliteter.

14 Konklusion

Formålet med dette projektforsøg var at udvikle *Goofy Candy Gun 3000* spillet, hvor 1 til 2 spillere dystet om at ramme et mål med projektiler affyret fra en slikkanon styret med en Wii-Nunchuck.

Som nævnt i Resultater, afsnit 12, endte projektet med en delvis implementering af use case 1 og en fuldtimplementeret use case 2.

Til use case 1, blev der konstrueret en kanon, som kan styres af en Wii-Nunchuck. For at aflæse data fra Wii-Nunchuck og omsætte data til en bevægelse i motoren, er der blevet implementeret et netværk af PSoC udviklingsboards, forbundet via en I2C-bus. Ifølge use case beskrivelsen, skulle der implementeres en brugergrænseflade til visning af statistikker og point. Disse funktionaliteter er ikke implementeret, med der er udviklet en demonstrations GUI, der viser opsætningen, dog uden nogen reelle funktionaliteter. Derudover skal der foretages nogle kalibreringer for kanonens motorstyring og affyringsmekanisme for at use case 1's krav er opfyldt til fulde.

For use case 2, er der implementeret en brugergrænseflade, hvor brugeren kan starte en systemtest og følge dennes fremskridt og resultater gennem testen. Brugergrænsefladen gør brug af en SPI-driver til Devkit8000, som muliggør kommunikationen med PSoC netværket via en SPI-bus. Denne use case er, ifølge den udfyldte accepttest, færdig-implementeret, uden mangler.

For projektet, blev der foretaget en MOSCOW-analyse (se afsnit 6 - Projektafgrænsning). I denne, blev mulige funktionaliteter prioriteret i en liste af *must have*, *could have*, *should have* og *won't have*. De højeste prioriterede funktionaliteter var:

- En motor til styring af kanonen
- En grafisk brugergrænseflade
- En Wii-nunchuck til styring af motoren
- En kanon med affyringsmekanisme
- En system test til diagnosering af fejl

Igennem projektet er disse højest prioriterede funktionaliteter implementeret i forbindelse med use case 1- og 2. De resterende funktionaliteter, der har lavere prioritet, er ikke implementeret.

15 Referencer

