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

26. maj 2016

Indhold

1 Forord

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

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 og diverse bilag.

Følgende denne rapport er en procesrapport der beskriver gruppens brug af udviklingsmodellen scrum[**?**]. Derudover beskriver procesrapporten udviklingsforløbet af projektet samt de værktøjer der er blevet anvendt til at fremme processen.

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							
Printudlægdesign 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	P	
Systemtest GUI					P		
Demo GUI					P		
Rotationsbegrænsning							
PSoC Software		P	P	S			
Motorstyring							
H-bro			P				P
Ultiboard Design			P				P
Lodning						P	P
Affyringsmekanisme							
Motorstyring						P	P
Rotationsdetektor						P	P
PSoC2 software						P	P
Mekanik - LEGO og træ						P	P
Lodning og ledninger						P	P

Tabel 1: Oversigt over ansvarsområder

1.1 Læsevejledning

PSoC Navngivning

Produktet, der beskrives i denne rapport, indeholder et netværk af PSoC4 udviklingsboards. For at kunne skelne mellem disse PSoC's er de blevet nummeret efter deres placering i forhold til Devkit 8000. Dermed er numrene efter PSoC ikke en indikation af, hvilken PSoC version der bruges, da der kun arbejdes med PSoC4 chippen under dette projektforløb. Tabel 2 giver en beskrivelse af ansvaret for hver PSoC samt deres navngivning.

Navn	Beskrivelse
PSoC0	Bindeled mellem Devkit 8000 og I2C netværket samt Wii-Nunchuck software
PSoC1	Software til motorstyring
PSoC2	Software til affyringsmekanisme

Tabel 2: Navngivning for PSoC4 udviklingsboards

Bilag

Når der refereres til bilag bruges følgende konvention:
(Bilag/Mappe/Undermappe/Filnavn)

Som et eksempel gives:
"For flere detaljer se (Bilag/Dokumentation/WiiNunchuck.pdf)".

Her findes filen *WiiNunchuck.pdf* så i mappen *Bilag*, under mappen *Dokumentation*.

2 Resumé

Denne rapport beskriver semesterprojektet for tredjesemesterstuderende, hvor en prototype for spillet Goofy Candygun 3000 er slut produktet. Ideen bag Goofy Candygun 3000 er et spilsystem hvor en eller to spillere skyder en slikkanon efter et mål for at opnå det største antal point. Brugeren interagerer med systemet gennem en touch skærm og en Wii-Nunchuck. Til projektet har IHA stillet nogle krav til hvilke elementer systemet skal indeholde (Bilag/Projektrapport/Semesterprojekt3Oplæg). Dette inkluderer en indlejret linux platform, en sensor, en aktuator og en PSoC platform.

Produktets prototype består af et netværk af PSoC4 udviklingsboards og en Wii-Nunchuck, som kommunikerer via I2C kommunikationsprotokollen. Brugeren interagerer med systemet gennem en grafisk brugergrænseflade på Devkit 8000's touch skærm. Via Wii-Nunchuck kan brugeren kontrollere kanonens sigte, samt affyre skud.

Ikke alle dele af produktet er fuldt implementeret. Implementationen af hoved use casen, som omhandler selve spillet, er påbegyndt, men ikke færdigimplementeret. Gennem prioritering med MoSCoW princippet [**?**] er kommunikationen mellem linux og PSoC platformene blevet prioriteret højest og dermed er use casen omhandlende systemtesten blevet færdigimplementeret.

3 Abstract

This paper describes the third semester project, in which a prototype for Goofy Candygun 3000 is the final product. The vision behind Goofy Candygun 3000 is game system where one or two players aim a candycannon at a target to score the highest amount of points. The system is controlled by the user through a touch screen and a Wii-Nunchuck. For this project IHA has required that the system includes an embedded linux platform, a sensor, an actuator and a PSoC platform.

Mainly, the paper describes the design and implementation of a prototype for Goofy Candygun 3000. Following this paper, a report about the usage of the agile development framework, scrum, and the group's workprocess throughout the project is described.

The prototype consists of a network PSoC4 developmentboards, which communicate amongst each other by the I2C-bus. The user interacts with the system through a graphical userinterface on the Devkit 8000's touch screen. Using the Wii-Nunchuck the user controls cannon's aim, and fires the cannon.

The prototype has not been fully implemented. The implementation of the main use case, which addresses gameplay functionalities, has not been fully implemented. Using the MoSCoW method, communication amongst the linux and PSoC platforms has been prioritized highly and therefore the use case addressing the systemtest has been implemented in full.

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 Cannon* [**?**] 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. Målet med projektet, er at bygge en funktionelt prototype, samt at dokumentere dette med en projektrapport og dens dertilhørende dokumentationdokument. 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. Semesterprojektet, præsenteret af *Ingeniørhøjskolen, Aarhus Universitet* (Bilag/Projektrapport/Semesterprojekt3Oplæg). 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:

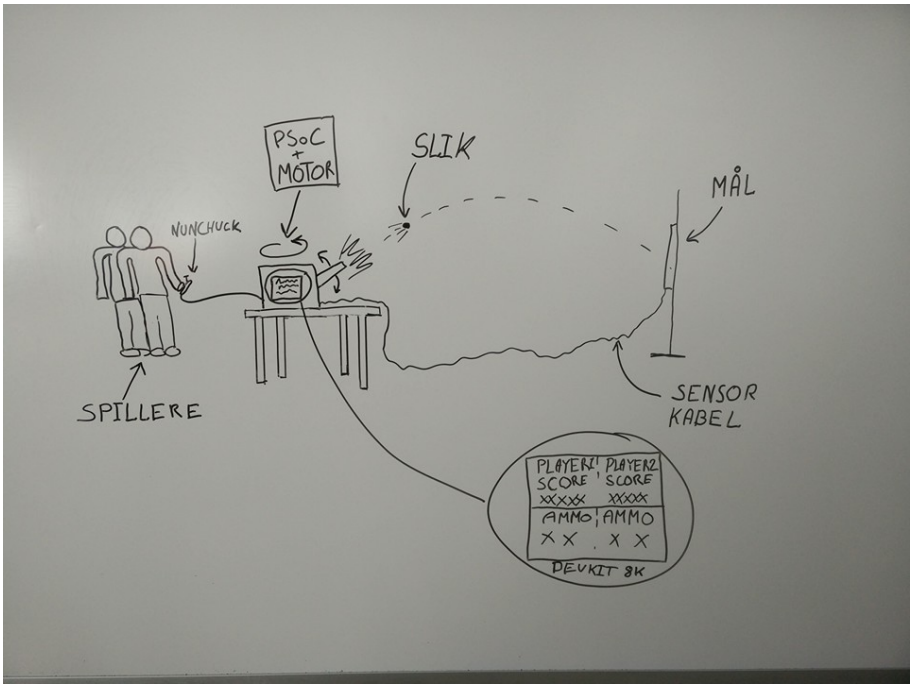
- Systemet *skal* via sensorer/aktuatorer interagere med omverdenen
- Systemet *skal* have en brugergrænseflade
- Systemet *skal* indeholde faglige elementer fra semesterets andre fag
- Systemet *skal* anvende en indlejret Linux platform og en PSoC platform

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

4.2 Systembeskrivelse

I dette projekt skal der udvikles en slik kanon, som skal bruges i et nyt spil som kommer til at hedde *Goofy Candygun 3000*. Denne slik kanon skal kunne skyde med slik. Kanonen, der affyrer slikket, skal styres af spillerne via en Wii-Nunchuck controller. Spillet vil have en brugergrænseflade, hvor brugeren kan vælge spiltype, og se statistikker og point for hvert spil.

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. Dette brugerscenarie er illustreret i det rige billede på figur 1.



Figur 1: Illustration af brugsscenarie for Goofy Candygun 3000

Det endelige produkt, hvis alle ønsker til produktet opfyldes, omfatter:

- En brugergrænseflade, hvor brugeren kan initiere både system test og selve spillet. Derudover kan brugergrænsefladen vise:

  Point

  Kanonens vinkel

  Antal resterende skud

- Består af 3 motorer, der drejer kanonen om forskellige akser

  Disse skal styres med en Wii-nunchuck controller

  To af motorerne styrer kanonen i forskellige retninger, og den sidste er til at affyre kanonen

- Et mål, der kan registrere spillernes skud

- Muligheden for at flere spillere kan spille sammen

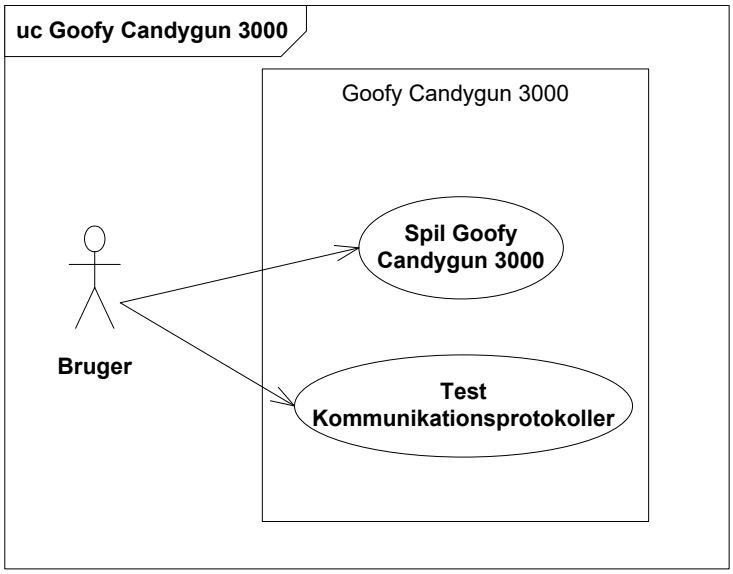
Brugergrænsefladen vil blive implementeret på et Devkit 8000. Denne vil kommunikere med et netværk af PSoC udviklingsboards, der tager input fra en Wii-Nunchuck, og fortolker disse til signaler, for at styre kanonens motorer.

5 Kravspecifikation

Ud fra systembeskrivelsen 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 defineret for 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

På tabel 3 vises aktørbeskrivelsen for brugeren for systemets use cases.

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.

Tabel 3: Aktørbeskrivelse for brugeren

5.2 Use case beskrivelse

I dette afsnit er der en kort beskrivelse af systemets use cases og dets ikke-funktionelle krav. De fully dressed use cases kan findes i dokumentationen, afsnit 1.4- *Fully Dressed Use Cases* side #ref.

5.2.1 Use case 1

Brugeren initierer use casen ved at starte spillet via brugergrænsefladen og vælge spiltype: oneplayer eller partymode. 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 exit-knappen på brugergrænsefladen, og denne vender tilbage til starttilstanden.

5.2.2 Use case 2

Brugeren initierer use casen ved 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.

5.3 Ikke-funktionelle krav

Til beskrivelse af produktets specifikationer er der udarbejdet nogle ikke-funktionelle krav. Kravene omhandler produktets dimensioner, kanonens rotationsevner og kanonens affyring. De ikke funktionelle krav findes i dokumentationen afsnit 1.5 - *Ikke funktionelle krav* side #ref.

5.4 Accepttestspecifikation

Ud fra systemets use cases og ikke-funktionelle krav, er der blevet udarbejdet en accepttestspecifikation, for at verificere systemets funktionaliteter, efter endt udviklingsforløb. Denne kan findes i dokumentationen afsnit 2 - *Accepttestspecifikation* side #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 tydeliggør, hvilke dele af systemet, der er højst prioriteret. 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 detektering af fejl i kommunikationsprotokoller
- Produktet should have:
  - En lokal ranglistestatistik
  - En multiplayer-indstilling til over to spillere
- Produktet could have:
  - Trådløs Wii-nunchuckstyring
  - Afspilning af lydeffekter
  - Et mål til registering af point
- Produktet won't have:
  - Et batteri til brug uden strømforsyning
  - Online ranglistestatistik

Must have kravene har den højeste 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 hardware for use case 1 samt hele use case 2 implementeres. Grunden til dette er, at den grundlæggende kommunikation mellem udviklingsboardene er en essentiel del, idét kommunikationen er grundlaget for begge use cases.

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* [**?**] (Bilag/Projektrapport/SysMLSpecification) som visuelt modelleringsværktøj til at skabe diagrammer. SysML blev valgt, da det er en anerkendt industristandard [**?**], hvilket betyder, at det er mere universelt brugbart, og dokumentationen bliver letlæselig for andre ingeniører.

SysML er et modelleringssprog, der bygger videre på det meget udbredte modelleringssprog UML [**?**]. Men hvor UML hovedsageligt er udviklet til brug i objektorienteret 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 anvendt 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 anvendelighed for modellering af hardware og software. En kort opsummering ses i det følgende.

Til modellering af hardware er der i rapport og dokumentation gjort brug af strukturdiagrammerne *Block Definition Diagrams* (BDD) (Et eksempel på et BDD kan ses på figur 4) samt *Internal Block Diagrams* (IDB) (Et eksempel på et IBD kan ses på figur 5). Disse diagrammer er brugt til at beskrive systemets hardwarekomponenter, deres signaler og forbindelserne mellem disse.

Til modellering af software i rapport og dokumentation er der gjort brug af struktur- og adfærdsdiagrammerne *Block Definition Diagrams*, *Sequence Diagrams* (SD) (Eksempel på SD kan ses på figur 8) og *State Machine Diagrams* (STM) (Et eksempel på STM kan ses på figur 30). Der er desuden gjort brug af aktivitetsdiagrammer til beskrivelse af algoritmer for systemets software. Disse diagrammer er brugt til at beskrive systemets softwarekomponenter i form af klasser, relationer mellem klasserne, samt hvordan disse klasser interagerer med hinanden og hvilke 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 kommunikeres med. Denne fremgangsmåde anvendes

også i denne rapport, dog er der nogle sekvensdiagrammer, der afviger ved at beskedudvekslingen repræsenterer handlinger påført 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 letforståelig måde.

7.2 Scrum

Projektarbejdet er planlagt og udført vha. scrum [**?**]. Hvordan gruppen har gjort brug af scrum, og hvilke erfaringer gruppen har gjort sig, bliver beskrevet i procesrapporten.

8 Analyse

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

8.1 Devkit 8000

Devkit 8000 (Bilag/Projektrapport/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. Devkit 8000 understøtter desuden de serielle kommunikationsbusser SPI og I2C, hvilket er typiske busser der bliver brugt til kommunikation med sensorer og aktuatorer.
Devkit 8000 platformen er også brugt i forbindelse med undervisning på IHA, hvilket betyder, at gruppen allerede har erfaring med software udvikling til denne platfrom.

8.2 Programmable System-on-Chip (PSoC)

PSoC [**?**] er en microcontroller, der kan programmeres 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 forskellige kommunikationsbusser, såsom SPI og I2C.

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 (I2C protokollen), så den nemt kan sættes op, til at kommunikere med resten af systemet.

8.4 Motor

Der er testet og stillet krav for hvad motoren skal kunne for, at den kan bruges i dette projekt. Kravene er som følger:

- Motoren skal være en DC motor
- Den skal have en lav omdrejningshastighed
- Den skal have trækkraft nok til at kunne rotere kanonen
- For at den kan fungere i systemet, skal den forsynes med 8V eller mere.
  - Ifølge Multisim virker H-broen ikke med spænding under 8V

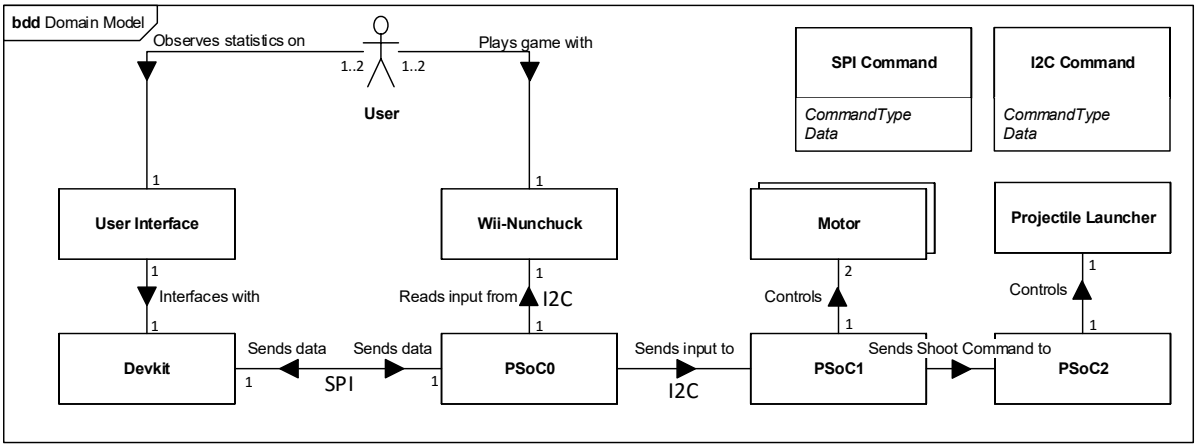
Efter nogle tests med forskellige motorer, er der fundet frem til, at en EV3 Large motor [**?**] overholder alle krav, som er stillet. Motoren forsynes med en spænding på 9V og har en omdrejningshastighed på 175rpm, hvilket gør motorens rotationer er langsommere, så den kan styres lettere. For mere information omkring motoren henvises der til databladet kilde [**?**]

9 Systemarkitektur

Dette afsnit præsenterer systemets arkitektur i en grad, der gør det muligt at forstå grænsefladerne mellem hardware- og softwarekomponenter. En mere detaljeret beskrivelse for systemarkitekturen, er dokumenteret i dokumentationen afsnit 3 - *Systemarkitektur*

9.1 Domænemodel

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



Figur 3: Systemets domænemodel

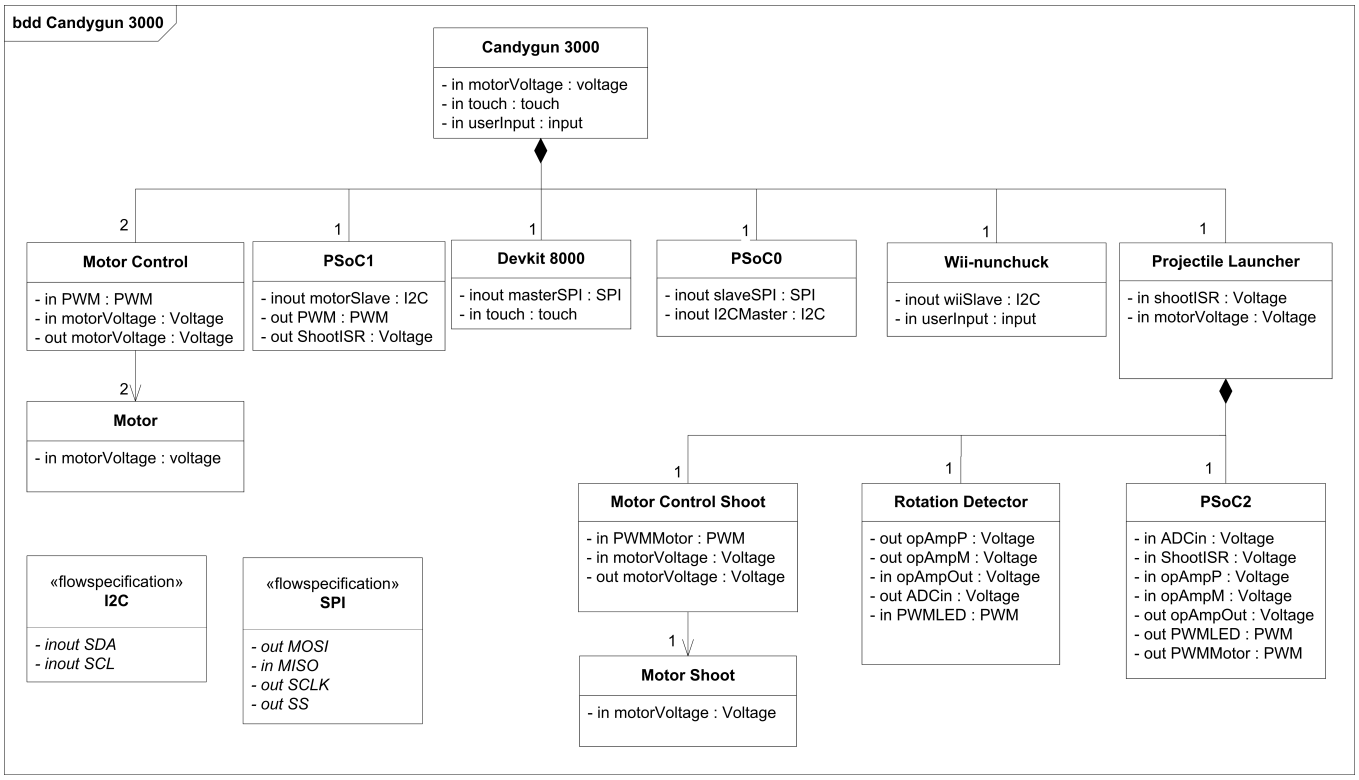
Her repræsenteres hardware som blokke forbundet med associationer. Associationerne viser grænsefladerne mellem de forbundne hardware komponenter samt retningen af kommunikationen. Af modellen fremstår konceptuelle kommandoer for grænsefladerne, som beskriver deres nødvendige attributter.

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, præsenteres i de følgende arkitekturafsnit. Den fulde beskrivelse af domænemodellen ses i dokumentationen afsnit 3.1-*Domænemodel* side #ref

9.2 Hardware

9.2.1 BDD

På figur 4 ses BDD'et for systemet.



Figur 4: BDD for systemets hardware

På figur 4 vises alle hardwareblokke fra domænemodellen (figur 3) med nødvendige indgange og udgange for de fysiske signaler. Yderligere ses det, at flowspecifikationer er defineret for de ikkeatomare forbindelser *I2C*[**?**] [**?**] og *SPI* [**?**], da disse er busser bestående af flere forbindelser.

9.2.2 Blokbeskrivelse

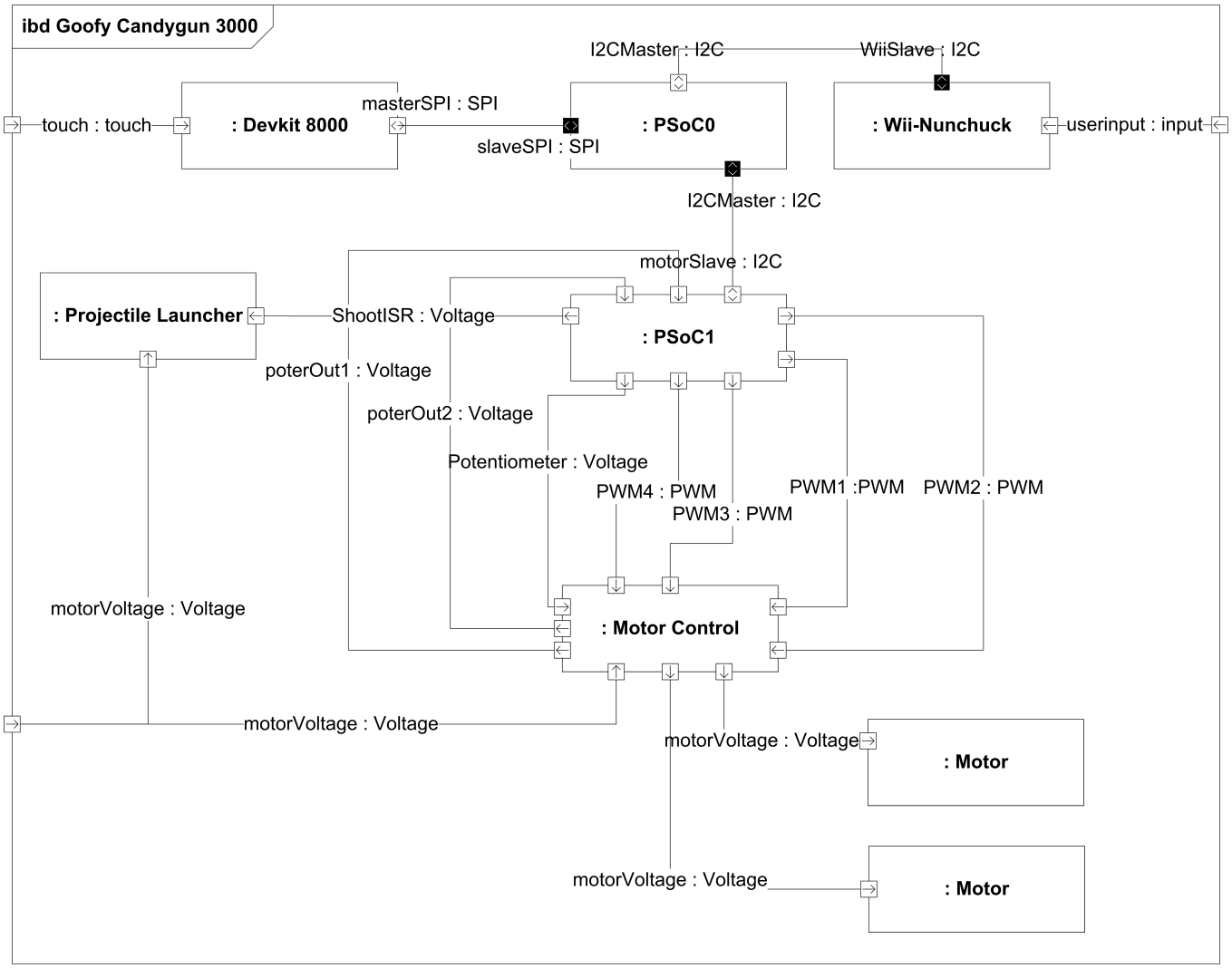
Følgende afsnit indeholder en blokbeskrivelse samt en flowspecifikation for I2C og SPI. I flowspecifikationen beskrives signalerne for I2C og SPI busserne. I blokbeskrivelsen beskrives hver enkelt blok, så der fås en ide om, hvad hver blok består af, hvis de består af flere ting. Derudover skal det give et overblik over, hvad hver blok bruges til i systemet.

Bloknavn	**Beskrivelse**
Devkit 8000	DevKit 8000 er en embedded linuxplatform med touchskærm, 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 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.
Motor Control	Motor Control blokken er Candy Gun 3000's motorer, der anvendes til at bevæge kanonen. Denne blok består af H-bro-blokken og rotationsbegrænsningsblokken.
Motor	Motor er motorene, der bruges til at bevæge platform og kanon.
H-bro	H-bro bruges til at styre motorens rotationsretning.
Rotationsbegrænsning	Rotationsbegrænsning er til at begrænse platformens rotation så den ikke kan dreje 360 grader. Den blok består af et potentiometer og en ADC, som sidder internt på PSoC0.
PSoC1	PSoC1 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 bussen. En fuld beskrivelse af bussen kan ses i dokumentationen afsnit 3.7.1-*SPI Protokol* side #ref
I2C (FlowSpecification)	I2C (FlowSpecification) beskriver signalerne der indgår i I2C bussen. En fuld beskrivelse af bussen kan ses i dokumentationen afsnit 3.7.2-*I2C Protokol* side #ref
Projectile Launcher	Affyringsmekanismen indeholder PSoC2, Motor Control Shoot, Motor Shoot og Rotation Detector og sørger dermed for affyring af kanonen.
PSoC2	PSoC2 indeholder en operationsforstærker til rotations-detektorkredsløbet og en ADC som aflæser rotationsdetektoren. Derudover står den for at sende PWM-signaler til LED'en i rotationsdetektoren og til motorstyringsblokken.
Motor Control Shoot	Denne blok står for at styre affyringsmekanismens motor.
Motor Shoot	Motor Shoot er motoren, der sidder i affyringsmekanismen.
Rotation Detector	Rotationsdetekoren detekterer, at der er blevet affyret et stykke slik og sender et signal til PSoC2 om dette.

Tabel 4: Blokbeskrivelse for BDD'et

9.2.3 IBD

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



Figur 5: IBD for systemets hardware

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

I tabel 5 ses beskrivelser af alle signaler i systemet. Et mere detaljeret IBD for hardwareenhederne *Projectile Launcher* og *Motor Control* kan ses i dokumentationen afsnit 3.3 - *IBD for Candy Gun 3000* side #ref.

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: 100kbps Beskrivelse: I2C bussen hvor der modtages data fra Nunchuck.
		motorMaster	Type: I2C Spændingsniveau: 0-5V Hastighed: 100kbps 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: 100kbps Beskrivelse: Indeholder formatteret Wii-Nunchuck data, som omsættes til PWM-signal.
		ShootISR	Type: voltage Spændingsniveau: 0-5V Beskrivelse: Giver 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: 0V-5V Beskrivelse: Den spænding, der viser, hvor motoren står henne
		PotenOut2	Type: Voltage Spændingsniveau: 0V-5V Beskrivelse: Den spænding, der 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 Rotationsbegrænsing 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 Rotationsbegrænsing 5V.
		PotenOut1	Type: Voltage Spændingsniveau: 0V-5V Beskrivelse: Den spænding, der viser, hvor motoren står henne.

		PotenOut2	Type: Voltage Spændingsniveau: 0V-5V Beskrivelse: Den spænding, der viser, hvor motoren står henne.
Motor	Denne blok beskriver, hvad motoren får.	motorVoltage	Type: Voltage Spændingsniveau: 0-5V Beskrivelse: Giver spænding til motoren. Gælder for begge.
Wii-nunchuck	Den fysiske controller, som brugeren styrer kanonen med.	wiiSlave	Type: I2C Spændingsniveau: 0-5V Hastighed: 100kbps Beskrivelse: Kommunikationslinje mellem PSoC1 og Wii-Nunchuck.
		userInput	Type: input Beskrivelse: Brugerinput fra Wii-Nunchuck.
SPI	Denne blok beskriver den ikkeatomiske 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: clocksignalet 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 ikkeatomiske I2C-forbindelse.	SDA	Type: CMOS Spændingsniveau: 0-5V Beskrivelse: Databussen mellem I2C master 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.
Projectile Launcher	Denne blok består blokkene PSoC2, Detector og Motor Control Shoot	shootISR	Type: Voltage Spændingsniveau: 0-5V Beskrivelse: Giver et højt signal, når kanonen skal skyde.
		motorVoltage	Type: Voltage Spændingsniveau: 9V Beskrivelse: Strømforsyning til motor.
PSoC2	Aflæser signaler fra rotationsdetektor og udsender PWM-signal til LED og Motor Shoot.	ADCin	Type: Voltage Spændingsniveau: 0,5-5V Beskrivelse: Rotationsdetektors output, som aflæses af ADC på PSoC2.

		shootISR	Type: Voltage Spændingsniveau: 0-5V Beskrivelse: Giver et højt signal, når motoren skal skyde.
		opAmpP	Type: Voltage Spændingsniveau: 0,5V Beskrivelse: Referencespænding til operationsforstærkerens positive indgang.
		opAmpM	Type: Voltage Spændingsniveau: 0,5V Beskrivelse: Virtuelt nul. Den ligger altid på 0,5V pga. negativ feedback på operationsforstærkeren.
		opAmpOut	Type: Voltage Spændingsniveau: 0,5-5V Beskrivelse: Udgang på operationsforstærker i rotationsdetektorkredsløbet.
		PWMLED	Type: PWM Spændingsniveau: 0-5V Frekvens: 10kHz PWM%: 0-100% Beskrivelse: Udsender PWM-signal til LED'en i rotationsdetektorkredsløbet.
		PWMMotor	Type: PWM Spændingsniveau: 0-5V Frekvens: 33,33kHz PWM%: 0-100% Beskrivelse: Udsender PWM-signal til LED'en i rotationsdetektorkredsløbet.

Detector	Rotationsdetektoren detekterer om der er skudt.	opAmpP	Type: Voltage Spændingsniveau: 0,5V Beskrivelse: Referencespænding til operationsforstærkerens positive indgang.
		opAmpM	Type: Voltage Spændingsniveau: 0,5V Beskrivelse: Virtuelt nul. Den ligger altid på 0,5V pga. negativ feedback på operationsforstærkeren.
		opAmpOut	Type: Voltage Spændingsniveau: 0,5-5V Beskrivelse: Udgang på operationsforstærker i rotationsdetektorkredsløbet.
		ADCin	Type: Voltage Spændingsniveau: 0,5-5V Beskrivelse: Rotationsdektors output, som aflæses af ADC på PSoC2.
		PWMLED	Type: PWM Spændingsniveau: 0-5V Frekvens: 10kHz PWM%: 0-100% Beskrivelse: Udsender PWM-signal til LED'en i rotationsdetektorkredsløbet.
Motor Control Shoot	Denne blok styrer Motor Shoot til affyringsmekanismen.	PWMMotor	Type: PWM Spændingsniveau: 0-5V Frekvens: 33,33kHz PWM%: 0-100% Beskrivelse: Udsender PWM-signal til LED'en i rotationsdetektorkredsløbet.

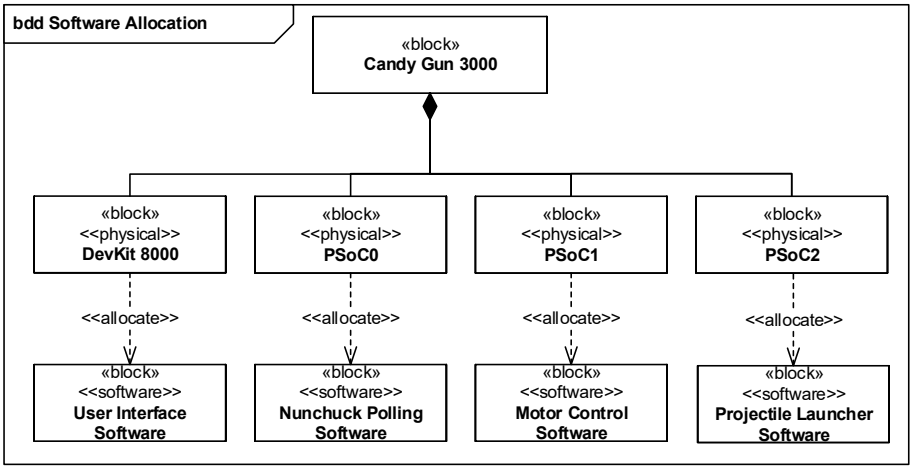
		motorVoltage	Type: Voltage Spændingsniveau: 9V Beskrivelse: Strømforsyning til motor.
Motor Shoot	Denne blok er affyringsmekanismens motor.	motorVoltage	Type: Voltage Spændingsniveau: 9V Beskrivelse: Strømforsyning til motor.

Tabel 5: Signalbeskrivelse

9.3 Software

9.3.1 Softwareallokering

Domænemodellen i figur 3, side 14, præsenterer systemets hardwareblokke. På figur 6 ses et software allokeringsdiagram, som viser på hvilke hardwareblokke de forskellige softwaredele er allokeret.



Figur 6: Systemets softwareallokeringer

På figur 6 ses, at systemet består af fire primære softwaredele: *User Interface Software*, *Nunchuck Polling Software*, *Motor Control Software* og *Projectile Launcher Software*. Disse er fordelt over de fire viste CPU'er.

I tabel 6 er hver allokeret softwarekomponent 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.

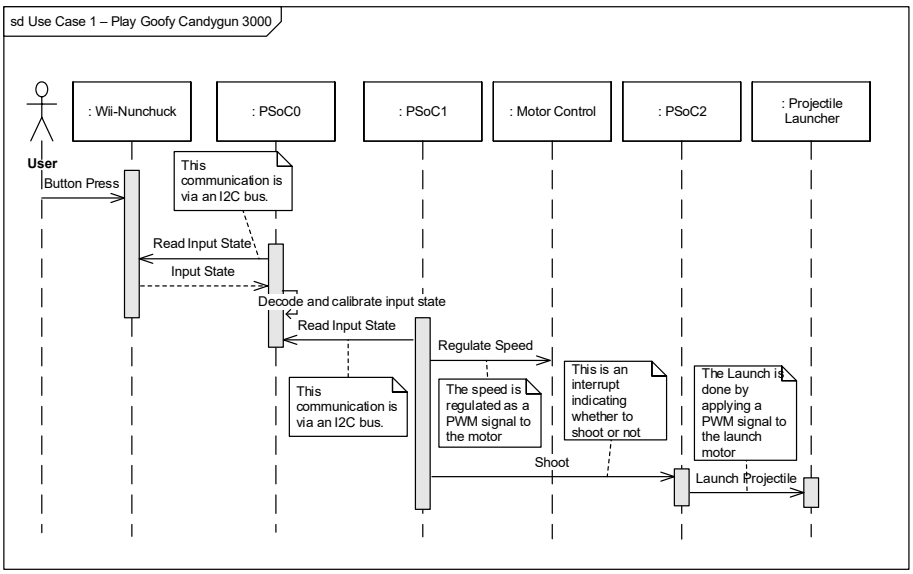
Tabel 6: Beskrivelse af den allokerede software

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 afsnit 10.2.4, i Design og Implementering.

Wii-Nunchuck Information Flow

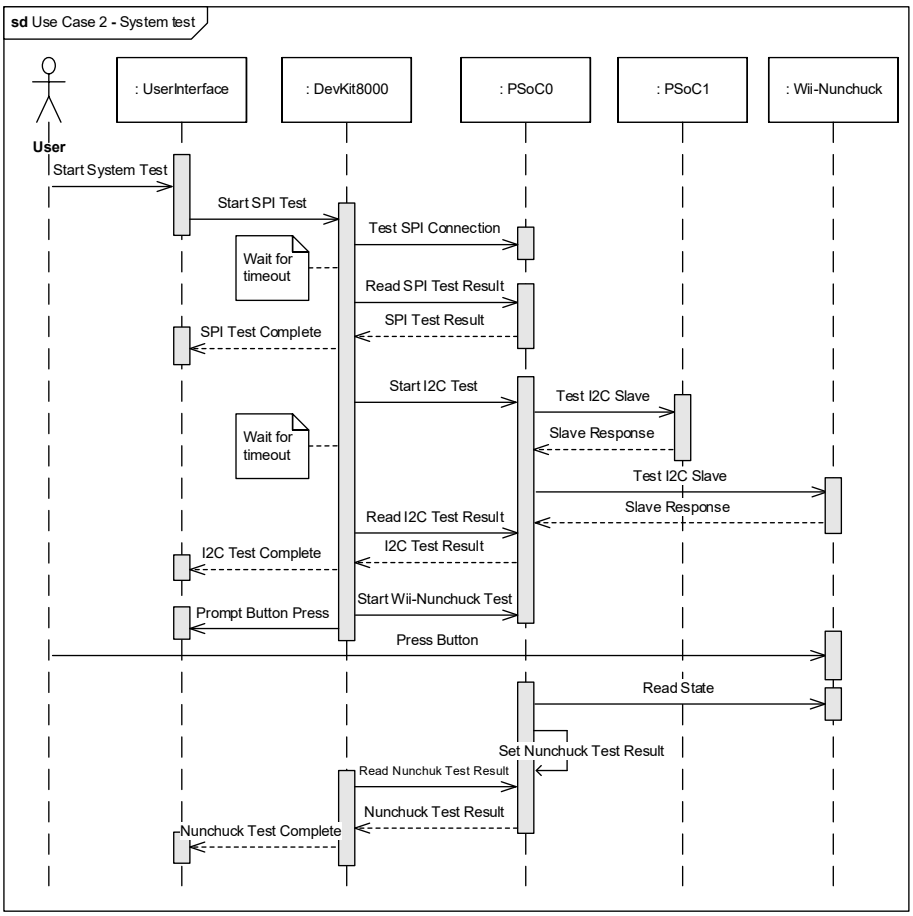
En essentiel del af use case 1 er at kunne styre motoren ved brug af Wii-Nunchuck controlleren. På figur 7 vises gennemløbet af Wii-Nunchucks inputdata fra Wii-Nunchucken til motoren med de relevante CPU'er angivet. Her ses det, at inputdata fra Wii-Nunchuck kontinuert bliver aflæst af PSoC0. Det bemærkes her, at grænsefladen mellem PSoC0 og Wii-Nunchuck er en I2C bus. Efter at PSoC0 har aflæst inputdata, overføres det til PSoC1. Grænsefladen mellem disse to PSoC's er også en I2C bus. PSoC1 kan til slut oversætte modtaget inputdata til PWM-signaler til motorstyring samt affyring.



Figur 7: Wii-Nunchuck Input Data Forløb

Systemtest

Use case 2 beskriver test af systemet før spillet startes. På figur 8 vises testsekvensen for en vellykket test.



Figur 8: System Test Forløb

Kommunikationen mellem Devkit 8000 og PSoC0 er initieret og styret af Devkit 8000. Da PSoC0 i dette design ikke har mulighed for at indikere, hvornår data er klar til aflæsning, skal Devkit 8000 vente på en *timeout*, før den aflæser data på PSoC0. Dette er muligt, idet systemtesten bliver afviklet sekventielt.

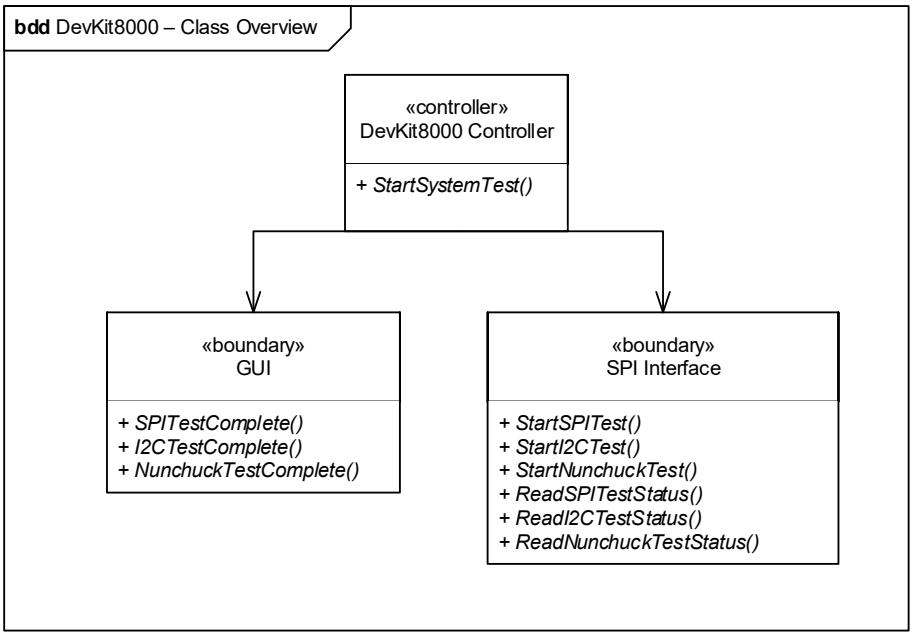
9.3.3 Samlede Klassediagrammer

På baggrund af sekvensdiagrammerne i afsnit 9.3.2 samt de detaljerede applikationsmodeller beksrevet i dokumentationen, afsnit 3.5-*Applikationsmodeller* side #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 for resten af udviklingsprocessen.

Figur 9, 10, 11, og 12 viser de samlede klassediagrammer for hver CPU.

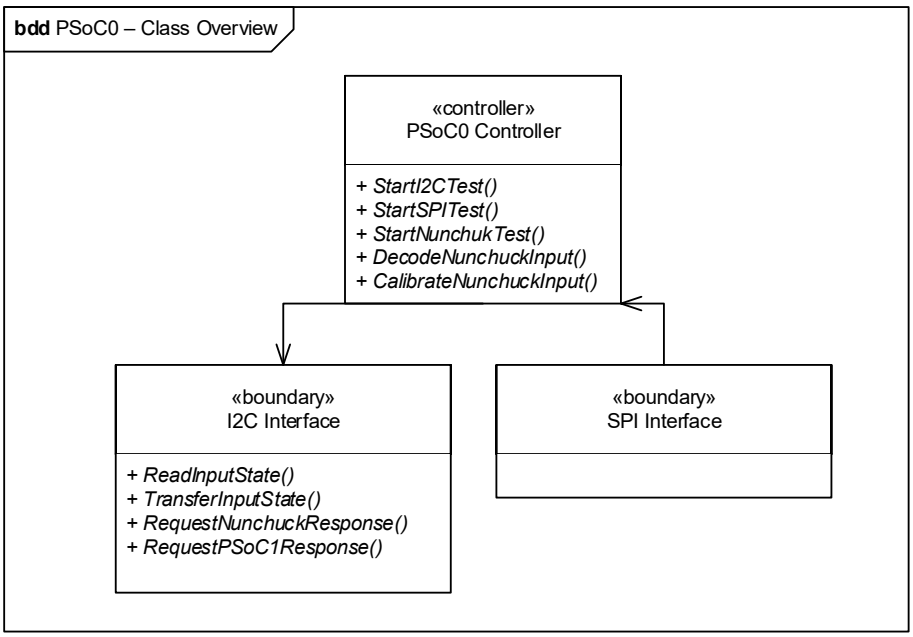
For disse klassediagrammer er der konceptuelle klasser, som går igen på flere diagrammer. Alle klassediagrammer har én klasse af typen *controller*. Disse klasser indeholder al funktionalitet, der er nødvendig for at kunne implementere systemets use cases. Derudover går klasserne *I2C Interface* og *SPI Interface* også igen på flere diagrammer. 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 Devkit 8000 controlleren skal have funktionalitet til start af systemtest, 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 resten af systemet via SPI bussen.



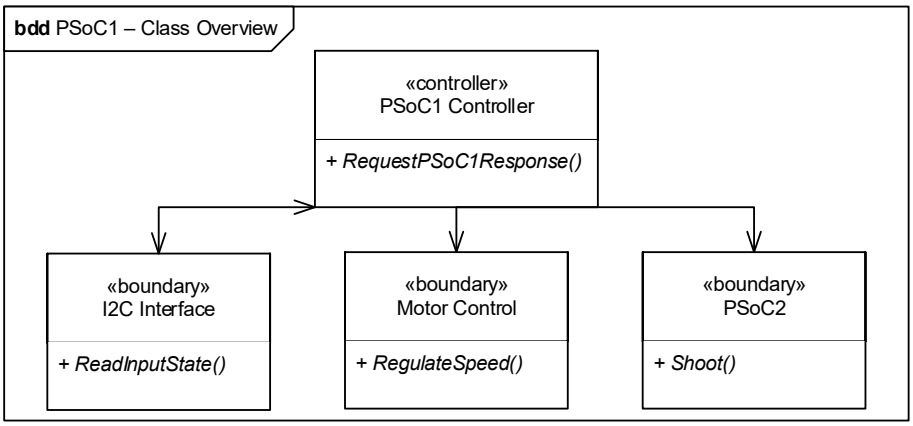
Figur 9: Samlet Klassediagram for Devkit 8000

På figur 10 ses det at PSoC0 controlleren skal have funktionalitet til at starte tests af systemets busser. Yderligere skal den også kunne aflæse, 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.



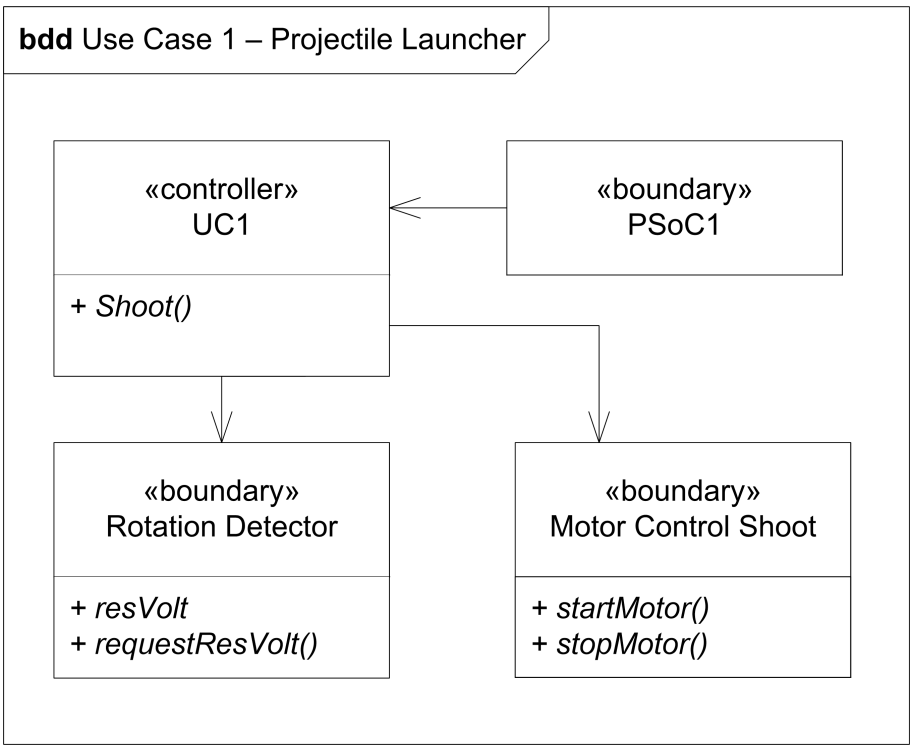
Figur 10: Samlet Klassediagram for PSoC0

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 Devkit 8000 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 7 ses en de anvendte kommandotyper samt en kort beskrivelse for hver af disse. En fuld beskrivelse af SPI-kommunikations protokollen kan ses i dokumentationen afsnit 3.7.1-*SPI Protokol* side #ref.

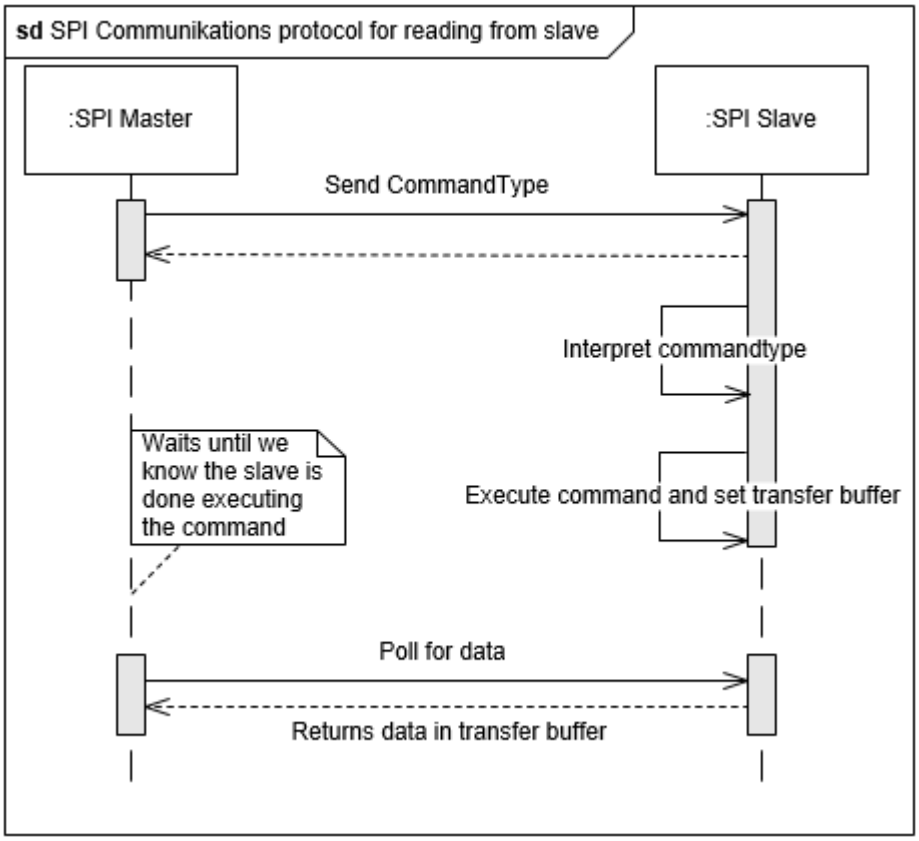
Der er til systemet valgt en SPI bus mellem Devkit 8000 og PSoC0, da der i dette tilfælde kun er brug for en bus med god support for én master og én slave. En SPI Bus skalerer ikke godt med multiple slaver i forhold til SPI, da en SPI bus skal have en fysisk forbindelse for hver enkelt slave der tilkobles. 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_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 7: SPI kommunikation kommandotyper

Kommandotyperne er udledt fra det samlede klassediagram for Devkit 8000, figur 9. På figuren kan det ses at *Devkit 8000 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-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 13.



Figur 13: Sekvensdiagram for aflæsning data fra en SPI-slave

Designvalg

SPI kommunikations protokollen er designet ud fra det grundlag at Devkit 8000, som SPI master, skal læse tilstande fra SPI slaven ved brug af en timeout model. Ved timeout model menes der at Devkit 8000 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 det 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 vil blive præsenteret her, men en fuld gennemgang kan findes i dokumentationen afsnit

3.7.2-*I2C Protokol* side #ref.

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, for at kommunikere med Nunchucken. Yderligere skulle Nunchuken's 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 predefineret protokol, der anvender adressering af hardwareenheder til identificering af, hvilken enhed der kommunikeres med. Derfor har hardwareblokkene som indgår i I2C kommunikationen fået tildelt adresser. På tabel 8 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 8: 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 9 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

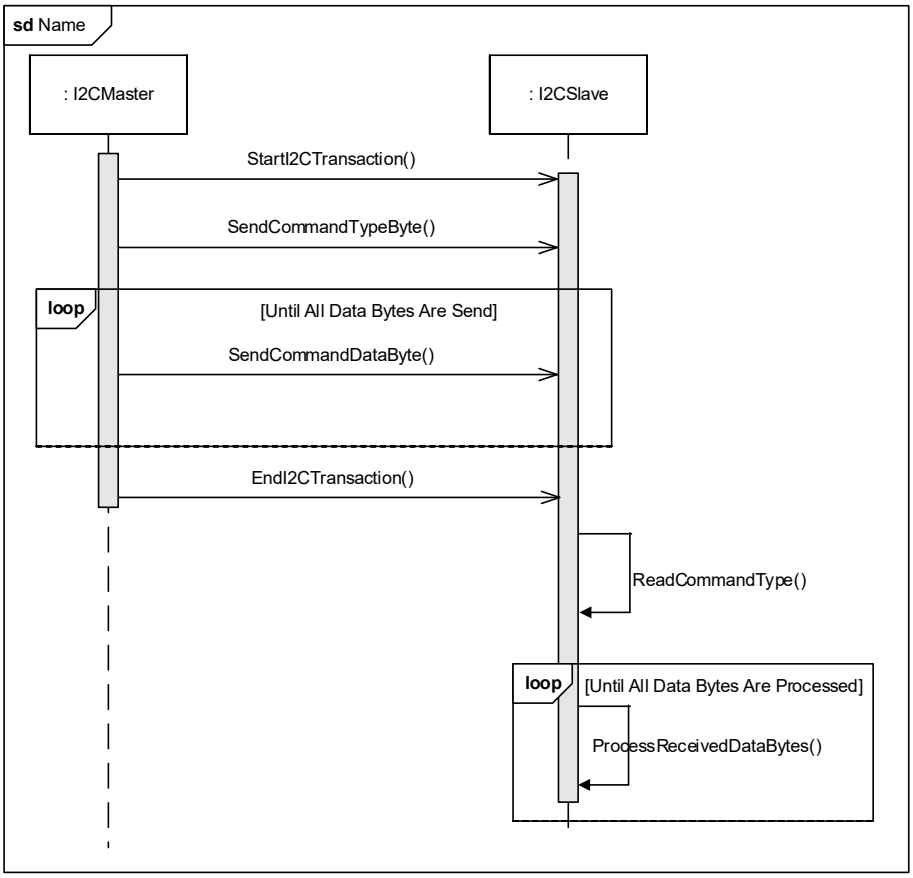
Tabel 9: I2C kommunikation kommandotyper

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 9 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.

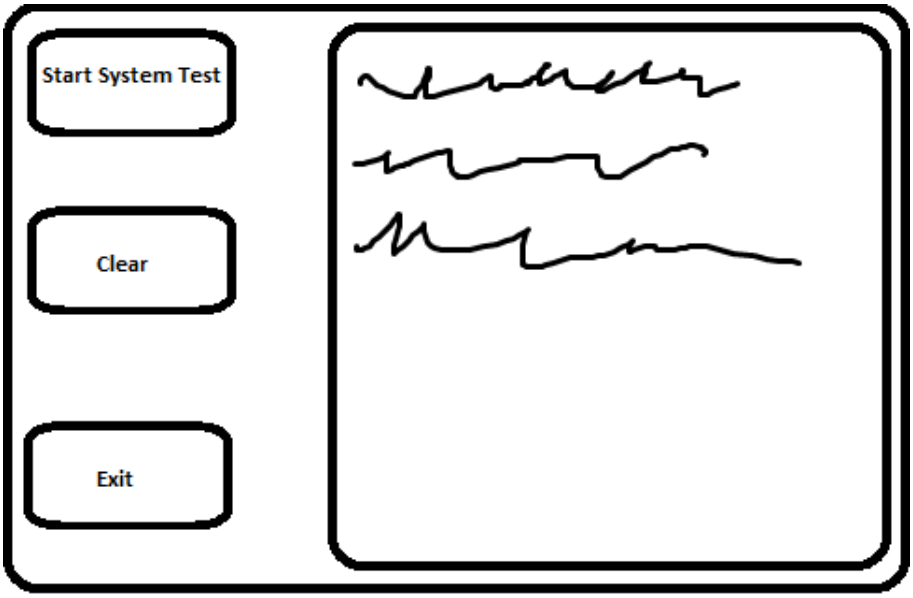
Designvalg

Den primære idé bag valget 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 14.

9.3.6 Brugergrænseflade

Ved hjælp af en brugergrænseflade, kan systemtesten styres fra Devkit 8000. Brugergrænsefladen er opbygget ud fra sekvensdiagrammet 8, ud fra dette kunne brugergrænsefladen skitseres som figur 15. Grænsefladen mellem Devkit 8000 og PSoC0 er en SPI-bus som ses på figur 9.2.3, og brugergrænsefladen er koblet til Devkit 8000 ved hjælp af interfacedriveren. Brugergrænsefladen skal sende startsekvensen til interfacedriveren, hvorefter dette sendes til PSoC0 og videre ud i systemet, som beskrevet i sekvensdiagrammet figur 8. 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 use case 2, hvilken er beskrevet i afsnit 5.2.2 og i dokumentationen afsnit 3.5.2-*Use case 2 - Test kommunikationsprotokoller* side #ref. 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 funktionaliteter, der faciliterer systemtesten. Et alternativ kunne være trådbaseret design. Kompleksiteten i dette design ville være overvældende i forhold til den ønskede funktionalitet og blev derfor fravalgt.

10 Design og Implementering

Følgende afsnit beskriver de løsninger der er implementeret for hardware og software komponenter for prototypen.

10.1 Hardware

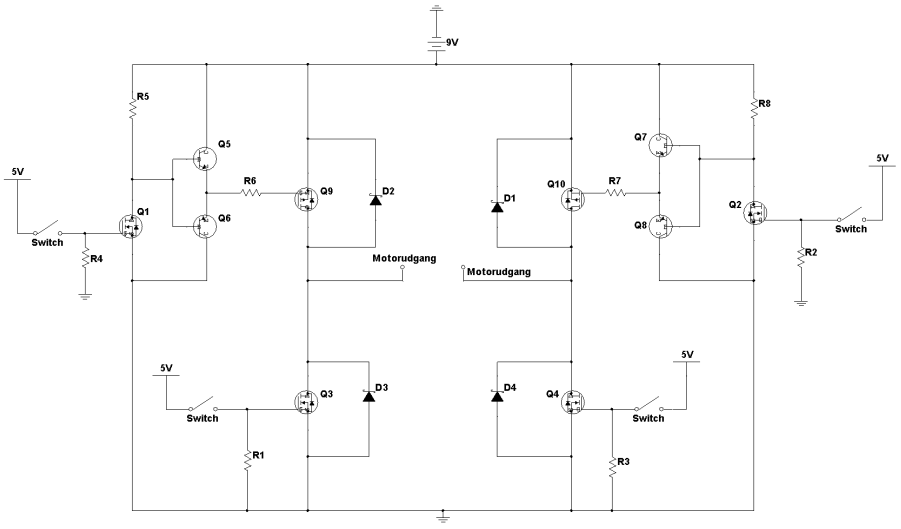
I Goofy Candygun 3000 indgår et antal hardwaredele, som skal udgøre det færdige system. Ud fra BDD'et ses det, at de to overordnede hardwaredele, der skal udvikles, er motorstyring til den vertikale og horisontale drejeretning, og en affyringsmekanisme.

10.1.1 Motorstyring

Motorstyringen skal sørge for at kanonen kan styres i de vertikale og horizontale 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 9.2.1. 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 figur 16 ses det færdige kredsløbsdiagram over H-broen.



Figur 16: Kredsløbsdiagram for H-broen

I tabel 10 ses relevante komponentværdier for H-broen. En fyldestgørende oversigt over disse ses i dokumentationen afsnit 4.5.2-*Motorstyring*, *Tabel 12, side #ref.*

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

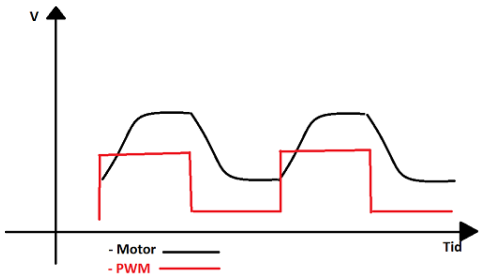
Tabel 10: Komponentbetegnelser på H-bro

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.

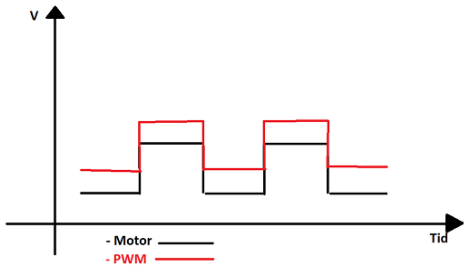
Da motoren gerne skal kunne skifte retning hurtigt og ofte, er det nødvendigt, at de to MOSFET's kan både åbne og lukke hurtigt. På figur 17 ses, hvordan Q9 åbner og lukker langsomt, hvilket skyldes kondensatoreffekten mellem benene på MOSFET'en.

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 afsnit 4.5.2-*Motorstyring side #ref.*

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. For mere information se dokumentationen afsnit 4.5.2-*Motorstyring - Rotationsbegrænsning #ref*

Figur 19: Opstilling for rotationsbegrænsning

Potentiometer
Det anvendte potentiometer har en størrelse på 47 $K\Omega$. 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 potentiometret roteres ændres modstanden i de to variable modstande og dermed sker der en ændring i outputspændingen.

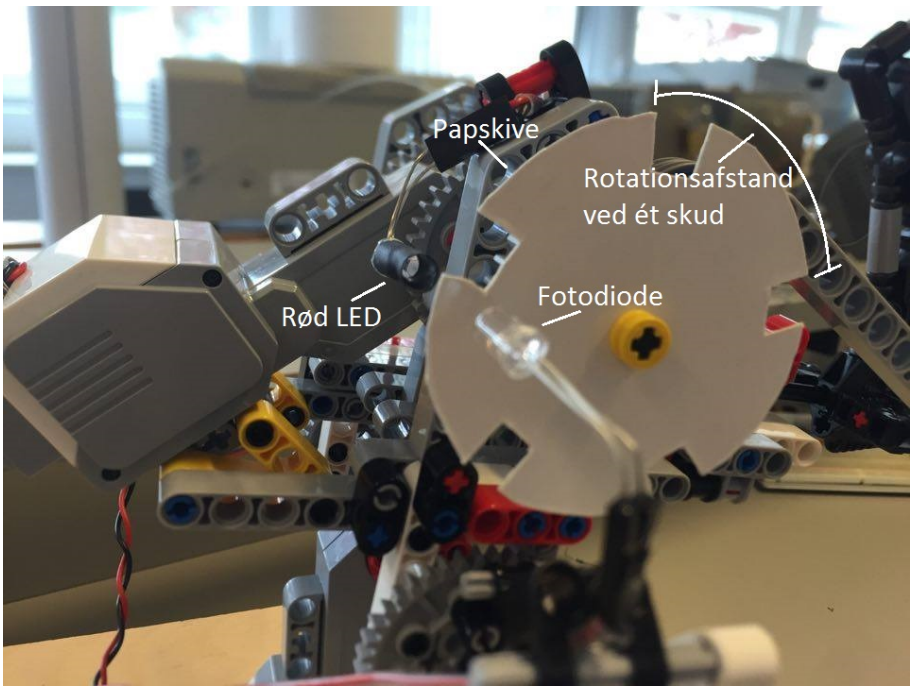
ADC
For at kunne aflæse spændingen på potentiometeret, anvendes en 12-bit AD converter af typen Sequencing Successive Approximation ADC. En sequencing SAR ADC indeholder et sample-hold kredsløb. Kredsløbet holder på et indgangssignal indtil det næste signal registres på kredsløbets indgang. Dermed har converteren tid til at bestemme outputværdien.

10.1.2 Affyringsmekanisme

Affyringsmekanismen består af en motor; et motorstyringskredsløb; et 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

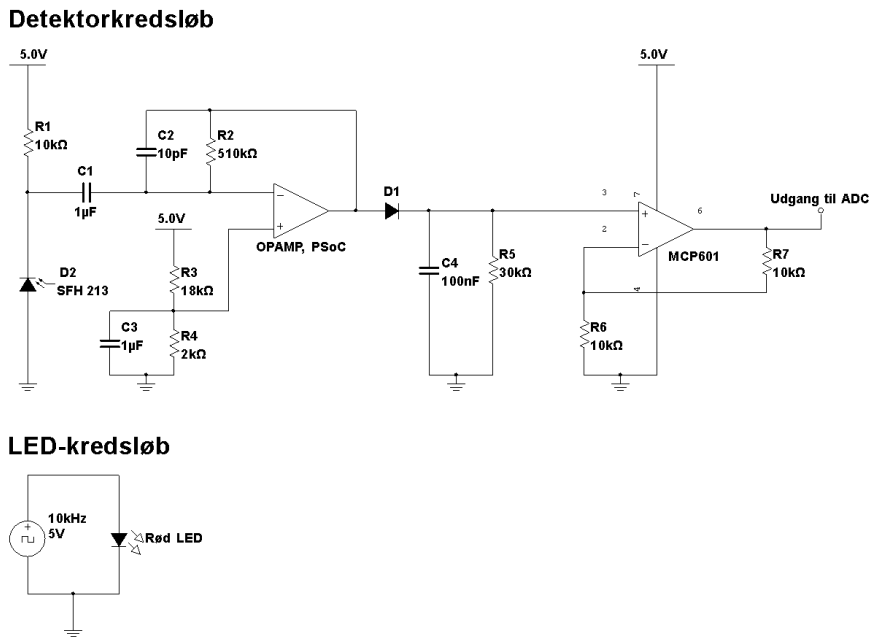
Når kanonen affyres, styres det af motoren, og som mekanikken er opbygget, er der et propertionelt forhold mellem omdrejning på motoren og antal skud, der affyres. Derfor er det væsentligt at vide, hvornår motoren har roteret en runde, så den kan stoppes, inden der igen skydes. Til det formål anvendes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

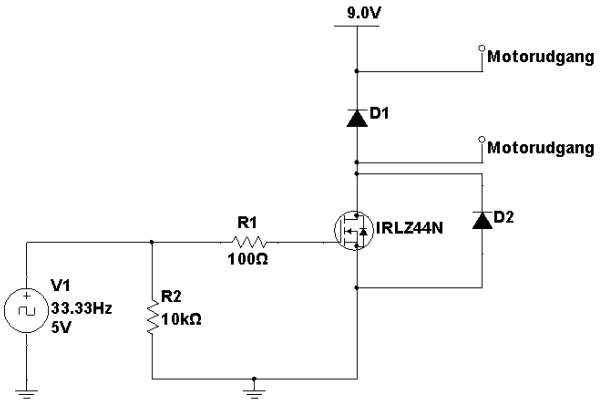
Detektoren skal kun sende et signal, når fotodioden ser lyset fra LED'en. Det er derfor vigtigt, at den ikke bliver forstyrret af dagslys og andre lyskilder. For at sikre dette, styres den røde LED af et PWM-signal, så LED'en blinker med en frekvens på 10 kHz. Detektoren opbygges tilsvarende af et båndpasfilter, 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

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.



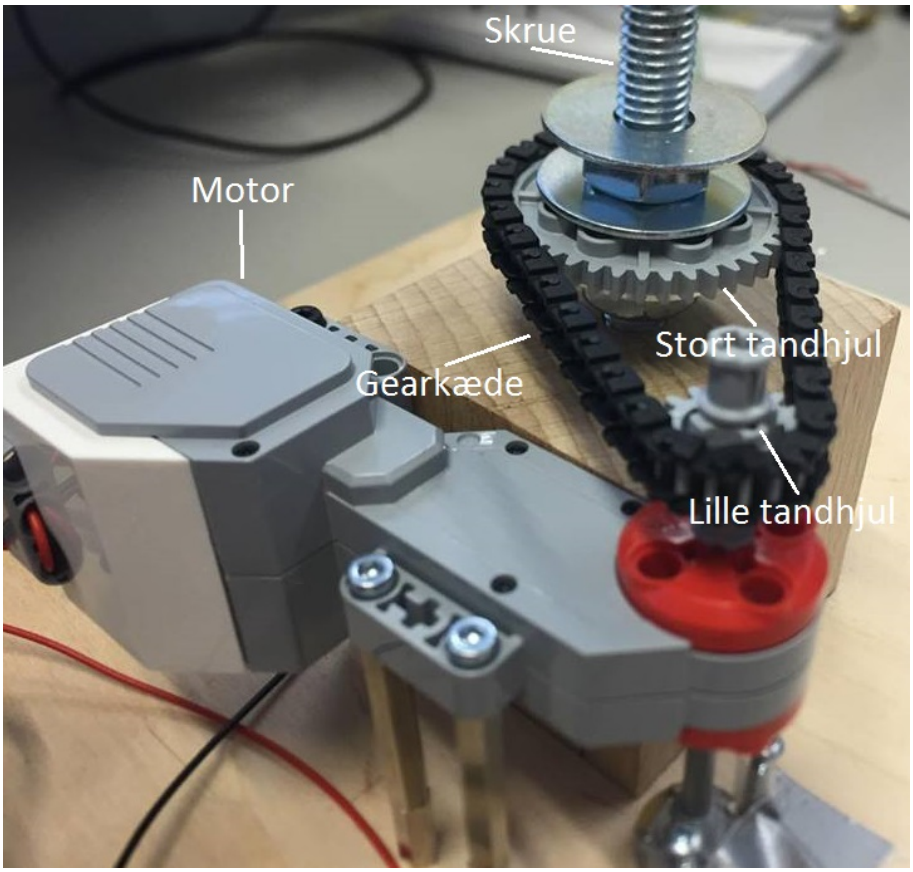
Figur 22: Diagram over motorstyring til motor på affyringsmekanisme

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.

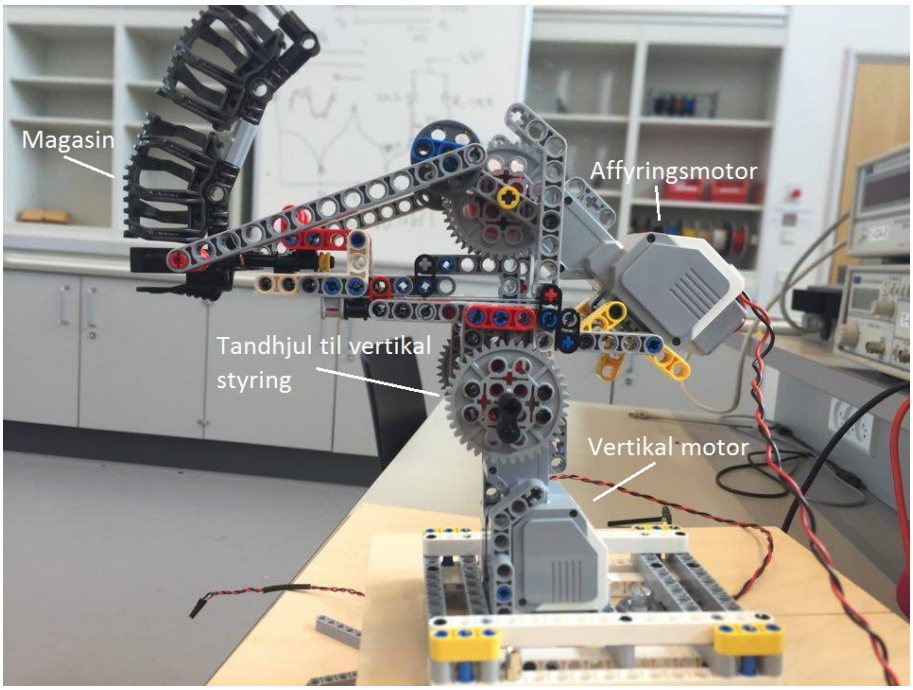


Figur 23: Horisontal mekanik

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

Følgende afsnit beskriver softwareimplementeringer relateret til kommunikation mellem systemets hardwareblokke.

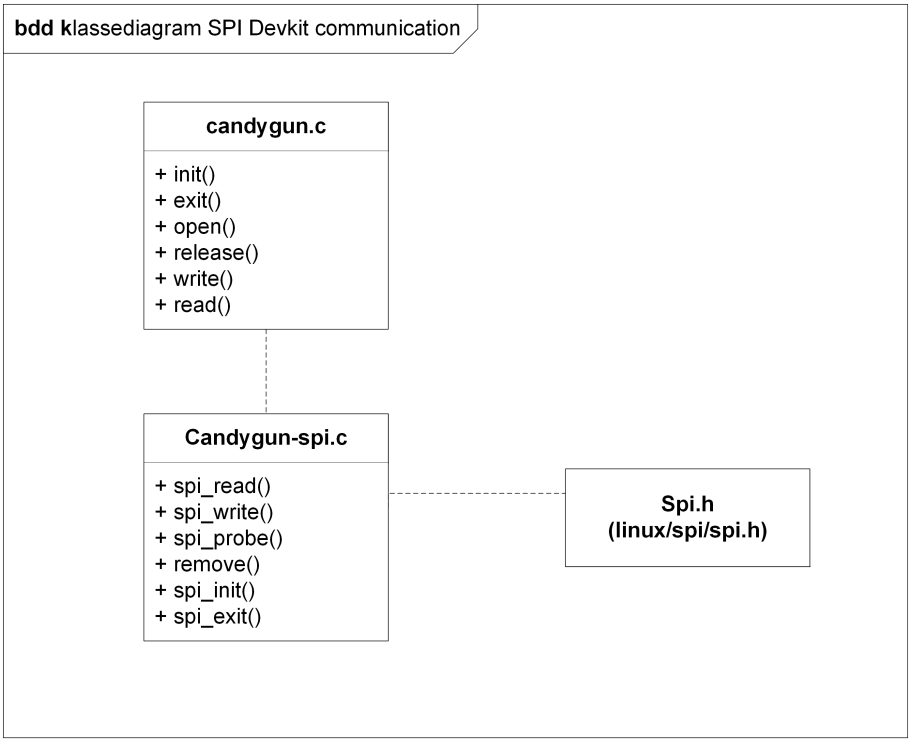
10.2.1 SPI - Devkit8000

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 indstilligner ses på tabel 11. Yderligere uddybning af de forskellige indstillinger kan ses i dokumentationen afsnit 4.1.1-*SPI Devkit 8000 - Candygun Driver #ref*

Tabel 11: 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 overskuelighedensskyld. De stiplede linjer i diagrammet indikerer at den ene klasse anvender den andens motoder, på samme måde som ved et bibliotek. spi.h, som også ses i diagrammet, er en indbygget del Linux, og er derfor ikke yderligere dokumenteret her.



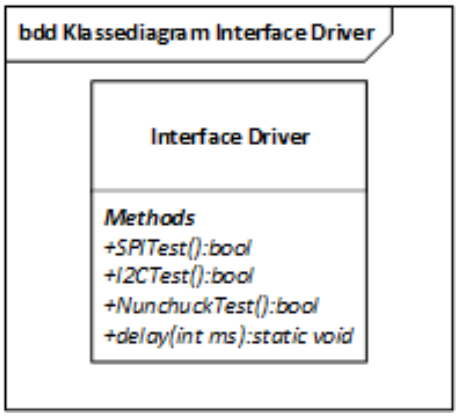
Figur 25: Klassediagram for SPI kommunikation på Devkit 8000

10.2.2 Brugergrænseflade

I det følgende afsnit bliver delene i den overordenede brugergrænseflade beskrevet.

Interface Driver

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

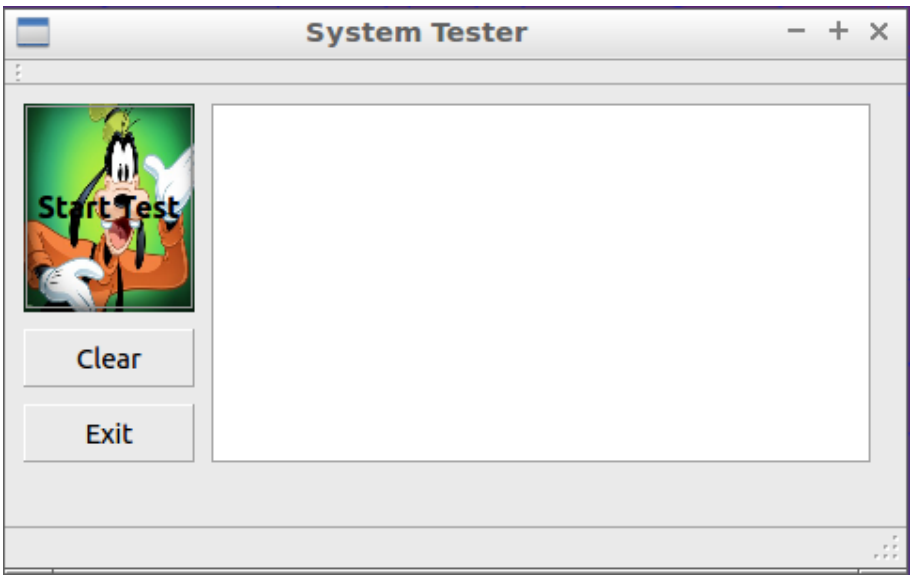


Figur 26: Klassediagram for Interfacedriver

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.

Systemtest GUI

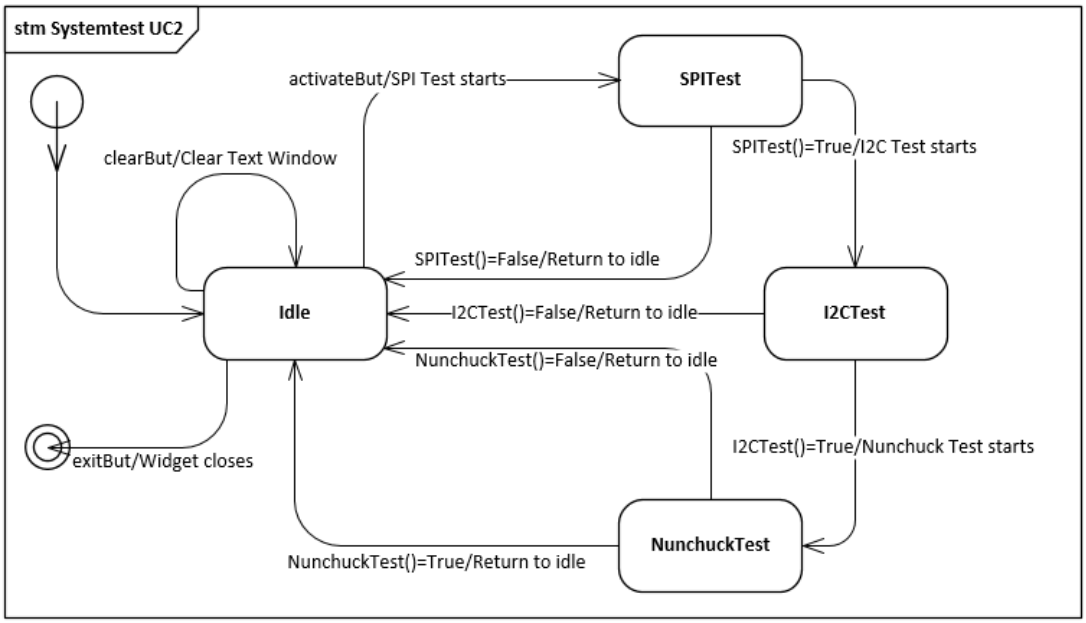
Use case 2 styres via Systemtest GUI'en fra Devkit 8000. Dette afsnit beskriver Systemtest GUI'ens design. Figur 27 viser GUI'en for use case 2.



Figur 27: Brugergrænseflade for usecase 2 - Test kommunikationsprotokoller

Systemtest GUI'en er lavet med det indbyggede design framework i QT Creator 5[**?**], se dokumentationen afsnit 7.1-*QT Creator #ref.* QT frameworket opretter "hovedvinduet"i GUI'en som en klasse. Knapperne tilføjes som private slots i klassen hvilket gør dem i stand til interagere i GUI'en. 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 GUI'en er assignet signal-typen "clicked()".

Her vises en statemachine for Systemtest GUI'ens forløb igennem usecase 2.

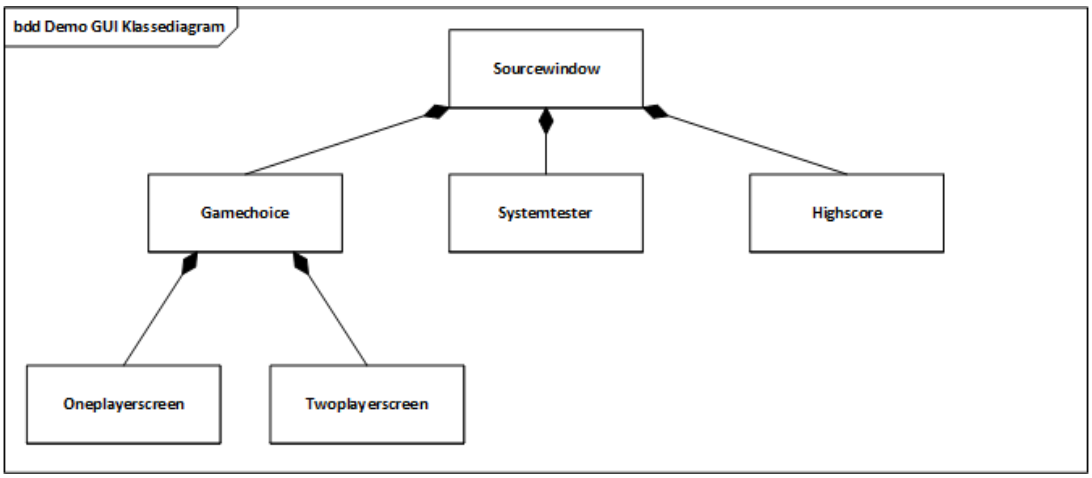


Figur 28: State machine for Systemtest GUI

Statemaskinens, figur 28, forløb beskrives i dokumentationen, afsnit 4.1.2-*Brugergrænseflade, Figur 48 #ref*
Systemtest GUI'en for UC2 er en simpel test-konsol. Den består af 3 knapper og et konsol vindue. GUI'en 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 GUI'en returnerer til idle tilstand. Når alle test er successful, skrives "System test successful, system is ready for use"til konsol vinduet, og GUI'en returnerer til idle tilstand. Den anden knap, Clear, clearer konsollen til blank tilstand. Den tredje knap, Exit, lukker GUI'en.
Koblingen i GUI'en hænger udelukkende sammen med interfacedriveren. Knapperne kalder funktioner fra interfacedriveren, og ved ændring i disse funktioner, ville foresage ændringer i GUI'en.

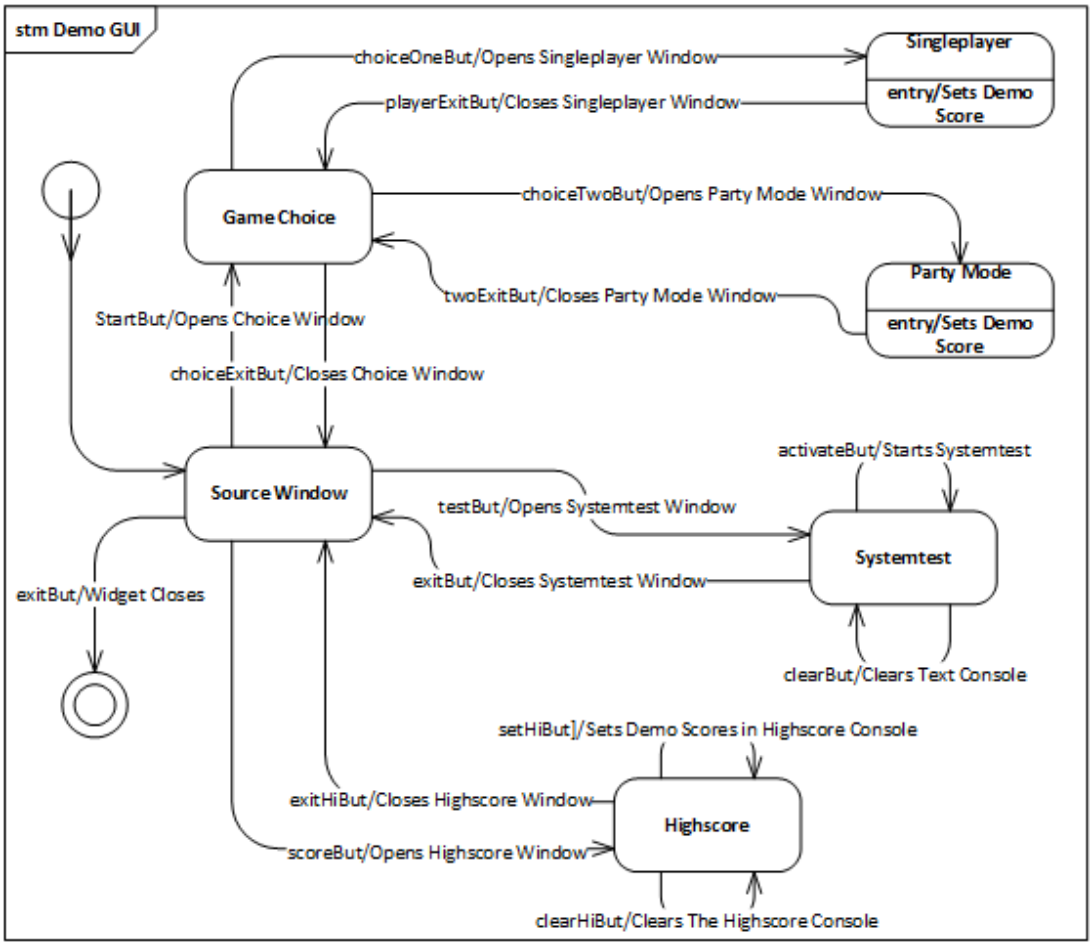
Demo GUI

I dette afsnit beskriver Demo GUI'en for usecase 1. Strukturen for denne GUI er lignende til Systemtest GUI'en. Denne demo er en repræsentation af hvad der kunne være en færdig implementeret GUI til usecase 1.



Figur 29: Klassediagram for Demo GUI

Demo'en består af seks vinduer, hvert vindue har en tilhørende klasse, se figur 29. Hovedvinduet har klassen sourcewindow. Sourcewindow består af fire knapper, hvor tre af dem åbner nye vinduer og den fjerde lukker GUI'en. De tre vinduer der kan åbnes fra sourcewindow er: Gamechoice, Highscore, og System Test. Highscore vinduet består af tre knapper og et konsol vindue. System Test er Systemtest GUI'en implementeret i Demo GUI'en, se afsnit 10.2.2 for detajler. Gamechoice består af tre knapper. To af knapper åbner to nye vinduer og den tredje knap lukker vinduet og returnerer programmet til sourcewindow. De to vinduer åbnes er Single Player og Party Mode. Single player består af et konsol vindue og en exit-knap, Party mode består af to konsol vinduer og en exit-knap.



Figur 30: State machine for Demo GUI

Statemaskinen, figur 30, forklarer, hvordan Demo GUI'ens vinduer skifter tilstand i forhold til event-triggers. Se dokumentationen afsnit 4.1.2-*Brugergrænseflade #ref*

10.2.3 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 (Bilag/Projektrapport/WiiNunchuck.pdf):

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 12

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 12: 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.

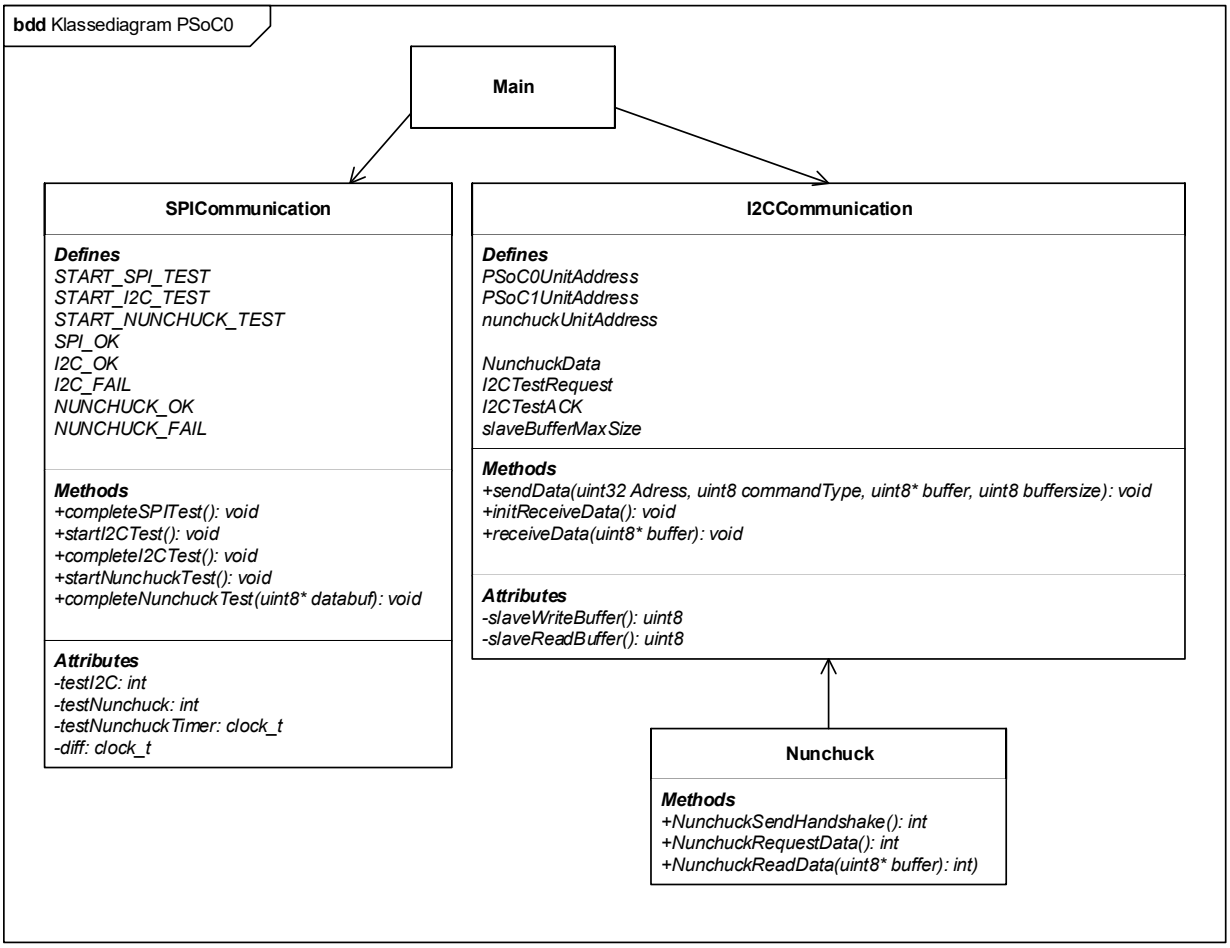
I projektet er de afkodede værdier for det analoge stick kalibreret med værdien -15 (0x0F i hexadecimal), altså ser den endelige formel for afkodning samt kalibrering således ud:

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

10.2.4 PSoC Software

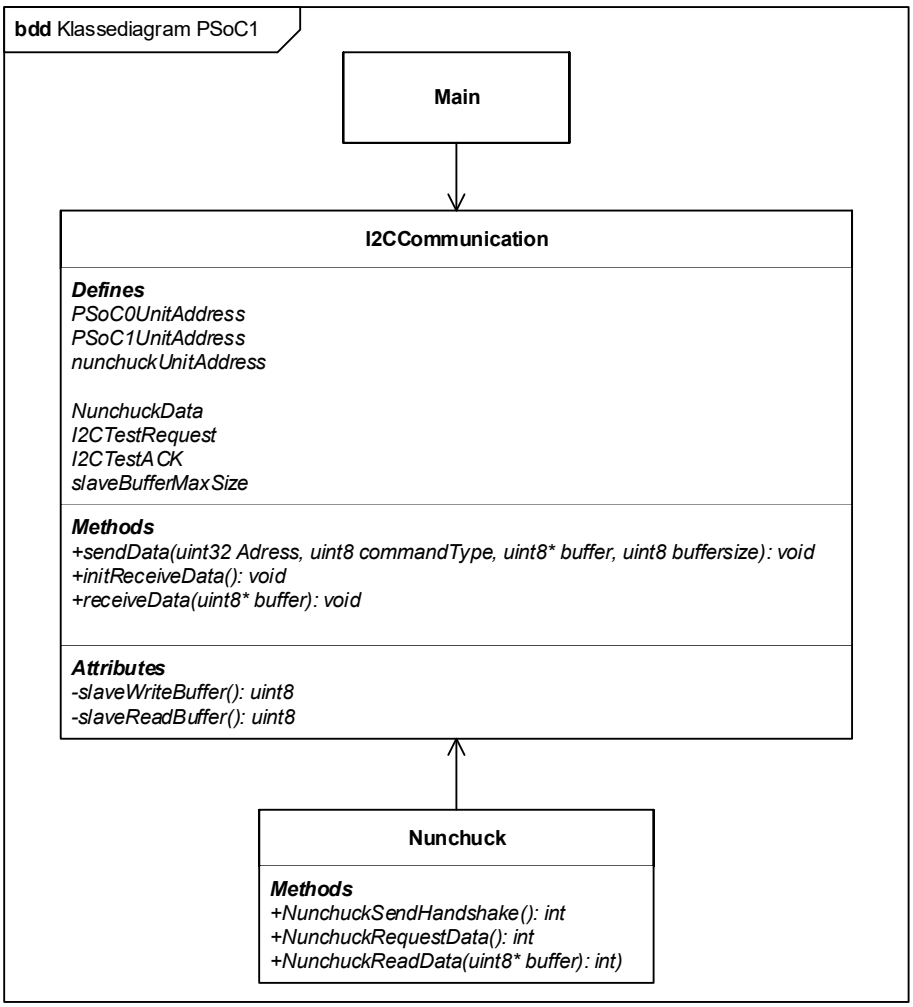
På figur 31 og 32 ses de endelige klassediagrammer for PSoC0 og PSoC1. Disse klassediagrammer er designet ud fra applikationsmodellerne i dokumentationen afsnit 3.5-*Applikationsmodeller #ref* og de samlede klassediagrammer for UC1- og 2 i systemarkitekturen figur 10 og 11. For at opretholde høj samhørighed, 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 31: Klassediagram for PSoC0

På figur 31 ses det samlede klassediagram for klasserne som bruges til PSoC0 softwaren. Her kan det ses at der er en *main* klasse. Denne indeholder programmets primære loop, som gentages indtil PSoC'en slukkes. I dette loop bliver der gjort brug af funktionaliteten fra klasserne *SPICommunication*, *I2CCommunication*, samt *Nunchuck*. Disse klasser beskrives i afsnit *I2CCommunication*, *SPICommunication* og *Nunchuck*. For mere detaljerede klasse- samt algoritmebeskrivelser ses dokumentation afsnit 4.2-*PSoC Software #ref*



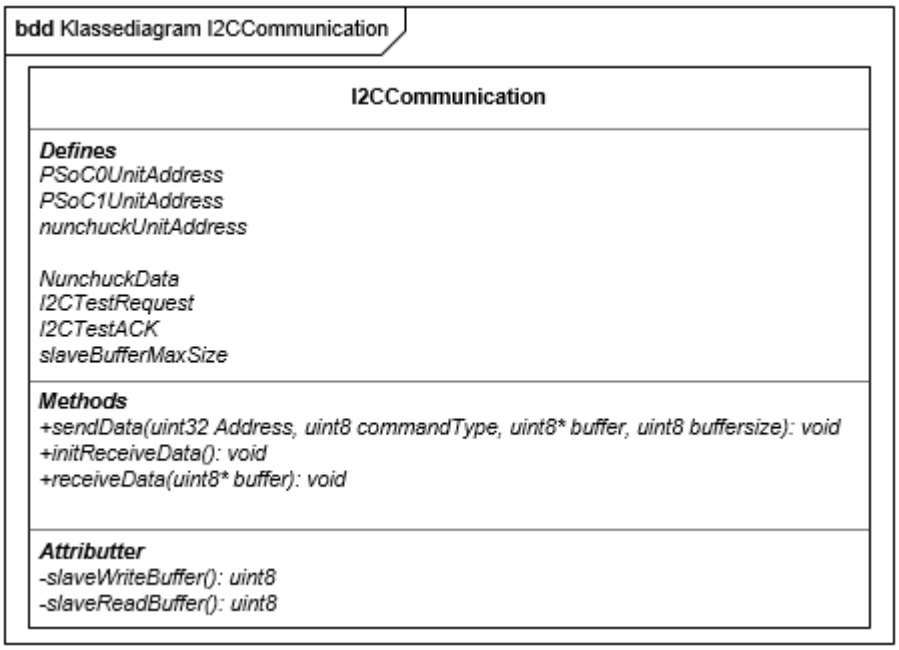
Figur 32: Klassediagram for PSoC1

På figur 32 ses det samlede klassediagram for klasserne som eksisterer på PSoC1 softwaren. Her kan det ses at der er en *main* klasse. Denne indeholder programmets primære loop, som gentages indtil PSoC'en slukkes. I dette loop bliver der gjort brug af funktionaliteten fra klasserne *I2CCommunication* og *Nunchuck*. Disse klasser beskrives ligeledes i afsnit *SPIcommunication* og *I2Ccommunication*. For mere detaljerede klasse- samt algoritmebeskrivelser ses dokumentation afsnit 4.2-*PSoC Software #ref*.

I2CCommunication

I systemarkitekturen, afsnit 9.3.3, på figur 10 og 11, er der udarbejdet klassediagrammer der beskriver funktionaliteterne for PSoC0- og 1 softwaren. Med hensyn til I2C-kommunikation klassen, er den overordnede funktionalitet, at kunne sende og modtage data via I2C-nettet. På figur 33 ses klassediagrammet

for I2CCommunication klassen, der netop har metoder og attributter til at dække disse behov.

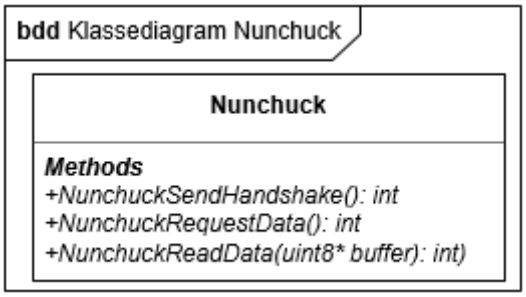


Figur 33: Klassediagram for I2CCommunication klassen

Den fulde klassebeskrivelse for I2CCommunication klassen kan findes i dokumentationen, afsnit 4.2.2-*I2CCommunication #ref.*

Nunchuck

Nunchuck klassen er seperat fra I2CCommunication klassen, på trods af at kommunikationen foregår over I2C-bussen, idét at Nunchucken bruger sin egen kommunikationsprotokol, som defineret i (Bilag/Projektrapport/WiiNunchuck.pdf). I systemarkitekturen, afsnit 9.3.3, på figur 10 og 11, er det udledt at softwaren skal have metoder til at skrive og læse fra nunchucken. På figur 34 ses klassediagrammet for den implementerede nunchuck klasse, der indeholder metoder der har disse funktionaliteter.

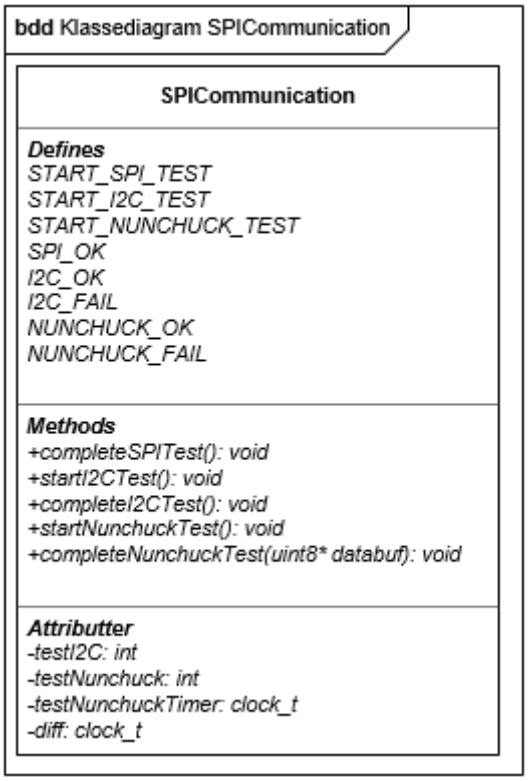


Figur 34: Klassediagram for klassen Nunchuck

Den fulde klassebeskrivelse for Nunchuck klassen og dens metoder kan ses i dokumentationen, afsnit 4.2.3-*I2CCommunication #ref*.

SPICommunication

I dette afsnit vil softwaren der specifikt omhandler SPI-kommunikationen mellem PSoC0 og DevKit8000 blive beskrevet fra PSoC-CPU'ens perspektiv. På figur 35 ses den klasse der omhandler SPI-kommunikationen i systemet. Ud fra klassediagrammet i systemarkitekturen, afsnit 9.3.3, på figur 10, er det blevet udledt at SPI-klassen skal kunne starte og gennemføre forskellige test. Figur 35 viser klassediagrammet for den implementerede klasse, der afspejler netop dette.

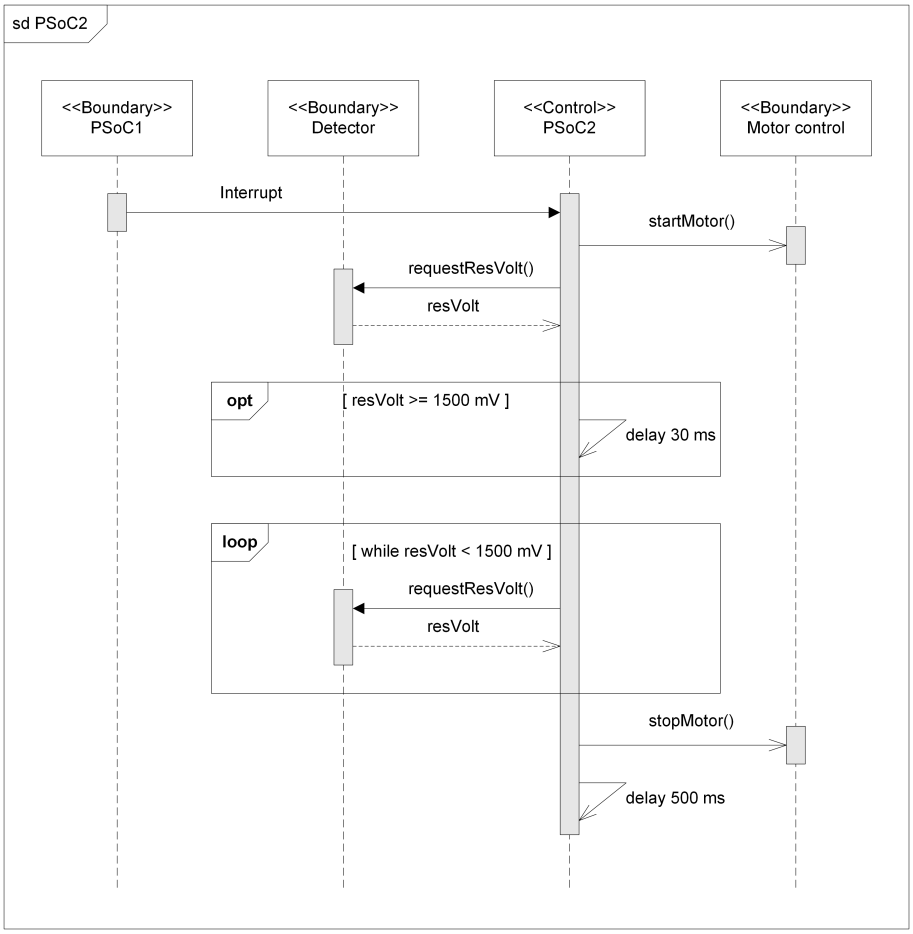


Figur 35: Klassediagram over klassen SPICommunication

En detaljeret klassebeskrivelse for SPICommunication klassen kan findes i dokumentationen, afsnit 4.2.1-*SPICommunication #ref.*

10.2.5 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, afsnit 4.2.6-*Affyringsmekanisme #ref.* I forbindelse med affyringen af kanonen køres en kodesekvens. Et sekvensdiagram for afviklingen af koden ses på figur 36.



Figur 36: 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 36. 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. Følgende tabel giver en oversigt over hvilke tests, der er blevet udført, samt en reference til hvor i dokumentationen testbeskrivelsen findes.

Test	**Reference**
Wii-Nunchuck	5.1.1
SPI Protokol	5.1.2
I2C Protokol	5.1.3
Brugergrænseflade	5.1.4
Rotationsdetektor	5.1.5
H-bro	5.2.1
Rotationsbegrænsing	5.2.2
Rotationsdetektor	5.2.3
Integrationstest	5.3
Accepttest	6

Tabel 13: Referencer til test afsnit

Modultest
Ved modultests blev én funktionalitet testet isoleret. Det vil sige med 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- og hardwarekomponenter.

Integrationstest
Ved integrationstesten blev alle elementer, der er inkluderet i use case 2, sammensat og systemet som helhed blev testet. Værdien af dette var, at alle hardware- og softwaremoduler som førhen kun var testet i isolation, blev testet i sammenhæng med andre moduler.

Accepttest
I samarbejde med vejlederen er der foretaget en accepttest af prototypen. Værdien af dette var, at test om alle de krav, der blev stillet til produktet, var overholdt.

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, afsnit 6-*Udfyldt Accepttest* side #ref.
På tabel 14 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 14: 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.

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

13 Fremtidigt arbejde

Noget, der kunne arbejdes videre på, er stabilisering af affyringsmekanismens vertikale bevægelighed, da kanonen er forholdsvis tung i begge ender. En ændring af H-broens opbygning, så spændingen over de to positive eller de to negative indgange kunne kortsluttes, ville medføre en mulighed for at motorbremse. Dermed ville den vertikale position af kanonen kunne låses under brug. En anden metode til at opnå bedre stabilitet, kunne være at geare mekanikken, så der opnås større trækkraft.

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 indeholde use case diagrammet en enkelt aktør, nemlig brugeren. Idét at use case 2 omhandler en systemtest, som den almene bruger ikke vil gøre brug af, kunne det være fordelagtigt at tilføje en *Admin* aktør der håndterer systemtesten. Derved interagerer brugeren kun med use case 1 - Spil Goofy Candygun 3000.

Systemets brugergrænseflade kunne opdateres med mulighed for at vise pointtavler, 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 gjort brug af Devkit 8000 og PSoC udviklingsboard, som har ubrugte funktionaliteter. Skulle produktet produceres til salg, ville være uhensigtsmæssigt at 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 projektforløb var at udvikle *Goofy Candy Gun 3000* spillet, hvor 1 til 2 spillere dyster 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 udviklingboards, forbundet via en I2C-bus. Ifølge use case beskrivelsen, skulle der implementeres en brugergrænseflade til visning statistikker og point. Disse funktionaliteter er ikke implementeret, med der er udviklet en demonstrations GUI, der viser opsætningen, dog uden nogen reele funktionaliteter. Derudover skal der foretages nogle kalibreringer for kanonens motorstyring og affyringmekanisme 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 mulliggør 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 resterene funktionaliteter, der har lavere prioritet, er ikke implementeret.

Litteratur

[1] SysML.org. *SysML Open Source Specification Project*, Tilgået 14 Maj, 2016.