3. Semesterprojekt - Goofy Candy Gun
Gruppe 3

Rieder, Kasper
201310514

Jensen, Daniel V.
201500152

Nielsen, Mikkel
201402530

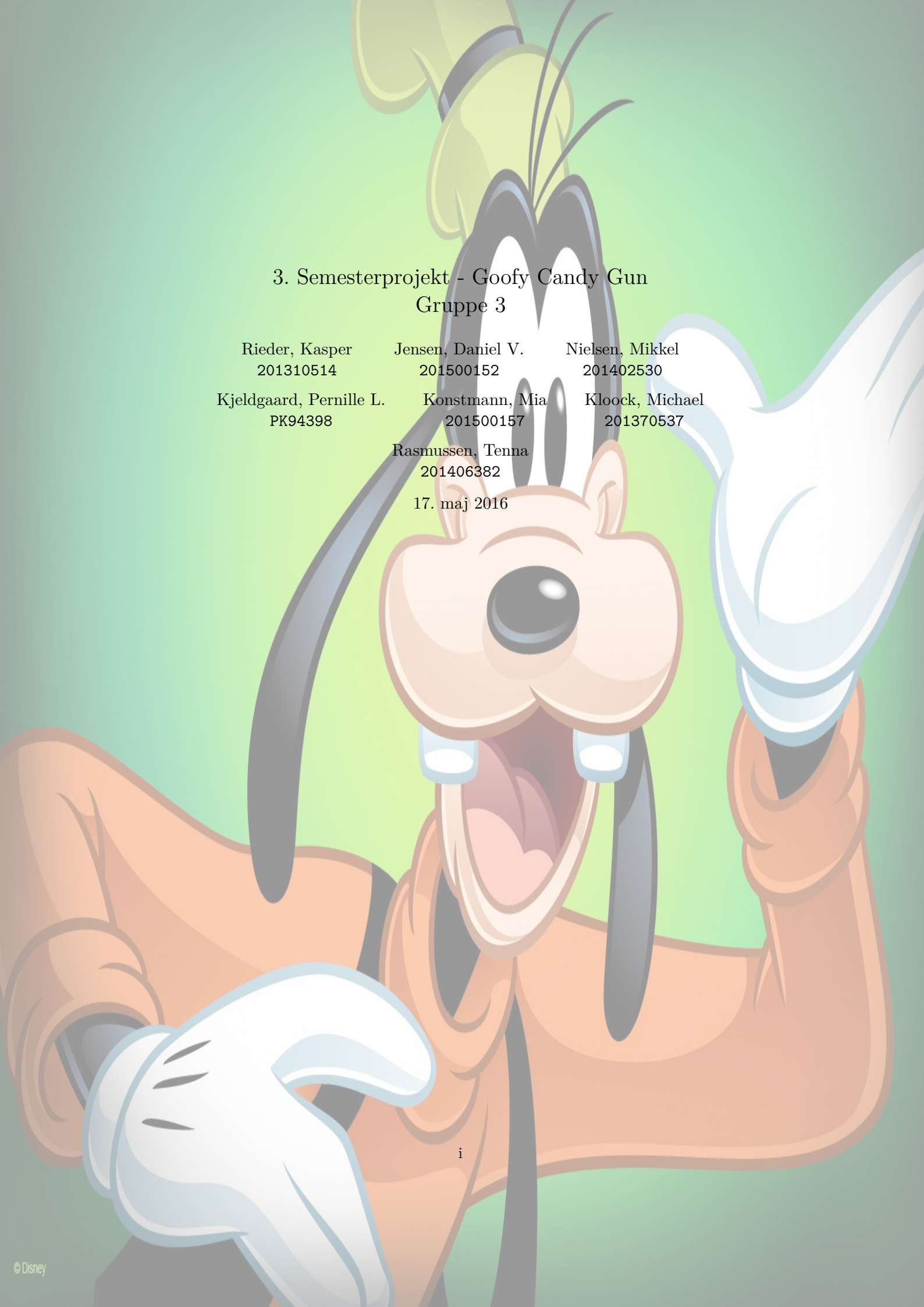
Kjeldgaard, Pernille L.
PK94398

Konstmann, Mia
201500157

Kloock, Michael
201370537

Rasmussen, Tenna
201406382

17. maj 2016



Indhold

Indhold ii

Figurer iv

1 Resumé 1

2 Abstrakt 1

3 Indledning 2
- 3.1 Krav til produktet . . . 2
- 3.2 Systembeskrivelse . . . 2
- 3.3 Rigt Billede . . . 4
- 3.4 Ansvarsområder . . . 5

4 Krav 6
- 4.1 Aktørbeskrivelse . . . 6
  - 4.1.1 Aktør - Bruger . . . 6
- 4.2 Use case beskrivelse . . . 7
  - 4.2.1 Use case 1 . . . 7
  - 4.2.2 Use case 2 . . . 7
- 4.3 Ikke-funktionelle krav . . . 7

5 Projektafgrænsning 8

6 Metode 9
- 6.1 SysML . . . 9
  - 6.1.1 Afvigelser fra standard brug . . . 9

7 Analyse 10
- 7.1 DevKit8000 . . . 10
- 7.2 Programmable System-on-Chip (PSoC) . . . 10
- 7.3 Wii-Nunchuck . . . 10

8 Systemarkitektur 11
- 8.1 Domænemodel . . . 11
- 8.2 Software Allokering . . . 11
- 8.3 Hardware . . . 13
  - 8.3.1 BDD . . . 13
  - 8.3.2 Blokbeskrivelse . . . 14
  - 8.3.3 IBD . . . 15
  - 8.3.4 Signalbeskrivelse . . . 17
- 8.4 Software . . . 20
  - 8.4.1 Informationsflow i systemet . . . 23
    - Wii-Nunchuck Information Flow . . . 23
    - System Test . . . 23
  - 8.4.2 SPI Kommunikations Protokol . . . 24
    - Designvalg . . . 26
  - 8.4.3 I2C Kommunikations Protokol . . . 26

Designvalg 28

9 Design og Implementering **29**
9.1 Valg og Begrundelse 29
9.2 Hardware 29
9.2.1 Motorstyring 29
H-bro 29
Detektor 29
9.2.2 Affyringsmekanisme 31
9.3 Software 31
9.3.1 SPI - Devkit8000 31
9.3.2 Interface Driver 31
9.3.3 Brugergrænseflade 33
9.3.4 Nunchuck 34
Afkodning af Wii-Nunchuck Data Bytes 34
Kalibrering af Wii-Nunchuck Analog Stick 35

10 Test **36**
10.1 Modultest 36
10.1.1 Software 36
Wii-Nunchuck 36
I2C Kommunikationsprotokol 37
SPI Kommunikationsprotokol 37
10.1.2 Hardware 37
10.2 Integrationstest 37
10.3 Accepttest 37

11 Udviklingsværktøjer **38**
11.1 PSoC 38
11.2 DevKit 8000 38
11.3 QT Creator 38

12 Resultater og Diskussion **39**
12.1 Perspektivering 39
12.2 Perspektivering til semesterets kurser 39
12.3 Ingeniørfaglige Styrker og Svagheder 39

13 Fremtidigt Arbejde **40**

14 Fejl og Mangler **41**

15 Referencer **42**

Figurer

1 Rigt Billede af det endelige produkt 4
2 Use case diagram 6
3 Systemets domænemodel 11
4 Systemets software allokeringer 12
5 Systemets software allokeringer 12
6 Systemets software allokeringer 13
7 Systemets software allokeringer 13
8 BDD af systemets hardware 14
9 IBD af systemets hardware 15
10 Klassediagram for DevKit8000 CPU 21
11 Klassediagram for PSoC0 CPU 22
12 Klassediagram for PSoC1 CPU 22
13 Wii-Nunchuck Input Data Forløb 23
14 System Test Forløb 24
15 Sekvensdiagram for aflæsning data fra en SPI-slave 26
16 Eksempel af I2C Protokol Forløb 28
17 potentiometer 29
18 ADC opbygning 30
19 Interface driver for UC2 32
20 Brugergrænseflade for usecase 2 33
21 State machine for brugergrænsefladen for usecase 2 34

1 Resumé

2 Abstrakt

3 Indledning

3.1 Krav til produktet

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

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

På baggrund af disse krav er der udarbejdet et system, der beskrives i afsnit 3.2.

Til dette projekt bliver systemet opbygget som en prototype. Grundet dette er der i afsnit 7 beskrevet nogle grundlæggende hardwarekomponenter til realisering af denne prototype.

3.2 Systembeskrivelse

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

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

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

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

Det endelige produkt omfatter:

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

  Pointvisning

  Kanonens vinkel

  Antal resterende skud

- En motor, der drejer kanonen om forskellige akser

  Dette styres med en Wii-nunchuck

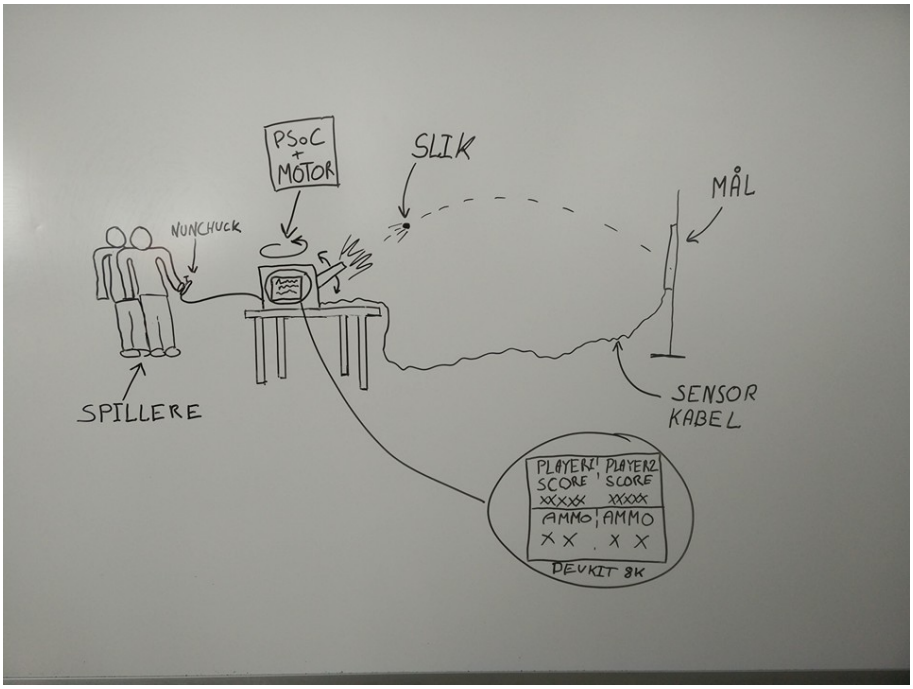
- Et mål, der kan registrere spillernes skud

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

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

3.3 Rigt Billede

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



Figur 1: Rigt Billede af det endelige produkt

3.4 Ansvarsområder

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

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

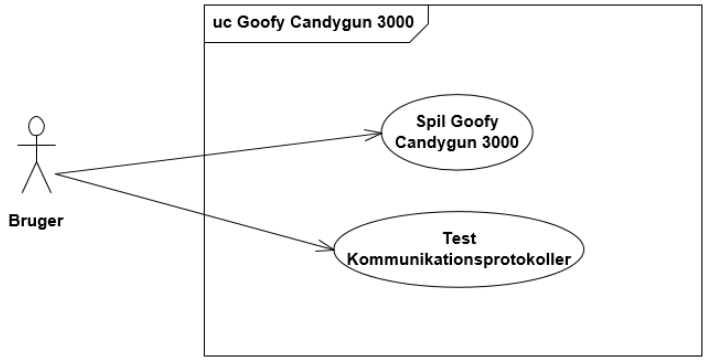
Ansvarsområder	DVJ	MPK	MN	KR	MMK	TR	PLK
I2C Kommunikationsprotokol PSoC Software	P	S		P			
Nunchuck PSoC Software	S	P		S			
SPI Kommunikationsprotokol PSoC Software	P	S		P			
SPI Driver Devkit Software		S		S		P	
Brugergrænseflade Devkit Software					P		
Motorstyring PSoC Software		P	P	S			
Use Case 2 Implementering	P	S		P			
H-bro							
Affyringsmekanisme							

Tabel 1: Ansvarsområder tabel

4 Krav

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

På figur 2 ses Use Case diagrammet for systemet.



Figur 2: Use case diagram

4.1 Aktørbeskrivelse

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

4.1.1 Aktør - Bruger

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

4.2 Use case beskrivelse

4.2.1 Use case 1

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

4.2.2 Use case 2

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

4.3 Ikke-funktionelle krav

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

5 Projektafgrænsning

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

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

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

6 Metode

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

6.1 SysML

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

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

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

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

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

6.1.1 Afvigelser fra standard brug

I SysML sekvensdiagrammer bruges beskedudveksling typisk til at repræsentere metoder på de objekter der kommunikeres med. Denne fremgangsmåde bruges i denne raport, dog er der nogle sekvensdiagrammer der afviger ved at beskedudvekslingen repræsenterer handlinger påført på objekter. Et eksempel på denne afvigelse kan ses i afsnit 8.4.1, figur 13. Denne afvigelse blev brugt for at kunne tydeliggøre interaktionen mellem systemets komponenter på en naturlig måde.

7 Analyse

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

7.1 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.

7.2 Programmable System-on-Chip (PSoC)

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

7.3 Wii-Nunchuck

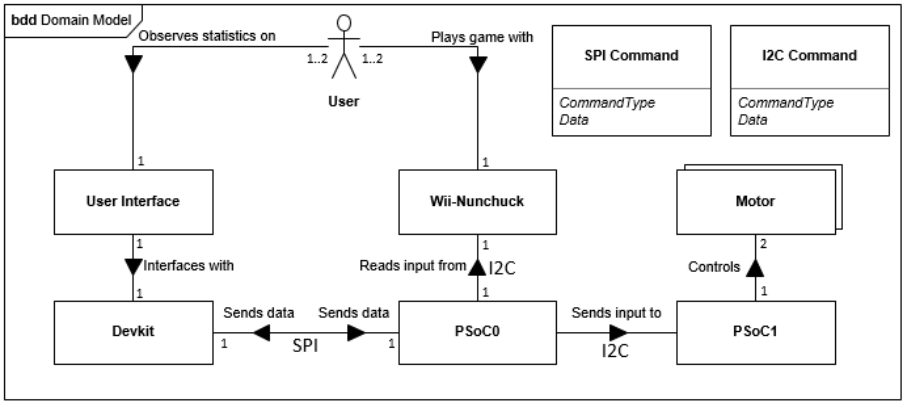
Til brugerstyring af systemets kanon er en Wii-Nunchuck controller valgt. Denne controller er valgt, da den har gode egenskaber til styring af en kanon. Wii-Nunchucken understøtter desuden en af PSoC'ens kommunikationsprotokoller, så den nemt kan kommunikere med resten af systemet.

8 Systemarkitektur

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

8.1 Domænemodel

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



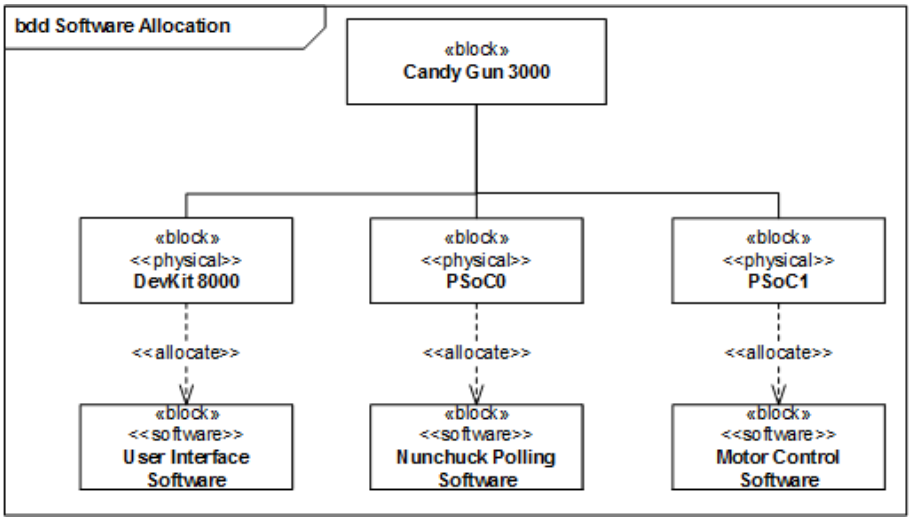
Figur 3: Systemets domænemodel

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

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

8.2 Software Allokering

Domænemodellen i figur 4 præsenterer systemets hardwareblokke. På figur 4 ses et software allokeringsdiagram, som viser hvilke hardwareblokke der har softwaredele af systemet allokeret på sig.

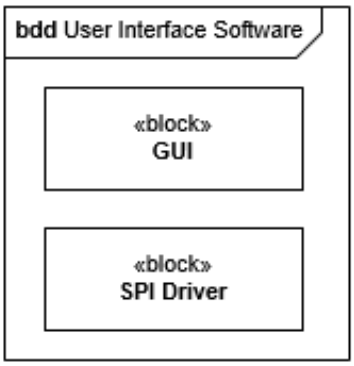


Figur 4: Systemets software allokeringer

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 figurerne: 5, 6, 7 ses et overblik over konceptuelle klasser der repræsenterer ansvarsområder de primære softwaredele får. For en beskrivelse af design og implementering af disse primære softwaredele henvises til **DESIGN OG IMPLEMENTERING #ref**.

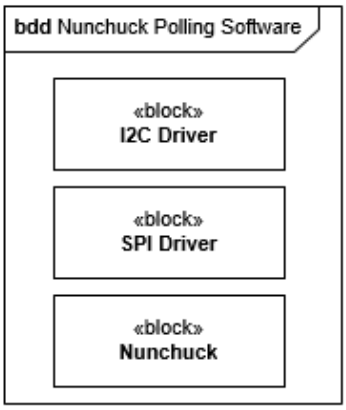
På figur 5 ses det at *User Interface Software*, allokeret på DevKit 8000, har ansvar for brugergrænsefladen samt en SPI Driver til kommunikation med PSoC0.



Figur 5: Systemets software allokeringer

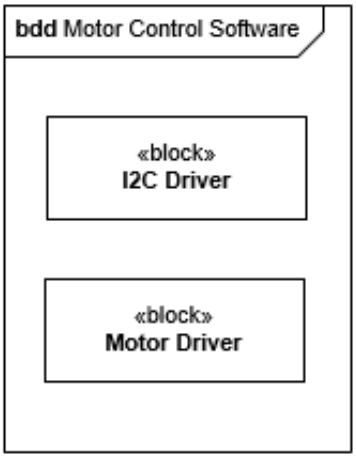
På figur 6 ses det at *Nunchuck Polling Software*, allokeret på PSoC0, har ansvar for en I2C Driver, SPI Driver, samt en Nunchuck API. I2C Driveren skal bruges til kommunikation med den fysiske Nunchuck controller samt PSoC1. SPI

Driveren skal bruges til kommunikation med DevKit 8000. Nunchuck API'en bruges for at tilgå data'en fra den fysiske Nunchuck controller.



Figur 6: Systemets software allokeringer

På figur 7 ses det at *Motor Control Software*, allokeret på PSoC1, har ansvar for en I2C Driver til kommunikation med PSoC0, samt en Motor Driver til motorstyring.

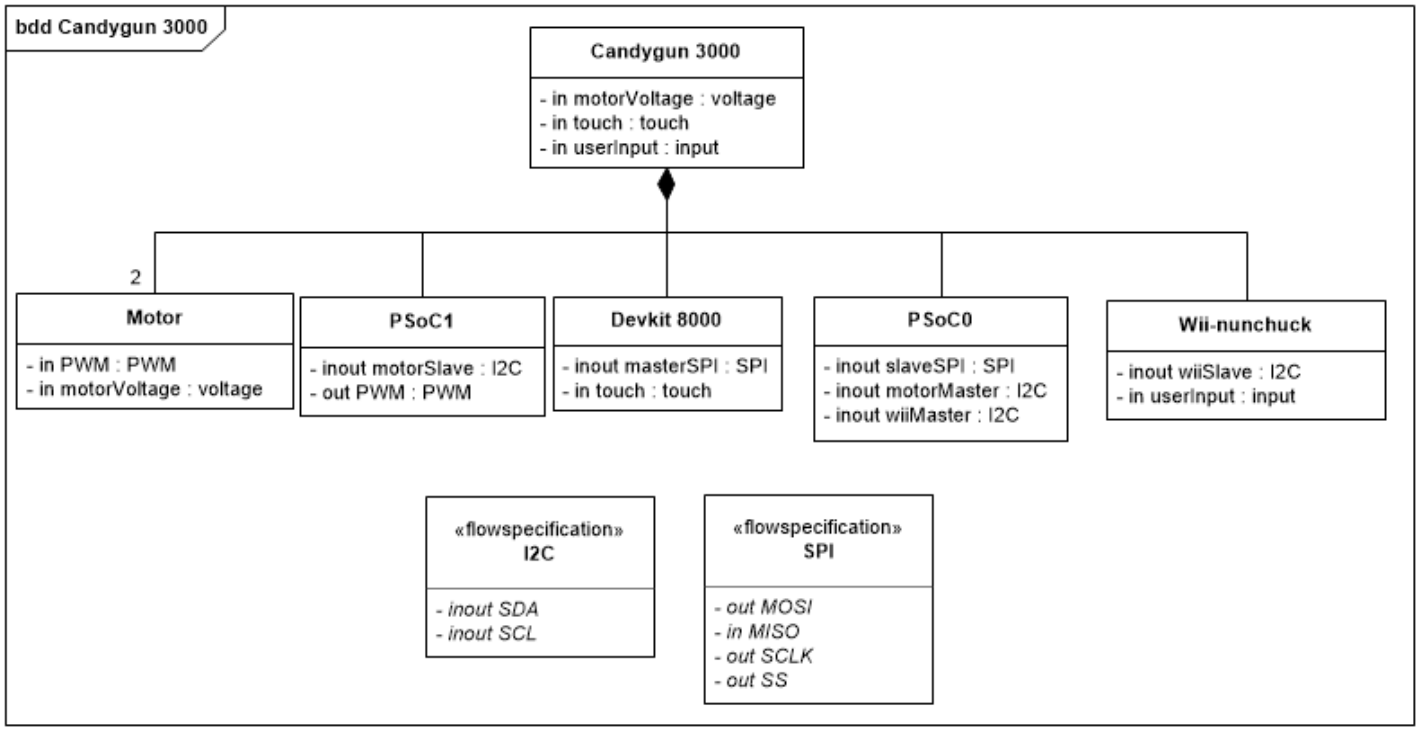


Figur 7: Systemets software allokeringer

8.3 Hardware

8.3.1 BDD

På figur 8 ses BDD'et for systemet.



Figur 8: BDD af systemets hardware

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

8.3.2 Blokbeskrivelse

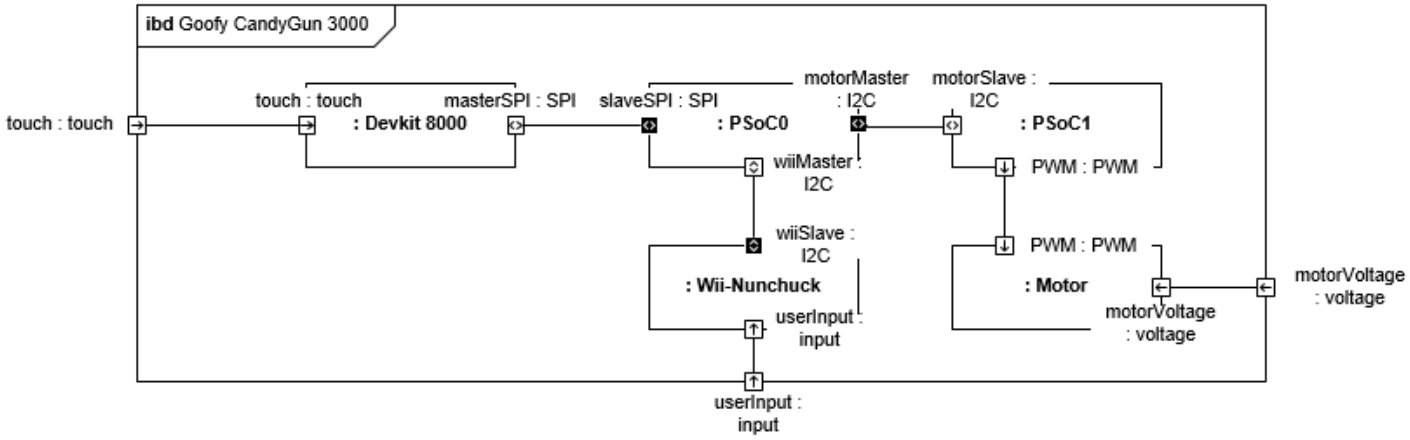
Følgende afsnit indeholder en blokbeskrivelse samt en flowspecifikation for I2C og SPI. I flowspecifikationen beskrives I2C og SPI forbindelserne mere detaljeret fra en masters synsvinkel.

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

Tabel 2: Blokbeskrivelse

8.3.3 IBD

På figur 9 ses IBD'et for systemet. Figuren viser hardwareblokkene med de fysiske forbindelser beskrevet i BDD'et (figur 8).



Figur 9: IBD af systemets hardware

Her vises alle hardwareblokke med de fysiske forbindelser beskrevet i BDD'et (figur 8).

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.

8.3.4 Signalbeskrivelse

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

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

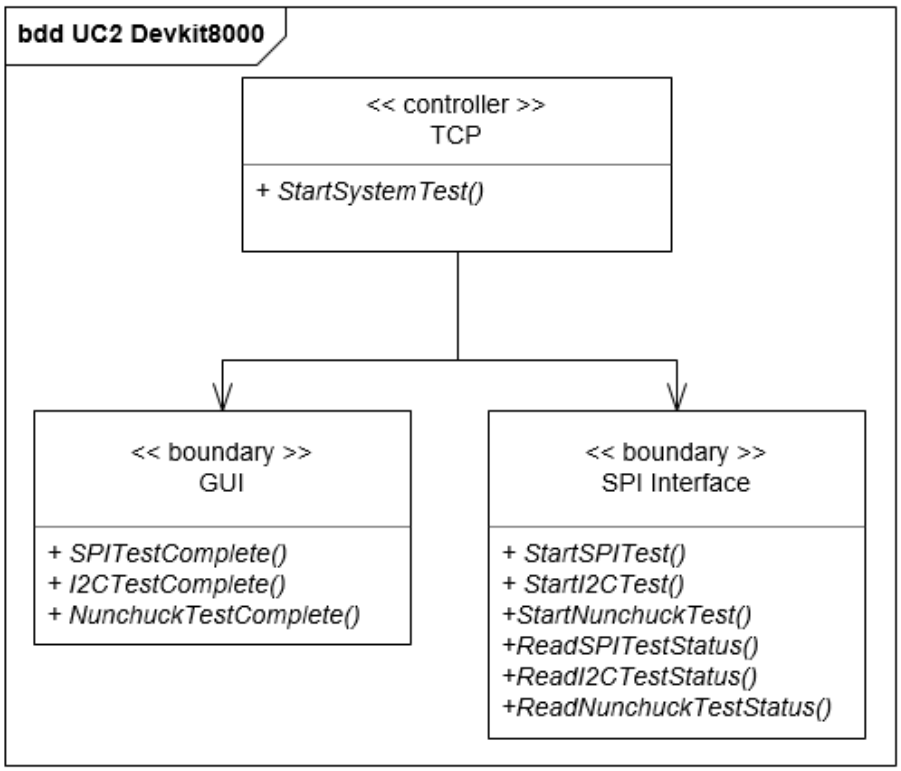
SPI	Denne blok beskriver den ikke-atomiske SPI forbindelse.	MOSI	Type: CMOS Spændingsniveau: 0-5V Hastighed: ?? Beskrivelse: Binært data der sendes fra master til slave.
		MISO	Type: CMOS Spændingsniveau: 0-5V Hastighed: ?? Beskrivelse: Binært data der sendes fra slave til master.
		SCLK	Type: CMOS Spændingsniveau: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

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

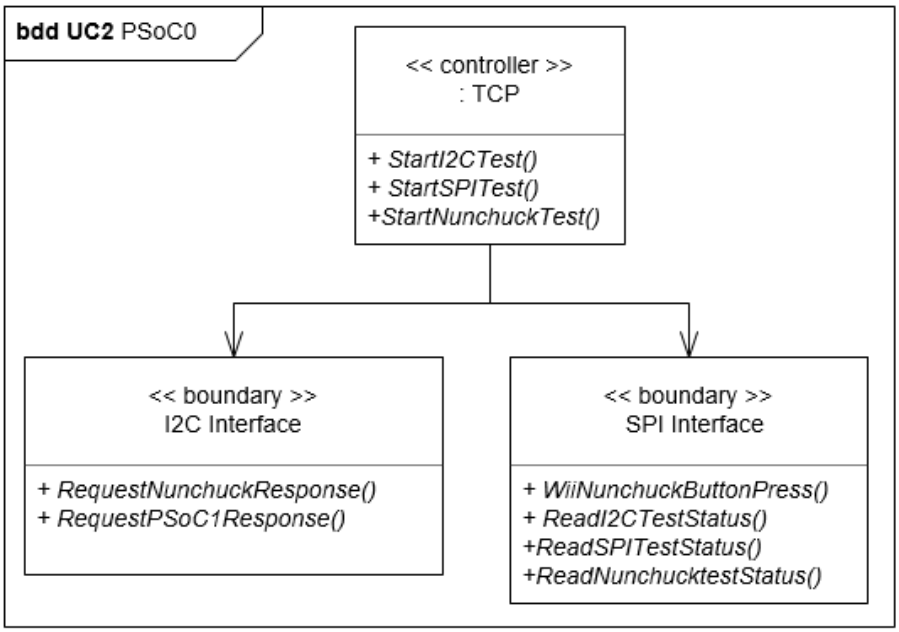
Tabel 3: Signalbeskrivelse

8.4 Software

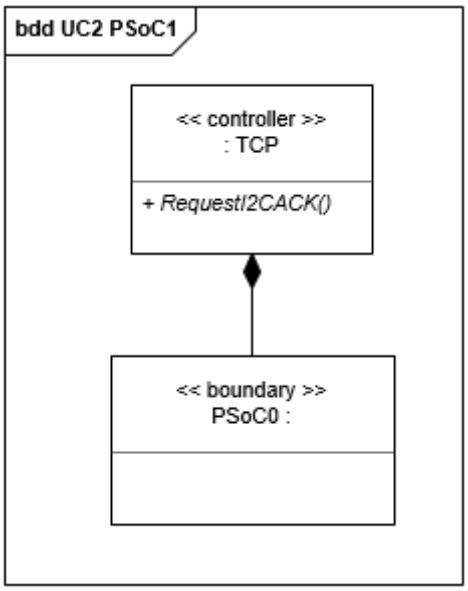
Softwaren til systemet er udarbejdet for hver CPU (se evt. software allokeringsdiagrammet på side 12 figur 4) ud fra en applikationsmodel, der kan ses i dokumentationen side (**Indsæt reference til applikationsmodel dokumentations #ref**). Ud fra disse applikationsmodeller er der udarbejdet klassediagrammer, der viser software klasserne på den givne CPU, samt klassernes relationer til hinanden. Et indledende forslag til klassediagrammerne for hver CPU i systemet kan ses på figur 10, 11 og 12.



Figur 10: Klassediagram for DevKit8000 CPU



Figur 11: Klassediagram for PSoC0 CPU



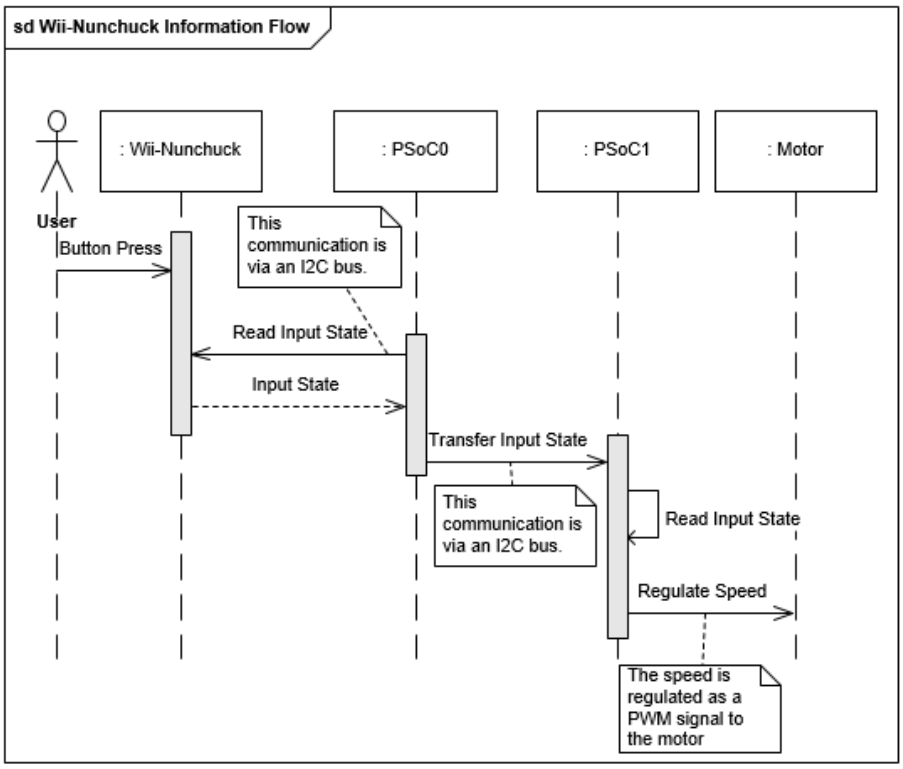
Figur 12: Klassediagram for PSoC1 CPU

8.4.1 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 13 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.

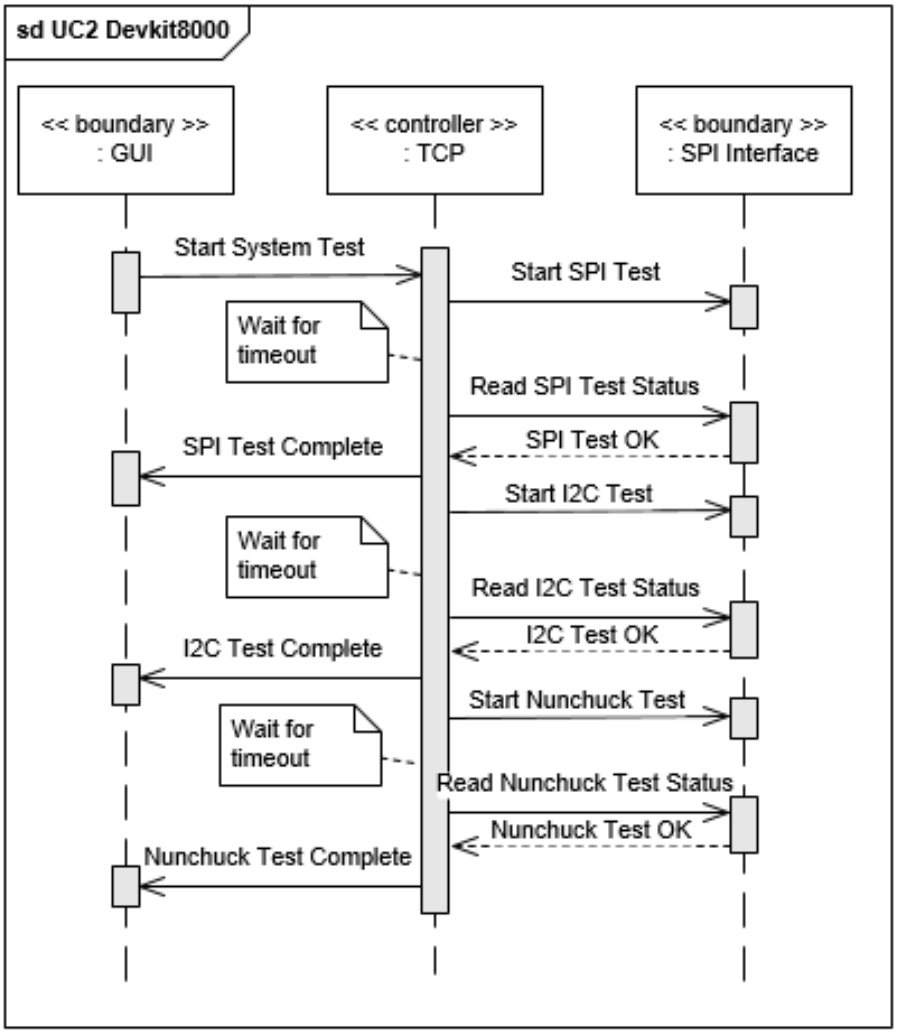


Figur 13: Wii-Nunchuck Input Data Forløb

System Test

Use Case 2 beskriver test af systemet før spillet startes. På figur 14 vises test sekvensen for en vellykket test. Det ses her at system testen sker sekventielt, og

tester systemets SPI og I2C busser.



Figur 14: System Test Forløb

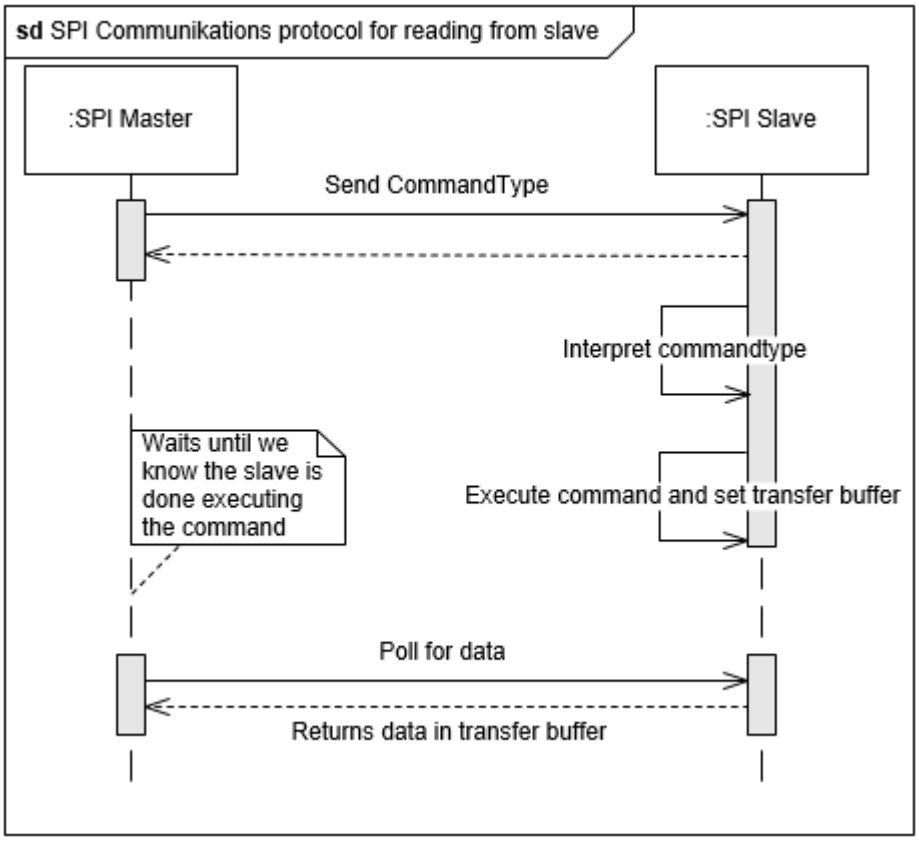
8.4.2 SPI Kommunikations Protokol

I afsnit 8.3.3 **IBD** ses på figur 9 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 4 ses en de anvendte kommandotyper samt en kort beskrivelse for hver af disse.

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

Tabel 4: SPI kommunikation kommandotyper

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



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

Designvalg

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

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

8.4.3 I2C Kommunikations Protokol

I afsnit 8.3.3 **IBD** ses på figur 9 at tre hardwareblokke kommunikerer via en I2C bus. Til denne I2C kommunikation er der defineret en protokol, som bestemmer hvordan modtaget data skal fortolkes. Denne protokol beskrives følgende.

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

hardwareblokkene som indgår i I2C kommunikationen fået tildelt adresser. På tabel 5 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 5: 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.

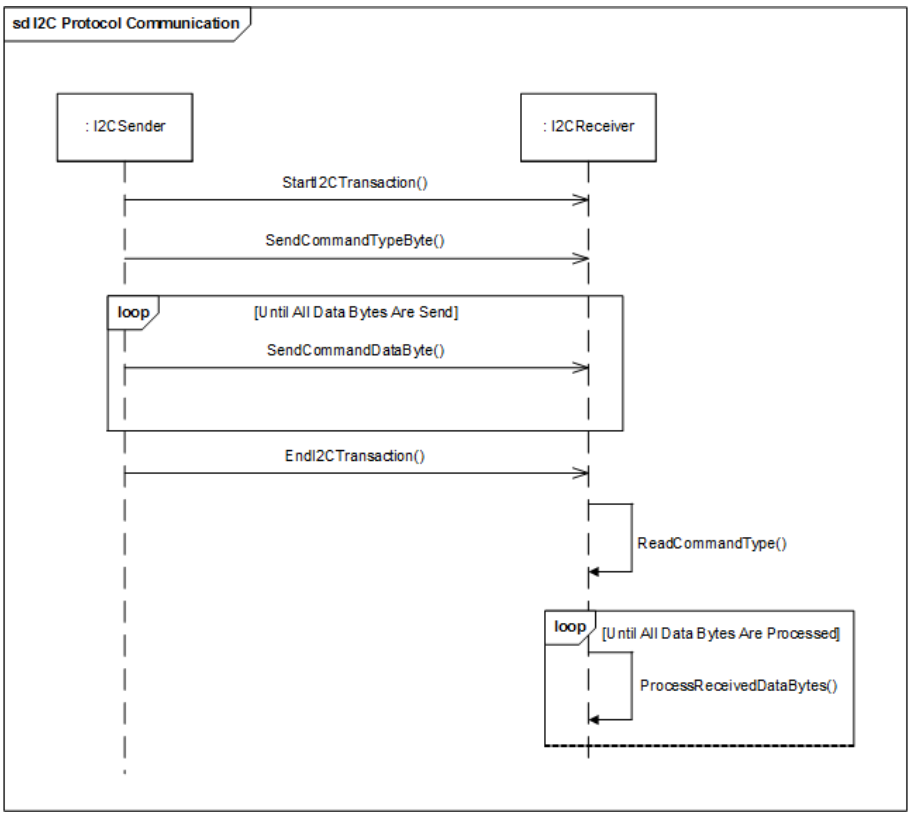
På tabel 6 ses de definerede kommandotyper og det tilsvarende antal af bytes der sendes ved dataveksling.

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

Tabel 6: I2C kommunikation kommandotyper

Kolonnerne *Binær Værdi* og *Hex Værdi* i tabel 6 viser kommandotypens unikke tal-ID i både binær- og hexadecimalform. Denne værdi sendes som den første byte, for at identificere kommandotypen.

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



Figur 16: Eksempel af I2C Protokol Forløb

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

Designvalg

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

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

9 Design og Implementering

9.1 Valg og Begrundelse

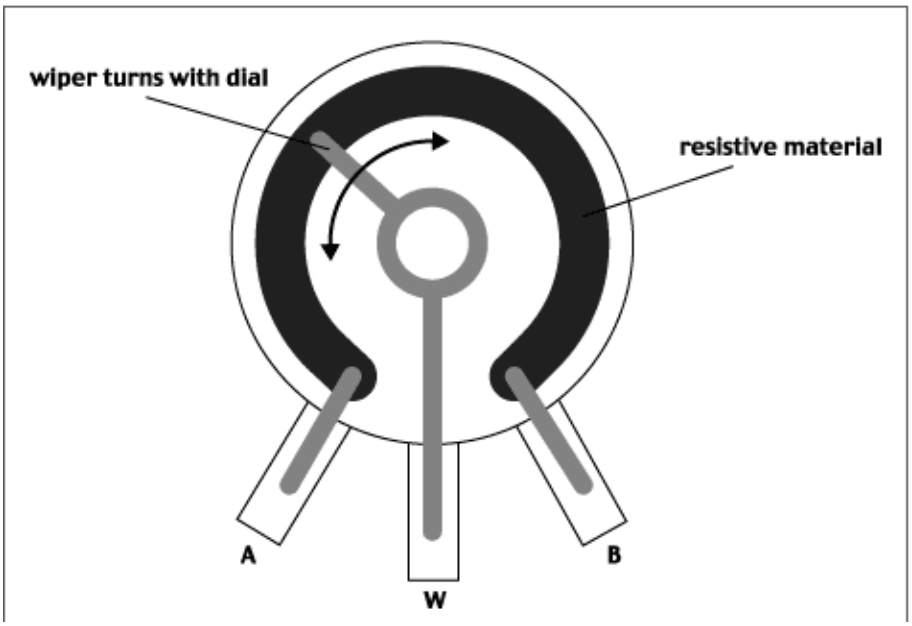
9.2 Hardware

9.2.1 Motorstyring

H-bro

Detektor

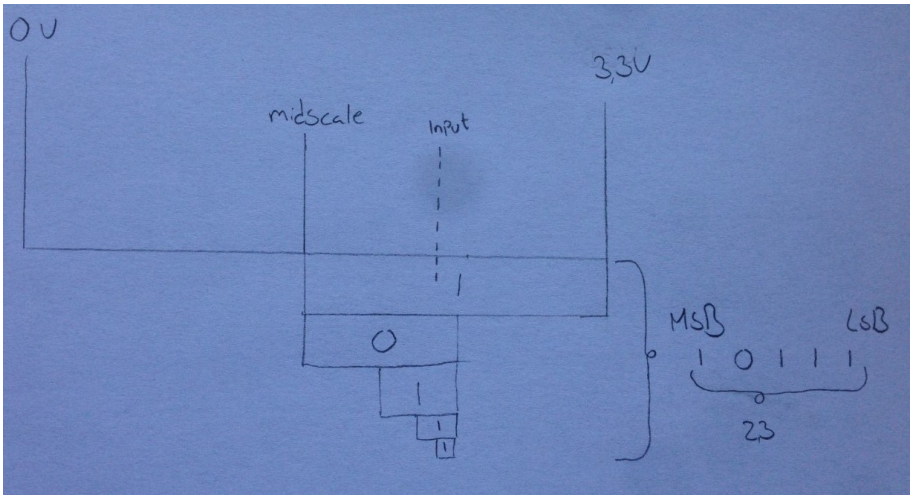
Denne detektor består af et potentiometer og en ADC, som er en del af "motorstyring" blokken, så ved hjælp af Potentiometeret kan man søger for at motoren ikke drejer for langt ud til begge sider. Der kunne være brugt en stepper motor til at syre det med, men for at få mere hardware med, blev det besluttet at det var en bedre løsning at gør det på denne måde.



Figur 17: potentiometer

måske et billede af opstilling i stedet for
Potentiometer har til formal at skulle holde styre på at motoren ikke køre ud længere end det hvad det fastsatte krav er. Der er blevet valgt at bruge et Potentiometer som er et linæret et, som vil sige at når man for hver ændring man laver på Potentiometeret, vil der også ske en ændring i output spændingen. Så når den bliver fast gjord til motoren vil man kunne se en ændring i den spænding som kommer ud fra potentiometeret, som bliver sendt ind i ADC'en, som er af typen Sequencing Successive Approximation ADC, som er placeret internt i PSoC'en. Så når output spændingen fra potentiometer bliver aflæst,

vil man kunne vide præcise hvor motoren er. ADC'en er en sample hold kreds, som vil sige den holder på dataen end til der er fundet en værdi, så der går noget tid i mellem hver værdi, så der er søgeret for at der ikke bliver lavet over samples og der ikke bliver lavet aliasing af det signal som kommer ind i ADC'en. For at finde den værdi som bliver sendt ind i ADC'en, bliver der lavet en skala som bliver halvert alt efter hvor værdien ligger på skalaen, som vist på figur 18



Figur 18: ADC opbygning

Der er blevet målt med en vinkel måler for at over holde kravene i kravspecifikationen**#ref Reference til kravspecikation (ikke funktionelle krav)**. Hvor der blev fundet frem til en min og max hvor motoren må køre i mellem.

Min=915 mV

Max = 2000mV

Disse værdier er aflæst fra PSoC'en, da den har en ref spændingen på 3.3V vil den have lavere værdier end hvad der bliver sendt ud fra Potentiometeret.

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

Der er en forskel på ca. 1.515 mellem potentiometer og ADC. Så ADC er ca. 1,515 mindre end hvad potentiometer sender ud.
For at forstå hvordan ADC'en og potentiometeret funger henvises der til dokumentatioen

9.2.2 Affyringsmekanisme

9.3 Software

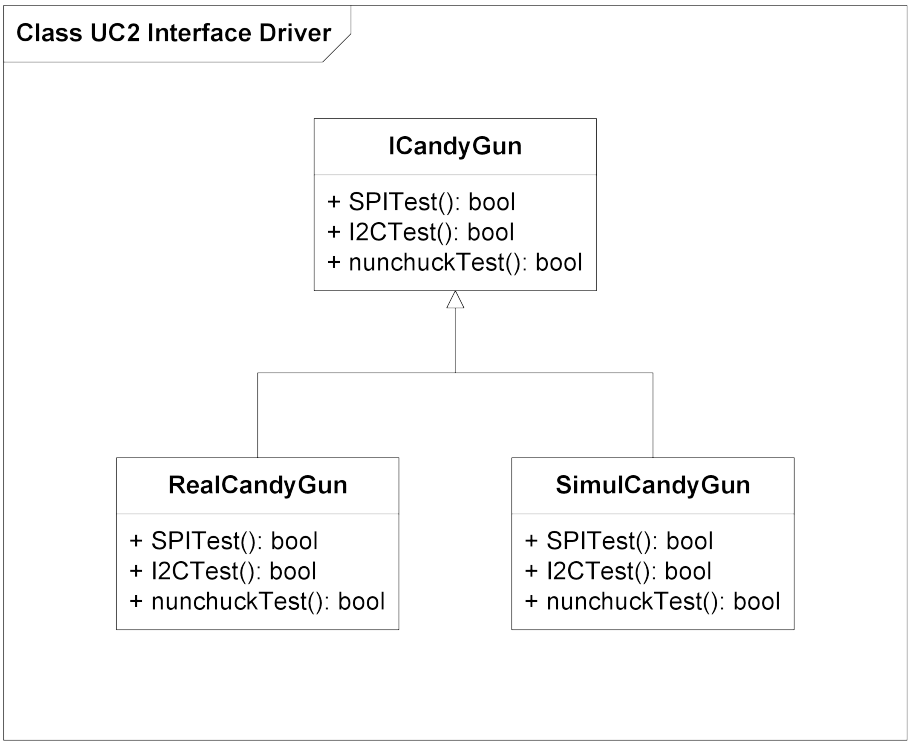
9.3.1 SPI - Devkit8000

Candydriveren sørger for SPI-kommunikationen fra Devkit8000 til PSoC0. Driveren er skrevet i c, hvilket er typisk for drivere til linuxplatforme.
SPI-kommunikationen er implementeret med SPI bus nummer 1, SPI chip-select 0 og en hastighed på 1 MHz (et godt stykke under max på 20 MHz for en sikkerhedsskyld). Desuden starter clocken højt og data ændres på falling edge og aflæses på rising edge. Dermed bliver SPI Clock Mode 3. Derudover sendes der 8 bit pr transmission, hvilket passer med SPI-protokollen for projektet.
For at kunne anvende driveren, når SPI er tilsluttet, er der oprettet et hotplug-modul, som fortæller kernen, at der er et SPI device, som matcher driveren. Det kan SPI-forbindelsen ikke selv gøre, som usb fx kan. Selve driveren er i candygun.c opbygget som en char driver. For at holde forskellige funktionaliteter adskilt er alle funktioner, der har med SPI at gøre, implementeret i filen candygun-spi-c. Så når der fx skal requestes en SPI ressource i init-funktionen i candygun.c, så anvender driveren en funktion fra candygun-spi.c til det. I probe-funktionen sættes bits_per_word til 8, da vi sender otte bit som nævnt tidligere. I exit-funktionen anvender candygun.c igen en funktion fra candygun-spi.c - denne gang til at frigive SPI ressourcen. I write-metoden gives der data med fra brugeren. I dette tilfælde udgøres brugeren af Interface driveren og dataet er en 8 bit kommando fra SPI-protokollen. Dog er dataet fra brugeren i første omgang læst ind som en charstreng. I write-metoden bliver det så lavet om til en int. For at overføre dataet på en sikker måde anvendes funktionen copy_from_user() til at overføre data fra brugeren. Write-funktionen fra candygun.c anvender derefter en write-funktion fra candygun-spi.c, hvor den sender brugerinputtet med. I den spi-relaterede write-funktion bliver bruger inputtet lagt i transfer bufferen og der NULL bliver lagt i receive bufferen, og med spi_sync-funktionen bliver det sendt.
Ofte ville der en spi read-funktion først indeholde en write-del, som fortalte SPI-slaven, hvad der skulle læses over i bufferen. Det ville typisk efterfølges af et delay og så en read-del. Men i dette projekt skal der ofte afventes et brugerinput, som ikke kan styres af et fast delay, og der skal generelt sendes en aktiv kommando før der læses. Derfor er det besluttet at read-funktionen kun indeholder en read-del i transmissionen. Dermed skal write-funktionen altid aktivt anvendes inden der læses, da PSoC0 ellers ikke ved, hvad der skal gøres/lægges i bufferen.
Når funktionen har modtaget resultatet fra transmissionen returneres det til brugeren med funktionen copy_to_user(), som igen sørger for at overførslen af data foregår på en sikker måde.

9.3.2 Interface Driver

Interface driveren fungerer som bindeled mellem brugergrænsefladen og candy-driveren på Devkit8000. Den indeholder tre funktioner. Funktionerne anvendes i use case 2 til at teste kommunikationsforbindelserne i resten af systemet. Interface driveren er designet og implementeret i C++ og gør brug af klassere-

lationen arv. Et klassediagram for interface driveren se på figur 19.

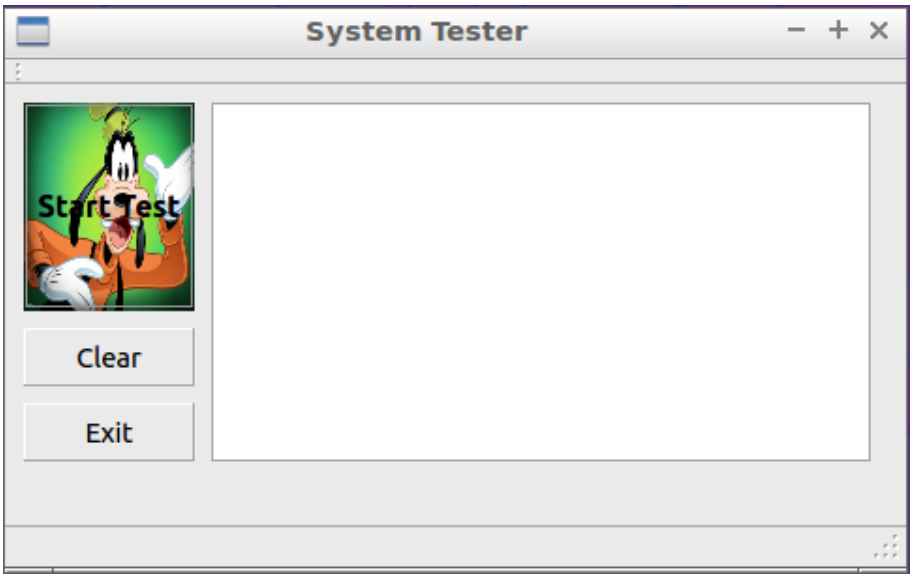


Figur 19: Interface driver for UC2

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

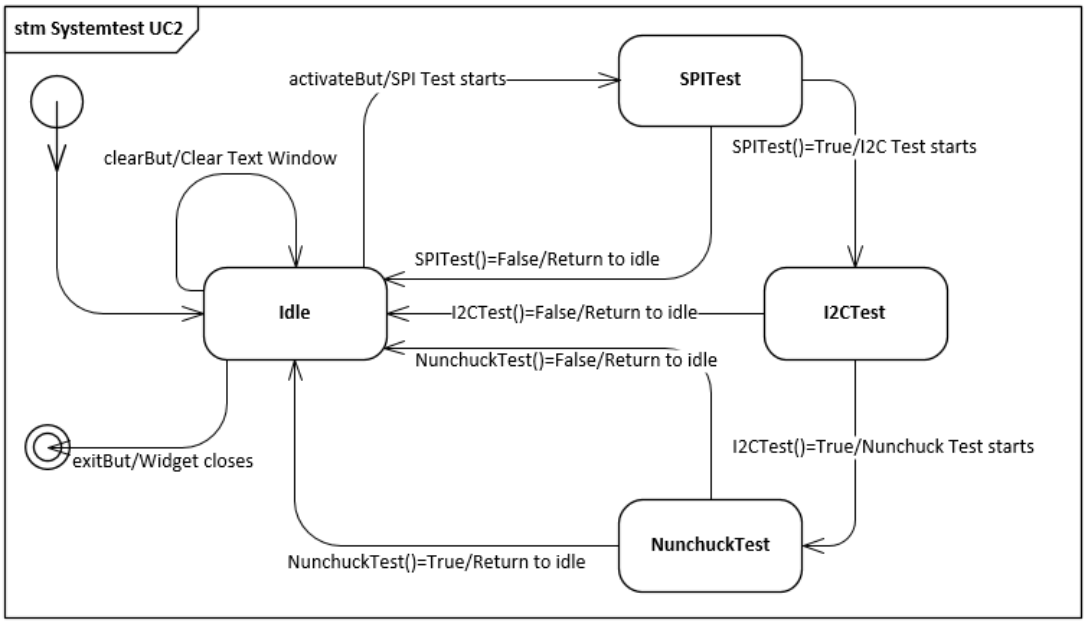
ét sekund og læser så svaret fra filen. Da der i nunchucktesten ventes på et brugerinput, og brugeren skal have lidt tid til at trykke på nunchuck-knappen, er der oprettet en while-løkke, som tjekker flere gange om testen returnerer true. Hvis testen ikke returnerer true ved første check, venter funktionen atter et sekund og tjekker igen. Det gør den op til 15 gange og melder derefter om fejl, hvis ikke den returnerer true inden.
Brugergrænsefladen anvender interface driveren ved at inkludere headerfilerne og oprette en ICandyGun pointer, der peger på en instans af én af de to afledte klasser. Ved at pakke kommunikationen til kernemodulet for candydriveren væk i funktioner kan brugergrænsefladen anvende funktionerne uden at kende til SPI-protokollen. Det sikrer igen lav kobling, og i tilfælde hvor det kunne ønskes, at SPI-kommunikationen kan erstattes af en anden kommunikationsform, kan det gøres uden, at der skal foretages ændringer i brugergrænsefladen.

9.3.3 Brugergrænseflade



Figur 20: Brugergrænseflade for usecase 2

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



Figur 21: State machine for brugergrænsefladen for usecase 2

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

9.3.4 Nunchuck

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

Afkodning af Wii-Nunchuck Data Bytes

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

AfkodetByte = (AflæstByte XOR 0x17) + 0x17

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

Kalibrering af Wii-Nunchuck Analog Stick

De afkodede bytes for Wii-Nunchuck's analoge stick har definerede standardværdier for dets forskellige fysiske positioner. Disse værdier findes i tabel 7

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 7: 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 Test

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

10.1 Modultest

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

10.1.1 Software

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

Wii-Nunchuck

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

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

Tabel 8 og 9 præsenterer modultest resultaterne for Wii-Nunchuck.

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

Tabel 8: Modultest af Wii-Nunchuck Handshake

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

Tabel 9: Modultest af Wii-Nunchuck Data Aflæsning

Det kan på tabel 8 og 9 ses at de egentlige resultaterne stemte overens med de forventede.

I2C Kommunikationsprotokol

I2C Kommunikationsprotokollen beskrevet i afsnit 8.4.3 blev modultestet ved to tests. Den første test er til for at verificere at kommandotyper bliver overført på I2C bussen i korrekt format. Den anden test er til for at verificere at modtaget I2C data fortolkes korrekt af software på PSoC0.

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

Tabel 10: Modultest af kommandotype på I2C Bussen

Forventet Resultat
Egentlig Resultat

Tabel 11: Modultest af kommando fortolknings software

SPI Kommunikationsprotokol

10.1.2 Hardware

10.2 Integrationstest

10.3 Accepttest

11 Udviklingsværktøjer

11.1 PSoC

11.2 DevKit 8000

11.3 QT Creator

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

Liste: PSoC Creator Analog discovery

12 Resultater og Diskussion

12.1 Perspektivering

12.2 Perspektivering til semesterets kurser

12.3 Ingeniørfaglige Styrker og Svagheder

13 Fremtidigt Arbejde

14 Fejl og Mangler

15 Referencer