

3. Semesterprojekt - Goofy Candy Gun Dokumentation - Gruppe 3

Rieder, Kasper Jensen, Daniel V. Nielsen, Mikkel
201310514 201500152 201402530

Kjeldgaard, Pernille L. Konstmann, Mia Kloock, Michael
PK94398 201500157 201370537

Rasmussen, Tenna Vejleder:
201406382 Gunvor Elisabeth Kirkelund

26. maj 2016

Indhold

Indhold	ii
1 Kravspecifikation	1
1.1 Aktør kontekst diagram	1
1.2 Use Case Diagram	1
1.3 Aktør beskrivelse	2
1.3.1 Aktør - Bruger	2
1.4 Fully Dressed Use Cases	2
1.4.1 Use Case 1 - Spil Goofy Candy Gun 3000	3
1.4.2 Use Case 2 - Test Kommunikationsprotokoller	4
1.5 Ikke funktionelle krav	5
2 Accepttestspezifikation	7
2.1 Use case 1 - Hovedscenarie	7
2.1.1 Use case 1 - Extension 1	8
2.1.2 Use case 1 - Extension 2	8
2.2 Use case 2 - Hovedscenarie	9
2.2.1 Use case 2 - Exception 1	9
2.2.2 Use case 2 - Exception 2	9
2.2.3 Use case 2 - Exception 3	10
2.3 Ikke-funktionelle krav	10
3 Systemarkitektur	11
3.1 Domænemodel	11
3.2 BDD for Candygun 3000	12
3.2.1 Blokbeskrivelse	13
3.3 IBD for Candygun 3000	13
3.3.1 Signalbeskrivelse	15
3.4 Software Allokéringsdiagram	24
3.5 Applikationsmodeller	25
3.5.1 use case 1 - Spil Goofy Candy Gun 3000	26
Applikationsmodel for Nunchuck Polling Software . . .	26
Applikationsmodel for Motor Control Software	28
Applikationsmodel for Projectile Launcher Software .	29
3.5.2 use case 2 - Test Kommunikationsprotokoller	32
Applikationsmodel for User Interface Software	33
Applikationsmodel for Nunchuck Polling Software . . .	36
Applikationsmodel for Motor Control Software	39
3.6 Samlede Klassediagrammer	41
3.7 Kommunikationsprotokoller	44
3.7.1 SPI Protokol	44
3.7.2 I2C Protokol	46
3.8 Brugergrænseflade	49
4 Design og implementering	50
4.1 Software Design	50
4.1.1 SPI Devkit 8000 - Candygun driver	50
Indstillinger for SPI	50

Opbygning af driver	50
Opbygning af write-metode	51
Opbygning af read-metode	52
Hotplug	53
Metodebeskrivelser	53
4.1.2 Brugergrænseflade	54
Interfacedriver	55
Klassebeskrivelse	55
Systemtest GUI	56
Klassebeskrivelse	56
Demo GUI	57
Klassebeskrivelse	58
4.2 PSoC Software	60
4.2.1 SPICommunication	62
Klassediagram	62
Klassebeskrivelser	63
SPI Indstillinger på PSoC	64
Test Opstart Algoritme	64
4.2.2 I2CCommunication	66
Klassediagram	66
Klassebeskrivelser	66
I2C indstillinger på PSoC	67
Send Data Algoritme	67
Receive Data Algoritme	69
4.2.3 Nunchuck	69
Klassediagram	70
Klassebeskrivelser	70
Nunchuck Read Data Algoritme	70
4.2.4 Rotationsbegrensning	72
4.2.5 Motorstyring	72
4.2.6 Affyringsmekanisme	73
4.3 Afkodning af Wii-Nunchuck Data Bytes	77
4.4 Kalibrering af Wii-Nunchuck Analog Stick	77
4.5 Hardwaredesign	78
4.5.1 Motor	78
4.5.2 Motorstyring	78
H-bro	78
Rotationsbegrensning	83
4.5.3 Affyringsmekanisme	85
Rotationsdetektor	85
Motorstyring	89
Kanon og platform	90
5 Modultest	94
5.1 Software	94
5.1.1 Modultest af Wii-Nunchuck	94
5.1.2 SPI Protokol	96
SPI Bus Test	96
5.1.3 I2C Protokol	97
Test af NunchuckData kommandotype	97

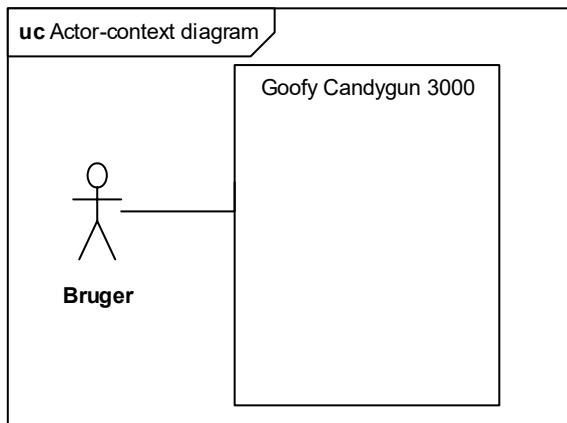
5.1.4	NunchuckData kommandotype test del 1	97
5.1.4	NunchuckData kommandotype test del 2	98
5.1.4	Systemtest GUI	100
5.1.4	Start Test-knap Test	100
5.1.4	Clear-knap Test	101
5.1.4	Exit-knap Test	102
5.1.5	Rotationsdetektor	102
5.2	Hardware	104
5.2.1	H-bro	104
5.2.2	Rotationsbegrensning	107
5.2.3	Rotationsdetektor	110
5.2.3	Test af relevante målepunkter på rotationsdetektor	111
5.3	Integrationstest - Use case 2	120
6	Udfyldt Accepttest	124
6.1	Use case 1 - Hovedscenarie	124
6.1.1	Use case 1 - Extension 1	125
6.1.2	Use case 1 - Extension 2	125
6.2	Use case 2 - Hovedscenarie	126
6.2.1	Use case 2 - Exception 1	126
6.2.2	Use case 2 - Exception 2	126
6.2.3	Use case 2 - Exception 3	127
6.3	Ikke-funktionelle krav	127
7	Udviklingsværktøjer	128
7.1	QT Creator	128
7.2	PSoC Creator	128
7.3	Analog Discovery	128
8	Referencer	129
	Litteratur	129

1 Kravspecifikation

Det følgende afsnit udpensler projektet ved specifikation af aktører, use cases, samt ikke-funktionelle krav.

1.1 Aktør kontekst diagram

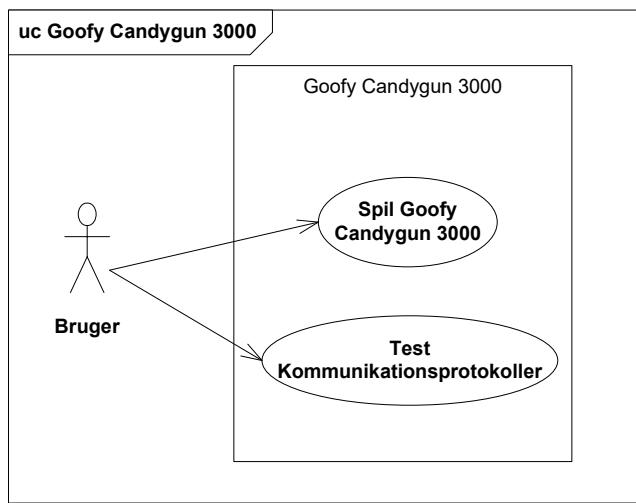
Figur 1 viser et kontekst diagram for Goofy Candygun 3000.



Figur 1: Kontekst diagram for slikkanonen

1.2 Use Case Diagram

Figur 2 viser et use case diagram for Goofy Candygun 3000.



Figur 2: Use case diagram for slikkanonen

1.3 Aktør beskrivelse

Det følgende afsnit beskriver de identificerede aktører for Goofy Candygun 3000.

1.3.1 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.

1.4 Fully Dressed Use Cases

Det følgende afsnit indeholder de *fully dressed use cases* for Goofy Candy Gun, som er identificeret på figur 2.

1.4.1 Use Case 1 - Spil Goofy Candy Gun 3000

Navn	Spil Goofy Candygun 3000
Mål	At spille spillet
Initiering	Bruger
Aktører	Bruger
Antal samtidige forekomster	Ingen
Prækondition	Spillet og kanonen er operationel. UC2 Test kommunikationsprotokoller er udført
Postkondition	Brugeren har færdiggjort spillet
Hovedscenarie	<ol style="list-style-type: none"> 1. Bruger vælger spiltype på brugergrænseflade 2. Bruger vælger antal skud til runde 3. Bruger fylder magasin med slik tilsvarende antal skud 4. Bruger indstiller kanon med analogstick på Wii-nunchuck 5. Bruger udløser kanonen med Wii-nunchucks trigger 6. System lader et nyt skud 7. Brugergrænseflade opdateres med spillets statistikker 8. Punkt 4 til 7 gentages indtil skud er opbrugt <ul style="list-style-type: none"> [Extension 1: Bruger vælger 2 player mode] [Extension 2: Bruger afslutter det igangværende spil] 9. Brugergrænseflade viser afslutningsinfo for runden 10. Bruger afslutter runde 11. Brugergrænseflade vender tilbage til starttilstand
Udvidelser/ undtagelser	<p>[Extension 1: Brugeren vælger 2 player mode]</p> <ol style="list-style-type: none"> 1. Bruger overdrager Wii-nunchuck til den anden bruger 2. Punkt 4 til 7 gentages indtil skud er opbrugt 3. Use case genoptages fra punkt 8 <p>[Extension 2: Bruger afslutter spillet]</p> <ol style="list-style-type: none"> 1. Brugergrænseflade vender tilbage til starttilstand 2. Use case afsluttes

1.4.2 Use Case 2 - Test Kommunikationsprotokoller

Navn	Test kommunikationsprotokoller
Mål	At teste kommunikations protokoller
Initiering	Bruger
Aktører	Bruger
Antal samtidige forekomster	Ingen
Prækondition	Systemet er tændt
Postkondition	Systemet er gennemgået testen og resultaterne er vist
Hovedscenarie	<ol style="list-style-type: none"> 1. Bruger vælger test system på brugergrænseflade 2. Devkit 8000 sender start SPI test til PSoC0 via SPI 3. PSoC0 sender acknowledge til Devkit 8000 via SPI [Exception 1: PSoC0 sender ikke acknowledge] 4. Brugergrænseflade meddeler om gennemført SPI test 5. Devkit 8000 sender start I2C test til PSoC0 via SPI 6. PSoC0 sender start I2C test til PSoC slaver via I2C 7. PSoC slaver sender acknowledge til PSoC0 via I2C [Exception 2: PSoC slaver sender ikke acknowledge] 8. PSoC0 meddeler om gennemført I2C test til Devkit 8000 via SPI 9. Brugergrænseflade meddeler om gennemført I2C test 10. Brugergrænseflade anmoder bruger om at trykke på knap 'Z' på Wii-nunchuck 11. Wii-nunchuck sender besked "Knap Z trykket" til PSoC0 via I2C [Exception 3: Wii-nunchuck sender ikke "Knap Z trykket"] 12. PSoC0 videresender besked om "Knap Z trykket" til Devkit 8000 via SPI 13. Brugergrænseflade meddeler om gennemført Wii-nunchuck test 14. Brugergrænseflade meddeler at test af kommunikationsprotokoller er gennemført

Udvidelser/ undtagelser	<p>[Exception 1: PSoC0 sender ikke acknowledge]</p> <ol style="list-style-type: none"> 1. Brugergrænseflade meddeler fejl i SPI kommunikation 2. UC2 afsluttes <p>[Exception 2: PSoC slaver sender ikke acknowledge]</p> <ol style="list-style-type: none"> 1. PSoC0 sender fejlmeldelse til Devkit 8000 2. Brugergrænseflade meddeler fejl i I2C kommunikation 3. UC2 afsluttes <p>[Exception 3: Wii-nunchuck sender ikke "Knap Z trykket"]</p> <ol style="list-style-type: none"> 1. PSoC0 sender fejlmeldelse til Devkit 8000 2. Brugergrænseflade meddeler fejl i I2C kommunikation med Wii-nunchuck 3. UC2 afsluttes
--------------------------------	--

1.5 Ikke funktionelle krav

1. Kanonen må ikke kunne rotere 360 °
2. Kanonen skal kunne affyre projektiler med en diameter på $1,25 \text{ cm} \pm 2 \text{ mm}$
3. Kanonens størrelse må maksimalt være 50h x 60b x 50d i centimeter
4. Fra aftryk på trigger til affyring må der maksimalt gå ti sekunder
5. Affyring af kanonen skal kunne afvikles minimum tre gange pr. minut
6. Figur 3 viser en skitse af hvordan den grafiskbrugergrænseflade kunne se ud



Figur 3: Skitse af brugergrænsefladen

2 Accepttestspezifikation

2.1 Use case 1 - Hovedscenarie

Step	Handling	Forventet observation/resultat	Faktisk observasjon/resultat	Vurdering (OK/FAIL)
1	Vælg single-player mode.	Brugergrænsefladen viser spilside for one-player mode og anmoder om, at der fyldes slik i magasinet og at kanon indstilles.		
3	Fyld slik i kanon. Indstil kanon til affyring med Wii-nunchuck.	Kanon indstiller sig svarende til Wii-nunchucks placering.		
4	Udløs kanon med trigger på wii-nunchuck.	Kanon udløses.		
5	Gentag punkt 4 og 5 to gange.	Punkt 4 og 5 gentages.		
6	Tryk på knap for at vende tilbage til starttilstand.	Brugergrænseflade vender tilbage til startside.		

2.1.1 Use case 1 - Extension 1

Step	Handling	Forventet observasjon/resultat	Faktisk observasjon/resultat	Vurdering (OK/FAIL)
1	Vælg party mode.	Brugergrænsefladen viser spilside for two-player mode og anmoder om valg af antal skud.		
2	Fyld slik i kanon. Indstil kanon til affyring med Wii-nunchuck.	Kanon indstiller sig svarende til Wii-nunchucks placering.		
3	Udløs kanon med trigger på Wii-nunchuck.	Kanon udløses.		
4	Giv Wii-nunchuck til den anden spiller.	Den anden spiller modtager Wii-nunchuck.		
5	Gentag punkt 5 til 6 indtil skud er opbrugt.	Punkt 5 til 6 gentages.		
6	Tryk på knap for at vende tilbage til starttilstand.	Brugergrænseflade vender tilbage til startside.		

2.1.2 Use case 1 - Extension 2

Step	Handling	Forventet observasjon/resultat	Faktisk observasjon/resultat	Vurdering (OK/FAIL)
1	Vælg single-player mode.	Brugergrænsefladen viser spilside for one-player mode og anmoder om, at kanon indstilles.		
2	Tryk på knap for afslutning af spil.	Brugergrænseflade vender tilbage til startside.		

2.2 Use case 2 - Hovedscenarie

Step	Handling	Forventet observation/resultat	Faktisk observation/resultat	Vurdering (OK/FAIL)
1.	Tryk på "Systemtest" på GUI	Systemtest brugergrænsefladen vises på Devkit 8000.		
2.	Tryk på "Start Test" på GUI"	Brugergrænsefladen udskriver at SPI og I2C testen er godkendt. Brugergrænsefladen anmoder brugereren om tryk på Z på Wii-nunchuck		
3.	Tryk 'Z' knappen på Wii-nunchucken	Brugergrænsefladen udskriver at Wii-testen er godkendt		

2.2.1 Use case 2 - Exception 1

Step	Handling	Forventet observation/resultat	Faktisk observation/resultat	Vurdering (OK/FAIL)
1.	Fjern SPI-kablet fra Devkit 8000			
2.	Tryk på "Systemtest" på GUI	Systemtest brugergrænsefladen vises på Devkit 8000.		
3.	Tryk på "Start Test" på GUI	Brugergrænsefladen udskriver at SPI forbindelsen mislykkedes		

2.2.2 Use case 2 - Exception 2

Step	Handling	Forventet observation/resultat	Faktisk observation/resultat	Vurdering (OK/FAIL)
1.	Fjern I2C-kabel fra PSoC0			
2.	Tryk på "Systemtest" på GUI	Systemtest brugergrænsefladen vises på Devkit 8000.		
3.	Tryk på "Start Test" på GUI	Brugergrænsefladen udskriver at SPI forbindelsen lykkedes og I2C forbindelsen mislykkedes		

2.2.3 Use case 2 - Exception 3

Step	Handling	Forventet observation/resultat	Faktisk observation/resultat	Vurdering (OK/FAIL)
1.	Tryk på "Systemtest" på GUI	Systemtest brugergrænsefladen vises på Devkit 8000.		
2.	Tryk på "Start Test" på GUI	Brugergrænsefladen udskriver at SPI og I2C forbindelsen lykkedes		
3.	Afvent timeout	Brugergrænsefladen udskriver at Nunchuck forbindelsen mislykkedes		

2.3 Ikke-funktionelle krav

Krav	Test	Forventet observation/resultat	Faktisk observation/resultat	Vurdering (OK/FAIL)
1	Bruger drejer kanonen så lang til venstre og højre som muligt.	Det observeres at kanonen ikke har roteret 360 °		
2	Et projektil på 1.25 cm i diameter \pm 5mm affyres fra kanonen.	Projektilet bliver affyret		
3	Mål produktet dimensioner med en lineal.	Dimensionerne overstiger ikke 50cm x 60cm x 50cm.		
4	Tryk på 'Z' knappen på Wii Nunchuck, og mål med et stopur hvor lang tid der går fra tryk, til kanonen bliver affyret.	Den målte tid er mindre end 10 sekunder.		
5	Kanonen affyres 3 gange, og et stopur startes ved første skud, og stoppes ved det tredje skud.	Den målte tid er mindre end 60 sekunder.		

3 Systemarkitektur

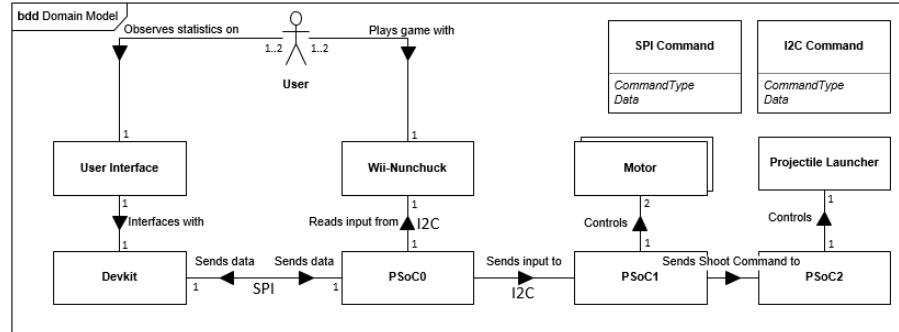
Systemarkitekturen har til formål at beskrive både hardware- og softwarekomponenter til sådan en grad at sammensætningen af hele systemet kan forstås. I dette afsnit beskrives arkitektur for både hardware og software.

I hardwarearkitekturen beskrives systemet ved at nedbryde det i blokke. I BDD'et er blokkenes porte og associationer til andre blokke angivet. Disse porte anvendes senere i afsnittet, hvor de benyttes i IBD'et.

I softwarearkitekturen udarbejdes der applikationsmodeller bestående af sekvensdiagrammer og klassediagrammer for hver use case, opdelt for hver CPU i systemet. Applikationsmodellerne har til formål at danne et overblik af de krav der stilles til softwaren for produktets uses cases. På denne måde kan de bruges som inspirerende grundlæg for software design og implementering. Applikationsmodellerne giver en overfladisk beskrivelse af CPU'ernes interaktioner.

3.1 Domænemodel

På figur 4 ses domænemodellen for systemet. Denne er lavet for at danne et overblik over systemet, og hvordan de forskellige dele overfladisk interagerer med hinanden.



Figur 4: Domænemodel for Candygun 3000

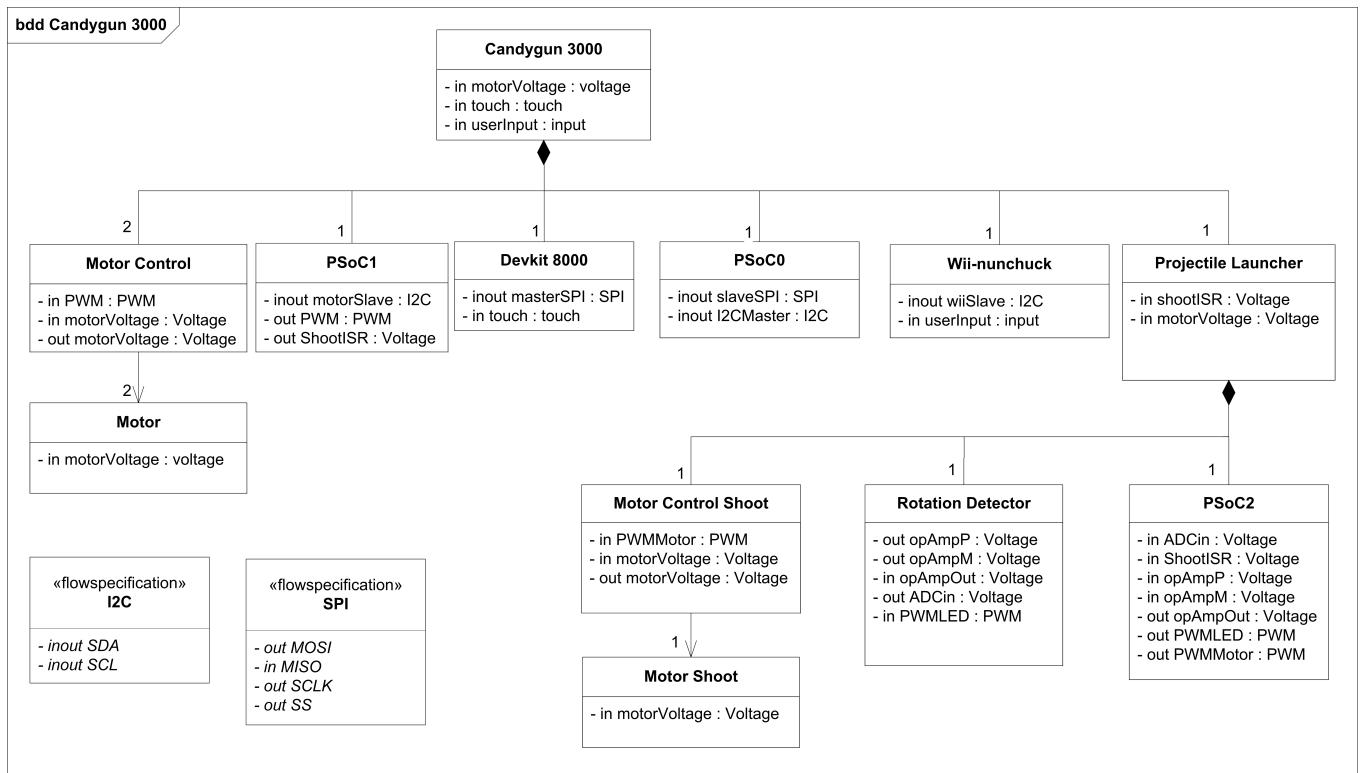
I domænemodellen ses, at brugeren interagerer med både Wii-nunchucken[9] [10] og med brugergrænsefladen. Brugergrænsefladen er en grænseflade til softwaren devkittet. Devkit 8000 kommunikerer med PSoC0, som læser den analoge data der kommer fra Wii-nunchucken. Denne data bliver derefter afkodet og videresendt til PSoC1, som styrer motorene ud fra dataene. PSoC1 sender også et interrupt til PSoC2, når 'Z' knappen på nunchucken bliver trykket. Når PSoC2 modtager interruptet, starter motoren for affyringsmekanismen i projectile launcher.

I "I2C Command"ses en grov estimering af interfacet mellem de forskellige I2C [4][8] enheder i systemet. CommandType bruges til at identificere hvilken data der bliver sendt. Data er en mængde tal-værdier der bliver tolket ud fra hvilken CommandType der er blevet sendt.

I "SPI Command" ses en estimering af interfacet mellem SPI [5] enhederne i systemet. "Commandtype" bruges til at identificere hvilken opgave systemet skal udføre (f.eks. 'I2C-test'). Data kan indeholde tal-værdier, som repræsenterer nunchuck værdier (buttonpress osv.). En mere detaljeret gennemgang af disse kommunikations protokoller kan findes i afsnit 3.7

3.2 BDD for Candygun 3000

På figur 5 ses et *Block Definition Diagram (BDD)*. Figur 5 skal give et overblik over hvordan relationerne mellem de forskellige hardwareblokke er. Så man får et overordnet billede af hvordan strukturen er for Candygun 3000. For at læse hvad hver blok skal bruges til, henvises der til Blokbeskrivelsen, tabel 3.2.1. På BDD'et figur 5 ses nødvendige indgange og udgange for de fysiske signaler. Yderligere ses det at flow specificationer er defineret for de ikke-atomare forbindelser I2C samt SPI, da disse busser består af flere forbindelser. Der henvises til afsnit 3.3 for en detaljeret model af de fysiske forbindelser mellem hardwareblokkene.



Figur 5: BDD for Candygun 3000

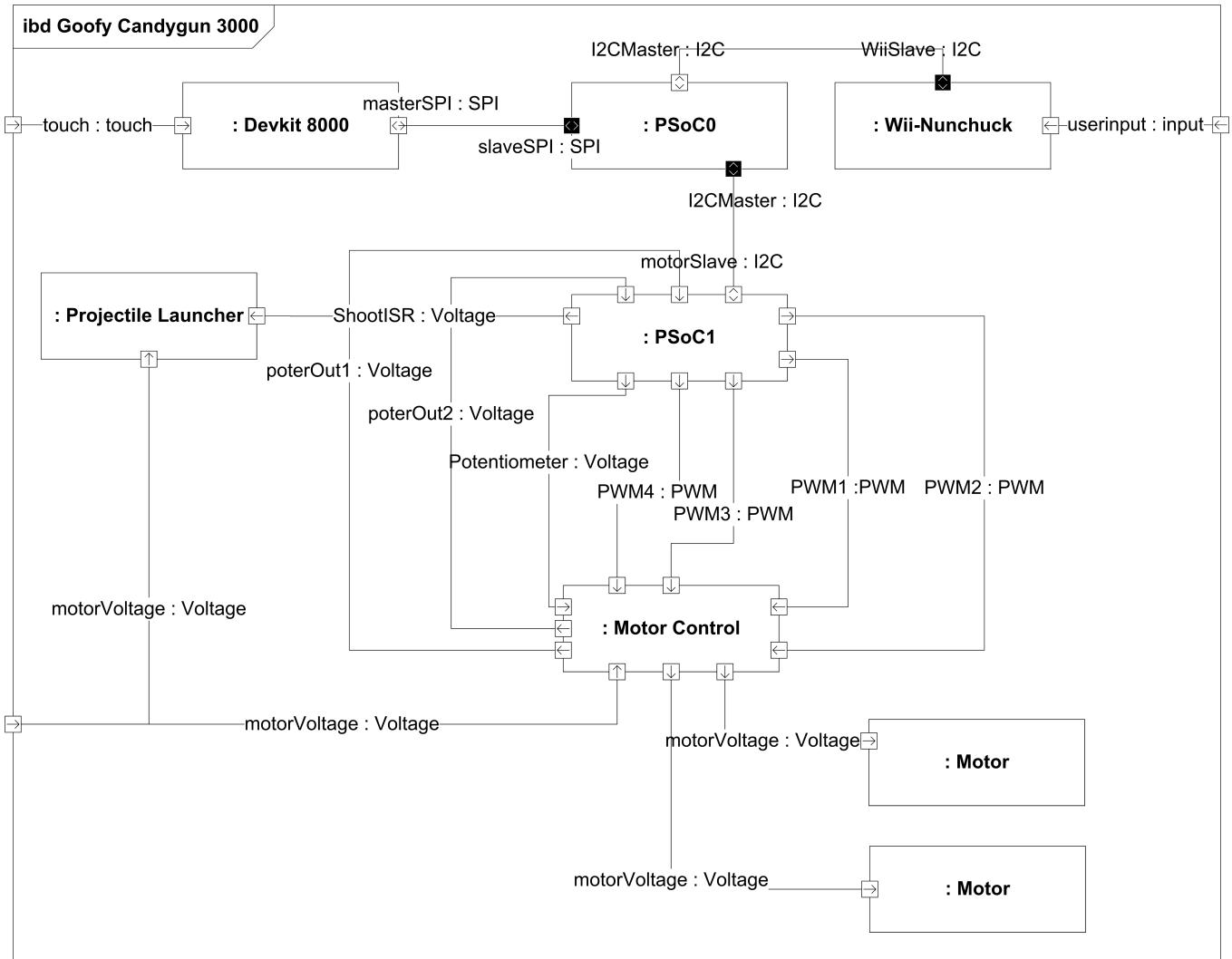
3.2.1 Blokbeskrivelse

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 master og SPI slave. Denne PSoC er bindeleddet mellem brugergrænsefladen og resten af systemets hardware.
MotorControl	MotorControl blokken er Candy Gun 3000's motorerer, der anvendes til at bevæge kanonen. Denne blok består af H-bro blokken og rotationsbegrænsnings blokken.
Motor	Motor er motorene der bruges til at bevæge platform og kanon.
H-bro	H-bro bruges til at styre mototrens rotationsretning
Rotationsbegrænsning	Rotationsbegrænsning er til at begrænse platformens rotation så denne ikke kan dreje 360 grader. Den blok består af et potentiometer og en ADC'en, som sidder internt på PSoC0
PSoC1	PSoC1 er PSoC hardware der indeholder software til I2C kommunikation og styring af Candy Gun 3000's motorer. PSoC1 fungerer som I2C slave.
SPI (FlowSpecification)	SPI (FlowSpecification) beskriver signalerne der indgår i SPI kommunikation.
I2C (FlowSpecification)	I2C (FlowSpecification) beskriver signalerne der indgår i I2C kommunikation.
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 rotationsdetektorkredsløbet, 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	Rotationsdetektoren detekterer, at der er blevet affyret et stykke slik og sender et signal til PSoC2 om dette.

Tabel 2: Blokbeskrivelse

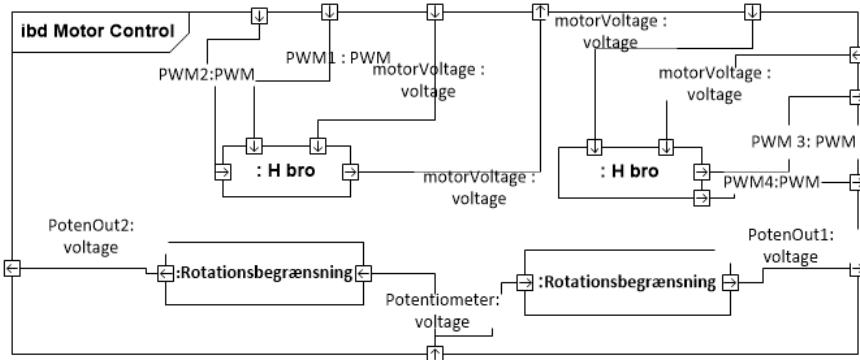
3.3 IBD for Candygun 3000

På baggrund af BDD'et er der lavet et *Internal Block Diagram (IBD)*. I IBD'et på figur 6 ses forbindelserne og portene mellem systemets blokke. Diagrammet viser grænsefladerne mellem blokkene og flowet i mellem disse.



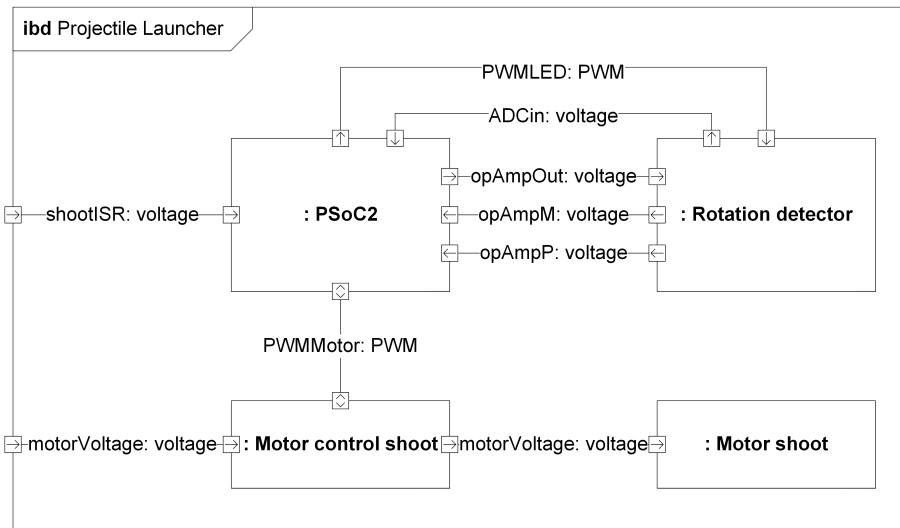
Figur 6: IBD for Candygun 3000

I IBD'et på figur 7 ses forbindelserne og portene mellem blokkene i motorControl blokken. Diagrammet viser grænsefladerne mellem disse blokke og flowet i mellem disse.



Figur 7: IBD for motor control

På figur 8 ses et IBD, der giver et overblik over affyringsmekanismens indgående og udgående signaler og forbindelserne mellem de interne blokke, som udgør affyringsmekanismen.



Figur 8: IBD over affyringsmekanismen

I tabel 3 ses en beskrivelse af alle signaler, der indgår i de tre IBD'er, som ses på figurerne 6, 7 og 8.

3.3.1 Signalbeskrivelse

I signalbeskrivelsen gælder det, at når et signal beskrives som 'højt' opereres der i et spændingsområde på 3,5V til 5V. På samme måde er signaler beskrevet som 'lav' defineret som spændinger indenfor et område fra 0V til 1,5V. Disse spændingsniveauer er defineret ud fra standarden for CMOS kredse [7].

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 hvor der sendes og modtages data.
		touch	Type: touch Beskrivelse: Brugertyk 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 hvor der sendes og modtages data.
		wiiMaster	Type: I2C Spændingsniveau: 0-5V Hastighed: 100Kpbs Beskrivelse: I2C bussen hvor der modtages data fra Nunchuck.
		motorMaster	Type: I2C Spændingsniveau: 0-5V Hastighed: 100kbit/sekund Beskrivelse: I2C bussen hvor der sendes afkodet Nunchuck data til PSoC1.

PSoC1	Modtager nunchuck-input fra PSoC0 og omsætter dataene til PWM signaler.	motorSlave	Type: I2C Spændingsniveau: 0-5V Hastighed: 100kbit/sekund Beskrivelse: Indeholder formatteret Wii-Nunchuck data som omsættes til PWM-signal.
		ShootISR	Type: voltage Spændingsniveau: 0-5V Beskrivelse: giv et højt signal når den skal skyde.
		PWM	Type: PWM Frekvens: 22kHz PWM %: 0-100% Spændingsniveau: 0-5V Beskrivelse: PWM signal til styring af motorens hastighed.
		PotenOut1	Type: voltage Spændingsniveau: en spænding 0V-5V alt efter hvad potentiometer står på Beskrivelse: den spænding viser hvor motoren står henne
		PotenOut2	Type: voltage Spændingsniveau: en spænding 0V-5V alt efter hvad potentiometer står på Beskrivelse: den spænding viser hvor motoren står henne
		PWM1	Type: PWM Frekvens: 3MHz PWM%: 0-100% Spændingsniveau: 0-5V Beskrivelse: PWM signal til styring af motorens hastighed.

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

		PotenOut2	Type: voltage Spændingsniveau: en spænding 0V-5V alt efter hvad potentiometer står på Beskrivelse: den spænding viser hvor motoren står henne
Motor	Denne blok beskriver, hvad motoren får.	Motorvoltage	Type: voltage Spændingsniveau: 0-5V Beskrivelse: giver spændning til motoren.(denne beskedelse glæder også for den anden motor)
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 Beskrivelse: Binært data der sendes fra master til slave.
		MISO	Type: CMOS Spændingsniveau: 0-5V Beskrivelse: Binært data der sendes fra slave til master.

		SCLK	Type: CMOS Spændingsniveau: 0-5V Hastighed: 1Mbps Beskrivelse: Clock signalet fra master til slave, som bruges til at synkronisere den serielle kommunikation.
		SS	Type: CMOS Spændingsniveau: 0-5V Beskrivelse: Slave-Select, som bruges til at bestemme hvilken slave der skal kommunikeres med.
I2C	Denne blok beskriver den ikke-atomiske I2C forbindelse.	SDA	Type: CMOS Spændingsniveau: 0-5V Beskrivelse: Databussen mellem I2C masteren og I2C slaver.
		SCL	Type: CMOS Spændingsniveau: 0-5V Hastighed: 100kbps Beskrivelse: Clock signalet fra master til lyttende I2C slaver, som bruges til at synkronisere den serielle kommunikation.
Projectile Launcher	Denne blok består blokkene PSoC2, Detector og Motor Control Shoot	shootISR	Type: Voltage Spændingsniveau: 0-5V Beskrivelse: Giv 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.

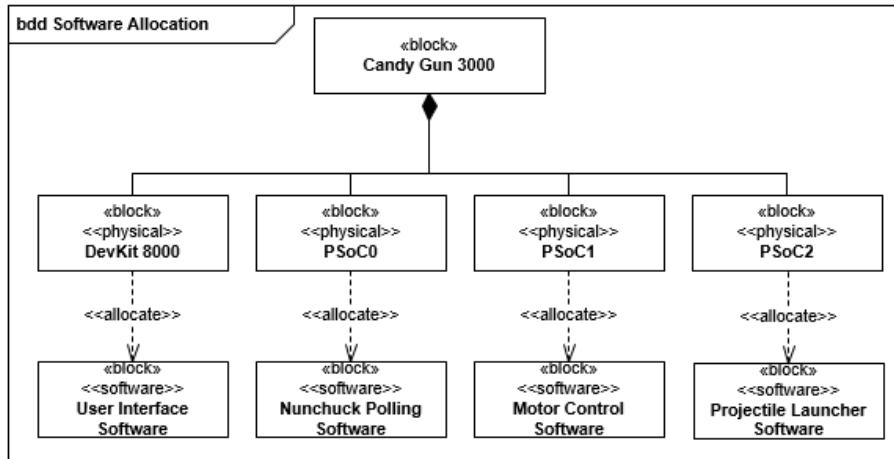
		PWMmotor	Type: PWM Spændingsniveau: 0-5V Frekvens: 33,33kHz PWM%: 0-100% Beskrivelse: Udsender PWM-signal til LED'en i rotationsdetektorkredsløbet.
Detector	Rotationsdetektoren detekterer om der er skudt.	opAmpP	Type: Voltage Spændingsniveau: 0,5V Beskrivelse: Referencespænding til operationsforstærkerens 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 affyrmekanismen.	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 affyrmekanismens motor.	motorVoltage	Type: Voltage Spændingsniveau: 9V Beskrivelse: Strømforsyning til motor.

Tabel 3: Signalbeskrivelse

3.4 Software Allokéringsdiagram

På figur 9 ses et software allokerings diagram. Dette diagram danner et overblik over hvilke CPU'er der findes i systemet, og hvilken software der skal allokeres på disse. Følgende applikationsmodeller tager udgangspunkt i softwaren der allokeres i figur 9.



Figur 9: Software allokations diagram

På tabel 4 er hvert allokeret software komponent beskrevet.

User Interface Software	Dette allokerede software er brugergrænsefladen som brugeren interagerer med på Devkit 8000 touch-skærmen.
Nunchuck Polling Software	Dette allokerede software har til ansvar at polle Nunchuck tilstanden og videresende det til PSoC1.
Motor Control Software	Dette allokerede software har til ansvar at bruge den pollede Nunchuck data fra PSoC0 til motorstyring samt affyringsmekanismen.
Projectile Launcher Software	Dette allokerede software har til ansvar at aktivere affyringsmekanismen når et knaptryk detekteres på Nunchuck.

Tabel 4: Beskrivelse af den allokerede software

3.5 Applikationsmodeller

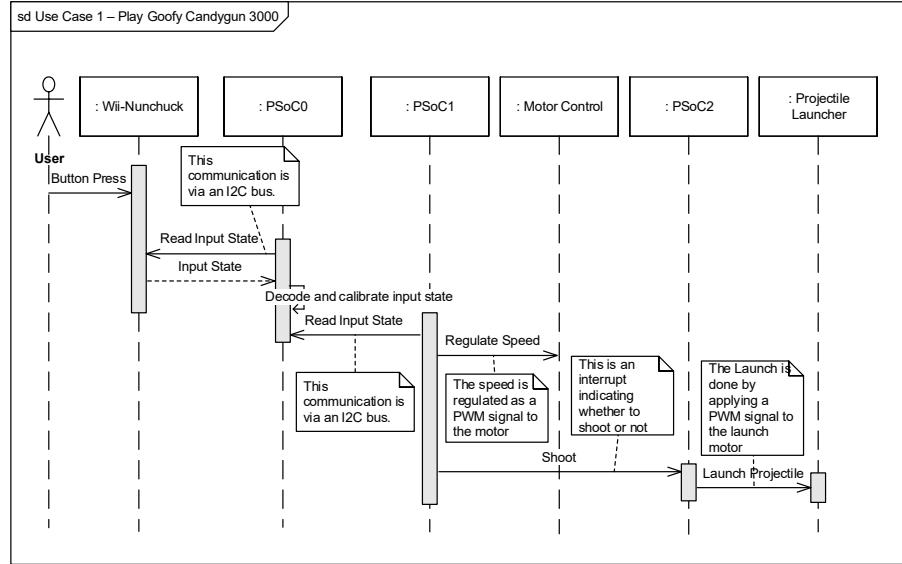
Til arkitekturfasen er der gjort brug af applikationsmodeller som et analyseværktøj. Der er lavet en applikationsmodel for hver use case, og for hvert software komponent identificeret i Software Allokéringsdiagrammet, tabel 4. Hver applikationsmodel består af et *Klasse Identifikations Diagram*, *Sekvensdiagram* og *Klassediagram*. Resultatet af hver applikationsmodel er et klassediagram der indeholder konceptuelle klasser med metoder relevant for den pågældende use case.

Med konceptuelle klasser menes der at de ikke nødvendigvis behøver at have et 1:1 forhold med de implementerede klasser. De konceptuelle klasser bruges som inspiration til de klasser og funktioner der muligvis skal modelleres i projektets problemdomæne.

3.5.1 use case 1 - Spil Goofy Candy Gun 3000

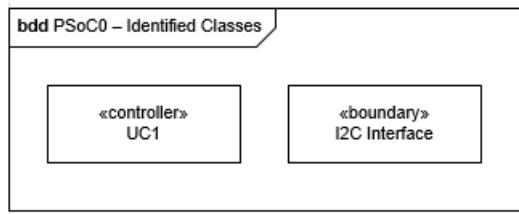
Følgende afsnit præsenterer applikationsmodeller relevante til use case 1 - Spil Goofy Candy Gun 3000, fordelt over systemets CPU'er.

På figur 10 ses et overordnet sekvensdiagram for hvordan Nunchuck data bliver overført igennem systemet og påvirker motorer samt affyringsmekanismen.



Figur 10: Overordnet sekvensdiagram for Wii-Nunchuck informations flow

Applikationsmodel for Nunchuck Polling Software



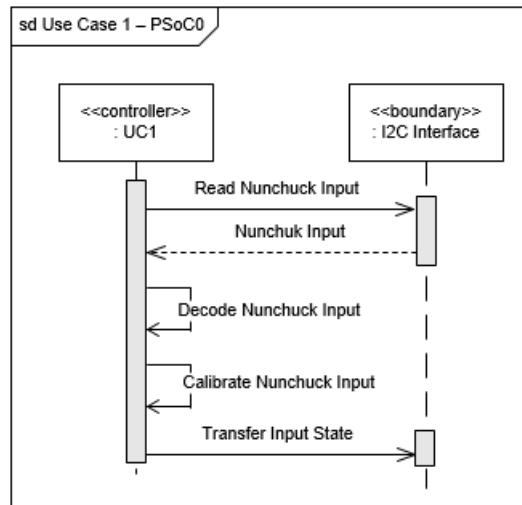
Figur 11: Klasseidentifikation for PSoC0

På figur 11 ses klasse identifikation for Nunchuck Polling Software. Der er til Nunchuck Polling Software for use case 1 identificeret to klasser: *UC1* samt *I2C Interface*.

Klassen *UC1* er en controller. Dette er klassen der har til ansvar at eksekvere funktionalitet relevant til use case 1. *I2C Interface* er en boundary klasse der repræsenterer grænseflade til I2C bussen.

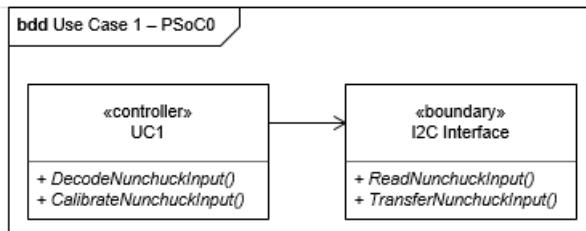
Det kan på sekvensdiagrammet, figur 12, ses hvordan klasserne interagerer med hinanden. Her kan det ses at klassen *UC1* aflæser Nunchuck input og herefter

dekoder det, kalibrerer det, og sender det videre ud på I2C bussen hvor andre slaver kan gøre brug af dataen.



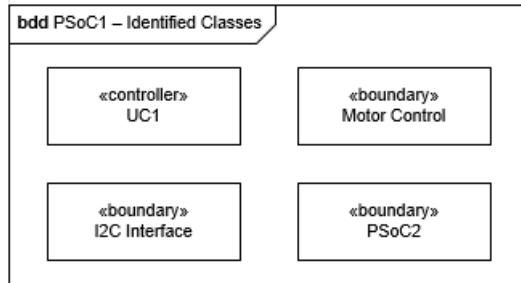
Figur 12: Sekvensdiagram for PSoC0

Figur 13 viser et endeligt klassediagram, hvor metoderne er udledt fra sekvensdiagrammet, figur 12. Her er det identificeret at controller klassen *UC1* har to metoder til at kunne dekode og kalibrere Nunchuck data. *UC1* gør brug af *I2C Interface* som grænseflade til at sende dataen ud på I2C bussen.



Figur 13: Klassediagram for PSoC0

Applikationsmodel for Motor Control Software

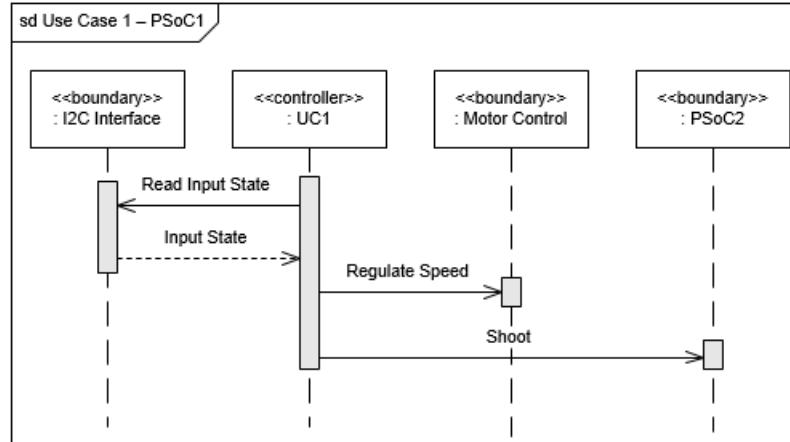


Figur 14: Klasseidentifikation for PSoC1

På figur 14 ses klasse identifikation for Motor Control Software. Der er til Motor Control Software for use case 1 identificeret fire klasser: *UC1*, *Motor Control*, *I2C Interface* og *PSoC2*.

UC1 er en controller. Denne klasse har til ansvar at eksekvere funktionalitet relevant til use case 1. *Motor Control* repræsenterer grænsefladen til motorstyringen. *I2C Interface* repræsenterer grænsefladen til I2C bussen. *PSoC2* repræsenterer en grænseflade til PSoC2.

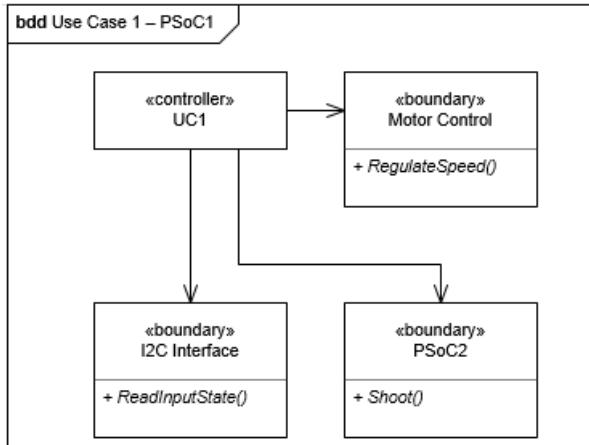
På figur 15 ses et sekvensdiagram af interaktionen mellem disse klasser. Her kan det ses at *UC1* klassen læser input data fra Nunchuck og videresender det til motor styringen samt *PSoC2*, så farten af motoren kan reguleres og affyringsmekanismen kan aktiveres.



Figur 15: Sekvensdiagram for PSoC1

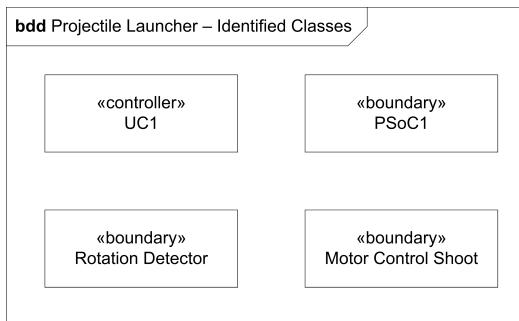
På figur 16 ses et endeligt klassediagram, hvor metoderne er udledt fra sekvensdiagrammet, figur 15. Her er det identificeret at controller klassen *UC1* har primært kommunikerer med grænsefladerne *I2C Interface*, *PSoC2* og *Motor*

Control. Her skal *UC1* kunne læse Nunchuck input fra *I2C Interface*. Yderligere skal den kunne regulere farten på *Motor Control* og affyre affyringsmekanismen.



Figur 16: Klassediagram for PSoC1

Aplikationsmodel for Projectile Launcher Software

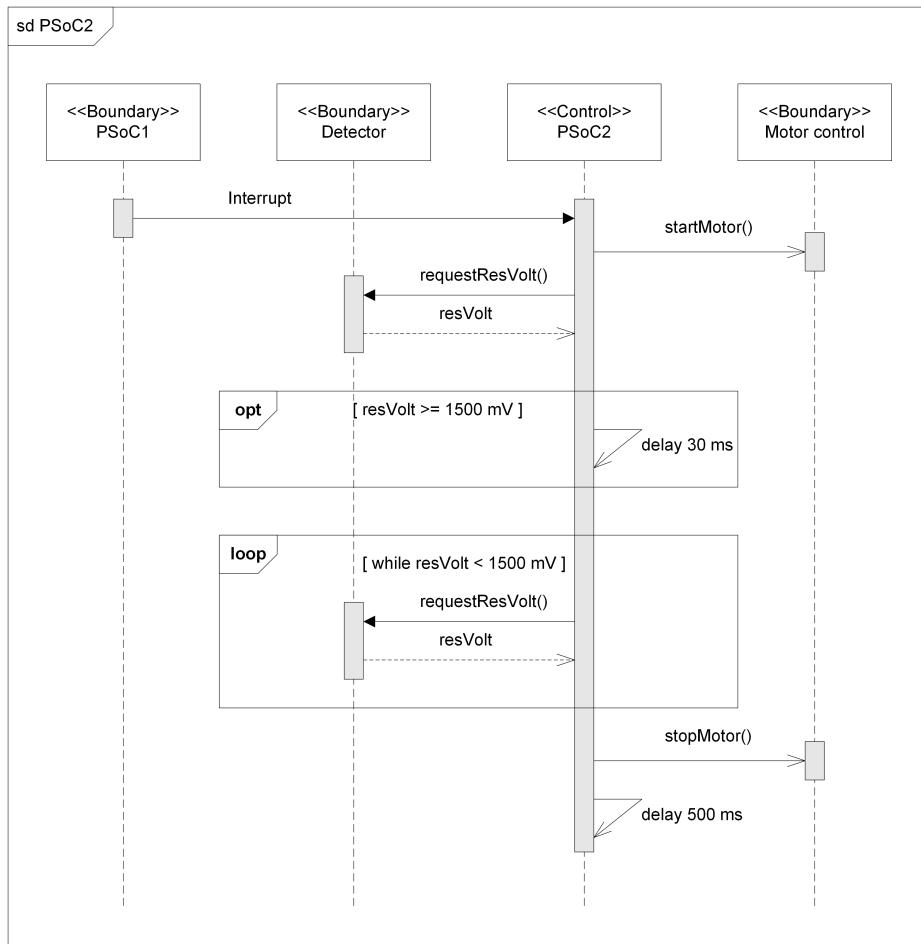


Figur 17: Klasseidentifikation for affyringsmekanismen

På figur 17 ses klasse identifikation for Projectile Launcher Software. Der er til Projectile Launcher Software for use case 1 identificeret tre klasser: *UC1*, *PSOC1* og *Projectile Launcher*.

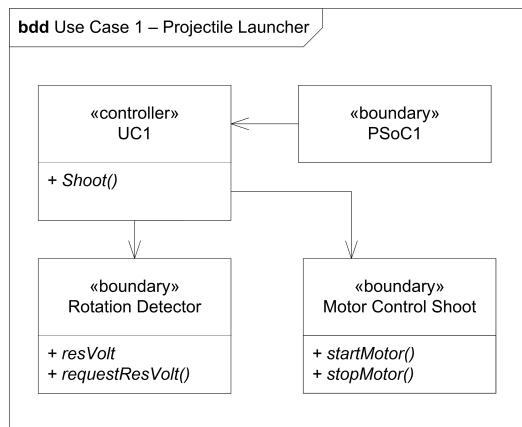
UC1 er en controller. Denne klasse har til ansvar at eksekvere funktionaliteten relevant til use case 1. *PSOC1* repræsenterer en grænseflade til PSOC1. *Projectile Launcher* repræsenterer en grænseflade til affyringsmekanismen.

På figur 18 ses et sekvensdiagram der viser klassernes interaktion. Her kan det ses at PSOC1 sender et signal til controlleren når der skal affyres et skud. Når dette signal er modtaget, bliver et aktiverings signal sendt ud til affyringsmekanismen.



Figur 18: Sekvensdiagram for PSoC2

På figur 19 ses det endelige klassediagram, hvor metoderne er udledt fra sekvensdiagrammet, figur 18. Det kan her ses *PSoC1* grænsefladen skal kunne sende en affyringsbesked til *UC1*. *UC1* skal derefter kunne aktivere affyringsmekanismen.

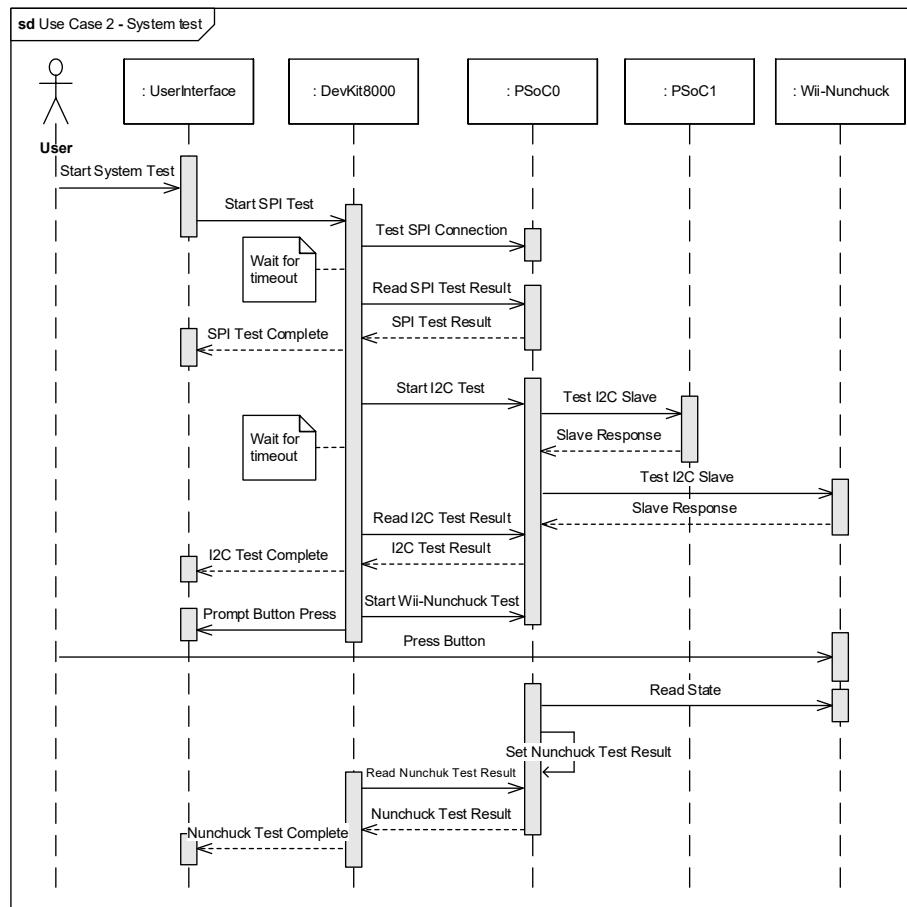


Figur 19: Klassediagramm für PSoC2

3.5.2 use case 2 - Test Kommunikationsprotokoller

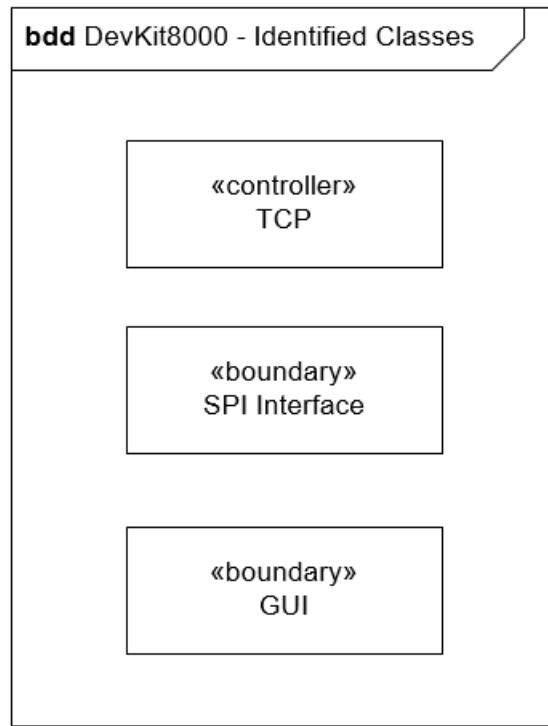
Følgende afsnit præsenterer applikationsmodeller relevante til use case 2 - Test Kommunikationsprotokoller, fordelt over systemets CPU'er.

På figur 20 ses et overordnet sekvensdiagram for use casen. Her starter brugeren testen gennem brugergrænsefladen. Først bliver SPI bussen mellem Devkit 8000 og PSoC0 testet. Herefter bliver I2C bussen testet ved at PSoC0 undersøger om de nødvendige I2C slaver (PSoC1 og Wii-Nunchuck) kan kommunikeres med. Til slut får brugeren en besked om at skulle trykke på en Wii-Nunchuck knap, hvorefter der bliver testet om knaptrykket skete eller ej.



Figur 20: Sekvensdiagram for use case 2 - Test Kommunikationsprotokoller

Applikationsmodel for User Interface Software

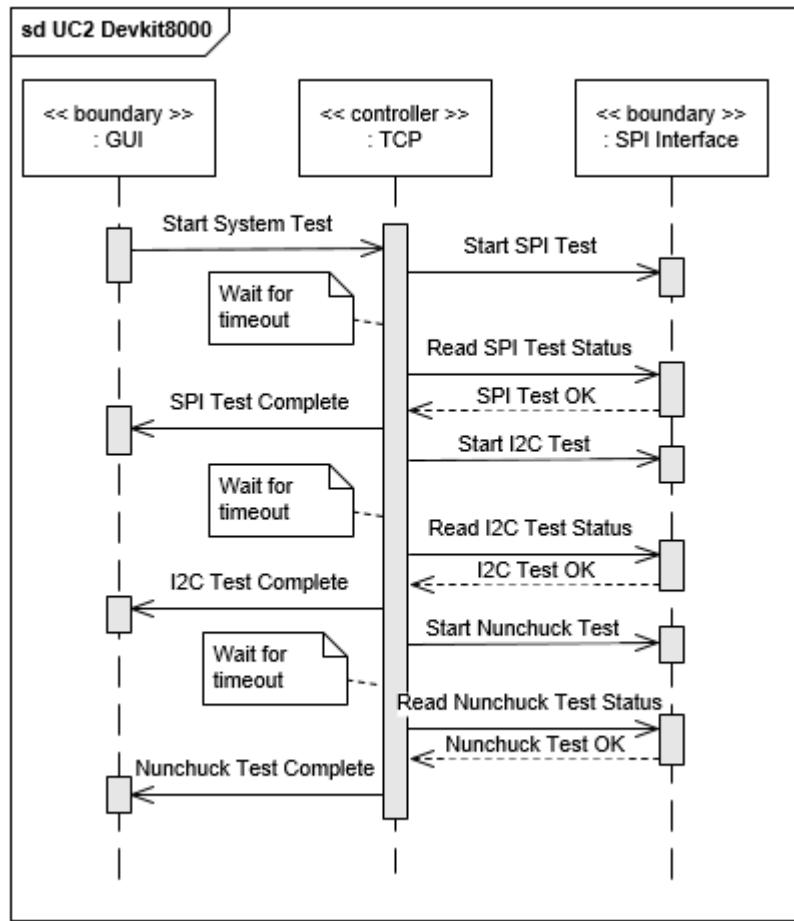


Figur 21: Klasseidentifikation for Devkit 8000

På figur 21 ses klasse identifikation for User Interface Software. Der er til User Interface Software for use case 2 identificeret tre klasser: *Test Communication Protocol (TCP)*, *SPI Interface*, og *GUI*.

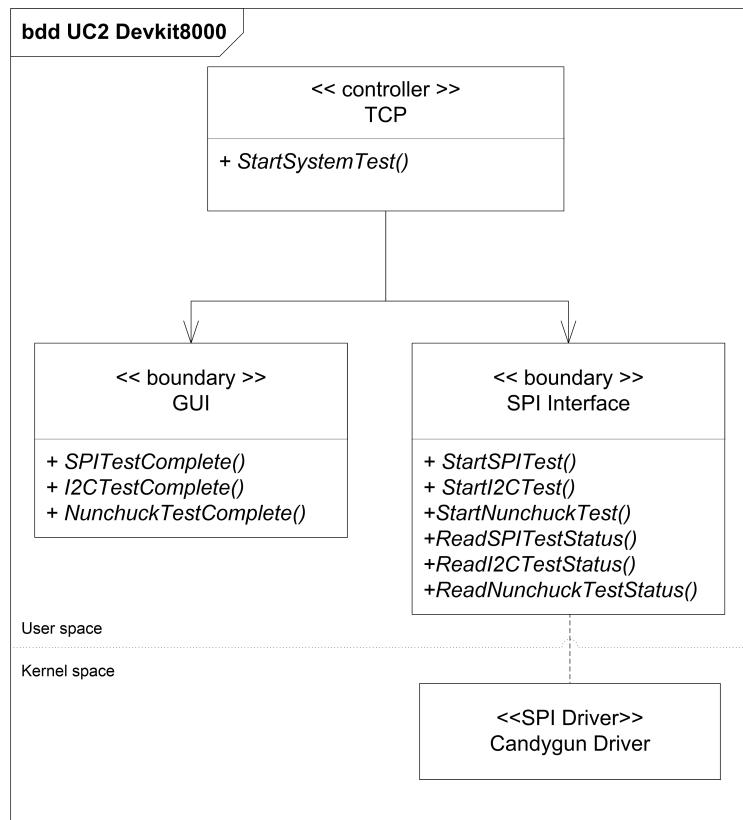
TCP er en controller. Denne klasse har til ansvar at eksekvere funktionalitet relevant til use case 2. *SPI Interface* repræsenterer en grænseflade til systemets SPI bus. *GUI* repræsenterer en grænseflade til systemets grænseflade.

På figur 22 ses et sekvensdiagram der viser klassernes interaktion. Som det fremkommer af diagrammet er det *TCP* klassen, der sørger for, at de forskellige tests bliver sat i gang ved at kommunikere med PSoC0 over SPI bussen via *SPI Interface* klassen. Når en test er færdiggjort meldes resultatet ud til brugeren via *GUI*'en.



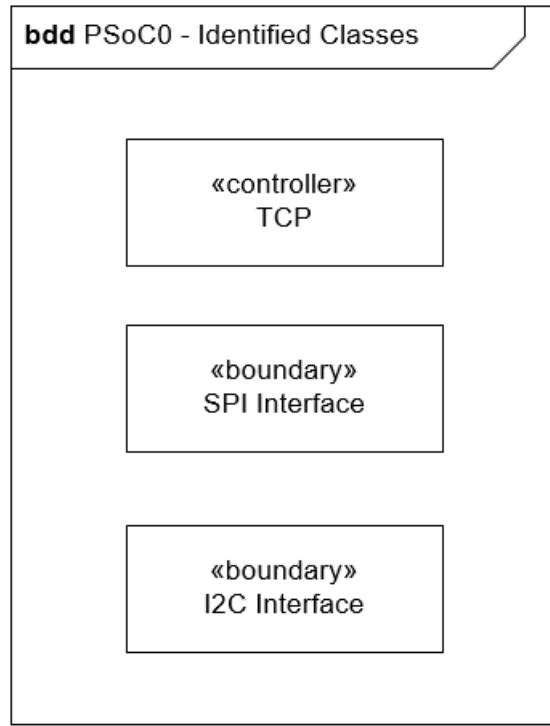
Figur 22: Sekvensdiagram for Devkit 8000

Ud fra sekvensdiagrammet, figur 22, er der udledt metoder til klasserne. De udledte metoder ses i klassediagrammet på figur 23. Her kan det ses at *GUI* grænsefladen skal kunne starte system testen på *TCP* klassen. *TCP* klassen skal kunne starte SPI-, I2C-, og Nunchucktest via *SPI Interface* grænsefladen. Yderligere skal *TCP* klassen også kunne notificere *GUI* grænsefladen når de test er færdiggjort.



Figur 23: Klassediagramm for Devkit 8000

Applikationsmodel for Nunchuck Polling Software

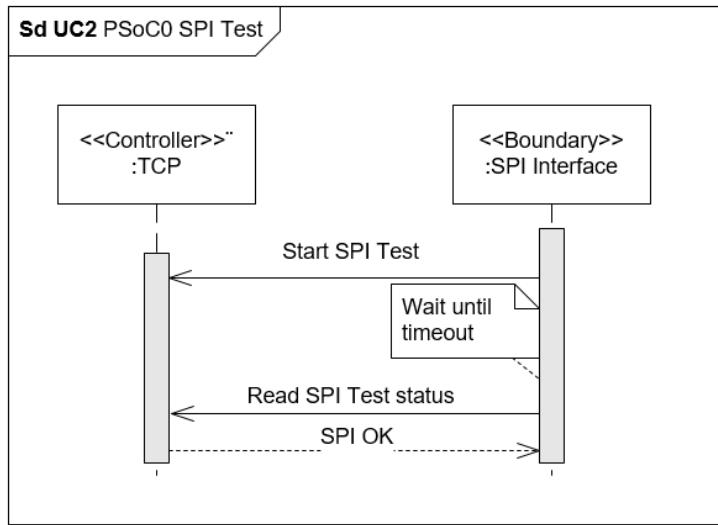


Figur 24: Klasseidentifikation for PSoC0

På figur 24 ses klasse identifikation for Nunchuck Polling Software. Der er til Nunchuck Polling Software for use case 2 identificeret tre klasser: *Test Communication Protocol (TCP)*, *SPI Interface*, og *I2C Interface*.

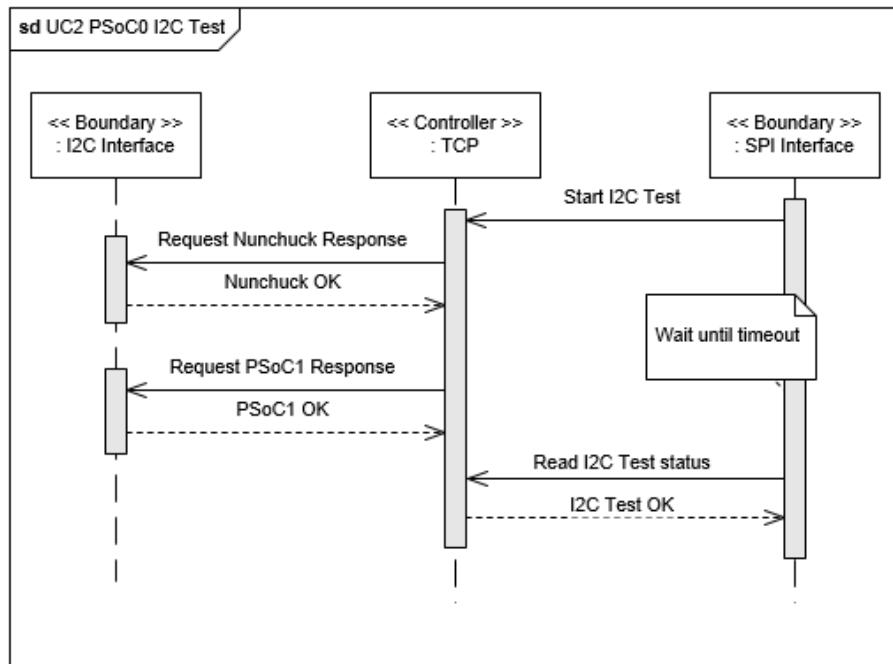
TCP er en controller. Denne klasse har til ansvar at eksekvere funktionalitet relevant til use case 2. *SPI Interface* repræsenterer SPI grænsefladen til systemets SPI bus. *I2C Interface* repræsenterer grænsefladen til systemets I2C bus.

På figur 25, 26 og 27 ses sekvensdiagrammer for Nunchuck Polling Software. Sekvensdiagrammerne er blevet opdelt i de 3 tests der gennemføres i use casen; SPI, I2C og Nunchuck kommunikations tests.



Figur 25: Sekvensdiagram for PSoC0 SPI test

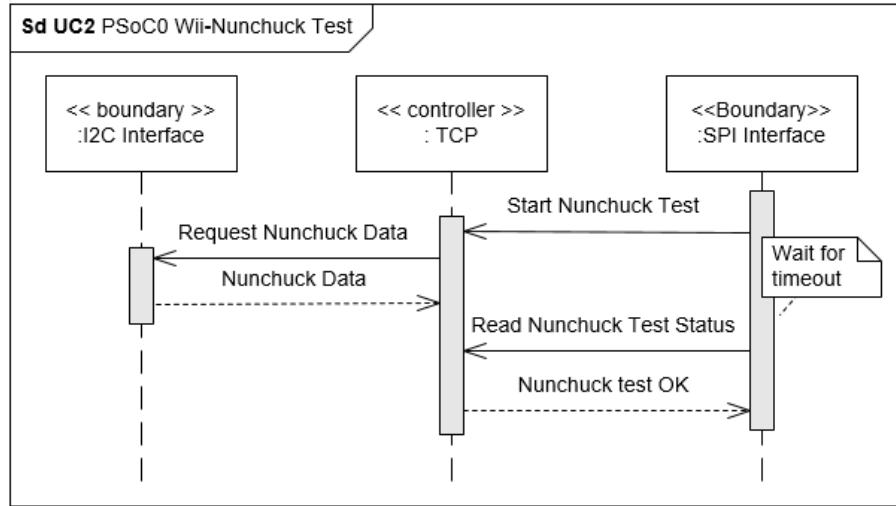
På figur 25 ses, at Devkit 8000 sender en besked til kontrolklassen via *SPI Interface* for at påbegynde SPI testen. Efter et bestemt timeout interval, læser Devkit 8000 resultatet af SPI testen, igen via *SPI Interface* grænsefladen.



Figur 26: Sekvensdiagram for PSoC0 I2C test

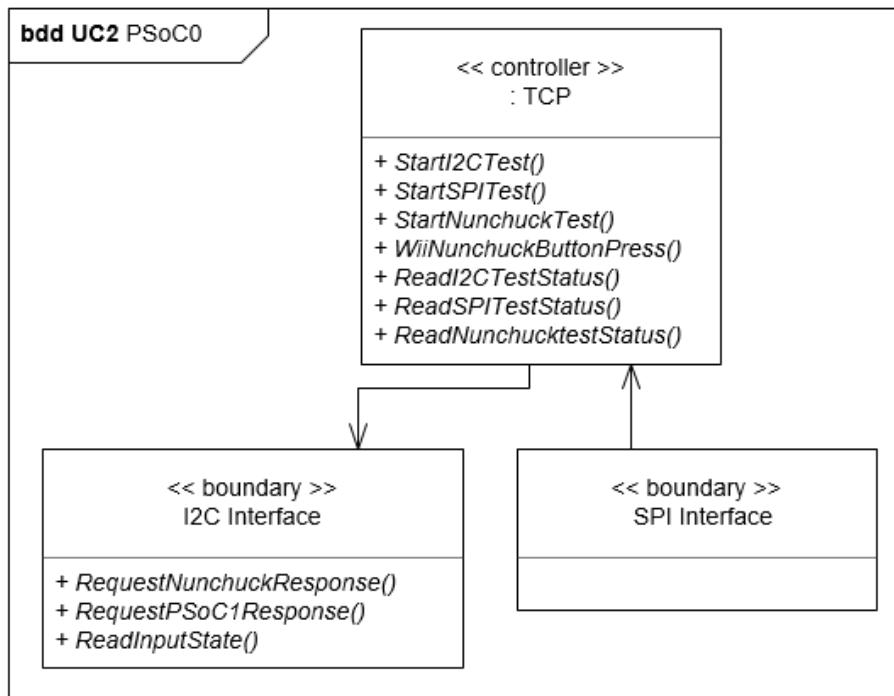
På figur 26 ses, at Devkit 8000 starter I2C testen, ved at sende en besked til *TCP* klassen. *TCP* klassen anmoder herefter om et acknowledge fra slaverne,

via *I2C Interface* grænsefladen, fra Nuchucken og PSoC1. Når disse enheder har svaret med et I2C OK er testen gennemført. Efter et bestemt timeout interval, læser Devkit 8000 resultatet af I2C testen, via *I2C Interface* grænsefladen.



Figur 27: Sekvensdiagram for PSoC0 Nunchuck test

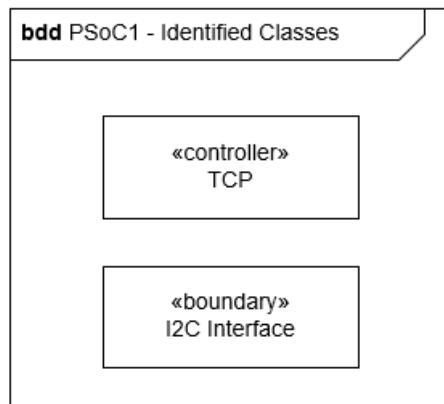
På figur 27 ses sekvensdiagrammet for Wii-Nunchuck testen. Devkit 8000 sender en *Start Nunchuck Test* besked til *TCP* klassen, hvorefter testen startes. *TCP* klassen anmoder om input data fra Nunchuck via *I2C Interface* grænsefladen, hvorefter devkittet efter et bestemt timeout aflæser test resultatet.



Figur 28: Klassediagram for PSoC0

På figur 28 ses det endelige klassediagram, hvor metoderne er udledt af sekvensdiagrammet, figur 27. Her kan det ses at *TCP* klassen skal kunne bede om tilbagemelding for I2C bussens slaver via *I2C Interface* grænsefladen. *SPI Interface* grænsefladen skal kunne starte de relevante tests for use casen, samt aflæse resultaterne.

Applikationsmodel for Motor Control Software

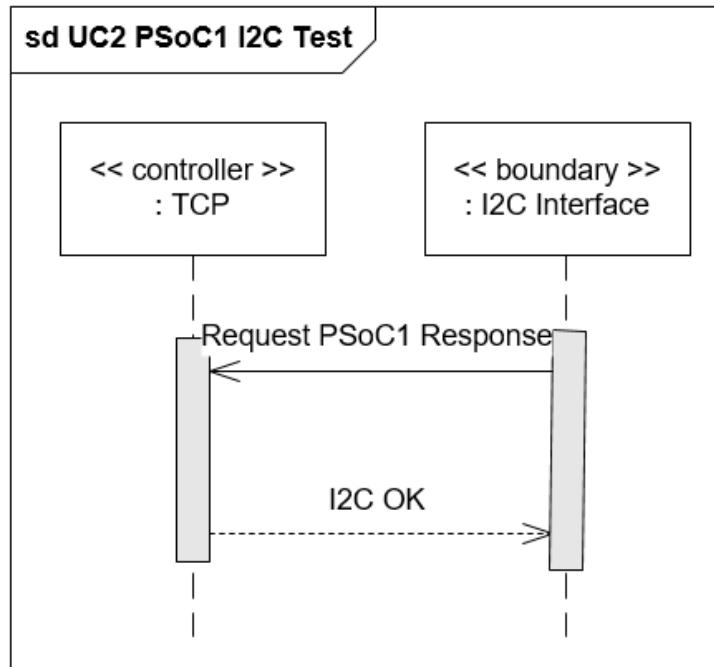


Figur 29: Klasseidentifikation for PSoC1

På figur 29 ses klasse identifikation for Motor control Software. Der er til Motor Control Software for use case 2 identificeret to klasser: *Test Communication Protocol (TCP)* og *I2C Interface*.

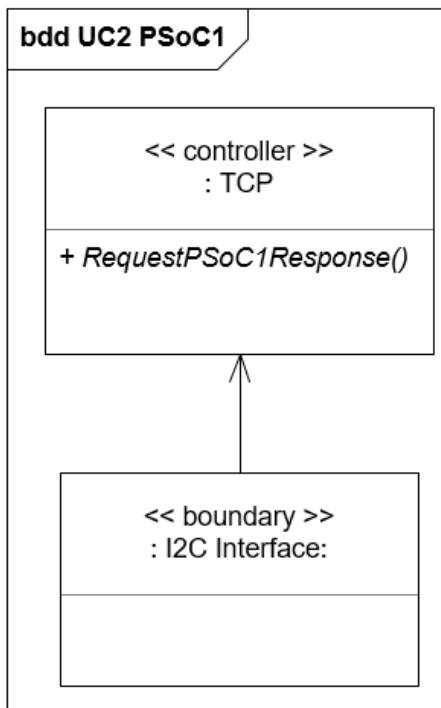
TCP er en controller. Denne klasse har til ansvar at eksekvere funktionalitet relevant til use case 2. *I2C Interface* repræsenterer grænsefladen til systemets I2C bus.

Sekvensdiagrammet for interaktion mellem de identificerede klasser ses på figur 30.



Figur 30: Sekvensdiagram for PSoC1

På figur 30 ses det at *I2C Interface* grænsefladen anmoder om respons fra *TCP* klassen for at bekräfte om I2C slaven kan kommunikeres med. Hvis dette er tilfældet sendes der I2C OK tilbage til boundaryklassen.



Figur 31: Klassediagram for PSoC1

Fra sekvensdiagrammet på figur 30 udledes det endelige klassediagram som ses i figur 31.

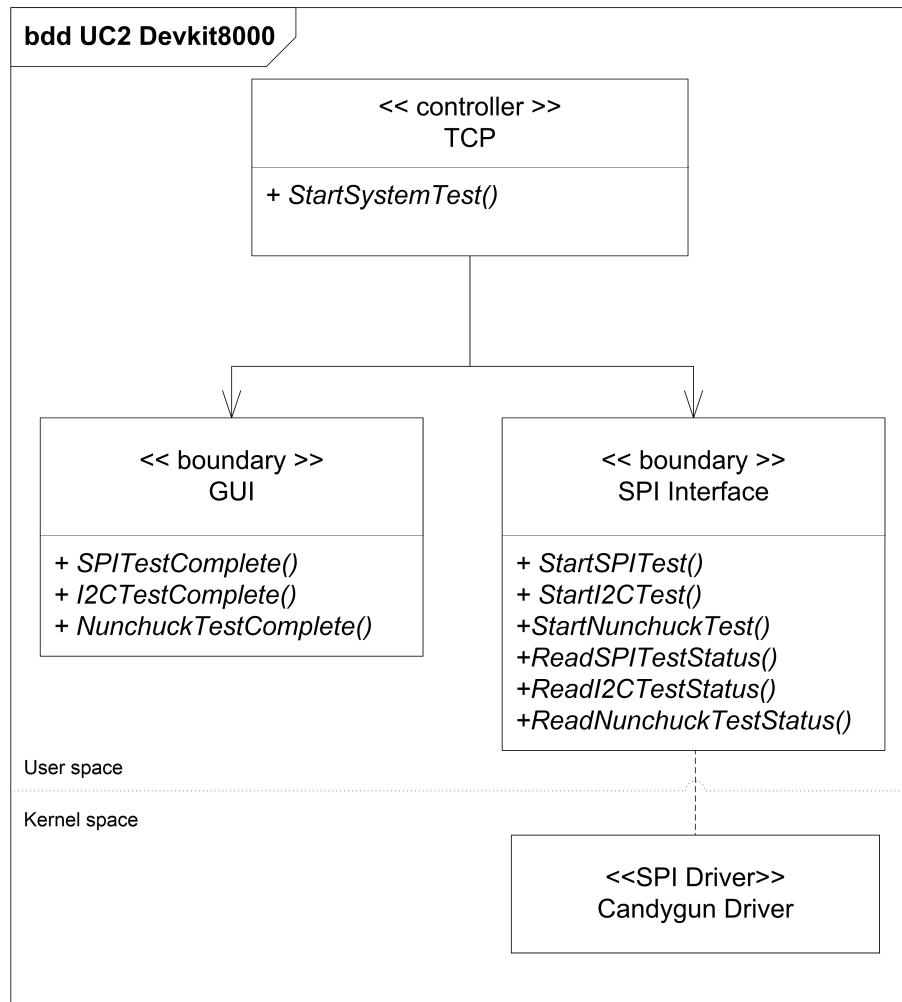
3.6 Samlede Klassediagrammer

Fra de detaljerede applikationsmodeller for use case 1- og 2, er der udledt samlede klassediagrammer for hver individuel CPU i systemet. Disse er med til at identificere konceptuelle klasser og funktionaliteter der skal overvejes i implementation og design af systemet, og har altså fungeret som et udgangspunkt til resten af udviklingsprocessen.

Figur 32, 33, 34, og 35 vises de samlede klassediagrammer for hver CPU. For disse klassediagrammer er der konceptuelle klasser som gentager sig selv. Alle klassediagrammer har én klasse af typen *controller*. Disse klasse indeholder alt funktionalitet der er nødvendig for at kunne implementere systemets use cases. Yderligere bliver klasserne *I2C Interface* samt *SPI Interface* gentaget. Disse repræsenterer klasser for de tilsvarende bustyper, I2C og SPI, og bruges af softwaren til at sende og modtage data på disse busser.

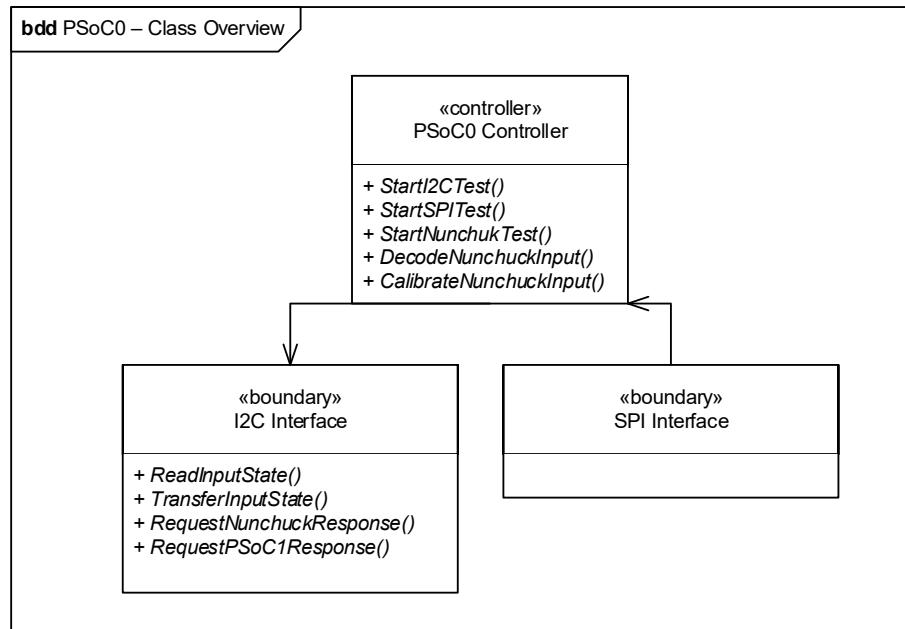
På figur 32 ses det at Devkit 8000 controlleren skal have funktionalitet til start af system test, i relation til use case 2. For at kunne udføre dette kommunikerer den med grænsefladen Graphical User Interface (*GUI*), for at kunne vise resultater

til brugeren. Desuden skal controlleren kommunikere med grænsefladen *SPI Interface*, for at sende data ud til resten af systemet via SPI bussen.



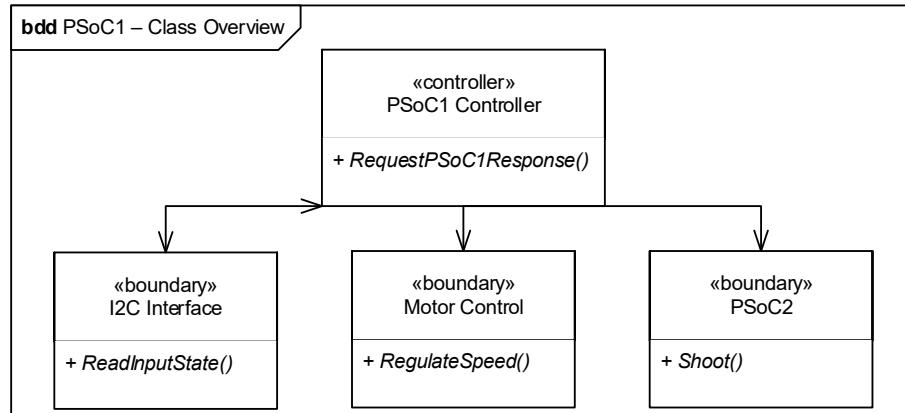
Figur 32: Samlet Klassediagram for Devkit 8000

På figur 33 ses det at PSoC0 controlleren skal have funktionalitet til at starte tests af systemets busser. Yderligere skal den også kunne dekode og kalibrere data der kommer fra Nunchuck. I relation til disse funktionaliteter skal den kunne modtage og sende data på systemets I2C og SPI busser.



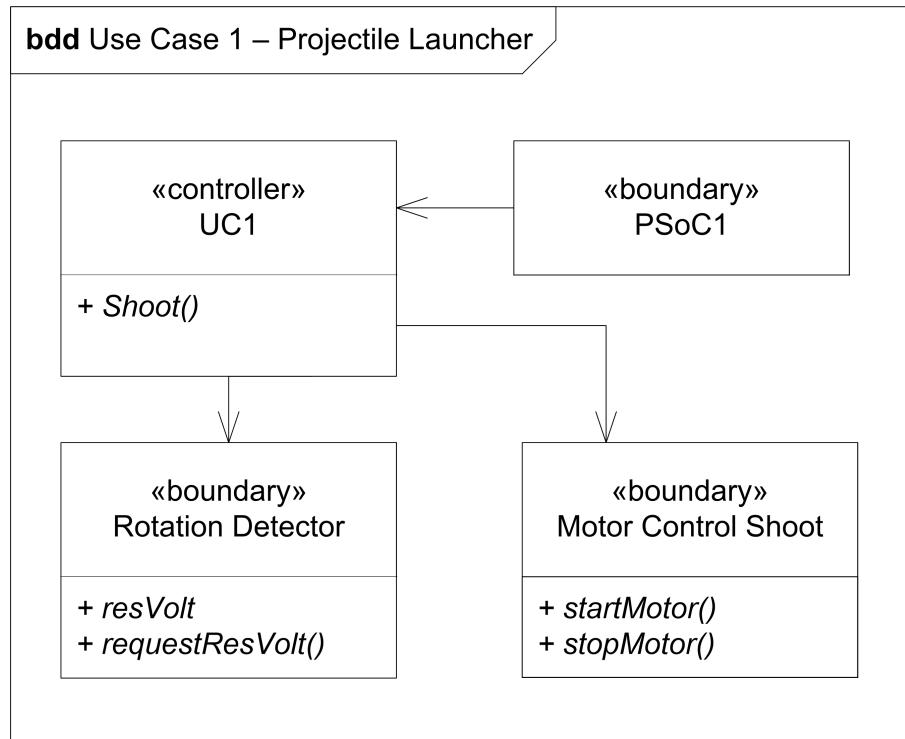
Figur 33: Samlet Klassediagram for PSoC0

På figur 34 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 34: Samlet Klassediagram for PSoC1

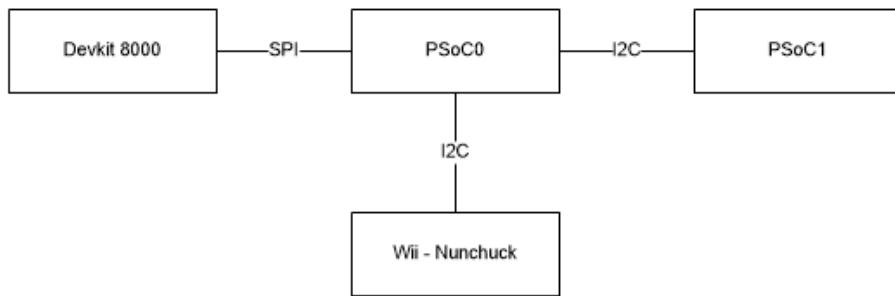
På figur 35 ses det at PSoC2 skal have funktionalitet til at aktivere afskydning, som gøres ved at kommunikere med rotationsdetektoren og motoren. Afskydningen aktiveres af grænsefladen *PSoC1*.



Figur 35: Samlet Klassediagram for PSoC2

3.7 Kommunikationsprotokoller

I dette afsnit beskrives de kommunikationsprotokoller der anvendes til at sende data mellem systemets komponenter på de anvendte bustyper - I2C og SPI. På figur 36 ses hvilke bustyper der anvendes mellem systemets komponenter.



Figur 36: Forbindelser mellem systemets komponenter

3.7.1 SPI Protokol

Til kommunikation mellem Devkit 8000 og PSoC0 anvendes der *Serial Parallel Interface (SPI)*[5]. SPI-forbindelsen består af fire signaler. Et af signalerne er

slave select (SS). Det muliggør kommunikation med flere slaver. Selve dataen sendes via signalerne *Master Out Slave In* (MOSI) og *Master In Slave Out* (MISO). Det foregår ved *Full Duplex*, hvor der altid sendes i begge retninger på én gang. Hvis der kun ønskes at sende i én retning, kan der sendes 0'er i den anden retning.

Derudover er der en *Clock* (SCLK), som sørger for at kommunikationen er synkron. I forhold til timing er det vigtigt, at både master og slave anvender samme *Timing Mode*, som styres af to bit, *SPI Clock Polarity Bit* (CPOL), som bestemmer om clock'en starter højt eller lavt, og *SPI Clock Phase Bit* (CPHA), som afgør om data samples på clock'ens rising- eller falling edge.

I dette projekt anvendes *Timing Mode 3*, hvor clock'en starter høj og sampler på rising edge.

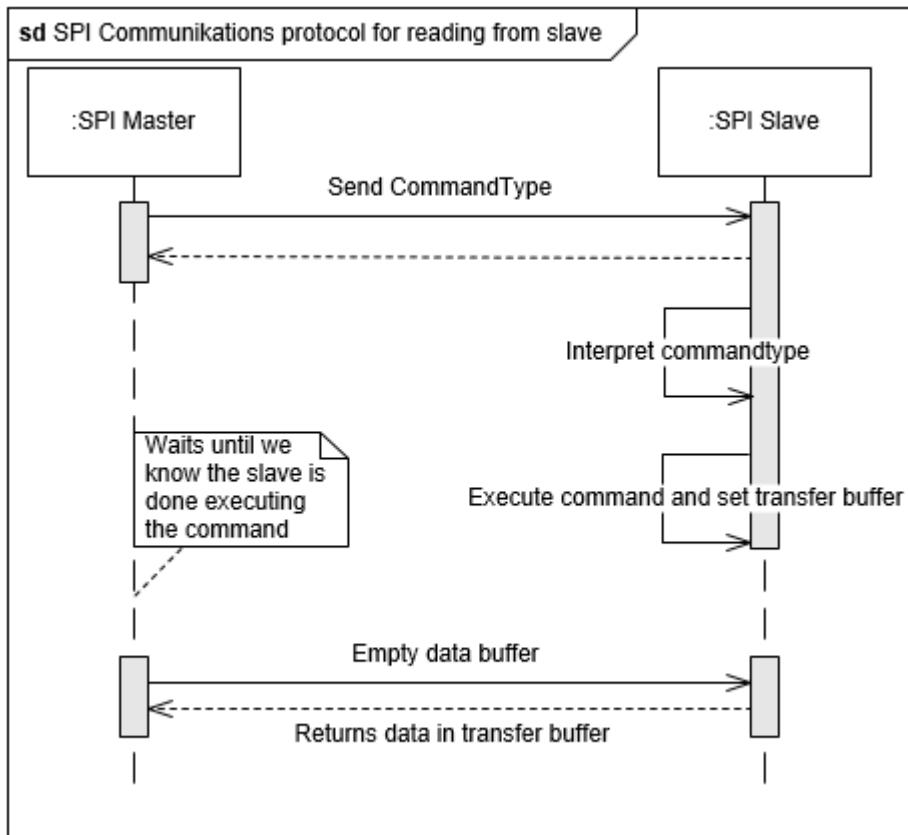
De kommandoer, der i projektet sendes via SPI, består af 8 bit.

På tabel 5 ses SPI-protokollen. Til venstre på tabellen ses navnet på de kommandoer, der sendes. Alle kommandoer med "TEST" i navnet sendes fra Devkit 8000 til PSoC0 via MOSI. De resterende kommandoer er svar på de forskellige tests, som sendes via MISO fra PSoC0 til Devkit 8000. Det er værd at lægge mærke til, at det er bevidst, at der ikke er en kommando for SPI_FAIL. Da SPI-forbindelsen ikke testes på PSoC0, men ved at PSoC0 sender SPI_OK tilbage til Devkit 8000 via SPI-forbindelsen. Dermed kan det først afgøres om SPI-forbindelsen fungerer korrekt, når Devkit 8000, modtager (eller ikke modtager) succes-beskeden fra PSoC0.

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

Tabel 5: Kommandotyper der anvendes ved SPI kommunikation

For at aflæse en besked fra SPI-slaven, skal slaven først klargøre dens SPI-transfer buffer. Dette gøres ved, at SPI-masteren sender en commandotype til slaven, som derefter tolker kommandotypen, og eksekverer kommandoen, hvor SPI-transfer bufferen bliver sat. Imens dette sker, venter SPI-masteren i et bestemt stykke tid, for at sikre at transfer-bufferen når at blive sat. På figur 37 ses et sekvensdiagram der illustrerer dette.



Figur 37: Sekvensdiagram for at læse fra SPI-slaven

3.7.2 I2C Protokol

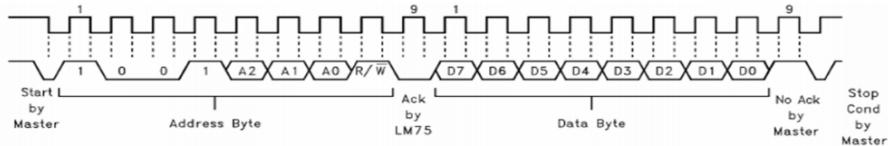
I2C[8] [4] er en bus bestående af to ledninger. Den ene ledning bruges som databus og navngives *Serial Data Line* (SDA). Den anden ledning bruges til clock signalet, der synkroniserer kommunikationen, og navngives *Serial Clock Line* (SCL). Enheder på I2C bussen gør brug af et master-slave forhold til at sende og læse data. En fordel ved I2C bussen er at netværket kan bestå af multiple masters og slaver, hvilket udnyttes i dette system da tre I2C komponenter skal sende data mellem hinanden.

I2C gør brug af en integreret protokol der anvender adressering af hardwareenheder for at identificere hvilken enhed der kommunikeres med. På tabel 6 ses addresserne 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

Tabel 6: Adresser der anvendes på I2C bussen

Den integrerede I2C protokol sender data serielt i pakker af 8-bit (1 byte). På figur 38 ses et timing diagram for aflæsning af 1 byte. Figuren er et udsnit fra databladet for en LM75, der anvender fire faste adressebits, tre vilkårlige bit og en read/write bit. Her ses at transmissionen af data begynder med en adresse-byte til slaven efterfulgt en acknowledge til masteren og herefter sendes en data-byte.

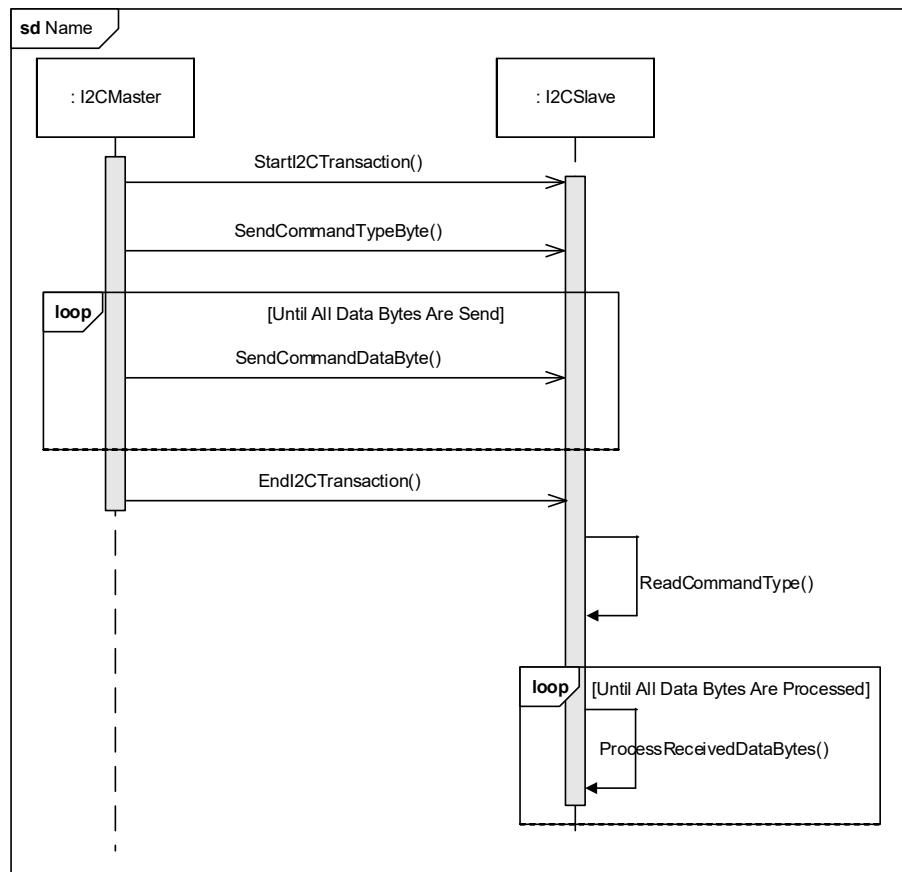


Figur 38: Timing Diagram af 1-byte I2C aflæsning

Goofy candygun gør brug af I2C protokollen via PSoC's I2C *Application Programming Interface* (API). Ved brug af denne API er der blevet udviklet en kommunikationsprotokol mellem systemets PSoCs, som gør det muligt at sende kommandoer og data.

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.

Kommandoens type definerer antallet af databytes modtageren skal forvente og hvordan disse skal fortolkes. På figur 39 ses et sekvensdiagram der, med pseudocommandoer, demonstrerer forløbet mellem en I2C afsender og modtager ved brug af kommunikations protokollen.



Figur 39: Eksempel af I2C Protokol Forløb

På figur 39 ses at afsenderen først 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 afsluttet I2C transaktion læser I2C modtageren typen af kommando, hvor den herefter kan fortolke N antal modtagne bytes afhængig af den modtagne kommandotype.

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

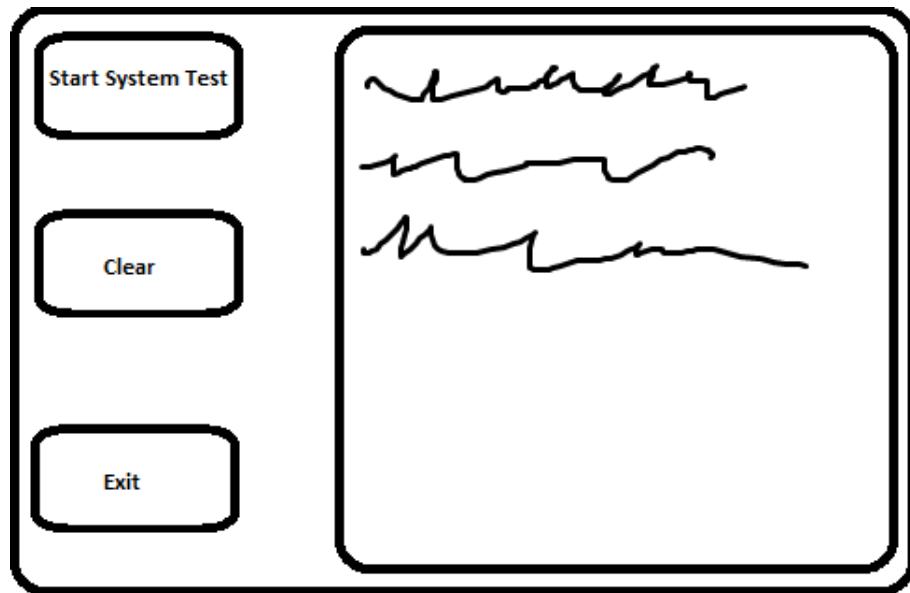
Kommandotype	Beskrivelse	Binær Værdi	Hex Værdi	Data Bytes
NunchuckData	Indholder aflæst data fra Wii Nunchuck controlleren	0010 1010	0xA2	Byte #1 Analog X-værdi Byte #2 Analog Y-værdi Byte #3 Analog Buttonstate
I2CTestRequest	Anmoder PSOC0 om at starte I2C-kommunikations test	0010 1001	0x29	Ingen databyte
I2CTestAck	Anmodning om at få en I2C OK besked fra I2C enhed	0010 1000	0x28	Ingen databyte

Tabel 7: Kommandotyper der anvendes ved I2C kommunikation

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

3.8 Brugergrænseflade

Ved hjælp af en brugergrænseflade, kan systemtesten styres fra Devkit 8000. Brugergrænsefladen er opbygget ud fra sekvensdiagrammet figur 20, ud fra dette kunne brugergrænsefladen skitseres som figur 40. Grænsefladen mellem Devkit 8000 og PSoC0 er en SPI-bus som ses på figur 6, og brugergrænsefladen er koblet til Devkit 8000 ved hjælp af en interfacedriver. Brugergrænsefladen skal sende startsekvensen til interfacedriveren, hvorefter dette sendes til PSoC0 og videre ud i systemet. Brugergrænsefladen skal aflæse svaret fra systemtesten og printe dette ud i en konsol.



Figur 40: Skitse af brugergrænseflade

Brugergrænsefladen er designet ud fra den sekventielle struktur i use case 2, se afsnit 1.4.2. 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. Interfacedriveren kunne oprettes som en separat tråd fra GUI'en, der ville køre den samme funktionalitet. Dette ville være unødvendigt i forhold til den sekventielle struktur i systemet. Kompleksiteten i dette design ville være overvældende i forhold til den ønskede funktionalitet og blev derfor fravalgt.

4 Design og implementering

4.1 Software Design

Dette afsnit indeholder designet for software komponenter og implementeringen af disse design.

4.1.1 SPI Devkit 8000 - Candygun driver

Candygun driveren sørger for SPI-kommunikationen fra Devkit8000 til PSoC0. Derved kan der sendes kommandoer og aflæses information fra PSoC0. Devkit 8000 fungerer som master og vil altid være den, der initierer en transfer.

Indstillinger for SPI

Indstillinger for SPI-kommunikationen ses i tabel 8. 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.

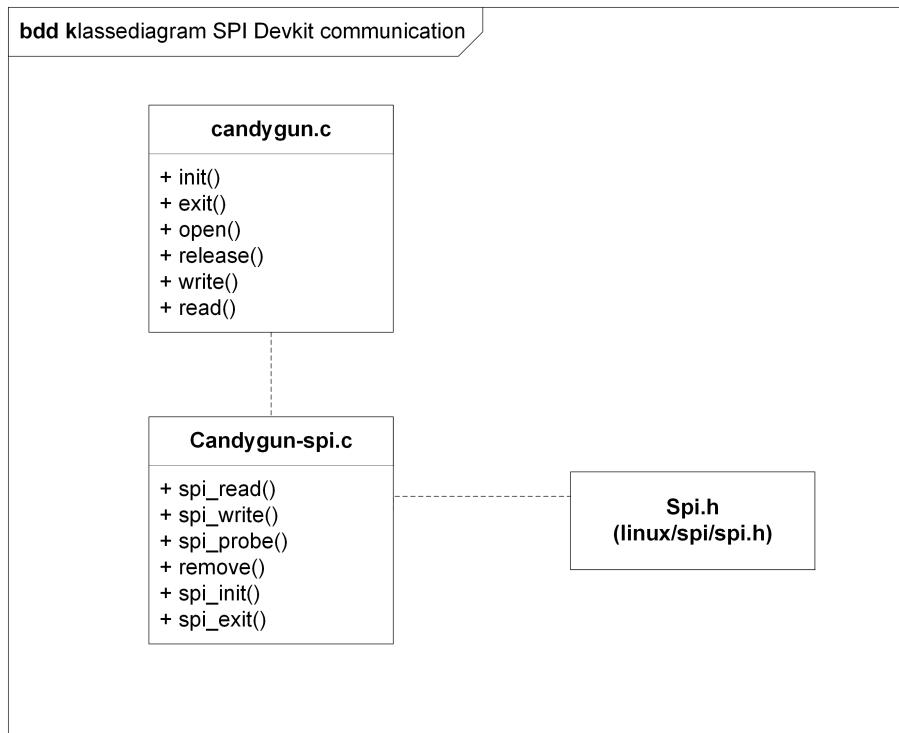
Tabel 8: Indstillinger for SPI

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

Opbygning af driver

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. Et klassediagram, som giver et illustrativt overblik over driveren ses på figur 41. 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 stippled 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 af Linux, og er derfor ikke yderligere dokumenteret her.

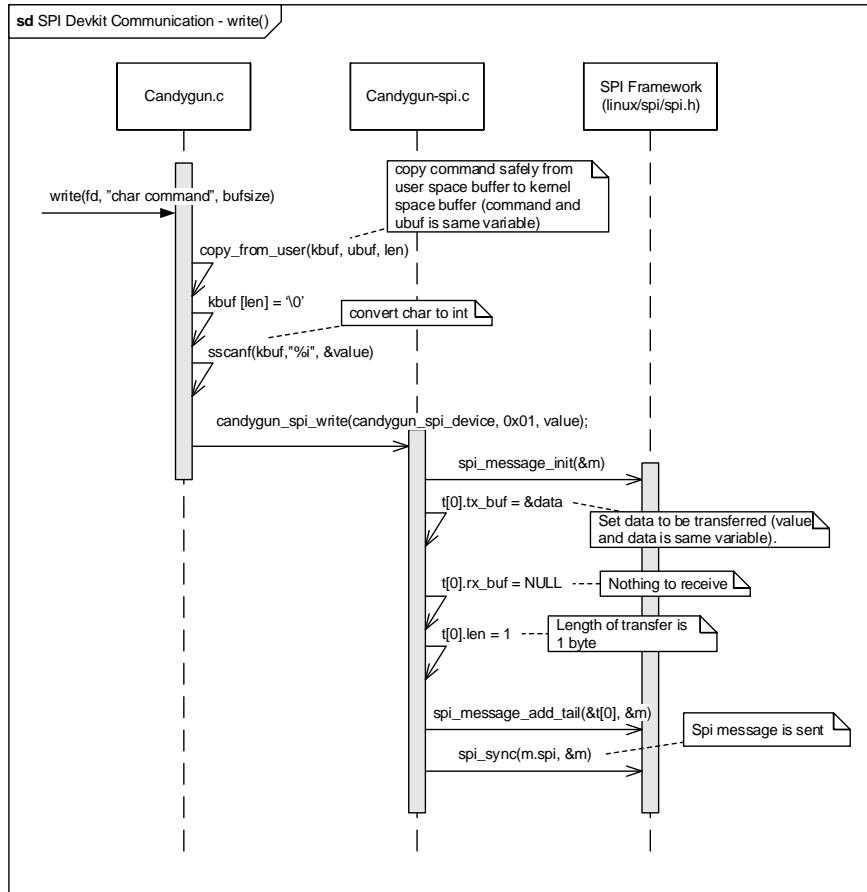
I probe-funktionen sættes bits_per_word til 8, da vi sender otte bit som nævnt tidligere. I exit-funktionen anvender candygun.c igen en funktion fra candygun-spi.c - denne gang til at frigive SPI ressourcen.



Figur 41: Klassediagram for SPI kommunikation på Devkit 8000

Opbygning af write-metode

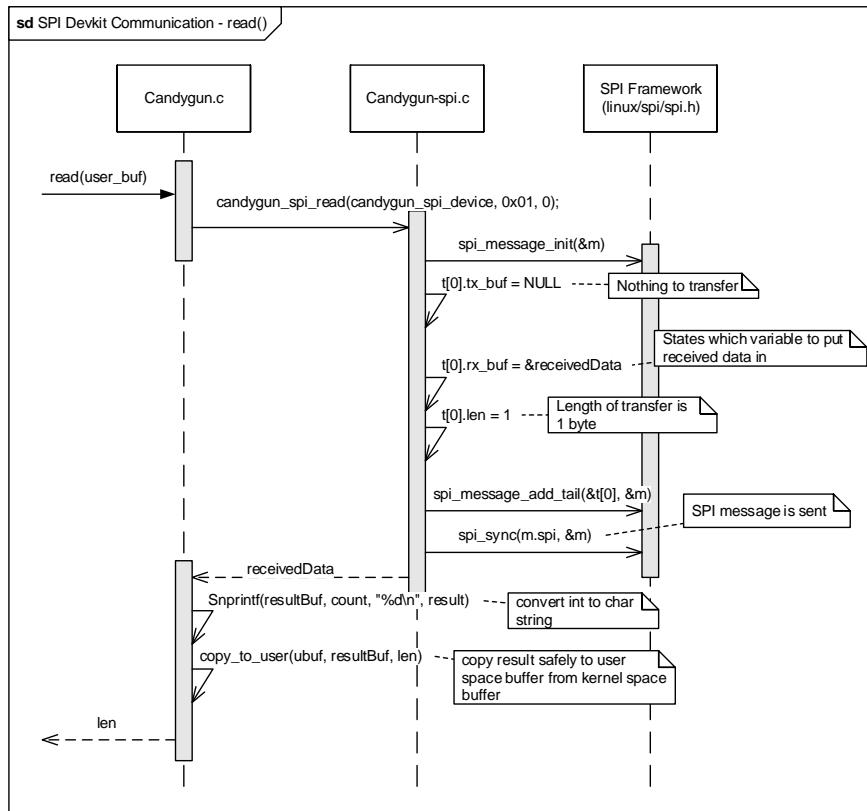
I write-metoden gives der data med fra brugerne. I dette tilfælde udgøres brugeren af Interface driveren og dataet er en 8 bit kommando fra SPI-protokollen. Dog er dataet fra brugerne i første omgang læst ind som en charstreng. I write-metoden bliver det så lavet om til en int. For at overføre dataet på en sikker måde anvendes funktionen `copy_from_user()` til at overføre data fra brugerne. 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. På figur 42 ses et sekvensdiagram for writemetoden, med fokus på opbygningen af den besked, der skal sendes.



Figur 42: Sekvensdiagram for SPI write kommunikation på Devkit 8000

Opbygning af read-metode

Ofte ville der en spi read-funktion først indeholde en write-del, som fortæller SPI-slaven, hvad der skal læses over i bufferen. Det ville typisk efterfølges af et delay og så en read-del. Men i dette projekt skal der ofte aftenes 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 indeholde 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. Et sekvensdiagram for read-funktionen ses på figur 43. Igen er hovedformålet at forklare, hvordan en SPI-message er opbygget i denne driver.



Figur 43: Sekvensdiagram for SPI read kommunikation på Devkit 8000

Hotplug

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.

Metodebeskrivelser

candygun.c:

static int __init candygun_cdrv_init(void)

I initfunktionen bliver devicenumre allokeret dynamisk, og kernen informeres om cdev-strukturen. Desuden anvendes `candygun_spi_init()` fra `candygun-spi.c` til at frigive en spi ressource.

static void __exit candygun_cdrv_exit(void)

Afregistrer device og klasse fra kernen. Anvender `candygun_spi_exit()` fra `candygun-spi.c` til at frigive SPI-ressource.

int candygun_cdrv_open(struct inode *inode, struct file *filep)
Kaldes når det kernemodul, der skal skrives til for at anvende driveren, åbnes.
Tjekker om device-numrene passer. Printer en kernebesked om at modulet åbnes.

int candygun_cdrv_release(struct inode *inode, struct file *filep)
Kaldes når kernemodulet lukkes. Printer en kernebesked om at modulet lukkes.

ssize_t candygun_cdrv_write(struct file *filep, const char __user *ubuf, size_t count, loff_t *f_pos)
Sørger for at klargøre en kommando fra userspace, og giver den med til funktionen candygun_spi_write(), som håndterer SPI-kommunikationen.

ssize_t candygun_cdrv_read(struct file *filep, char __user *ubuf, size_t count, loff_t *f_pos) Anvender candygun_spi_read(), som håndtere læsning ved SPI. Sørger derefter for at omdanne returværdien til en string og copiere den sikkert til userspace.

candygun-spi.c

int candygun_spi_init(void)
Requester en SPI-ressouce.

void candygun_spi_exit(void)
Frigiver SPI-ressource.

static int candygun_spi_probe(struct spi_device *spi)
Probe køres når driveren bliver "insmod"et i kernen, for at se om der er et SPI-device, der matcher modalias for driveren.

static int candygun_remove(struct spi_device *spi)
Fjerner SPI-device på chip-select.

int candygun_spi_write(struct spi_device *spi, u8 addr, u8 data)
Håndterer opbyggelsen af en SPI-message, sætter receivebufferen til NULL, da der ikke skal læses, og anvender funktioner fra linux/spi/spi.h til at udføre SPI-transfer.

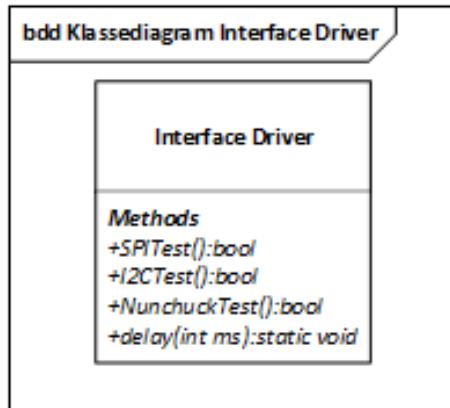
int candygun_spi_read(struct spi_device *spi, u8* value)
Håndterer opbyggelsen af en SPI-message, sætter transferbufferen til NULL, da der ikke skal skrives, sætter en kerne buffer til receive, og anvender funktioner fra linux/spi/spi.h til at udføre SPI-transfer.

4.1.2 Brugergrænseflade

I dette afsnit beskrives de forskellige dele af brugergrænsefladen.

Interfacedriver

Interface driveren er bindeled mellem brugergrænsefladen og candydriveren. Interface driveren er implementeret i c++. Den indeholder fire funktioner til use case 2. De håndterer test af de forskellige kommunikationsforbindelser i systemet. Candygun klassen implementerer de rigtig funktioner med forbindelse til det restende system, og anvender de nødvendige funktioner til at skrive til og læse fra et kernemodul. På figur 44 ses klassediagrammet for Interface Driveren.



Figur 44: Klassediagram for Interface driveren på Devkit8000

Klassebeskrivelse

bool SPITest():

Denne metode initierer SPI-test ved at skrive SPI-kommandoen "241" til "dev/candygun", som er den node, der oprettes af Candydriveren. Dernæst indeholder funktionen et delay på 1 sek, for at give SPI-testen tid til at blive udført. Til sidst læser funktionen fra "dev/candygun" og tjekker om den får den korrekte returnværdi, "209". Ved korrekt returnværdi returnerer funktionen true. Ved fejl returnerer funktionen false.

bool I2CTest():

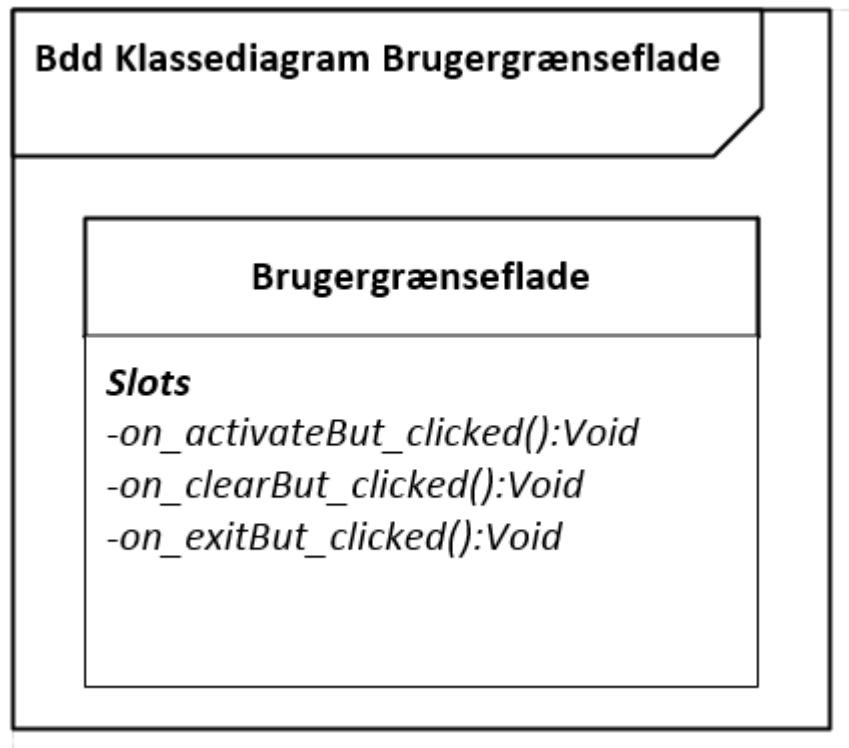
Denne metode initierer I2C-test ved at skrive I2C-kommandoen "242" til "dev/candygun", som er den node, der oprettes af Candydriveren. Dernæst indeholder funktionen et delay på 1 sek, for at give I2C-testen tid til at blive udført. Til sidst læser funktionen fra "dev/candygun" og tjekker om den får den korrekte returnværdi, "210". Ved korrekt returnværdi returnerer funktionen true. Ved fejl returnerer funktionen false.

bool nunchuckTest(): Denne metode initierer nunchuck-test ved at skrive nunchuck-kommandoen "251" til "dev/candygun", som er den node, der oprettes af Candydriveren. Metoden tjekker "dev/candygun" efter 6 sekunder, om der er trykket på nunchuckknappen. Det ses ved den korrekte returnværdi, "211". Ved korrekt returnværdi returnerer funktionen true.

static void delay(int ms): Denne metode initierer et delay. Det valgte antal ms bliver lagt til det interne systemur. Når systemuret når til det sammenlagte tidspunkt, fortsættes programmet.

Systemtest GUI

Usecase 2 styres via Systemtest GUI'en fra Devkit8000. Dette afsnit beskriver Systemtest GUI'ens design.



Figur 45: Klassediagram for Systemtest GUI

Klassebeskrivelse

void on_activateBut_clicked();

Dette slot er startknappen i brugergrænsefladen. Ved tryk bliver knappens signal broadcastet og knappens funktion bliver kørt. Forløbet for dette slot beskrives i figur 46. Der skrives til konsolvinduet, derefter testes der på SPITest(). Hvis der returnes true, skrives dette til konsollen og programmet fortsætter til I2CTest(). Hvis der returnes false, skrives dette til konsollen og programmet returnerer til idle-tilstand. Nu testes der på I2CTest(). Hvis der returnes true, skrives dette til konsollen og programmet fortsætter til NunchuckTest(). Hvis der returnes false, skrives dette til konsollen og programmet returnerer til idle-tilstand. Der testes der på NunchuckTest(). Hvis der returnes true, skrives

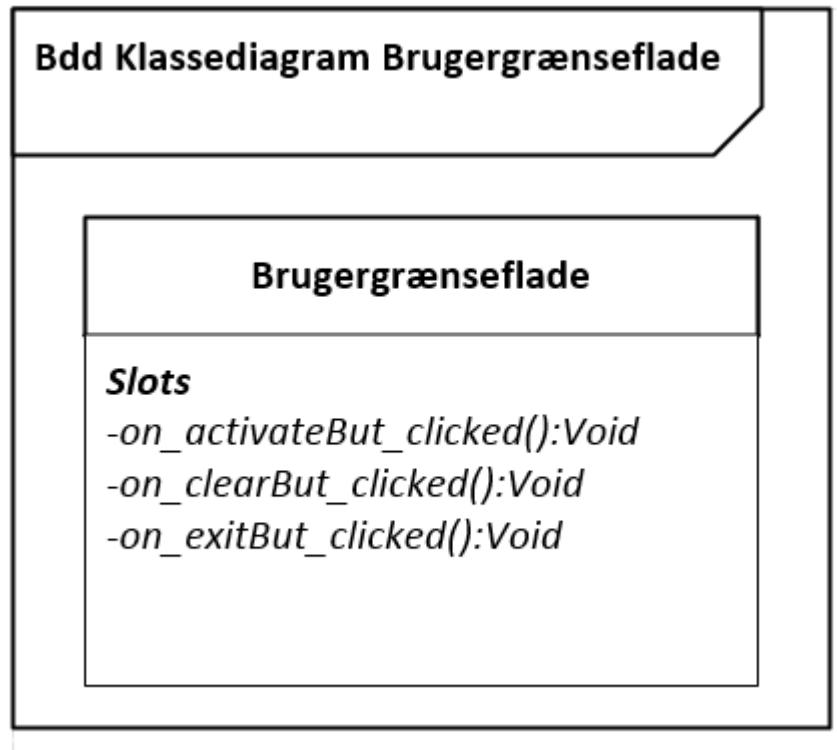
dette til konsollen, og der skrives at systemtesten er gennemført. Hvis der returnes false, skrives dette til konsollen. Programmeret returnerer til idle-tilstand.

void on_clearBut_clicked();

Dette slot er clearknappen i brugergrænsefladen. Knappens funktionalitet er en clearing af konsol vinduet.

void on_exitBut_clicked();

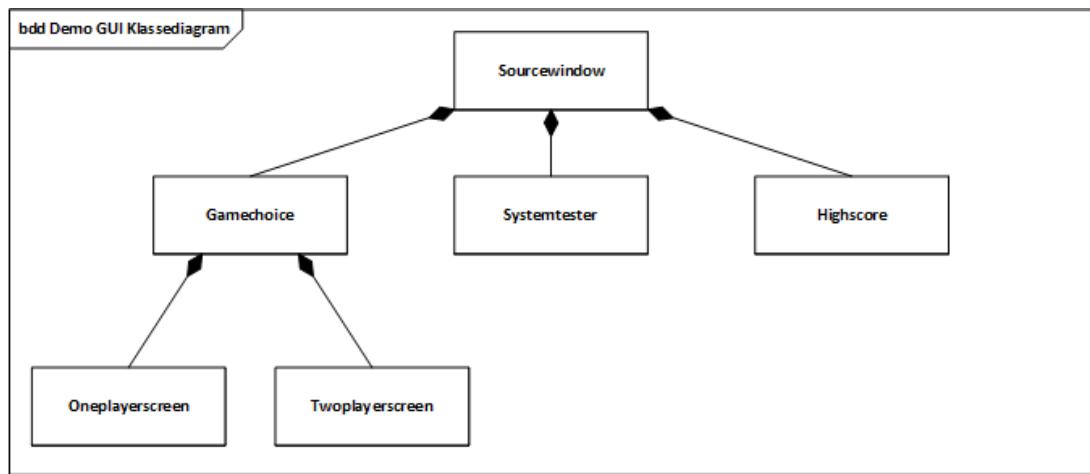
Dette slot er exitknappen i brugergrænsefladen. Knappens funktionalitet består i, at programmet lukkes.



Figur 46: Statemachine for Systemtest GUI

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 47: Klassediagram for Demo GUI

Klassebeskrivelse

Sourcewindow:

Klassens vindue består af fire knapper med følgende

Slots:

`void on_exitBut_clicked();`

Denne knap har den samme funktion som exitBut fra metodebeskrivelsen afsnit #ref

`void on_scoreBut_clicked();`

Denne knap åbner vinduet til Highscore-klassen.

`void on_startBut_clicked();`

Denne knap åbner vinduet til Gamechoice-klassen.

`void on_testBut_clicked();`

Denne knap åbner vinduet til Systemtester-klassen, se afsnit for #Systemtestgui for beskrivelse.

Highscore:

Klassens vindue består af et konsol vindue og tre knapper med følgende

Slots:

`void on_setHiBut_clicked();`

Denne knap skriver nogle testværdier til konsol vinduet.

`void on_clearHiBut_clicked();`

Denne knap clearer konsol vinduet.

`void on_exitHiBut_clicked();`

Denne knap lukker Highscore vinduet og GUI'en returnerer til sourcewindow.

Gamechoice:

Klassens vindue består af tre knapper med følgende

Slots:

`void on_choiceOne_clicked();`

Denne knap åbner vinduet til Oneplayerscreen-klassen, og den opretter et oneplayerscreen objekt og kalder en slot-funktionen `setTestScores` fra oneplayerscreen.

`void on_choiceTwo_clicked();`

Denne knap åbner vinduet til Twoplayerscreen-klassen, og den opretter et twoplayerscreen objekt og kalder en slot-funktionen `setTestScores` fra twoplayerscreen.

`void on_choiceExitBut_clicked();`

Denne knap lukker Gamechoice vinduet og GUI'en returnerer til sourcewindow.

Oneplayerscreen:

Klassens vindue består af et konsol vindue, en knap med følgende

Slots:

`void on_playerExitBut_clicked();`

Denne knap lukker Oneplayerscreen vinduet og GUI'en returnerer til Gamechoice.

`void setTestScores();`

Dette slot skriver testværdier til Oneplayerscreen's konsol vindue.

Twoplayerscreen:

Klassens vindue består af to konsol vinduer, en knap med følgende

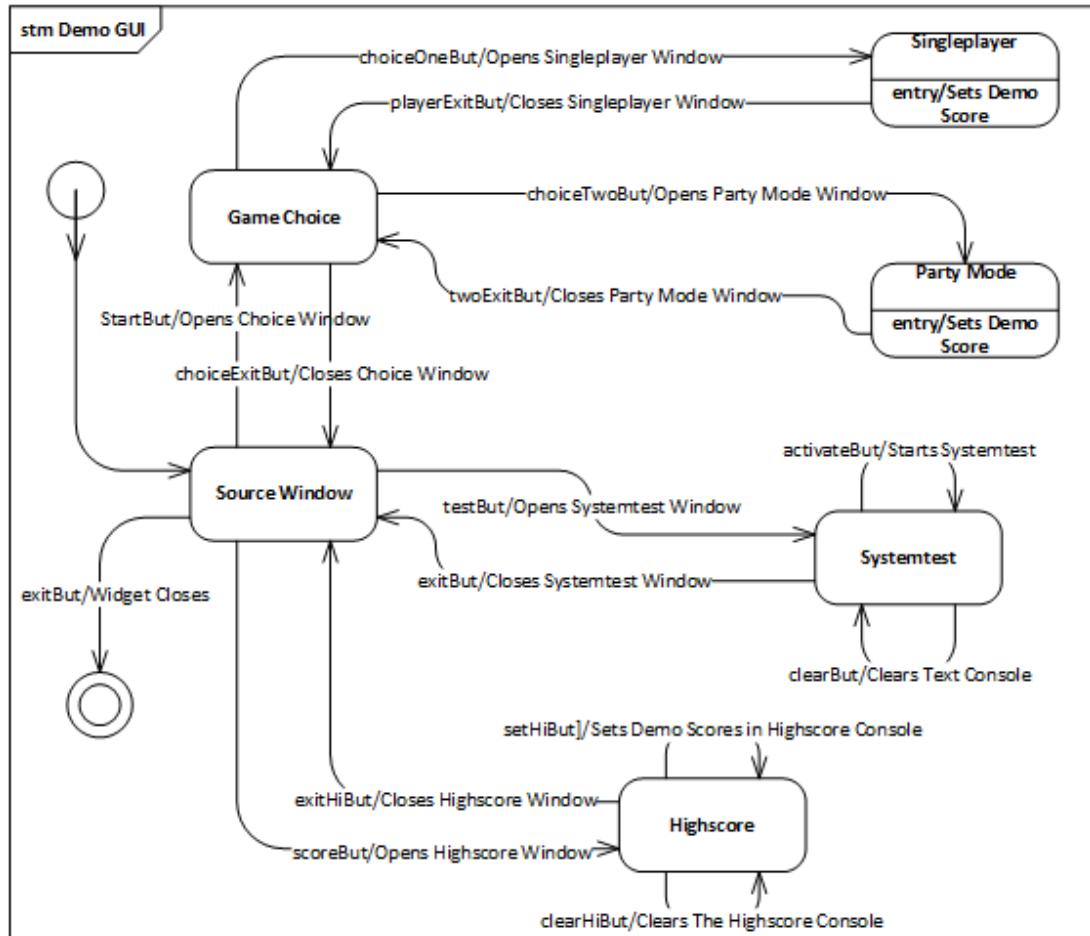
Slots:

`void on_twoExitBut_clicked();`

Denne knap lukker Twoplayerscreen vinduet og GUI'en returnerer til Gamechoice.

`void setTestScores();`

Dette slot skriver testværdier til Twoplayerscreen's konsol vinduer.

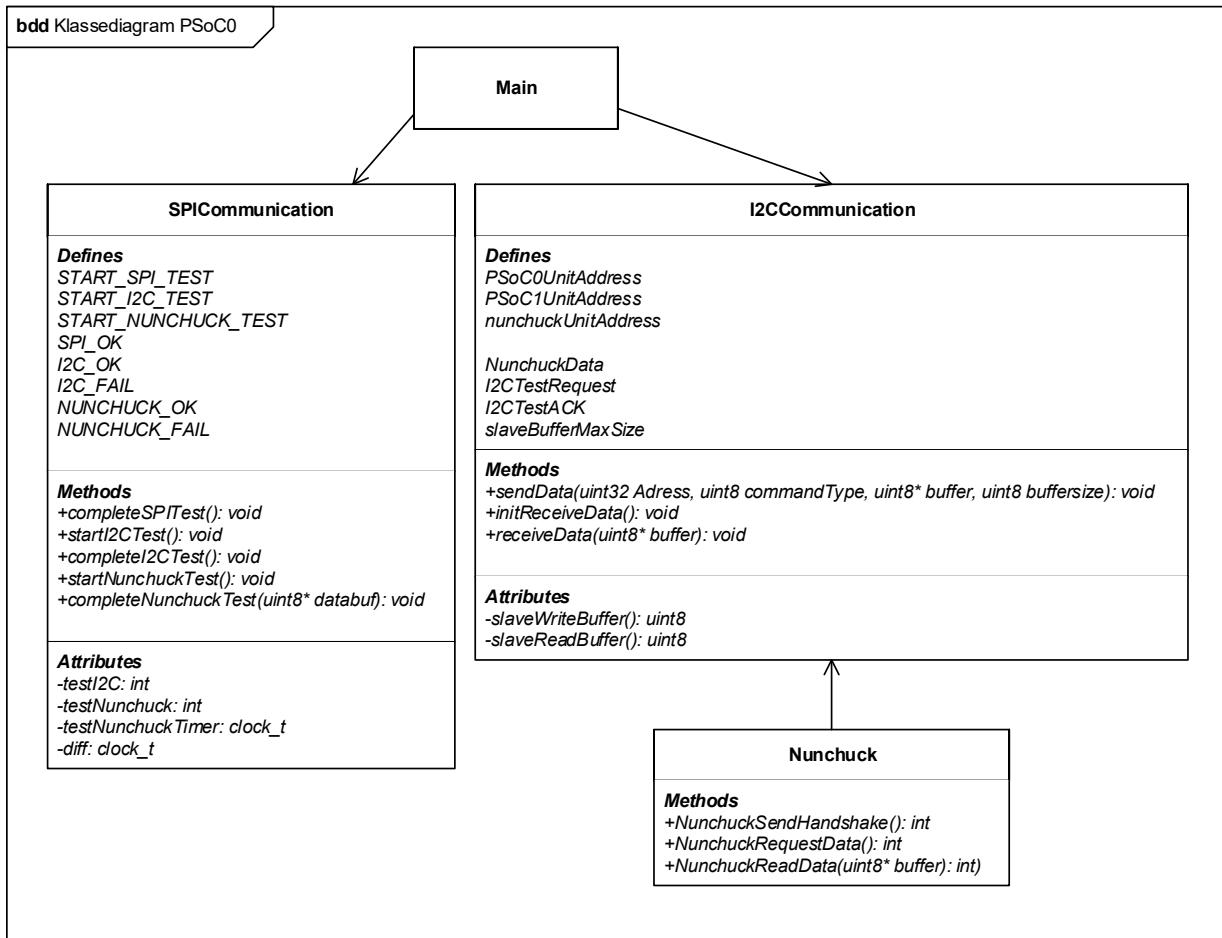


Figur 48: State machine for Demo GUI

Denne State Machine giver en visuel repræsentation af skiften mellem GUI-vinduer. Hvert knaptryk er et event og slot-funktionen for den pågældende knap bliver kørt. Se klassebeskrivelsen, 4.1.2 for detaljeret beskrivelse.

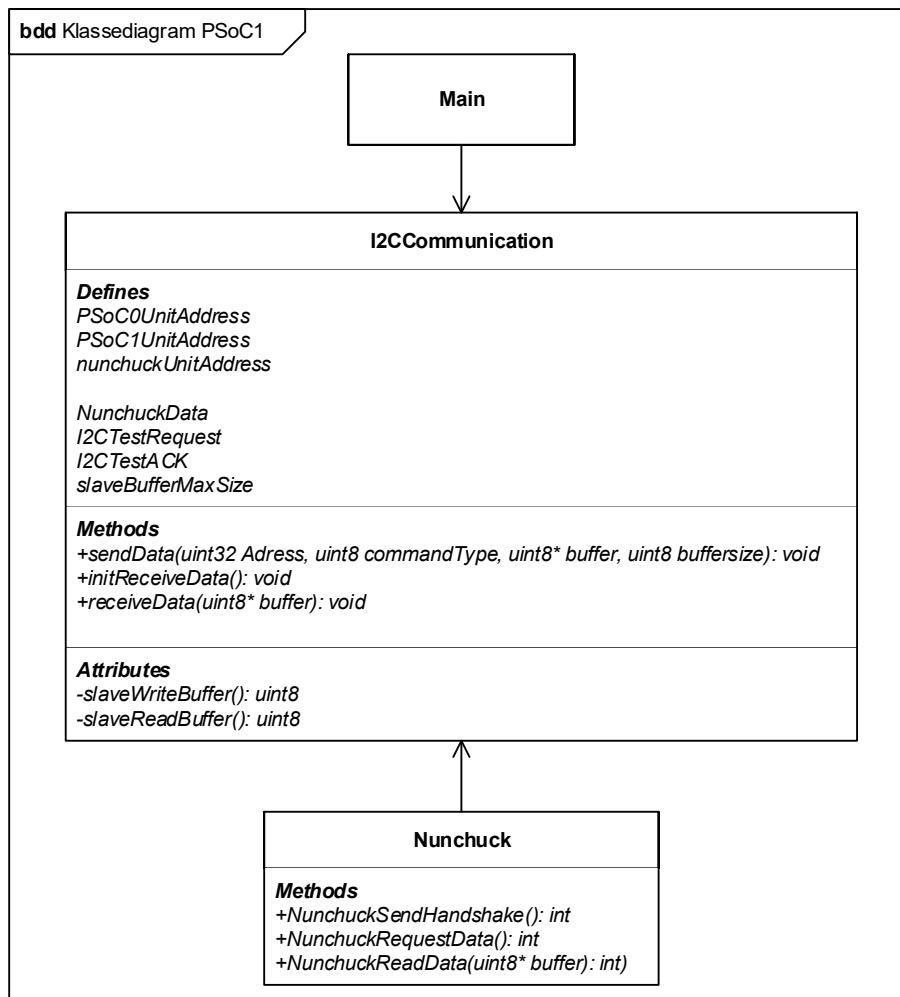
4.2 PSoC Software

På figur 49 og 50 ses de endelige klassediagrammer for PSoC0 og PSoC1. Disse klassediagrammer er designet ud fra applikationsmodellerne for use case 1 og 2 i afsnit 3.5.1 og 3.5.2, samt de samlede klassediagrammer for use case 1 og 2 i systemarkitekturen afsnit 3.6 figur 33 og 34. For at opretholde høj samhørighed, er der lavet en klasse for hver grænseflade. Dette kan ses på figur 49, da separate klasser er oprettet for grænsefladerne *SPI*, *I2C*, samt *Nunchuck* controlleren. De efterfølgende afsnit vil beskrive klasserne og deres funktioner.



Figur 49: Klassediagram for PSoC0

På figur 49 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 4.2.1, 4.2.2 og 4.2.3.



Figur 50: Klassediagram for PSoC1

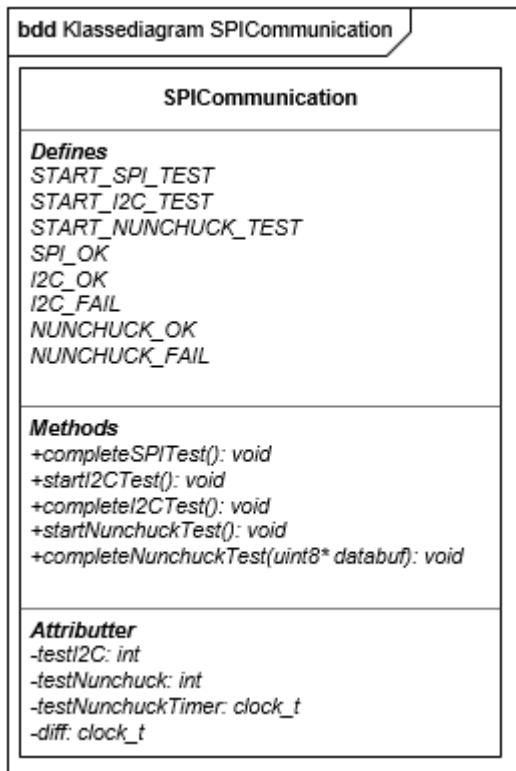
På figur 50 ses det samlede klassediagram for klasserne som eksisterer på PSoC1 softwaren. Her kan det ses at der er en *main* klasse. Denne indeholder programmets primære loop, som gentages indtil PSoC'en slukkes. I dette loop bliver der gjort brug af funktionaliteten fra klasserne *I2CCommunication* og *Nunchuck*. Disse klasser beskrives ligeledes i afsnit 4.2.1 og 4.2.2.

4.2.1 SPICommunication

I dette afsnit vil softwaren der specifikt omhandler SPI kommunikationen mellem PSOC0 og Devkit 8000 blive beskrevet. Dette gøres med et klassediagram og klassebeskrivelser.

Klassediagram

På figur 51 ses klassediagrammet over SPICommunication klassen.



Figur 51: Klassediagram over SPICommunication

Klassebeskrivelser

Klassen *SPICommunication* har til ansvar at gennemføre SPI-, I2C-, samt Nunchucktesten som bliver anmodet om af Devkit 8000 under use case 2. Dette gøres ved nedenstående metoder.

void completeSPITest()

Denne metode gemmer *SPI_OK* i PSoC'ens SPI-transfer buffer.

void completeI2CTest()

Denne metode gennemfører I2C-Testen. Dette gøres ved at der sendes en besked til alle enheder på I2C-nettet, og hvis der ikke registreres nogen fejl på denne besked, bliver *I2C_OK* gemt i PSoC'ens SPI-transfer buffer. Registreres der en fejl, bliver *I2C_FAIL* gemt i SPI-transfer buffer.

void completeNunchuckTest(uint8* databuf)

Denne metode gennemfører Nunchuck-testen. Dette gøres ved at der startes en timer på 5 sekunder. Hvis der sker et tryk på 'Z'-knappen på nunchucken indenfor disse 5 sekunder, vil *NUNCHUCK_OK* blive gemt i SPI-transfer bufferen. Hvis der ikke registreres nogen tryk inden for de 5 sekunder, er det *NUNCHUCK_FAIL* der gemmes.

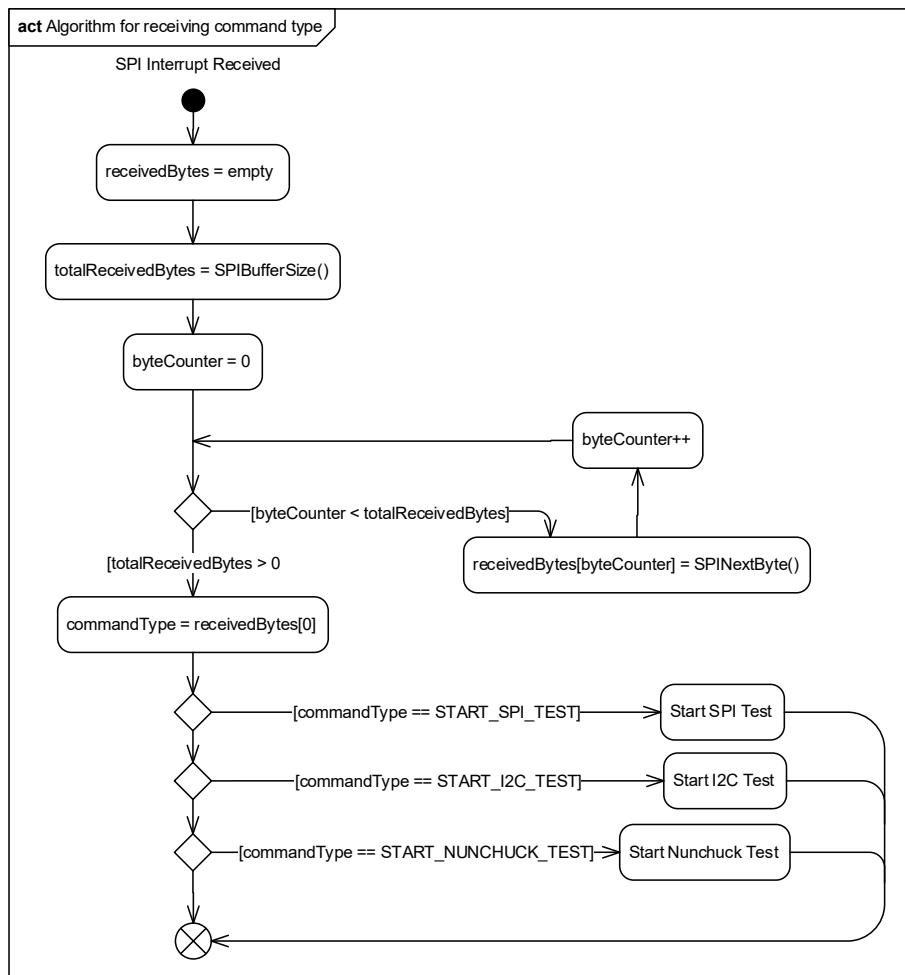
I klassediagrammet er der beskrevet en række "Defines". Disse defines bruges i klassen som unikke ID'er der indikerer om en test er gennemført OK eller om den er fejlet.

SPI Indstillinger på PSoC

I forbindelse med SPI bussen skal der sættes nogle indstillinger på PSoC0, idéet at denne kommunikerer med Devkit 8000 via SPI. PSoC'en sættes til slave, idéet at det er Devkit 8000 der aflæser og skriver til PSoC0. SCLK sættes til CPHA=1 og CPOL=1. Disse er valgt arbitrært, dog ud fra den forudsætning, at de skal stemme overens med indstillingerne på Devkit 8000, idéet disse indstillinger beskriver hvornår databit aflæses eller sættes i forhold til clock'en. *Data Rate* sættes til 1Mbps, da dette er en stabil overførselshastighed. Denne indstilling skal være ens for PSoC og Devkit 8000. Den sidste indstilling der sættes, er *transfer* og *read* buffer size. Disse er valgt til at være 8-bit, da projektets SPI kommunikationsprotokol ikke har brug for større datamængder. Denne indstilling skal være ens for både SPI master og SPI slave.

Test Opstart Algoritme

På figur 52 ses et aktivitetsdiagram for algoritmen som eksekverer de forskellige tests afhængig af den modtagne kommandotype fra Devkit 8000.



Figur 52: Aktivitetsdiagram der viser algoritmen som bestemmer hvilket test der skal eksekveres

På tabel 9 ses en forklaring af de variable der indgår i algoritmen.

Variable	Beskrivelse
receivedBytes	Denne variabel er et array som indeholder modtagede bytes fra SPI bussen.
totalReceivedBytes	Denne variabel indeholder antallet af modtagede bytes fra SPI bussen.
byteCounter	Denne variabel er en tæller til algoritmens for løkke.
commandType	Denne variabel indeholder kommandotypen af den modtagede kommando.

Tabel 9: Variable der indgår i aktivitetsdiagrammet figur 52

Aktivitetsdiagrammet, figur 52, viser at algoritmen bliver eksekveret hver gang

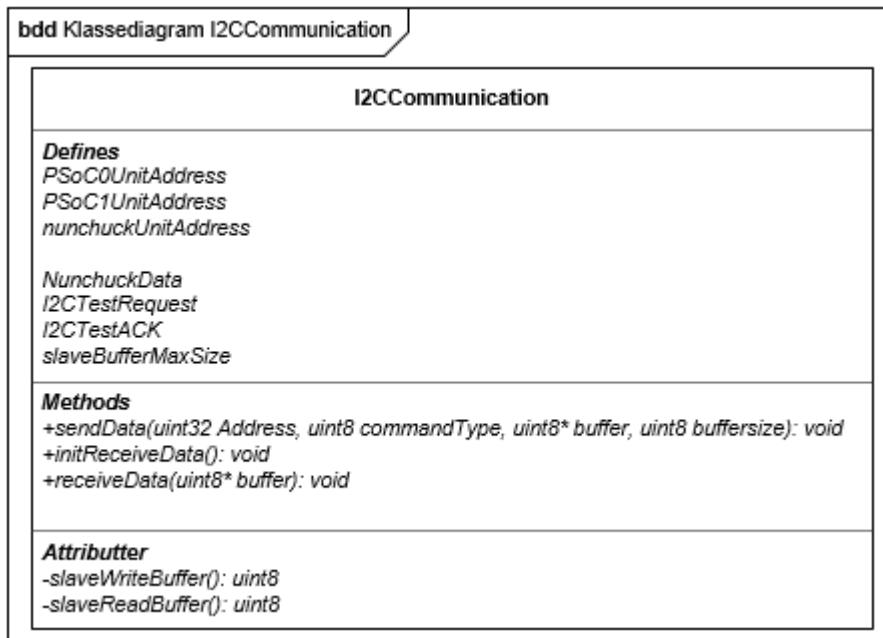
et SPI interrupt modtages. Et SPI interrupt opstår når nyt data fra SPI bussen er modtaget og klart til at blive læst fra bufferen på PSOC'en. Den første betydelige hændelse er at variablen *totalReceivedBytes* bliver sat til antallet af modtagede bytes fra SPI bussen. Herefter køres en for løkke igennem, hvor alle modtagede bytes fra PSOC'ens buffer bliver overført til variablen *receivedBytes*. Til slut tjekkes den første modtagede byte, da denne indeholder kommandotypen modtaget på SPI bussen. Ud fra denne værdi bliver den korrekte test eksekveret.

4.2.2 I2CCommunication

I dette afsnit vil softwaren der omhandler I2C-kommunikation blive beskrevet. Dette inkluderer et klassediagram, klassebeskrivelser og indstillingerne for I2C kommunikationen på PSoC'en.

Klassediagram

På figur 53 ses klassediagrammet for I2CCommunication.



Figur 53: Klassediagram for I2CCommunication klassen

Klassebeskrivelser

Som det ses på klassediagrammet figur 53 indeholder klassen flere metoder. Disse metoder blive beskrevet her.

Klassen *I2CCommunication* har til ansvar at skrive og læse data fra I2C bussen.

```
void sendData(uint8 Address, uint8 commandType, uint8* buffer,
uint8 bufferSize)
```

Denne metode sender, via PSoC Creators I2C-API, den data der ligger i *buffer* af kommandotypen *commandType* til slaven med adressen *Address*.

void initReceiveData()

Denne metode initialiserer de to buffers, *slaveWrite* og *slaveRead*, der kræves på en I2C-slave. Dette gøres ved brug af PSoC Creators I2C-API.

void receiveData(uint8* buffer)

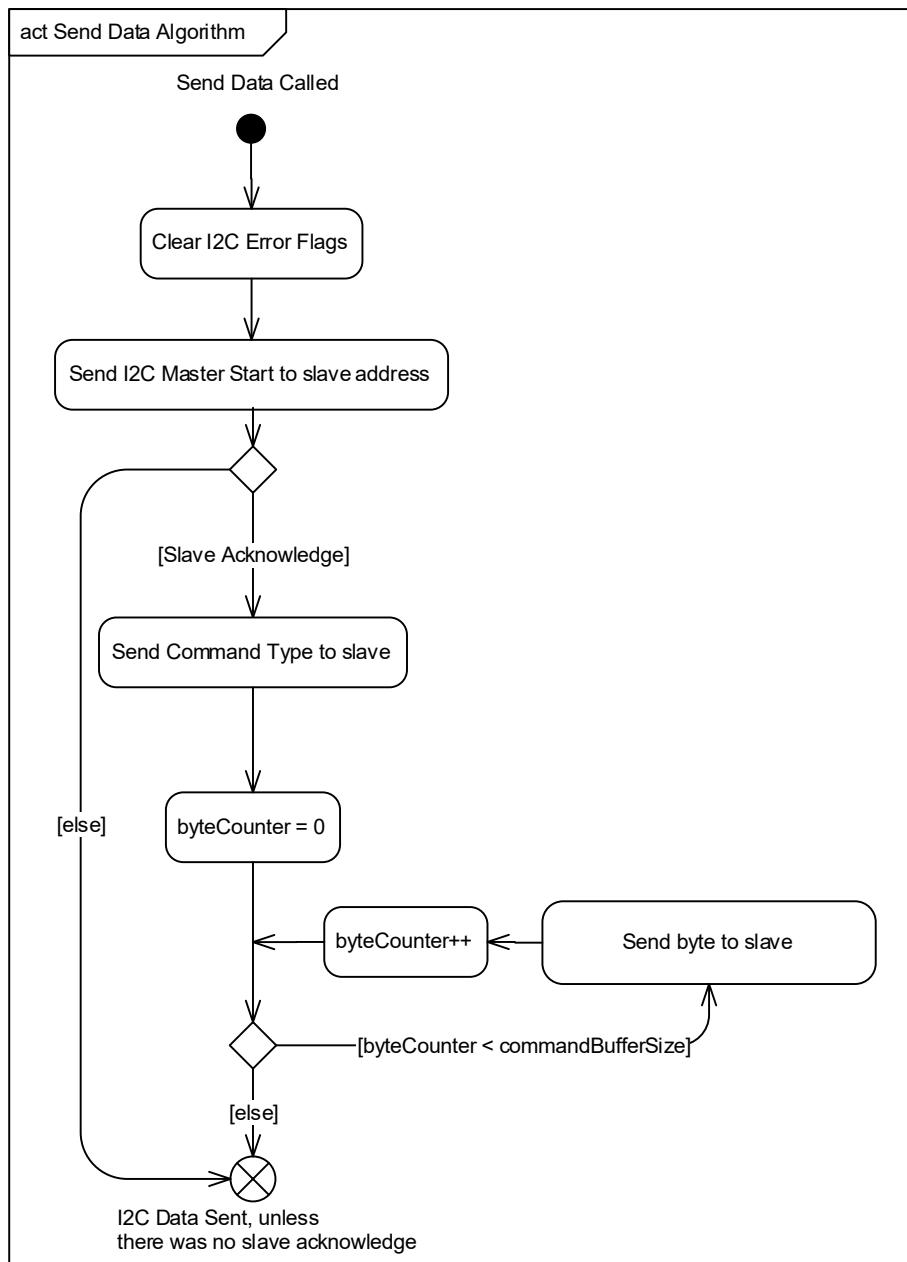
Denne metode venter på at slaveRead bufferen er blevet fyldt. Når dette er sket, bliver slaveRead bufferen kopieret over i *buffer*.

I2C indstillinger på PSoC

I forbindelse med at have en I2C kommunikation på PSoC'en, er der et par indstillinger der skal sættes. Hver PSoC der gør brug af I2C, sættes til at være en *Multi-master-slave*. Dette gøres for at alle I2C enheder kan gøre brug af *I2CCCommunication* klassen, idét at denne indeholder funktioner for både master og slave. En anden vigtig indstilling er I2C bussens *Data Rate*. Denne er sat til 100kbps. Enhver enhed på I2C nettet skal også have en adresse. Adressefordelingen ses i afsnit 3.7.2 på tabel 6.

Send Data Algoritme

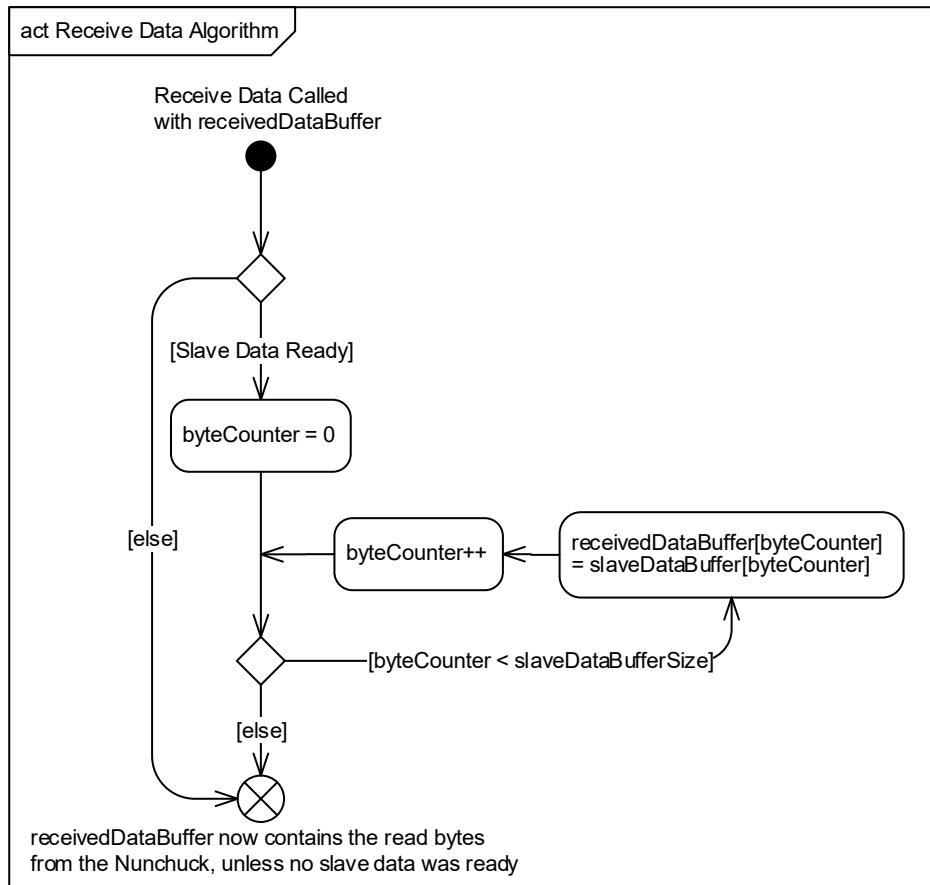
På figur 54 ses et aktivitetsdiagram for algoritmen der sender databytes ud på I2C bussen.

Figur 54: Aktivitetsdiagram over *sendData* metodens algoritme

Algoritmen starter når funktionen *sendData* bliver eksekveret. Herefter bliver potentielt tidligere fejltilstande for I2C softwaren nulstillet. I2C Masteren forsøger herefter at starte en transaktion med den slave der er blevet angivet i funktionskaldet. Hvis slaven findes på I2C bussen, bliver kommandotypen sendt, efterfulgt af alle bytes angivet i funktionskaldets buffer.

Receive Data Algoritme

På figur 55 ses et aktivitetsdiagram for algoritmen der modtager bytes på I2C bussen.



Figur 55: Aktivitetsdiagram over *receiveData* metodens algoritme

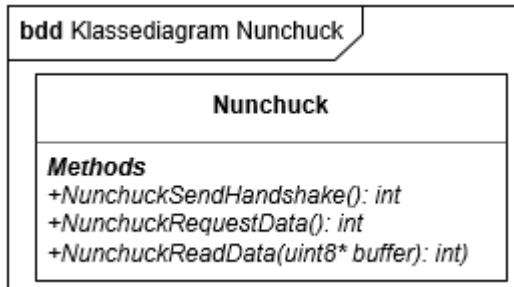
Algoritmen starter når funktionen *receiveData* bliver eksekveret. Klienten der kalder funktionen angiver desuden en buffer ved navn *receivedDataBuffer*. Denne buffer udfyldes med den modtagne I2C data. Det bliver tjekket om der er data klar til aflæsning på PSOC'ens I2C modtager buffer. Denne buffer er navngivet *slaveDataBuffer* i aktivitetsdiagrammet. Hvis dette er tilfældet, bliver alle bytes fra *slaveDataBuffer* overført til *receivedDataBuffer*. På denne måde vil *receivedDataBuffer* til slut indeholde alle bytes modtaget fra I2C bussen.

4.2.3 Nunchuck

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

Klassediagram

På figur 56 ses klassediagrammet for Nunchuck klassen.



Figur 56: Klassediagram for Nunchuck klassen

Klassebeskrivelser

Metoderne fra klassediagrammet figur 56 vil blive beskrevet i dette afsnit.

Nunchuck klassen har til ansvar at læse data fra Wii-Nunchuck controlleren. For at aflæse og dekode data, er formler og andet information taget fra (Bilag/-Dokumentation/WiiNunchuck.pdf).

int NunchuckSendHandshake()

Denne metode sender et *handshake* til Nunchuck enheden. Handshaket bruges til at parre PSoC'en med nunchucken. Metoden returnerer et '0' hvis der opstår en fejl.

int NunchuckrequestData()

Denne metode sender et 0x00 til nunchuck'en, og derved beder nunchuck'en om at klargøre data til overførsel. Metoden returnerer et '0' hvis der opstår en fejl.

int NunchuckreadData(uint8* buffer)

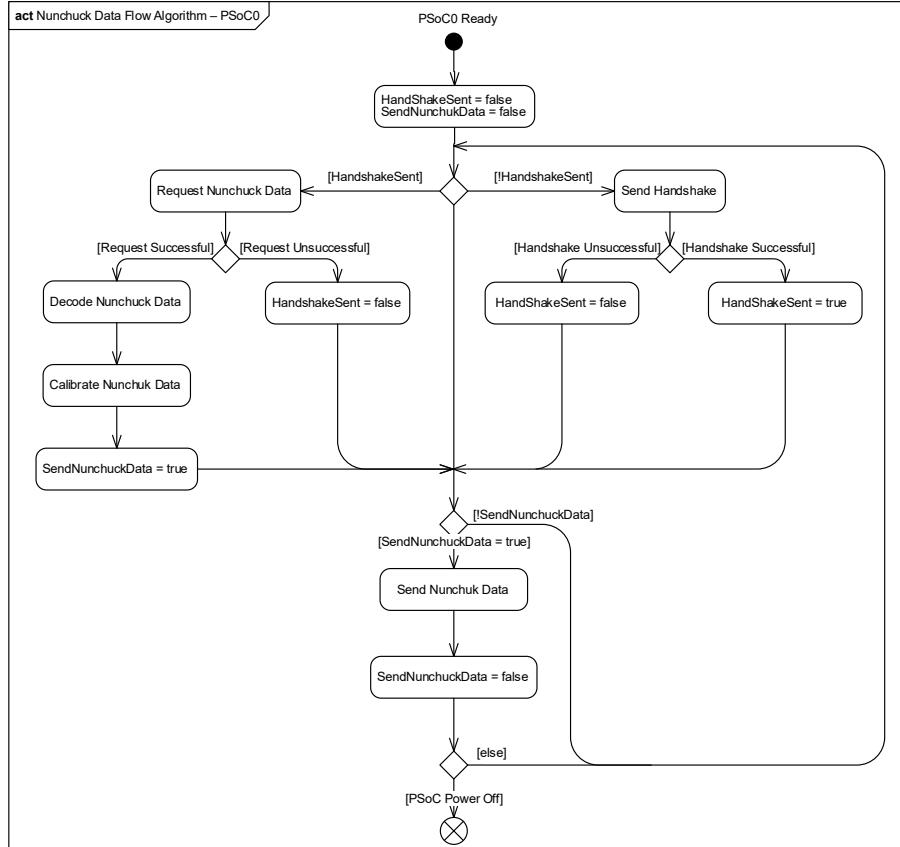
Denne metode bruger PSoC Creator's I2C-API til at læse data fra nunchuck'en (data der blev klargjort fra NunchuckrequestData()). Disse data bliver derefter dekrypteret og gemt i "buffer", så de bliver tilgængelige uden for metodens scope.

I klassediagrammet er der en sektion kaldet *Defines*. Disse Defines bruges i implementeringen til forskellige formål. *PSoC0UnitAddress*, *PSoC1UnitAddress* og *nunchuckUnitAddress* bruges til at definere adresserne for I2C-nettets slaver. *NunchuckData*, *I2CTestRequest* og *I2CTestACK* er kommando typer der bruges til at bestemme hvilken kommando type der er blevet sendt/modtaget, og hvor mange bytes der skal forventes at være gemt i databufferen. Se dokumentationen afsnit 3.7.2 tabel 7.

Nunchuck Read Data Algoritme

På figur 57 ses et aktivitetsdiagram for algoritmen der aflæser Nunchuck'ens tilstand ved at modtage dets bytes, for herefter at dekode og kalibrere dem, så

værdierne kan bruges til styring og affyring af kanonen.



Figur 57: Aktivitetsdiagram over *NunchuckReadData* metodens algoritme

På tabel 10 ses variablerne der indgår i aktivitetsdiagrammet på figur 57.

Variable	Beskrivelse
HandShakeSent	Denne variabel indikerer om der er sendt et handshake til Nunchuck'en eller ej.
SendNunchuckData	Denne variabel indikerer om der er aflæst data fra Nunchuck som kan videresendes til PSoC1 for motorstyring.

Tabel 10: Variable der indgår i aktivitetsdiagrammet figur 57

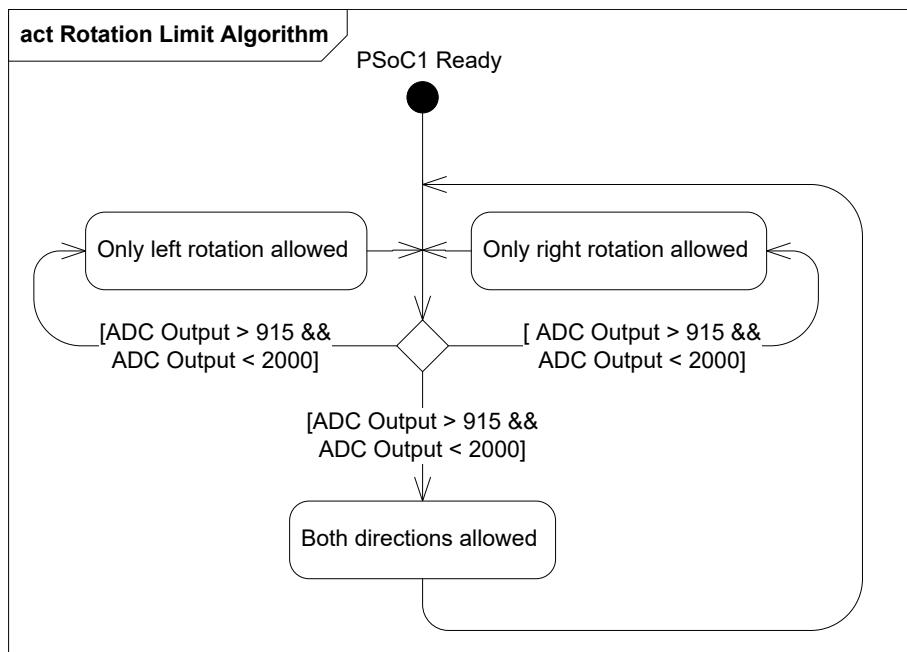
Algoritmen gentager sig selv indtil PSoC'en slukkes, så input fra Nunchuck'en konstant aflæses. Den grundlæggende idé er, at der først tjekkes på om et *handshake* er sendt til Nunchuck'en eller ej, hvilket er et krav for at kunne aflæse data fra den. I tilfældet hvor det ikke er afsendt, vil algoritmen blive ved med at forsøge indtil dette lykkedes. Herefter vil algoritmen kontinueret aflæse data fra Nunchuk'en, dekode det og til sidst kalibrere det. Hvis der er en ukendt

årsag ikke kan aflæses data, bliver *handshakeSent* sat til false.

Hvis der til slut er Nunchuck data at sende, vil denne data blive afsendt til PSoC1, og *SendNunchuckData* sættes til false, så en aflæsning sker igen.

4.2.4 Rotationsbegrænsning

I forbindelse med begrænsning af motorens rotation er der nogle indstillinger der skal sættes, se afsnit 4.5.2. ADC'en skal indstilles til at gøre brug af en kanal i single mode, da der ét inputsignal fra potentiometret og denne skal bestemmes i forhold til stel. For at styre motorens bevægelse i et interval er følgende algoritme implementeret.

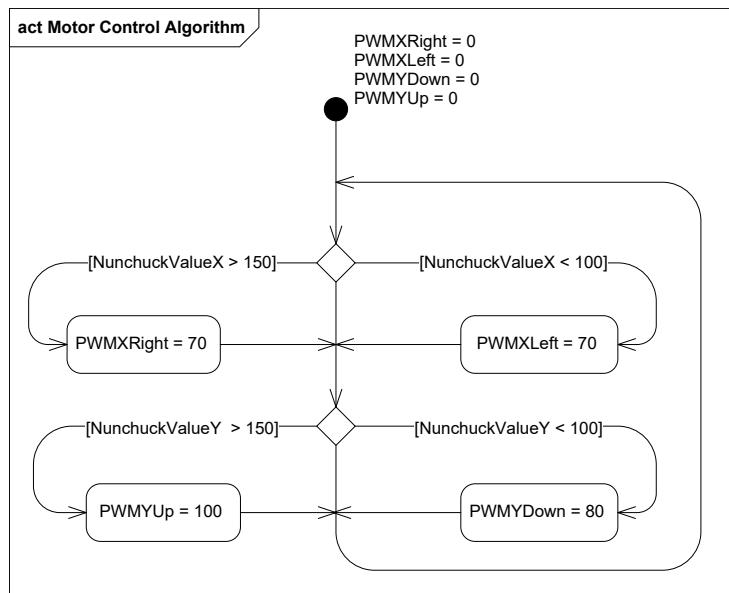


Figur 58: Aktivitetsdiagram for rotationsbegrænsningsalgoritmen

På figur 58 ses et aktivitetsdiagram for algoritmen for rotationsbegrænsning. Der er fastsat to værdier for ydergrænser. Når motoren bevæger sig udover ydergrænserne, blokeres motoren for den givne retning indtil motoren er tilbage i intervallet.

4.2.5 Motorstyring

Til styring af motorene gøres der brug af fire PWM blokke, to til styring af X-aksen og to til styring af Y-aksen, se afsnit 4.5.2. Disse PWM blokke er indstillet med en clock frekvens på 3MHz. Til styring af motorene er følgende algoritme implementeret.

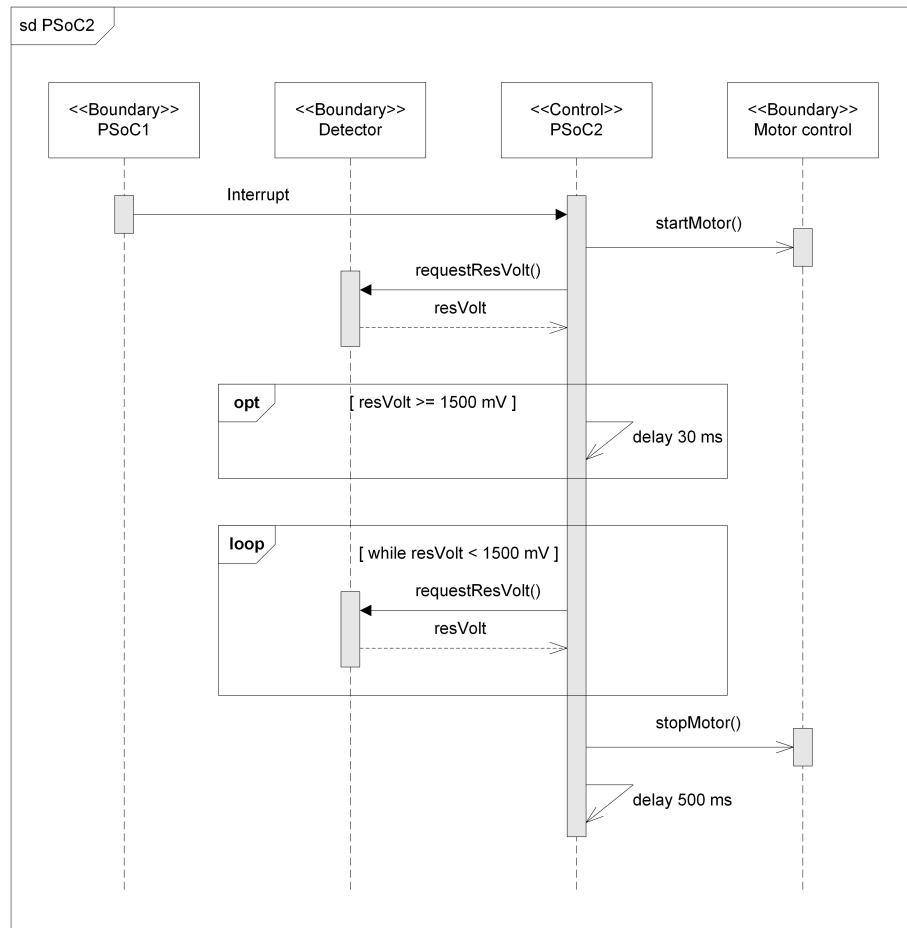


Figur 59: Aktivitetsdiagram for motorstyringsalgortimen

På figur 59 ses aktivitetsdiagrammet for motorstyringsalgoritmen. Der ses at retningensignalerne bestemmes ud fra Wii-Nunchuck'ens inputværdier fra X- og Y-aksen.

4.2.6 Affyringsmekanisme

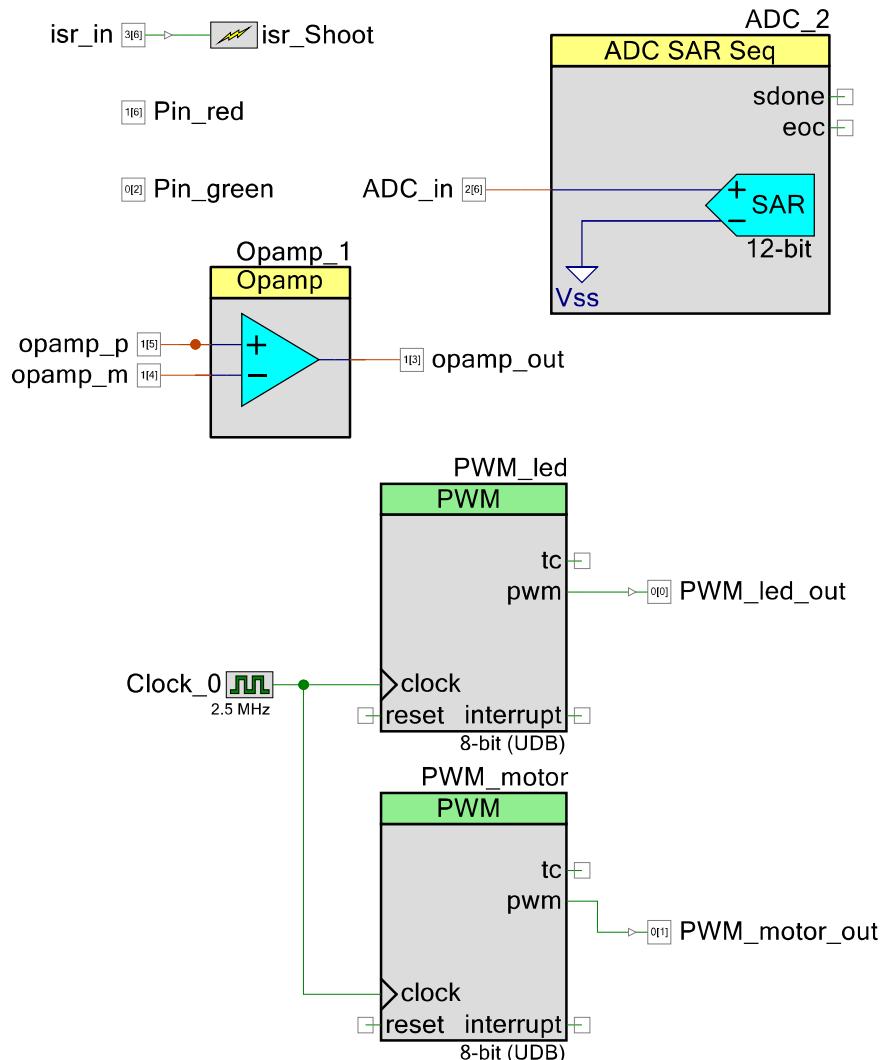
PSoC2 anvendes til styring af affyringsmekanismen. Til rotationsdetektorens operationsforstærker er der anvendt en, der er indbygget i PSoC'en. Derudover styres bl.a. interrupt og PWM-signaler i forbindelse med affyringsmekanismen ved hjælp af PSoC2. Desuden aflæses spændingen fra rotationsdetektoren af en SAR ADC, som også findes på PSoC2. Til at håndtere affyringen er der opstillet en kodesekvens, som køres ved modtagelse af et interrupt. Et sekvensdiagram for afviklingen af koden i interruptet ses på figur 60.



Figur 60: Sekvensdiagram for PSoC2 software ved affyring af kanonen

For at affyringssekvensen initieres, skal der modtages et interrupt. Det sker, når brugeren affyrer kanonen, og PSoC1 sender et højt signal, som går ind på en inputpind på PSoC2, som så er programmeret til at trigge på rising edge og derved køre interruptrutinen.

Det første der sker i interruptrutinen er, at ADC-værdien aflæses som en spænding i millivolt. Blokken til ADC'en, som er anvendt i koden ses på figur 61, og hvilke porte på PSoC2, de er forbundet til ses på figur 62. Spændingen fra ADC'en lægges over i variablen "resVolt". Så startes motoren, derved begynder kanonen at affyre. PSoC'ens røde LED er af debugginghensyn som udgangspunkt tændt, når der ikke affyres, når interruptet kommer, slukkes den, og den grønne PSoC-LED tændes i stedet, for til debugging at indikere, at interruptrutinen køres.



Figur 61: PSoC2 software PSoC-blokke

Dernæst anvendes spændingen fra resVolt, der blev aflæst fra ADC'en så til at vurdere om fotodioden, kan se lyset fra den røde LED. Når de ikke kan se hinanden er spændingen 500 mV, og når de kan se hinanden er spændingen mellem 4000 og 5000 mV. Normalt indikerer det, at de kan se hinanden, at affyringen er færdig, og at motoren skal stoppes, men hvis de kan se hinanden på dette tidspunkt ved interruptets start, betyder det, at de, inden affyringen er startet, allerede er placeret så de kan se hinanden. Hvis det er tilfældet køres et delay på 30 ms, for at sikre at motoren er drejet, så de igen ikke kan se hinanden. Derefter aflæses spændingen ved ADC'en igen - nu for at tjekke om affyringen er afsluttet.

	Name	Port	Pin	Lock
ADC_in	P2[6]	8	<input type="checkbox"/>	<input checked="" type="checkbox"/>
isr_in	P3[6]	17	<input type="checkbox"/>	<input checked="" type="checkbox"/>
opamp_m	P1[4]	41	<input type="checkbox"/>	<input checked="" type="checkbox"/>
opamp_out	P1[3]	40	<input type="checkbox"/>	<input checked="" type="checkbox"/>
opamp_p	P1[5]	42	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Pin_green	P0[2]	26	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Pin_red	P1[6]	43	<input type="checkbox"/>	<input checked="" type="checkbox"/>
PWM_led_out	P0[0]	24	<input type="checkbox"/>	<input checked="" type="checkbox"/>
PWM_motor_out	P0[1]	25	<input type="checkbox"/>	<input checked="" type="checkbox"/>

Figur 62: PSoC2 ports for software pins

Aflæsningen gentages indtil fotodioden kan se den røde LED, og motoren dermed skal stoppes. Affyringen sker på langt under et sekund, og det er derfor ikke problematisk med polling. Alternativt, hvis affyringen varede længere tid, kunne det have været implementeret med et interrupt, men i dette tilfælde kan det klares ved at tjekke ADC'en flere gange.

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

Både PWM-signalet til affyring af motoren og til generering af 10 kHz til den røde LED styres af PSoC2. De to PWM-blokke deles om en clock, der er sat til en frekvens på 2,5 MHz. Forbindelsen mellem clock'en og PWM-blokkene ses på figur 61, og hvilke porte PWM-blokkenes udgange er forbundet til ses på figur 62. PWM til den røde LED, har en periode på 250, dermed bliver PWM-signalet 10 kHz, som det også ses af ligning 1.

$$f_{LED} = \frac{f_{clock}}{\text{periode}_{LED}} \quad (1)$$

$$f_{LED} = \frac{2,5\text{MHz}}{250}$$

$$f_{LED} = 10\text{kHz}$$

Derudover er dutycyclen for LED'en 10 %. PWM-signalet til motoren har en periode på 75, derved bliver frekvensen 33,33 kHz. Beregningen for dette ses på ligning 2. En DC-motor skal gerne styres af et PWM-signal på over 20 kHz, for at rotere uden problemer. Dermed passer de 33,33 kHz godt.

$$f_{LED} = \frac{f_{clock}}{periode_{motor}} \quad (2)$$

$$f_{LED} = \frac{2,5MHz}{75}$$

$$f_{LED} = 33,33kHz$$

4.3 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/Dokumentation/WiiNunchuck.pdf):

$$AfkodetByte = (AflæstByte XOR 0x17) + 0x17$$

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

4.4 Kalibrering af Wii-Nunchuck Analog Stick

De afkodede bytes for Wii-Nunchuck's analoge stick har definerede standardværdier for dets forskellige fysiske positioner. (Bilag/Dokumentation/WiiNunchuck.pdf) Disse værdier findes i tabel 11

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 11: 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:

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

4.5 Hardwaredesign

På baggrund af BDD'et er der fundet følgende hardwareblokke, der skal udarbejdes:

- Tre motorer
- Motorstyring
- Affyringsmekanisme

Disse beskrives i de følgende afsnit.

4.5.1 Motor

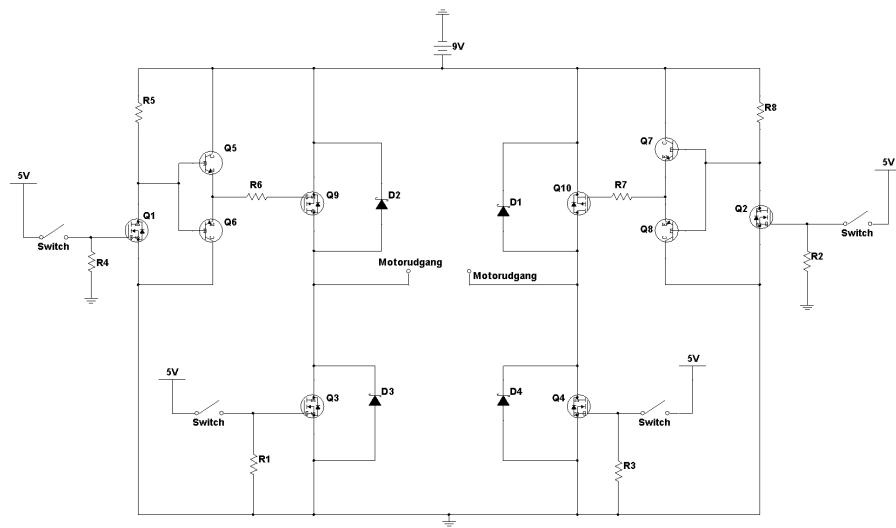
Der er valgt at bruge en DC-motor i alle tre tilfælde [6]. De to motorer skal bruges til at styre kanonen i to akser, og den sidste skal bruges i affyringsmekanismen.

4.5.2 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

Der blev først designet en H-bro, som bestod af to N-MOSFET's af typen IRLZ44(Bilag/Dokumentation/IRLZ44.pdf) og to P-MOSFET's af typen ZVP3306A. Det viste sig dog, at den P-MOSFET, der var brugt, ikke kunne klare strøm, som motoren skulle bruge, hvilket betød, at den blev brændt af. Derfor blev denne H-bro modifieret, så de to P-MOSFET's blev udskiftet med to MOSFET's af typen IRF9Z34N(Bilag/Dokumentation/irf9z34n p mosfet.pdf), der kan trække en større strøm.



Figur 63: Multisim kredsløb for H-bro

Betegnelse	Komponent
Q1	IRLZ44 (MOSFET N-kanal)
Q2	IRLZ44 (MOSFET N-kanal)
Q3	IRLZ44 (MOSFET N-kanal)
Q4	IRLZ44 (MOSFET N-kanal)
Q5	BC547
Q6	BC557
Q7	BC547
Q8	BC557
Q9	IRF9Z34N (MOSFET P-kanal)
Q10	IRF9Z34N (MOSFET P-kanal)
R1	10kΩ
R2	10kΩ
R3	10kΩ
R4	10kΩ
R5	100Ω
R6	100Ω
R7	100Ω
R8	10kΩ
D1	IN5819
D2	IN5819
D3	IN5819
D4	IN5819

Tabel 12: Komponentbetegnelser på H-bro

- MOSFET'er

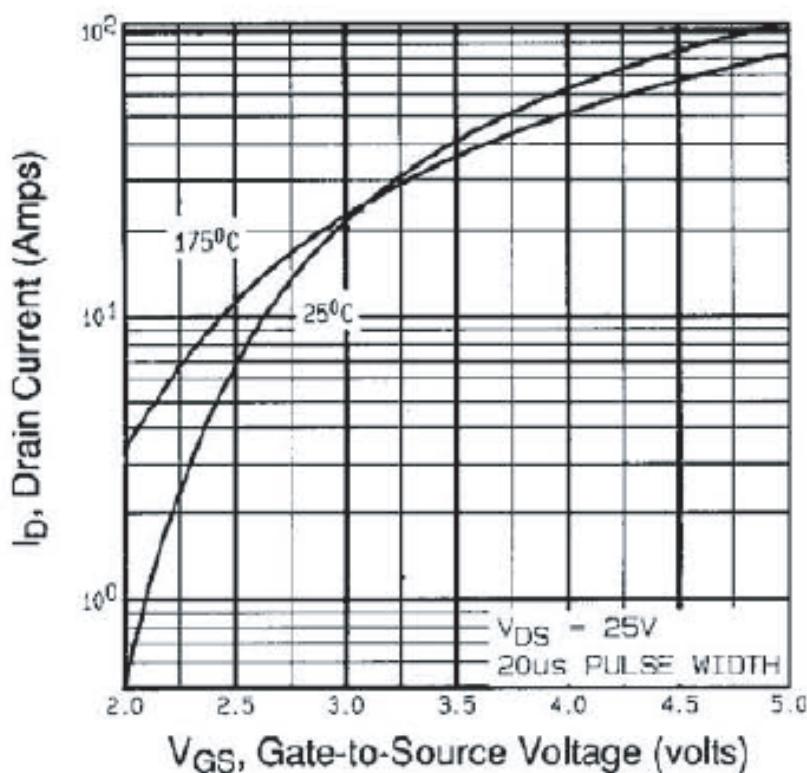
Til at styre motoren er der bygget en H-bro, som består af fire mosfet, hvor to af dem er af typen IRF9Z34N (mosfet P-channel, som er Q9

og Q10 på figur 63) og de to andre mosfet er af typen IRLZ44 (mosfet N-Channel, som er Q3 og Q4 på figur 63). Det er valgt at bruge mosfet for at kunne styre H-broen, da det ved denne er muligt at lukke og åbne for spændingen, og de bliver styret af spænding, i forhold til transistorer, som bliver styret af strøm.

- MOSFET N-kanal

Der er i denne H-bro brugt en N-kanals-MOSFET af typen IRLZ44 (Bilag/Dokumentation/IRLZ44.pdf). Denne MOSFET skal bruges til at trække spændingen fra den tilsvarende P-MOSFET til stel, så motoren kan begynde at køre. Det sker, når der kommer 5V ind på gate-benet.

MOSFET'en fungerer på den måde, at når der kommer positiv spænding ind på gate-benet åbner den, så der kommer forbindelse til stel og når der kommer 0V ind på dette lukker den igen.



Figur 64: Gate-to-Source Voltage(Bilag/Dokumentation/IRLZ44.pdf)

På figur 64 ses det, at når der er en gate-to-source-spænding på 5V, vil der MOSFET'en kunne klare, at der løber en strøm på op til 100A i følge datablad(Bilag/Dokumentation/IRLZ44.pdf). Det vil

altså ikke komme til at påvirke motoren, da denne kun kan trække en strøm på cirka 60mA.

- MOSFET P-kanal

Der er valgt at bruge en P-MOSFET af typen IRF9Z34N (Bilag/Dokumentation/irf9z34n p mosfet.pdf). Denne MOSFET skal bruges til at trække de 9V ned til motoren, så denne kan køre. Samtidig sørger den for, at de 9V ikke løber ned til motoren så længe, der ikke er negativ spænding på gate-benet. Denne type MOSFET kan trække en strøm på 6,7A ifølge databladet (Bilag/Dokumentation/irf9z34n p mosfet.pdf). Det vil altså ikke komme til at påvirke motoren, da den kun kan trække en strøm på cirka 60mA.

For at der kan løbe spænding igennem IRF9Z34N, skal den have en negativ spænding for at åbne og en spænding på over 0V for at lukke.

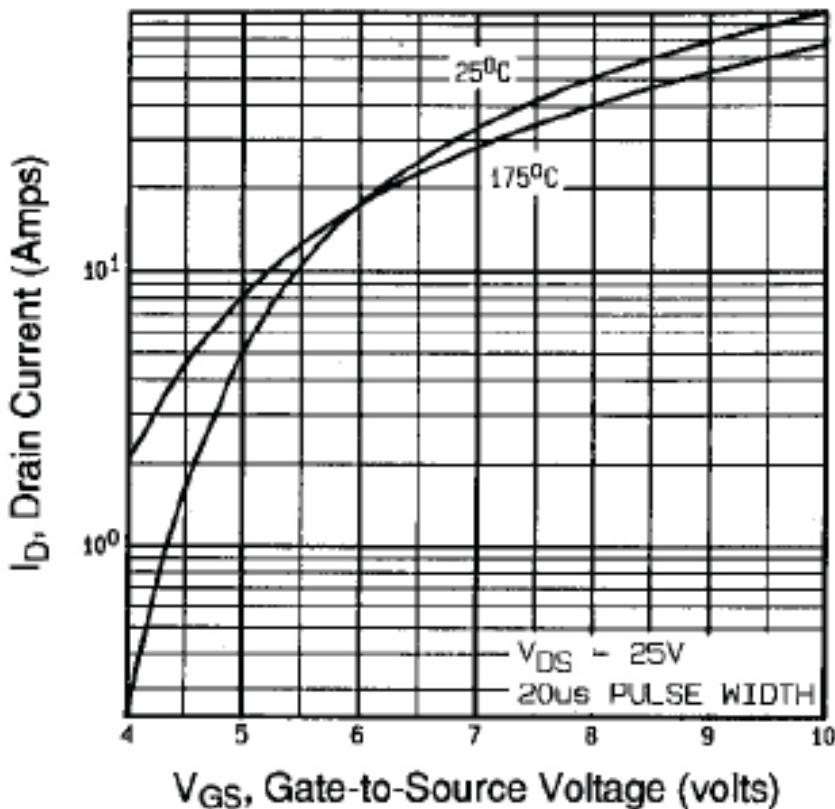


Fig. 3 - Typical Transfer Characteristics

Figur 65: Gate-to-Source Voltage (Bilag/Dokumentation/irf9z34n p mosfet.pdf)

På figuren 65 ses det, at når der er en gate-to-source-spænding på 5V,

vil der kunne løbe en strøm på omkring 5A igennem MOSFET'en, hvilket er mere end nok til at få motoren til at køre.

- Dioder

Som det se på figur 63 er der sat en diode af typen IN5819 (Bilag/Dokumentation/IN5819.pdf) over fire af MOSFET'ene (Q9, Q10, Q3 og Q4). Disse skal fungere som beskyttelse af MOSFET'en. Det, dioden gør, er, at den sikrer, at den spænding, som er tilbage i motoren, når der lukkes for MOSFET'en, ikke løber tilbage ind i MOSFET'en og brænder den af.

- Modstande

- Pull down modstande:

Der er blevet brugt fire pull-down-modstande (R1, R2, R3 og R4 som ses på figur 63). Disse sørger for, at signalet vil blive holdt lavt, når der ikke sendes signal ind på MOSFET'ens gateben. Hvis der blev sendt signal ind imens der også blev sendt signal ind fra den anden side af H-broen ville MOSFET'en blive brændt af. Altså skal modstanden være lille nok til, at de små spændinger kan løbe til stel, når der ikke er PWM-signal, men samtidig stor nok til, at spændingen ikke løber til stel, når der er signal på gatebenet. Der er derfor valgt en modstand med en værdi på $10k\Omega$.

- Andre modstande

- * R6 og R7

Grunden til, at R6 og R7(på figur 63), er der, er for at sikre, at transistorernes Absolute Maximum Ratings omkring strømmen, som ikke må overstige 100mA ifølge databladet (Bilag/Dokumentation/BC557.pdf) og (Bilag/Dokumentation/BC547.pdf)

$$R6 = R7 = \frac{9V}{100mA} = 90\Omega$$

Der blev valgt en modstand med en værdi på 100Ω i stedet, for at være på den sikre side.

- * R5 og R8 (jf. figur 63)

Grunden til at R5 og R8 er indsat i kredsløbet er, at der ifølge databladet kun kan løbe en strøm på omkring 30A igennem N-MOSFET, hvis Vgs er på 10V. Da Vgs, i dette projekt, kun er sat til 5V, vil MOSFET'en altså ikke kunne klare en alt for stor strøm. Derfor er R5 og R8 sat ind for at forhindre, at MOSFET'en ikke brænder af. (Bilag/Dokumentation/IRLZ44.pdf).

Der blev fundet en modstand ved at regn med at der 9V og at mosfet kun kan klar en strøm under 30A så der blev regnet med de 30 A selv om man vidste godt det ikke var det som den kunne klare, men der skulle en større modstand ind, men det var så man havde noget at gå ud fra

$$R8 = R5 = \frac{9V}{30A} = 0.3\Omega$$

0.3 var alt for lille så der blev prøvet op end til der blev fundet en som passet, hvor det blev en på $10K$ så er man sikker på der ikke sker noget med mosfet'en

- Transistorer

- Q5 og Q7(Bilag/Dokumentation/BC557.pdf) (jf. figur 63) Disse transistorer sidder i kredsløbet, fordi det tager tid for P-MOSFET'en at blive opladt helt og dermed åbne helt, på grund af kondensatoreffekten mellem benene på MOSFET'en.
- Q6 og Q8(Bilag/Dokumentation/BC547.pdf) (jf. figur 63) Disse to transistorer sidder der for at hjælpe med at lukke P-MOSFET'en igen. Inden Q5 og Q7 blev sat ind tog det tid for at lukke P-MOSFET'en, men da de to blev sat ind, kunne de hjælpe til med at aflade MOSFET'en hurtigere.

Rotationsbegrænsning

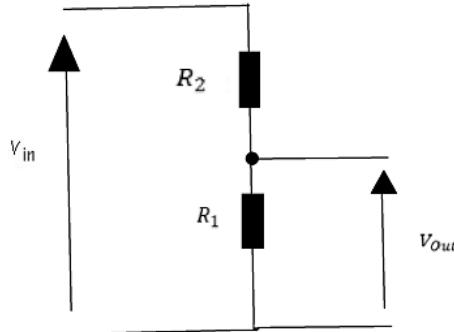
Platformen, som styres af motoren, må ikke kunne rotere 360 grader. Dette ses i ikke funktionelle krav, afsnit 1.5. For at begrænse motorens bevægelse, anvendes et potentiometer samt en ADC. Når motoren bevæger sig, ændres potentiometerets modstandsværdi, og dermed ændres spændingsniveauet. På figur 66 ses den endelige opstilling af rotationsbegrænsningen.



Figur 66: Opstilling for rotationsbegrænsning

Potentiometer

Den første del af rotationsbegrænsningen er et potentiometer, som fungerer efter spændingsdelerprincippet, som vist på figur 67.



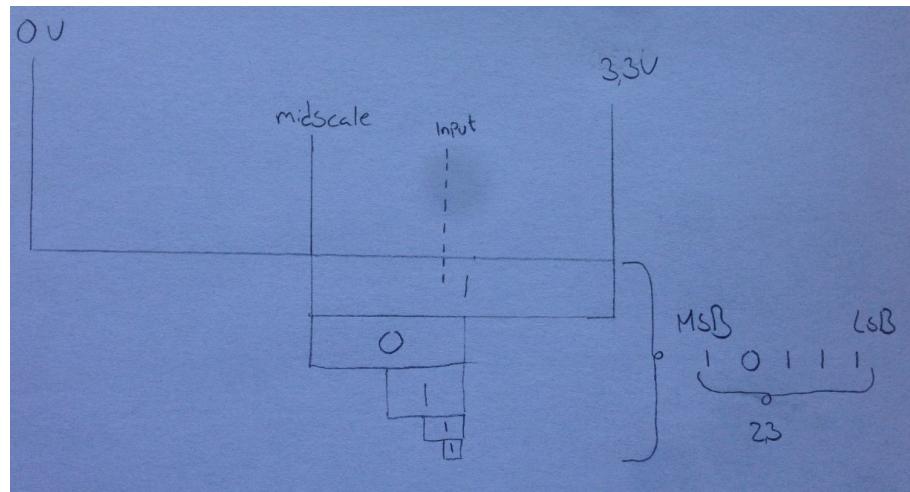
Figur 67: Spændingsdeler formlen for potentiometeret

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. Når skaftet på potentiometret roteres ændres modstanden i de to variable modstande, R_1 og R_2 . På figur 67 ses en konceptuel afbildning af de variable modstande i potentiometret, hvor der ses at udgangsspændingen er spændingen over R_1 .

ADC

For at kunne aflæse spændingen på potentiometeret, anvendes en 12-bit AD converter(Bilag/Dokumentation/ADC_SAR_SEQ_P4_v2_30.pdf) af typen Sequencing Successive Approximation ADC. En sequencing SAR ADC indeholder et sample-hold kredsløb. Kredsløbet holder på et indgangssignal indtil det næste signal registreres på kredsløbets indgang. Dermed har converteren tid til at bestemme outputværdien.

ADC'en fungerer ved at midscale indstilles til halvdelen af referencespændingen. Inputsignalet sammenlignes med midscale, hvis inputsignalet er højere end midscaleværdien sættes MSB 1 og hvis signalet er lavere bliver denne sat til 0. Herefter bliver midscale værdien rekursivt sat til havdelen af intervallet, som inputsignalet ligger indenfor, og bittene ned til LSB bliver sat. Bittene der sættes, bliver gemt i et register. Når konverteringen er gennemført, kan værdien af inputsignalet aflæses i dette register.



Figur 68: Illustration af AD konvertering

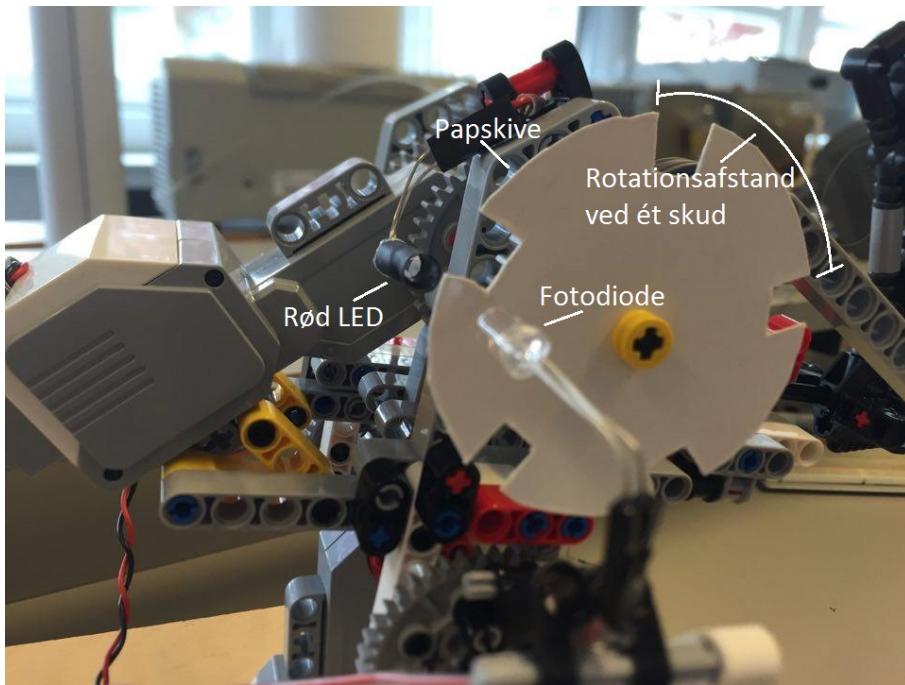
På figur 68 ses et eksempel over de første fem trin i en konvertering. Til dette projekt arbejdes der med en 12-bit AD converter, dermed ville denne konvertering forsætte 12 trin ned.

4.5.3 Affyringsmekanisme

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

Rotationsdetektor

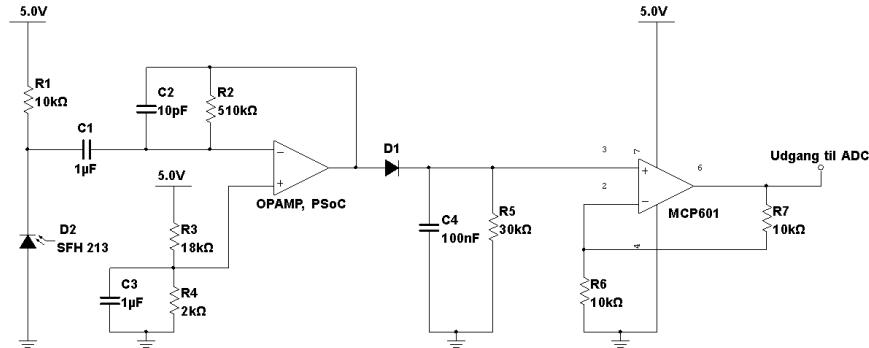
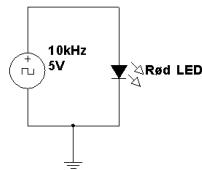
Når kanonen affyres, styres det af motoren, og som mekanikken er opbygget, er der et proportionelt forhold mellem omdrejning på motoren og antal skud, der affyres. Derfor er det væsentligt at vide, hvornår motoren har roteret en runde, så den kan stoppes, inden der igen skydes. Til det formål anvendes detektoren. Billedet på figur 69 illustrerer hvordan detektoren anvendes.



Figur 69: Detektorens placering på affyringsmekanismen

Den røde LED og fotodioden anbringes på affyringsmekanismen, som det ses på figur 69. De vender ind mod hinanden, men er adskilt af papskiven. Papskiven er forbundet til motoren's 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 70 ses et kredsløbsdiagram for detektoren og LED'en.

Detektorkredsløb**LED-kredsløb**

Figur 70: Kredsløbsdiagram for detektoren

Båndpasfiltret er opbygget af et højpasfilter, et lavpasfilter og en operationsforstærker. Da PSoC'en har en indbygget operationsforstærker, anvendes denne. Software design og implementering af den ses under afsnit 4.2.6. Højpasfilterets afskæringsfrekvens er beregnet på følgende måde:

$$f_H = \frac{1}{2\pi R_H C_H} = \frac{1}{2\pi \cdot 10k\Omega \cdot 1\mu F} = 15,9 Hz \quad (3)$$

Og lavpasfilterets afskæringsfrekvens er beregnet på følgende måde:

$$f_L = \frac{1}{2\pi R_L C_L} = \frac{1}{2\pi \cdot 510k\Omega \cdot 10pF} = 31,2 kHz \quad (4)$$

I begge beregninger gælder det, at R er filterets modstandsværdi og C er filterets kondensatorværdi.

Da båndpasfilteret har en meget stor båndbredde kan det ikke undgås, at der slipper nogle DC-lyssignaler igennem alligevel. Kondensatoren C1 skal sørge for at frasortere de fleste af disse. Men da filteret virker som ønsket gør det ikke noget.

Af hensyn til operationsforstærkeren, er der valgt en referencespænding på 0,5 V på den positive indgang. Det er opnået ved en spændingsdeler, for hvilken beregning ses herunder:

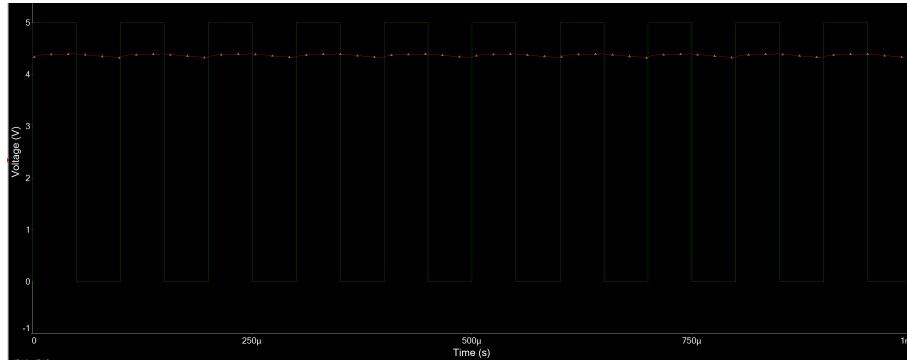
$$\begin{aligned} U_{Ref} &= U_{VCC} \cdot \frac{R_2}{R_1 + R_2} \\ \Rightarrow 0,5V &= 5V \cdot \frac{R_2}{18k\Omega + R_2} \\ \Rightarrow R_2 &= 2k\Omega \end{aligned} \quad (5)$$

hvor U_{Ref} er den ønskede referencespænding, U_{VCC} er forsyningsspændingen, R_1 ønskes at have en værdi på $18k\Omega$ og R_2 er den modstandsværdi, der ønskes beregnet. Kondensatoren C3 sikrer, at referencespændingen holdes på samme niveau hele tiden.

Der er negativ feedback på operationsforstærkeren, hvilket sikrer, at der oprettholdes samme spænding, 0,5 V, på begge indgange i operationsforstærkeren. Når fotodioden kan se den røde LED, genererer den en strøm, som bliver omsat til en spænding i kredsløbet. Operationsforstærkeren vil opretholde 0,5 V på den negative indgang. Den vil derfor regulere udgangen for at ophæve de ændringer, som fotodioden skaber på den negative indgang. Udgangssignalet vil dermed afspejle det PWM-signal, som den røde LED sender.

Når fotodioden kan se lyset fra den røde LED er signalet, som kommer fra udgangen af operationsforstærkeren, et firkantsignal med en frekvens på 10 kHz,. Når fotodioden ikke kan se det røde lys, er spændingen på udgangen 0,5 V. Det ønskes omdannet til et signal, der går højt, når PWM-signalet starter, og går lavt, når PWM-signalet er væk igen. For at opnå dette, blev der lavet en envelopedetector, som er opbygget af en diode, en modstand og en kondensator. Dioden sikrer, at skulle der komme negative spændinger, så vil de blive frasorteret. Da dioden er af typen Scottky har den et lavt spændningsfald over sig, hvilket gør, at der vil være større afvigelser end hvis der var anvendt en anden slags diode.

Kondensatoren er dimentioneeret efter, at den bliver opladet på de første udsving fra firkantsignalet. Modstanden er dimentioneeret, så spændingen ikke aflades mellem svingningerne på 10 kHz signalet. En simulering af dette kan ses på figur 71.



Figur 71: Simulering af envelope detectoren

På simuleringen ses det, at envelopedektoren virker, da det fremgår, at kondensatoren ikke når at aflade, inden der kommer en ny puls fra PWM-signalet. Derudover ses det også, at den når at lade op inden den første puls fra PWM-signalet kommer. Det betyder, at det signal der kommer ud ligger på cirka 4,3V.

Outputtet fra enveloppedektoren var dog noget lavt. HEJ. Det lå mellem 1,5V og 2V. Derfor blev der indsats en ikke-inverterende forstærker for at fordoble signalet. Forstærkeren består af en opamp og to modstande. For at eftervise, at signalet fordobles, hvis de to modstande i forstærkerkredsløbet er ens, blev følgende beregning foretaget (Bilag/Dokumentation/Analogteknik.pdf):

$$\begin{aligned} U_O &= \left(1 + \frac{R_2}{R_1}\right) \cdot U_P & (6) \\ \Rightarrow U_O &= \left(1 + \frac{10k\Omega}{10k\Omega}\right) \cdot 2V = 4V \end{aligned}$$

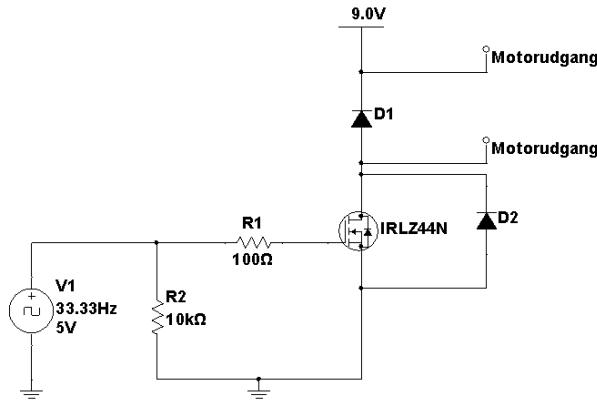
Herefter var det muligt at aflæse et tydeligt firkantsignal med en peak-to-peak værdi på 3,5V.

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

Der kunne overvejes at indsætte en kondensator fysisk tæt på systemet og parallelt med R1 og fotodioden. Denne ville sikre, at VCC-niveauet opretholdes, hvis der pludselig skete et fald i VCC.

Motorstyring

Til at styre affyringsmekanismens motor er der bygget et kredsløb med en MOSFET af typen IRLZ44 som primære komponent. Denne skal fungere som en switch, da den skal sørge for, at motoren kun kører, når der bliver sendt PWM-signal ind i den. Kredsløbsdiagrammet kan ses på figur 72.



Figur 72: Diagram over motorstyring til motor på affyringsmekanisme

Der ønskes i dette tilfælde at kunne styre motorenens hastighed. Derfor anvendes et PWM-signal, som sendes ind på gatebenet og styrer motoren. Hastigheden kan så styres ved hjælp af PWM-signalets dutycycle. Hvis ikke MOSFET'en var der, ville motoren køre med fuld hastighed hele tiden, hvilket i dette tilfælde ikke ville være optimalt.

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.

Modstanden R2 er en pull-down-modstand. Den hjælper altså til at trække de små spændinger til stel, når PWM-signalet er slukket. Når PWM-signalet er tændt, er den stor nok til, at strømmen ikke vil løbe til stel. Modstanden R1 skal sørge for, at der ikke kommer til at være en spænding på 9V på microprocessoren. Dette kunne ske, hvis der sker en kortvarig fejl i MOSFET'en.

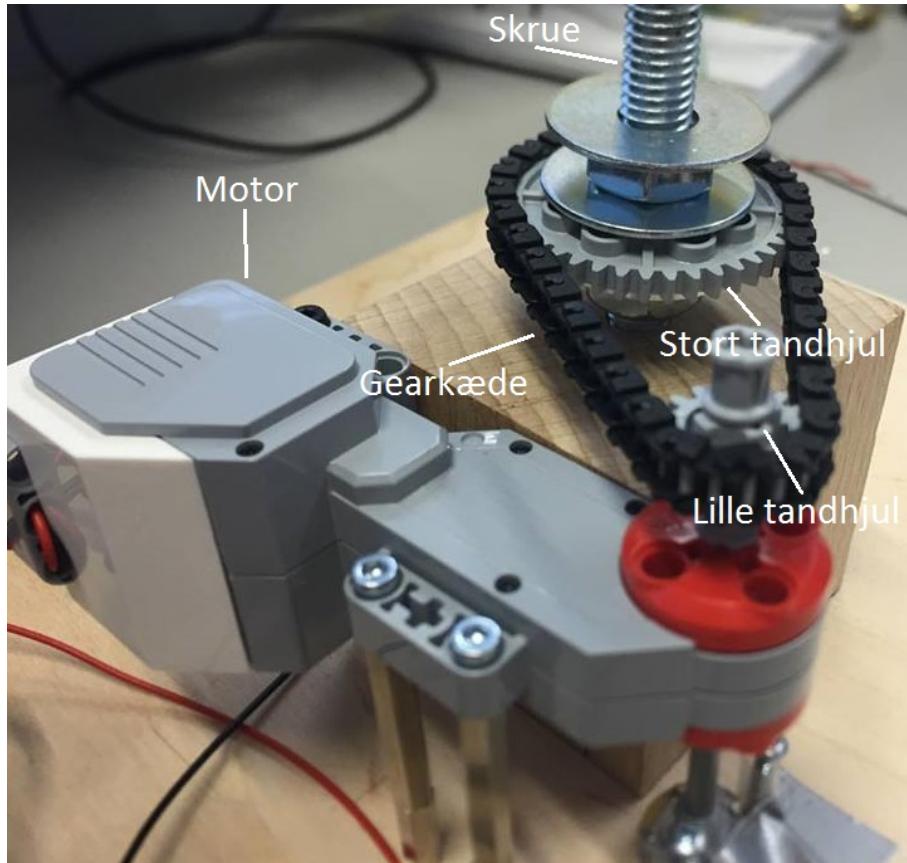
Kanon og platform

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 73. 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 indsatt en møtrik, så den kan skrues på skruen. Forholdet mellem det store og det mellemste tandhjul gør, at rotationshastigheden bliver gearet ned med et forhold på 1:0,6. Forholdet er fundet ved hjælp af følgende beregning:

$$\begin{aligned} gearing &= \frac{t_m}{t_s} \\ gearing &= \frac{24}{40} = 0,6 \end{aligned} \tag{7}$$

Her er t_m antallet af tandhjul på det mellemste tandhjul og t_s er antallet af tandhjul på det store tandhjul.

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.

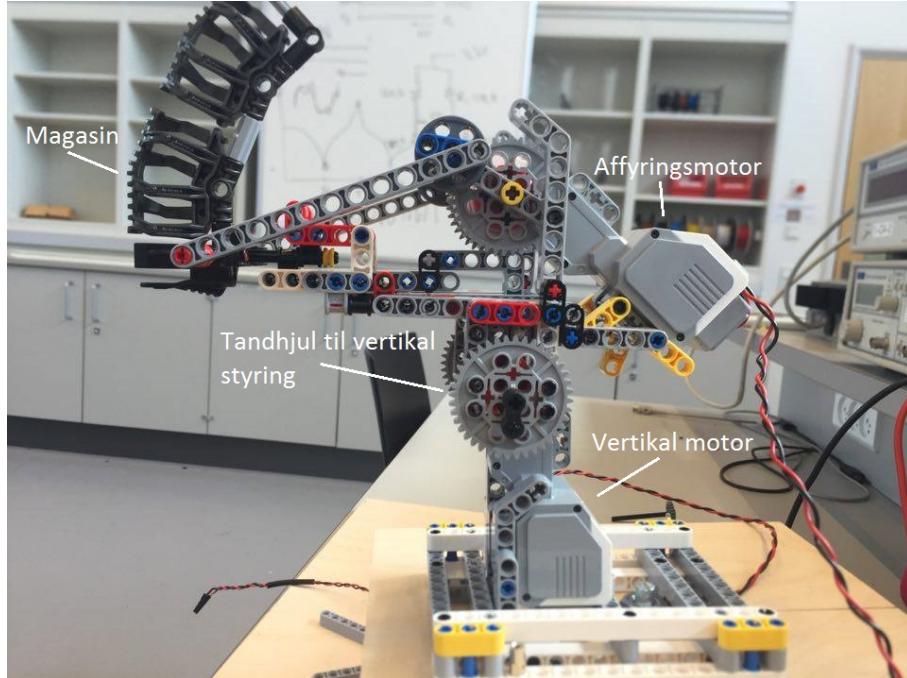


Figur 73: Horisontal mekanik

Mekanikken for den vertikale retning er bygget af LEGO. Et billede af opbygningen ses på figur 74. Styringen af den vertikale bevægelse bliver håndteret af den vertikale motor, som ses på figur 74. 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.

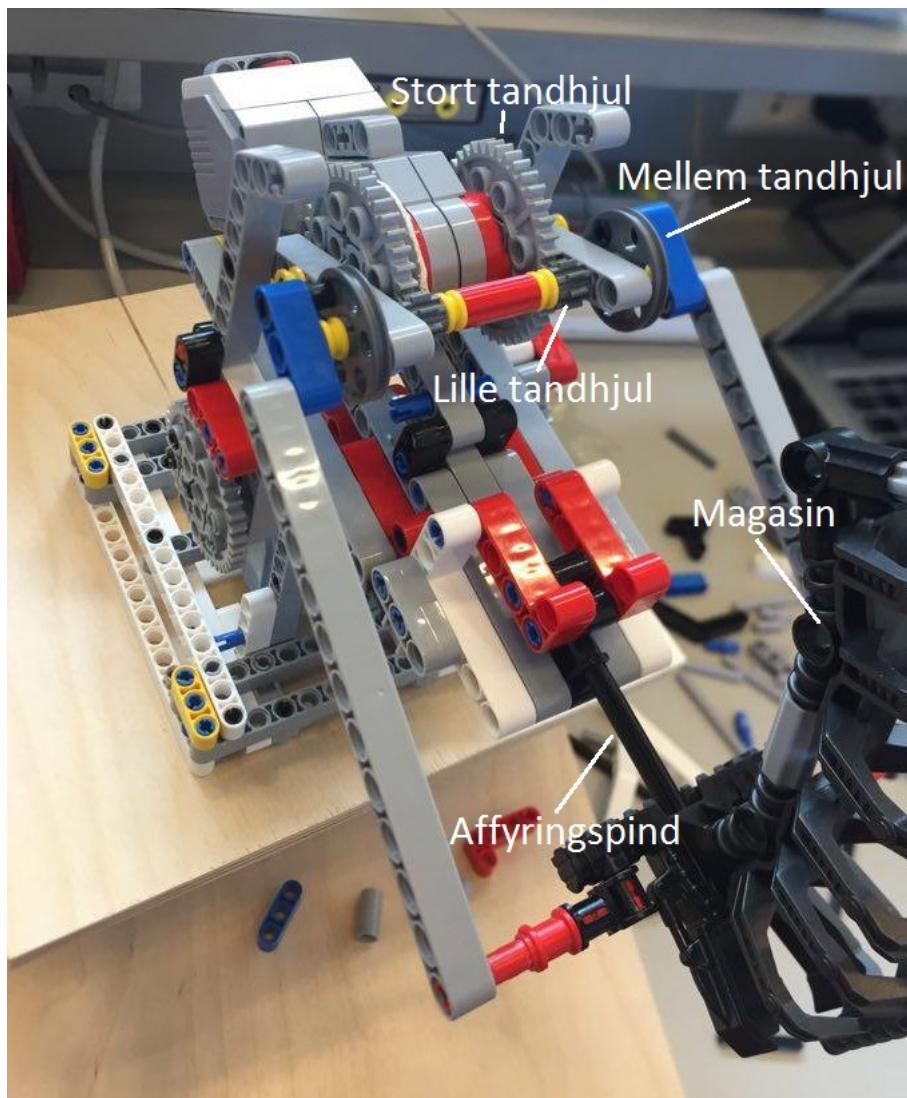
Ligesom den vertikale styring er selve kanonen også bygget i LEGO. Den har et magasin, som det fremgår af figur 74. Der kan kommes slik i magasinet, som så bliver affyret. Affyringen styres af den anden motor på figur 74. Når affyringsmotoren drejer bliver to større tandhjul roteret. De to tandhjul er 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 74: Kanon i LEGO

For at beregne hvor stor gearing der er mellem de forskellige tandhjul opstilles nogle beregninger. Her defineres t_s til at være antallet af tænder på det store tandhjul, t_m er antallet af tænder på det mellemste tandhjul og t_l er antallet af tænder på det mindste tandhjul. På figur 75 ses, hvilke tandhjul der henvises til.



Figur 75: Kanon i LEGO set oppefra

Forholdet mellem det store og det lille tandhjul er fundet ved beregningen

$$\text{gearing} = \frac{t_s}{t_l} \quad (8)$$

$$\text{gearing} = \frac{40}{8} = 5$$

Med denne gearing, får kanonen mulighed for at skyde 5 gange så kraftigt, som den ville have gjort uden gearing.

5 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.

5.1 Software

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

5.1.1 Modultest af Wii-Nunchuck

På PSoC0 er der software til aflæsning af Wii-Nunchuck input data (Bilag/Dokumentation/WiiNunchuck.pdf). Følgende afsnit beskriver test af dette software.

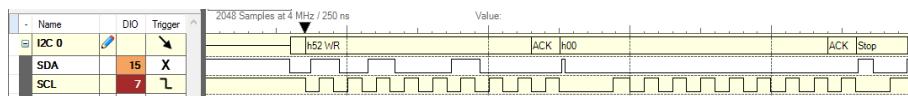
Aflæsning af Wii-Nunchuck sker i to skridt, som begge verificeres ved modultest. Først skal der sendes et *Handshake* fra PSoC0 til Wii-Nunchuck for at initialisere data udveksling, og herefter sker data udveksling hver gang PSoC0 sender en anmodning om det. Disse to skridt modultestes her.

PSoC0 blev programmeret til kontinuert aflæsning af Wii-Nunchuck. For at verificere data udveksling mellem PSoC0 og Wii-Nunchuck blev I2C bussen målt ved brug af Logic Analyzer fra Analog Discovery.

Data udveksling sker i to skridt. Først sender PSoC0 en byte med værdien 0 (0x00 i hexadecimal). Herefter sker den faktiske aflæsning, PSoC0 aflæser Wii-Nunchuck. Begge skridt testes her.

Test af Wii-Nunchuck Handshake

Den første forventede I2C besked er en *0x00* byte fra PSoC0 for at starte en ny aflæsning. På figur 76 ses aflæsningen af I2C bussen på tidspunktet hvor anmodningen til Wii-Nunchuck bliver udført. Dette er en tidslinje læst fra venstre til højre.



Figur 76: Aflæsning af I2C kommunikation mellem PSoC0 og Nunchuck

Det kan på figur 76 ses at den første besked der måles er af typen "WR" (Write) til addressen 0x52 (Wii-Nunchuck I2C Slave Addressen). Hertil kommer et tilhørende *ACK* (Acknowledge) fra Wii-Nunchuck. Til sidst sendes dataen Ox00 efterfulgt af et ACK fra Wii-Nunchuck. Til sidst afsluttes I2C transaktionen

ved "Stop".

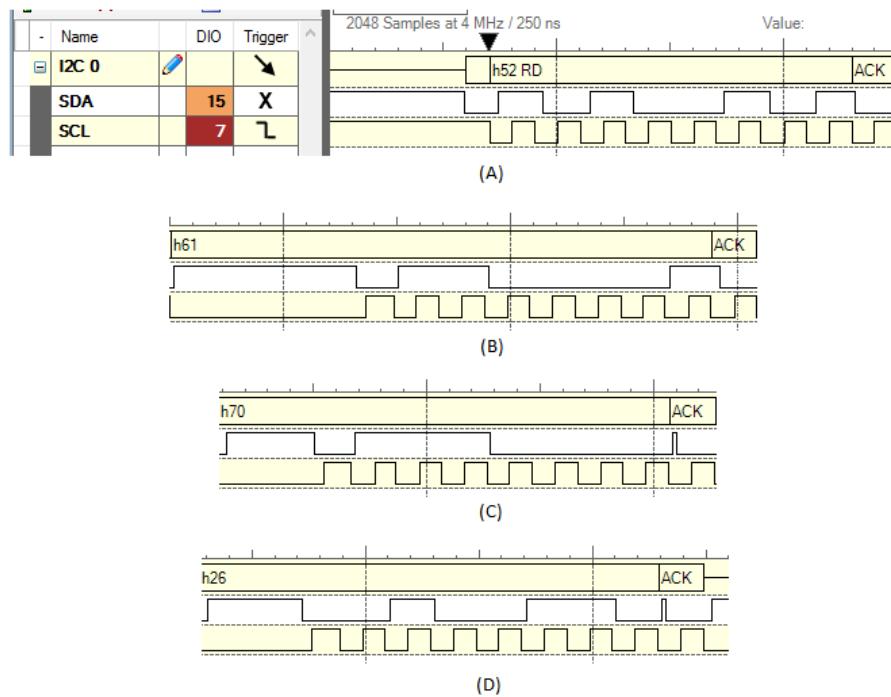
Det kan altså konkluderes at målingen er i overensstemmelse med forventningen om at en 0x00 byte skal sendes til Wii-Nunchuck for opstart af dataudveksling.

Aflæsning af Wii-Nunchuck

Efter vellykket afsendelse af 0x00 byten sker den egentlige aflæsning af Wii-Nunchuk input dataen.

Her forventes en række databytes i intervallerne beskrevet i bilag #ref [9]

På figur 77 ses I2C beskederne der bliver udvekslet mellem PSoC0 og Wii-Nunchuck efter vellykket Wii-Nunchuck Handshake.



Figur 77: Tidslinje af aflæste I2C beskeder af PSoC0 fra Wii-Nunchuck

På figur 77 ses logic analyzerens aflæsning af returværdierne fra Nunchucken. Disse værdier blev alle aflæst til at stemme overens med det forventede resultat.

Ud fra disse to modultests, er det konkluderet at dette modul fungerer efter hensigten.

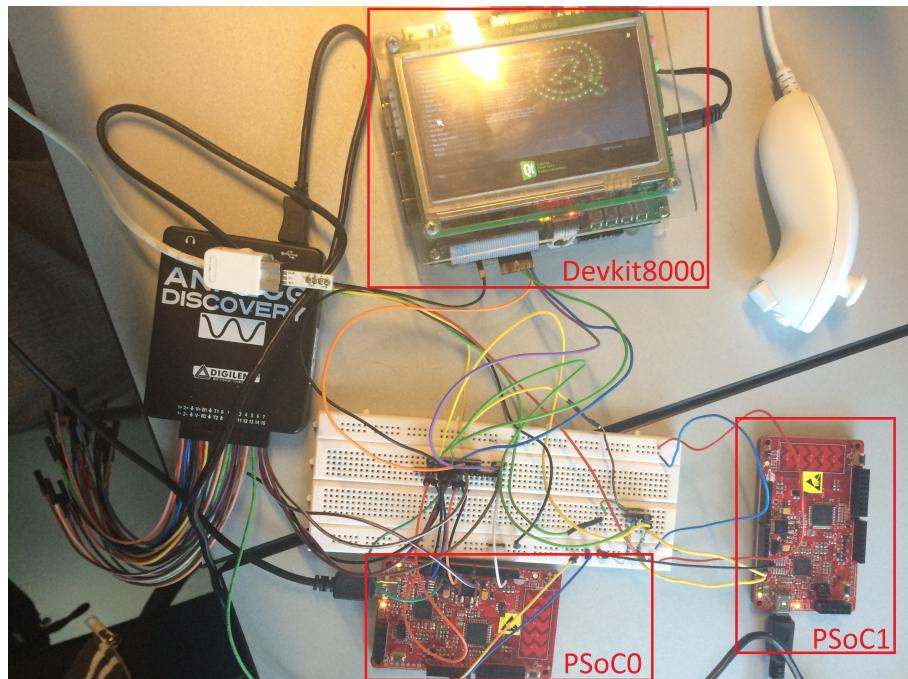
5.1.2 SPI Protokol

Devkit 8000 kommunikerer over en SPI bus. Kommunikationen følger SPI komunikations protokollen, som er beskrevet i afsnit 3.7.1. Dette afsnit beskriver test af denne protokol.

SPI Bus Test

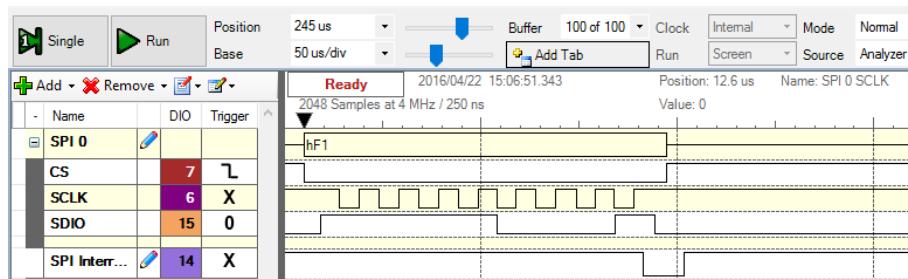
Test af SPI bussen udføres i to dele. Første del af testen udføres ved at sende kommandotypen for start af SPI test i terminalen, og derefter verificere den sendte data vha. Analog Discovery. Den anden del af testen, er at aflæse returnbeskeden på bussen med Analog Discovery. Målet med denne test, er at læse kommandotypen "SPI_OK", der indikerer en successful aflæsning af SPI-slaven. På figur 78 ses test opsætningen. Devkit 8000, PSoCs, SPI-bus og Analog Discovery er forbundet igennem fumlebrættet. PSoC0 og PSoC1's I2C-bus forbindes til nunchucken igennem fumlebrættet, adskilt fra SPI-forbindelserne.

SPI kommandotyper kan ses i afsnit 3.7.1, tabel 5.



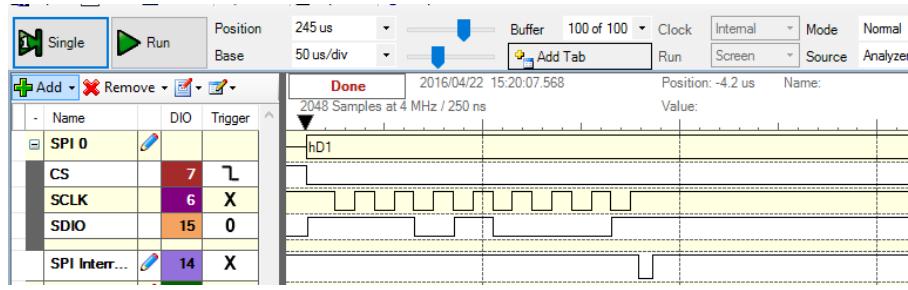
Figur 78: Opsætning for SPI-test

I testen afsendes SPI test kommandotypen, som har værdien 0xF1 jf. tabel 5. For at verificere, at kommandotypen er sendt ud på SPI-bussen korrekt, blev Analog Discovery's 'Logic Analyzer' brugt til at aflæse SPI-bussen. Det var forventet at 0xF1 blev aflæst på bussen. Det afmålte resultat stemte overens med det forventede. Målingen ses på figur 79.



Figur 79: Måling af kommandotype for start SPI test

Hvis SPI-kommunikationen fungerer korrekt, er det forventet at der aflæses 0xD1 på SPI-bussen, da dette betyder 'SPI_OK' jf. tabel 5. Måling af returbeskeden ses på figur 80.



Figur 80: Måling af returbesked for start SPI test

På figur 80 ses at returbeskeden har den forventede værdi 0xD1, som indikerer at SPI bus testens første del er gennemført uden fejl.

Ud fra testenes resultater kan det konkluderes at implementeringen af SPI protokollen fungerer efter hensigten.

5.1.3 I2C Protokol

PSoC0 og PSoC1 kommunikerer via I2C protokollen beskrevet i afsnit 3.7.2. Dette afsnit beskriver test af denne protokol. Følgende test tager udgangspunkt i kommandotypen *NunchuckData* beskrevet i tabel 7, i afsnit 3.7.2.

Test af NunchuckData kommandotype

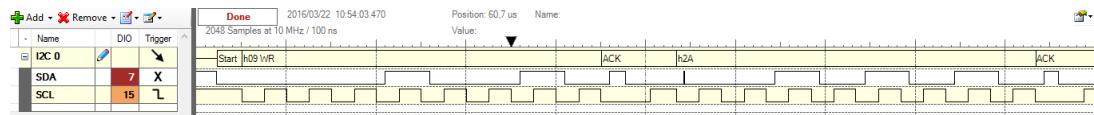
Testen blev udført i to dele. I første del måles I2C bussen ved brug af Analog Discovery's Logic Analyzer; for at verificere at den forventede kommandotype bliver overført via bussen. Anden del verificerer at den overførte data er modtaget korrekt via PSoC Creator's debugger.

NunchuckData kommandotype test del 1

I testen afsendes, som nævnt i afschnittets indledning, kommandotypen Nunchuck-Data. Som vist i tabel 7, afsnit 3.7.2 har denne kommandotype ID'et 0xA2,

hvor de efterfølgende 3 data bytes indeholder input dataen fra Wii-Nunchuck.

Det forventede resultat af målingen er at første byte er kommandoentypens ID, som har værdien 0x2A. Kommandoens data - de efterfølgende bytes - verificeres først i anden del, disse indgår altså ikke i følgende måling. Målingen ses på figur 81.



Figur 81: Tidslinje af målt I2C kommandotype

Det kan ses på figur 81 at I2C overførslen starter med en I2C *write*, som får et successfult acknowledge fra slaven PSoC1. Herefter kan det ses at den næste byte der sendes har værdien 0x2A. Denne byte er kommandoentypens ID, og er altså som forventet 0x2A.

Det kan altså verificeres at kommandoen overføres via I2C bussen. Dataens integritet er dog ikke inkluderet i denne del, og testes i del 2.

NunchuckData kommandotype test del 2

For at verificere integriteten af den data der sendes mellem PSoC0 og PSoC1, bruges PSoC Creators indbyggede debugger. Igennem er det kommandotyphen NunchuckData der sendes mellem de to enheder, hvor de medfølgende data bytes fortolkes.

Testen gennemføres ved at fortolke den modtagne data tre gange, hvor nunchucken er i forskellige tilstande (hvilken retning det analoge stik er trykket) i hver test. Værdierne sammenlignes med de forventede standardværdier som ses i tabellen på side 3 i (Bilag/Dokumentation/WiiNunchuck.pdf). Da testene kun er fokuserede på, hvilken retning den analoge stick er presset, er det altså kun receivedDataBuffer[1] (den analoge stick x-akse) og receivedDataBuffer[2] (den analoge pinds y-akse) der er relevante for testen. Når den analoge stick er presset til venstre, forventes det ifølge tabel 11 i afsnit 4.4 at receivedDataBuffer[1] er lig 0x1E og receivedDataBuffer[2] er 0x7B. Når den analoge stick er presset op, forventes det at receivedDataBuffer[1] er 0x7E og receivedDataBuffer[2] er 0xDF. Når der ikke er noget input på Nunchucken forventes det at receivedDataBuffer[1] er 0x7E og receivedDataBuffer[2] er 0x7B. Målingerne for testene kan ses på figur 82, 83 og 84.

Name	Value
+ receivedDataBuffer	0x2000012C (All)
receivedDataBuffer[2]	0x7D 'Y'
receivedDataBuffer[3]	0xFB '373'
receivedDataBuffer[1]	0x7F '177'
Click here to add	

Figur 82: Aflæst Nunchuck data i modtager-buffer. Intet input på Nunchuck'en

Name	Value
+ receivedDataBuffer	0x2000012C (All)
receivedDataBuffer[2]	0x7C ' '
receivedDataBuffer[3]	0xC3 '303'
receivedDataBuffer[1]	0x1E '036'
Click here to add	

Figur 83: Aflæst Nunchuck data i modtager-buffer. Den analoge stick er presset til venstre på Nunchuck'en

Name	Value
+ receivedDataBuffer	0x2000012C (All)
receivedDataBuffer[2]	0xDF '337'
receivedDataBuffer[3]	0xB3 '263'
receivedDataBuffer[1]	0x82 '202'
Click here to add	

Figur 84: Aflæst Nunchuck data i modtager-buffer. Den analoge stick er presset frem på Nunchuck'en

På figur 83 ses afmålingen af modtager-bufferen når Nunchucks analoge stick er presset helt til venstre. ReceivedDataBuffer[1] blev aflæst til 0x1E og receivedDataBuffer[2] blev aflæst til 0x7C. ReceivedDataBuffer[1] stemmer overens med forventningerne. ReceivedDataBuffer[2] har en lille afvigelse (oversat til decimaltal blev der målt 124, hvor der forventes 123). Denne afvigelse kan skyldes, at det analoge stick ikke blev presset direkte til venstre, men at den også er blevet presset en smule frem under målingen.

På figur 84 ses afmålingen af modtager-bufferen når Nunchucks analoge stick er presset frem. ReceivedDataBuffer[1] blev aflæst til 0x82, hvor det var forventet 0x7E. Dette er en afvigelse fra de forventede resultater med 4, og kan

skyldes at det analoge stick ikke var helt centreret idét den blev presset frem under målingen. ReceivedDataBuffer[2] blev aflæst til 0xDF, hvilket stemmer overens med de forventede målinger.

På figur 82 ses afmålingen af modtager-bufferen når der ikke er noget brugerinput på nunchuckens analoge stick. ReceivedDataBuffer[1] blev aflæst til 0x7F, hvor det forventede resultat var 0x7E. Denne afvigelse kan skyldes at det analoge stick ikke stod helt i midten under målingen (Det analoge stick er lidt "løs" og kan defor godt finde hvile i en position der ikke er fuldt centreret). ReceivedDataBuffer[2] blev aflæst til 0x7D, hvor det forventede resultat var 0x7B. Igen kan denne afvigelse skyldes at det analoge stick ikke var i centrum under målingen.

Ud fra testen kan det konkluderes at implementeringen af I2C-protokollen fungerer efter hensigten.

5.1.4 Systemtest GUI

Devkit 8000 styres ved hjælp af en brugergrænseflade. I dette afsnit udføres en modultest af brugergrænsefladen ved at observere beskeder udskrevet til dens terminal.



Figur 85: Test setup for modultest af brugergrænseflade

Test af brugergrænsefladen udføres i tre dele og resultater registreres visuelt. Den første del af testen udføres ved tryk på Start Test-knappen, derefter verificeres det i konsol vinduet at systemtesten udføres. Den anden del udføres ved tryk på Clear-knappen. Målet ved denne del er at konsol vinduet cleares. Sidste del af testen udføres ved tryk på Exit-knappen. Målet ved denne del er at programmet lukkes på DevKittet.

Start Test-knap Test

I testen trykkes der på knappen "Start Test", og knappens funktion køres i programmet.

Forventet resultat

Konsol printer følgende:

Starting System test
Starting SPI Test
Spi Test Successful
Starting I2C Test
I2C Test Successful
Starting Nunchuck Test
Please press the Z button within 6 seconds
Nunchuck Test Successful Programmet returnerer til idle-tilstand og konsol vinduet bliver ikke clearet.

Visuelt resultat

Som det ses på figur 86 bliver det forventet resultat opnået, og konsollen printer som forventet. Programmet er returneret til idle-tilstand og teksten er ikke blevet clearet.



Figur 86: Visuel respons af Start Test-knap funktionalitet

Clear-knap Test

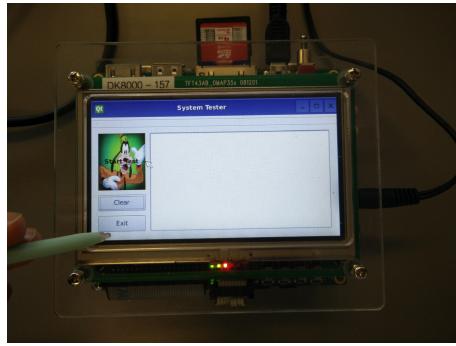
I testen trykkes der på knappen "Clear", og knappens funktion køres i programmet.

Forventet resultat

Konsol vinduet, hvor i der er skrevet resultatet fra forrige test, bliver clearet. Programmet returnerer til idle-tilstand.

Visuelt resultat

Som det ses på figur 87 bliver konsol vinduet clearet og programmet returnerer til idle-tilstand.



Figur 87: Visuel respons af Clear-knap funktionalitet

Exit-knap Test

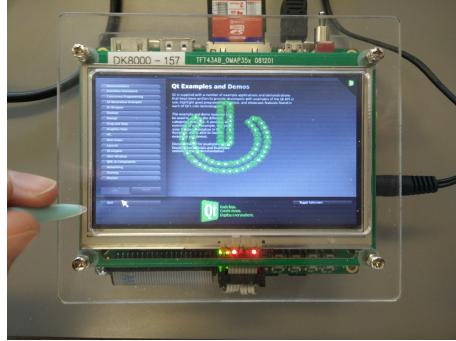
I testen trykkes der på knappen "Exit", og knappens funktion køres i programmet.

Forventet resultat

Programmet lukkes.

Visuelt resultat

Som det ses på figur 88 lukkes programmet.



Figur 88: Visuel respons af Exit-knap funktionalitet

Ud fra disse resultater af testene, fungerer brugergrænsefladen efter hensigten.

5.1.5 Rotationsdetektor

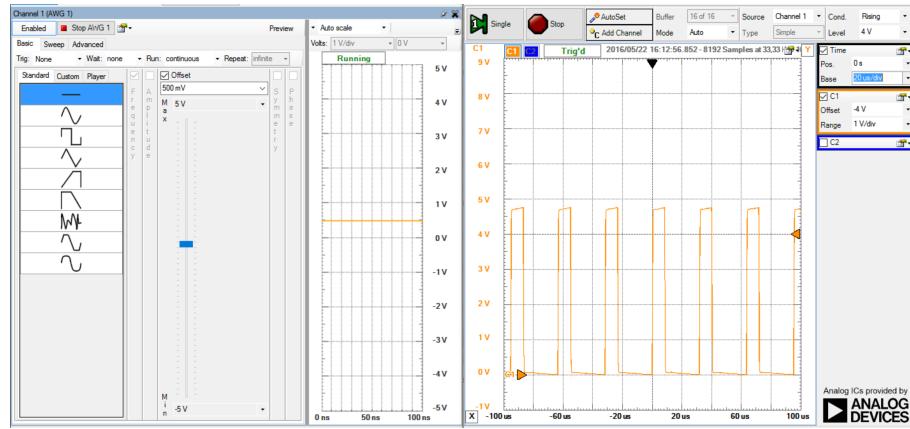
TÍ tabel 13 ses, at ADC'en virker som den skal. Den skal sende PWM-signal ud, når den får et signal, der er højere end 1500mV, og det gør den.

Indgangssignal	Forventet PWM	Målt PWM
0,5V	Ja	Ja
1600mV	Nej	Ja
5V	Nej	Nej

Tabel 13: Modultest af ADC

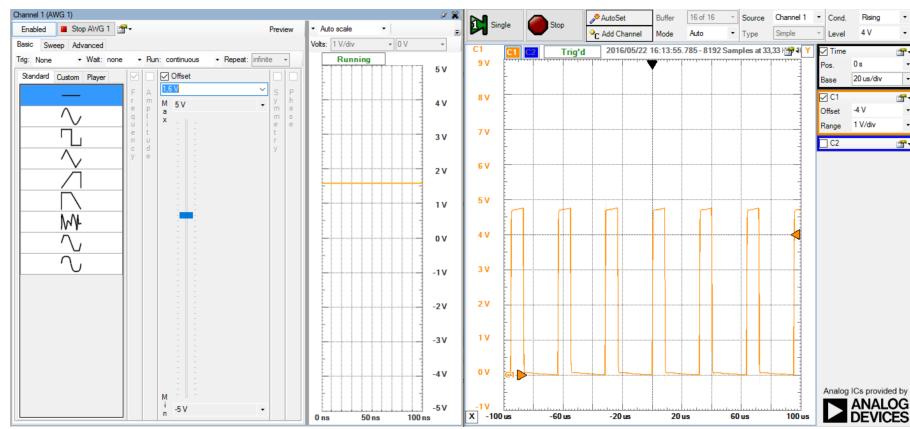
I det følgende ses screenshots fra modultesten af ADC'en brugt i affyringsmekanismen.

Det ses på figur 89, at ADC'en udsender PWM-signal, når den får et indgangssignal på 0,5V.



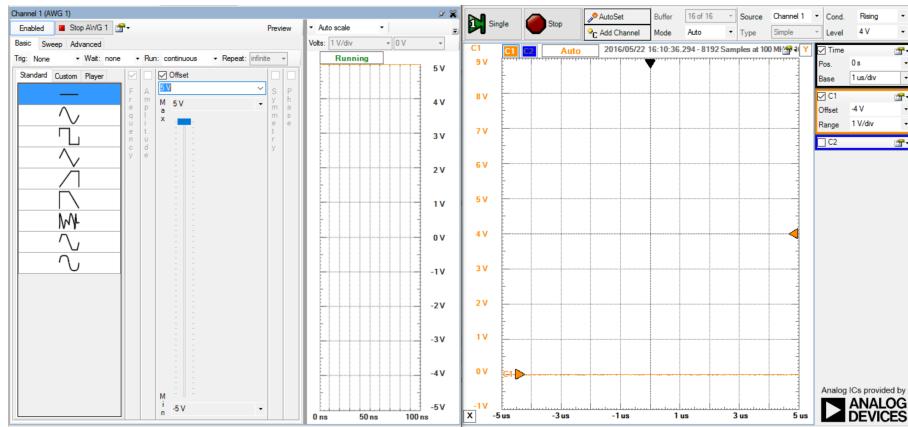
Figur 89: Modultest af ADC med et indgangssignal på 0,5V

Det ses på figur 90, at ADC'en udsender PWM-signal, når den modtager et signal på 1,6V.



Figur 90: Modultest af ADC med et indgangssignal på 1,6V

Det ses på 91, at ADC'en ikke udsender PWM-signal, når den får et indgangssignal på 5V.



Figur 91: Modultest af ADC med et indgangssignal på 5V

Testen er vellykket, da det ses, at ADC'en stopper med at udsende PWM-signal, når den når en hvis grænse.

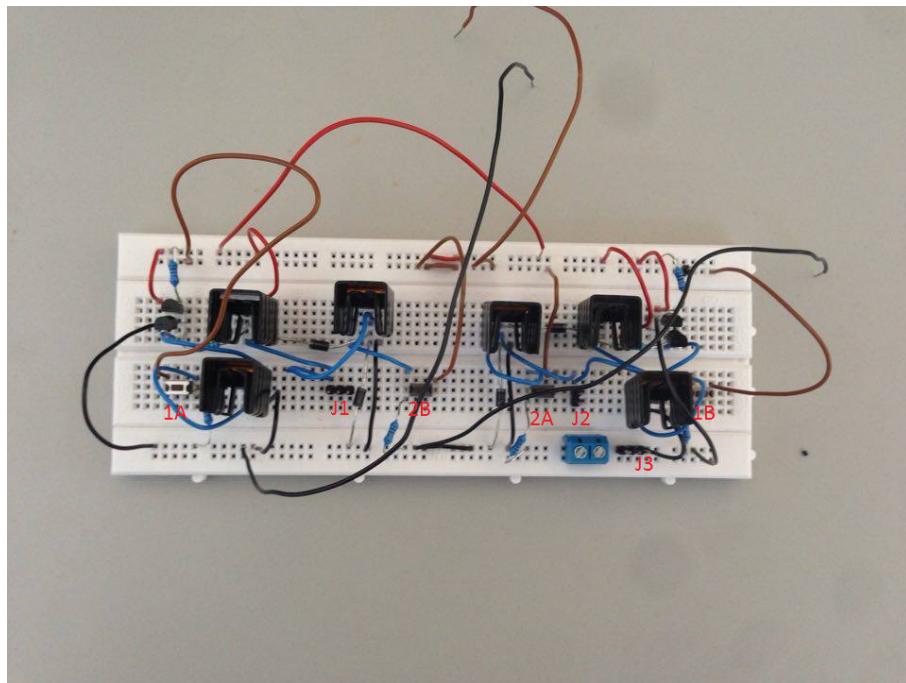
5.2 Hardware

5.2.1 H-bro

Formål

Formålet med denne test er at vise at en motor kan styres i begge retninger med H-broen.

Opstilling



Figur 92: Opstilling af H-bro på fumlebræt

Element	Beskrivelse
J1, J2	Output Pins
J3	Stel Pin
1a, 2a	Knapper til at styre motor til højre
1b, 2b	Knapper til at styre motor til venstre
Sorte ledninger	Stel
Røde ledninger	9V
Brune ledninger	5V
Blå ledninger	Interkomponente forbindelser

Tabel 14: Elementer i testopstillingen

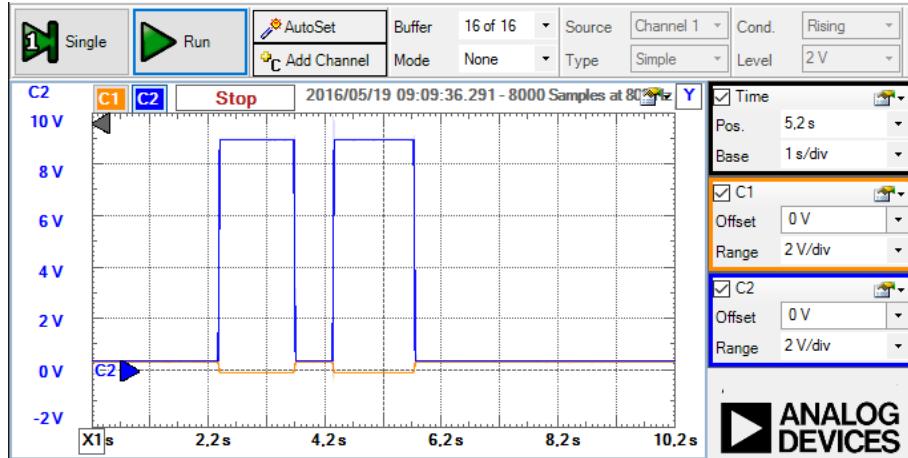
Til test af H-broens motorstyring i begge retninger blev to oscilloskop kanaler forbundet til output pin J1 og J2, samt oscilloskopenes stel til J3. Fumlebrættet forsynes med 9V, 5V og stel fra en spændingskilde.

Testen blev udført i to dele. Først testes H-broen i højre omdrejningsretning ved at trykke på 1a og 2a, derefter i venstre omdrejning ved at trykke 1b og 2b.

Forventet resultat

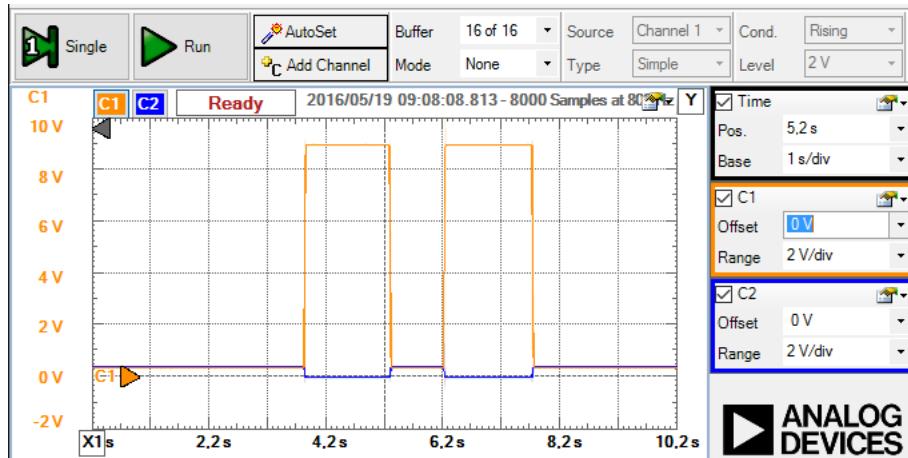
I første del af modultesten forventes det at når 1a og 2a blev nedtrykket vil den blå kurve stige til 9V samt den orange kurve vil falde til 0V. Ved anden del af testen forventes det at når 1b og 2b blev nedtrykket vil den orange kurve stige til 9V samt den blå kurve falde til 0V.

Opnæet resultat



Figur 93: Modultest for højre omdrejningsretning

Som det fremgår af figur 93, ses det at, når 1a og 2a nedtrykkes får det blå retningssignal forbindelse til stel og dermed trækker denne alt spændingen op til 9V. Da alt spænding trækkes af det blå retningssignal, bliver det orange retningssignal groundet. Dermed kan motoren styres via H-broen i højre omdrejningsretning.



Figur 94: Modultest for H-bro, orange går høj

Ved anden del af modultesten nedtrykkes 1b og 2b. På figur 94 ses resultatet af dette. Det observeres at knappene nedtrykkes får det orange retningssignal forbindelse til stel og stiger til 9V, samt at det blå retningssignal falder til 0V. Dermed er det bevist at motoren kan styres vi H-broen i venstre omdrejningsretning.

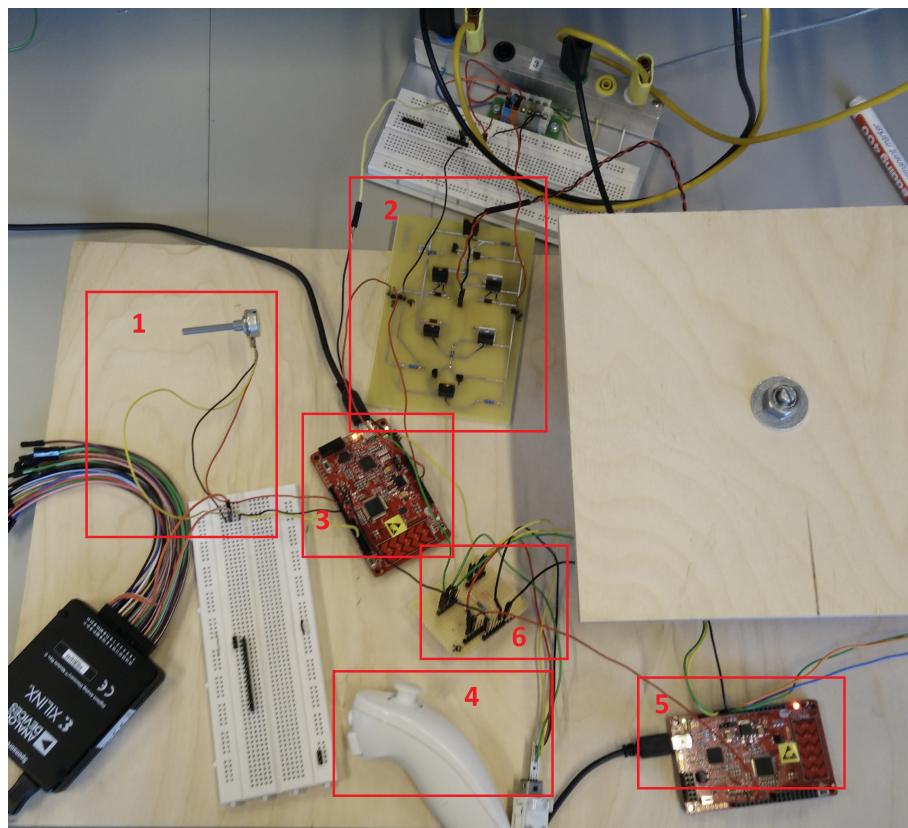
Ud fra disse to test kan det konkluderes at en motor kan styres i begge omdrejningsretninger via H-broen. Fra figurene 93 og 94 kan det også observeres at der ikke fremkommer støj, når retningssignalerne stiger til 9V, dermed vil der ikke være forstyrrende elementer der kommer til at påvirke styringen af motoren.

5.2.2 Rotationsbegrænsning

Formål

Formålet med denne test er at bevise at motoren ikke kan køre ud over de ydergrænser, men kan bevæge sig frit i det tilladte interval.

Opstilling



Figur 95: Testopstilling for rotationsbegrænsning

Bemærkning: Ved denne testopstilling er potentiometeret ikke fastsat til motoren, dermed følger output spændingen ikke motorens rotationer. Denne test bruges til at bevise at retningssignalene blokeres, når potentiometeret bevæges udover de tilladte grænser. Dette gøres manuelt.

Element	Beskrivelse
1	Potentiometer
2	H-bro
3	PSoC1
4	Wii-Nunchuck
5	PSoC0
6	Print med pull-up modstande til I2C kommunikation
Sorte ledninger	Stel
Røde ledninger	5V
Gule ledninger	Data linjer mellem moduler
Grønne ledninger	Clock signal

Tabel 15: Elementer i testopstillingen

Test af rotationsbegrænsning blev udført i tre dele. I første del testes om motoren kan styres i begge retninger når outputspændingen fra potentiometret ligger mellem 915mV og 2000mV. I anden del testes der for at retningssignalet til venstre blokeres når potentiometerets outputspænding overstiger 2000mV. I tredje del testes der for at retningssignalet til højre blokeres når potentiometers outputspænding falder under 915mV.

Til at monitorere outputspændingen fra potentiometeret er et Analog Discovery tilsluttet til dens udgange gennem et fumlebræt. Til at for at aflæse værdier fra AD converteren sættes PSOC1 i debug mode. Resultater verificeres visuelt.

Forventet resultat

Første del: Potentiometeret indstilles til midterposition. Det observeres at motoren kan roteres i begge retninger.

Anden del: Potentiometret indstilles til at have den størst mulig modstandsværdi. Det observeres at motoren kun kan roteres til højre.

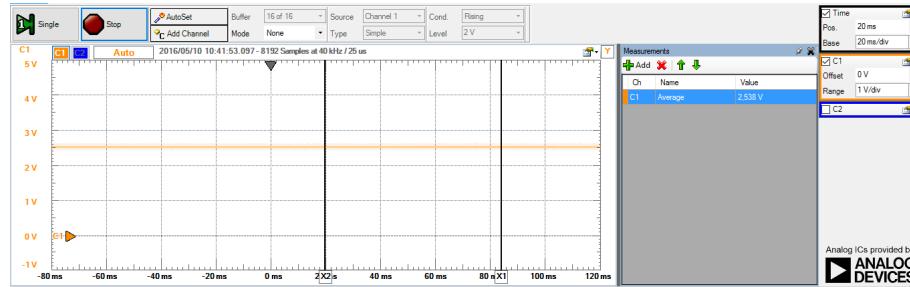
Tredje del: Potentiometret indstilles til at have lavest mulig modstandsværdi. Det observeres at motoren kun kan roteres til venstre.

Da ADC'en har en referencespænding på 3.3V og outputtet af potentiometret er mellem 0V-5V er foholdet givet ved:

$$\frac{5V}{3.3V} = 1.515 \quad (9)$$

Dermed forventes der et tilnærmelsesvis forhold mellem ADC'ens og potentiometrets output.

Opnået resultat



Figur 96: Første del: Potentiometer output 2538mV

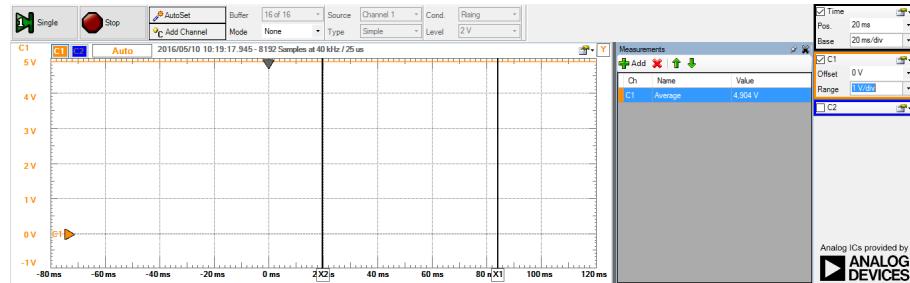


Figur 97: Første del: ADC output 1595mV

Som det fremgår på figur 96 er output spændingen af potentiometeret 2538mV og på figur 97 ses det at outputspændingen af ADC'en har en værdi på 1595mV. Tages der hensyn til det fornævnte forhold, ses det at den forventede outputspænding er:

$$1595mV * 1.515 = 2416mV \quad (10)$$

Da den faktiske spænding fra potentiometeret 2538mV stemmer det faktiske resultat overens med det forventede resultat. Det observeres at motoren kan styres i begge retninger.



Figur 98: Anden del: Potentiometer output 4904mV



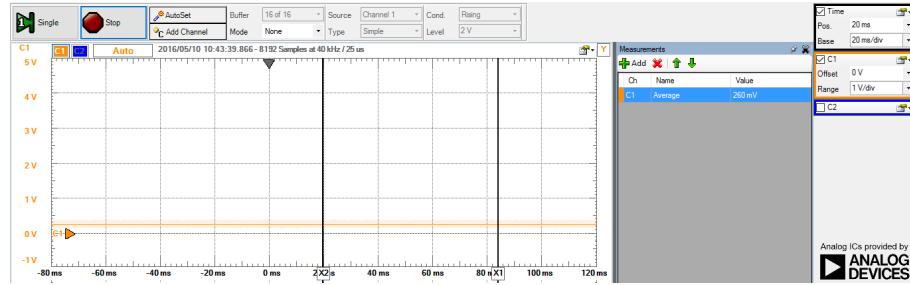
Figur 99: Anden del: ADC output 3252mV

Som det fremgår på figur 98 er output spændingen af potentiometeret 4904mV og på figur 99 ses det at outputspændingen af ADC'en har en værdi på 3252mV.

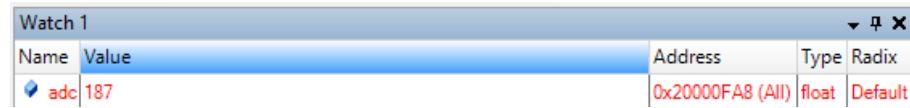
Tages der hensyn til det fornævnte forhold, ses det at den forventede outputspænding er:

$$3252mV * 1.515 = 4926mV \quad (11)$$

Da den faktiske spænding fra potentiometeret 4904mV stemmer det faktiske resultat overens med det forventede resultat. Det observeres at motoren kun kan styres i højre.



Figur 100: Tredje del: Potentiometer output 260mV



Figur 101: Tredje del: ADC output 187mV

Som det fremgår på figur 100 er output spændingen af potentiometeret 260mV og på figur 101 ses det at outputspændingen af ADC'en har en værdi på 187mV. Tages der hensyn til det fornævnte forhold, ses det at den forventede outputspænding er:

$$187mV * 1.515 = 280.5mV \quad (12)$$

Da den faktiske spænding fra potentiometeret 260mV stemmer det faktiske resultat overens med det forventede resultat. Det observeres at motoren kun kan styres til venstre.

Ud fra disse tre deltests kan det konkluderes at rotationsbegrænsningen fungerer efter hensigten.

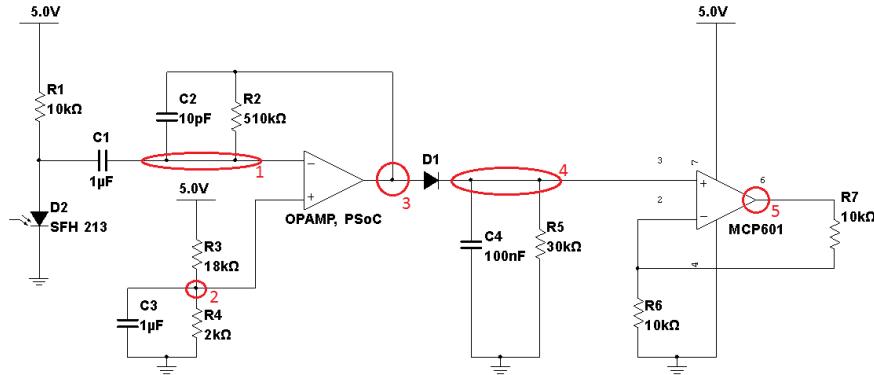
5.2.3 Rotationsdetektor

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

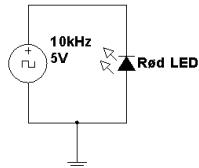
Test af relevante målepunkter på rotationsdetektor

På figur 102 ses, hvor der er målt for at teste rotationsdetektoren.

Detektorkredsløb



LED-kredsløb



Figur 102: Målepunkter på rotationsdetektor

Fotodiode får ikke signal fra LED

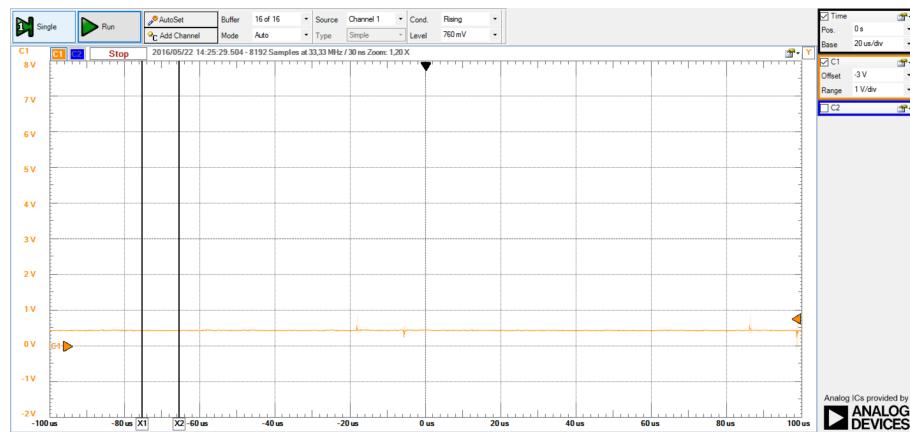
I tabel 16 ses resultaterne fra modultesten af rotationsdetektoren, når fotodioden ikke får noget lyssignal. Det ses, at det virtuelle nul fra spændingsdeleren ligger på 0,4V, og at forstærkeren fordobler signalet, hvilket var meningen.

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

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

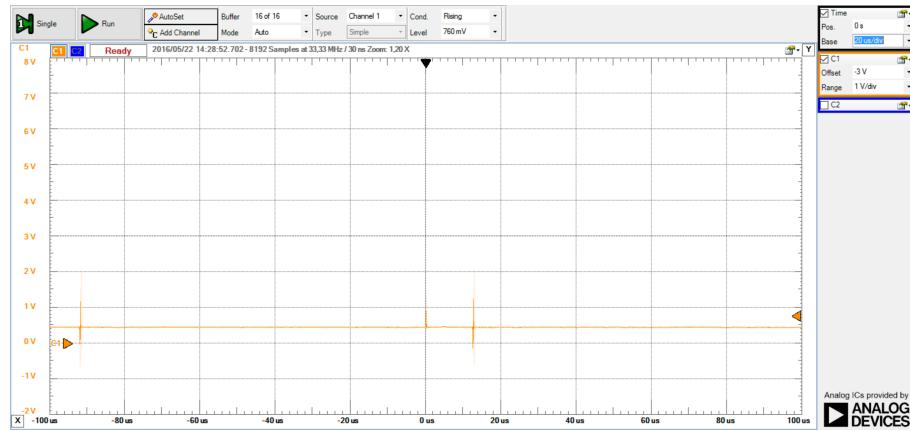
I det følgende ses screenshots af resultater fra modultesten.

Det ses på figur 103, at det virtuelle nulpunkt ligger på 0,4V. Det ligger lidt under det forventede, men det er en ubetydeligt lille forskel.



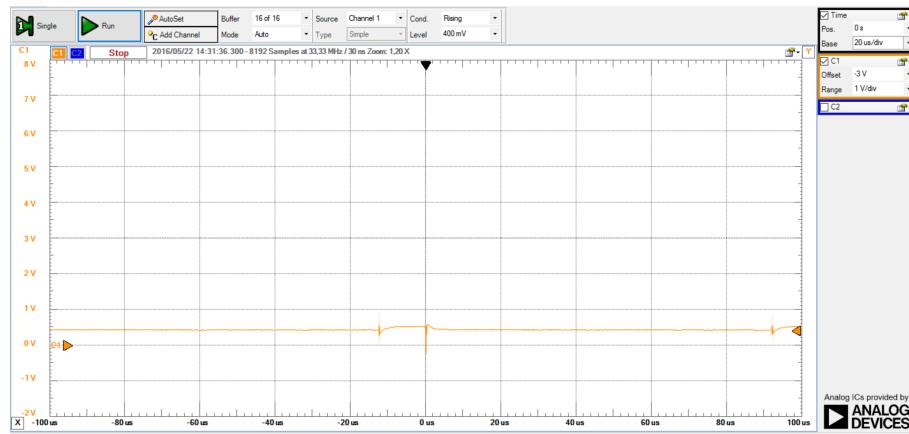
Figur 103: Modultest af det virtuelle nulpunkt i rotationsdetektorkredsløbet, hvor fotodioden ikke får signal fra LED'en

På figur 104 ses det, at spændingsdeleren virker, da der er en referencespænding på 0,45V, og ønsket var, at den skulle være 0,5V.



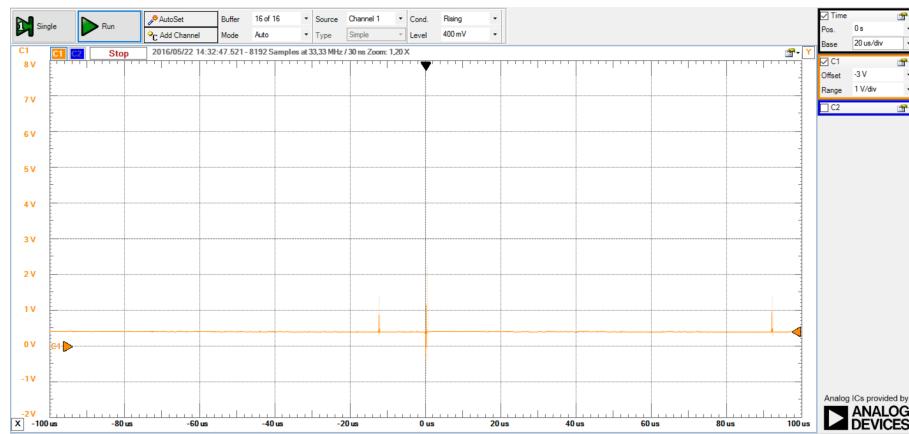
Figur 104: Modultest af referencespændingen i rotationsdetektorkredsløbet, hvor fotodioden ikke får signal fra LED'en

Det ses på figur 104, at PSoC'ens operationsforstærkers udgang ligger på 0,4V, hvilket ligger meget tæt på den forventede værdi på 0,5V. Derfor er det vist, at operationsforstærkeren virker, når fotodioden ikke får signal fra LED'en.



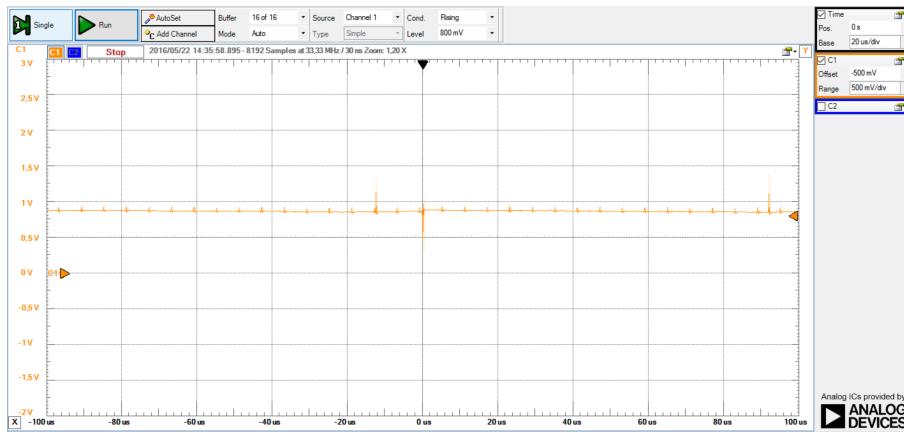
Figur 105: Modultest af udgangen på PSoC'ens operationsforstærkers udgang, hvor fotodioden ikke får signal fra LED'en

Det ses på figur 105, at envelopedetektoren virker, når fotodioden ikke får lyssignal, da der er målt en værdi på 0,4V, hvilket ligger meget tæt på de forventede 0,5V.



Figur 106: Modultest af enveloppedetektoren i rotationsdetekkredsløbet, hvor fotodioden ikke får signal fra LED'en

Det ses på figur 106, at forstærkeren virker, da der er målt 0,85V på udgangen, og den forventede værdi var 1V. De 0,15V er en ubetydelig lille forskel.



Figur 107: Modultest af forstærkeren i rotationsdetekorkredsløbet, hvor fotodioden ikke får signal fra LED'en

Fotodiode får signal fra LED

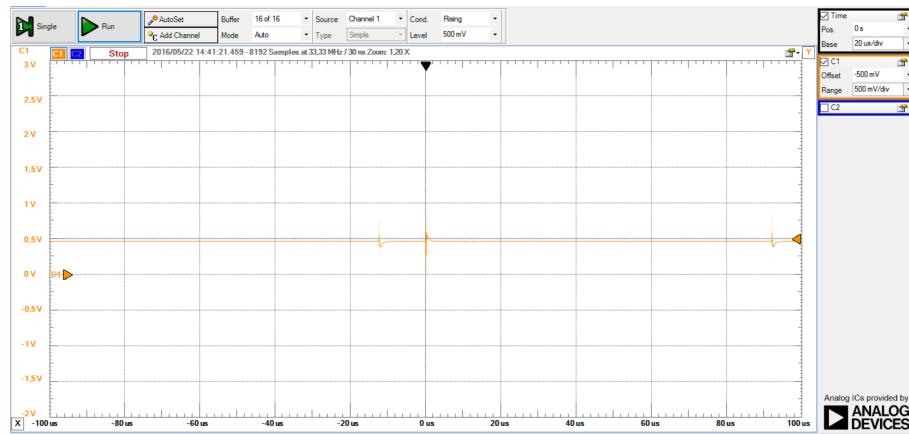
I tabel 17 ses resultaterne fra modultesten af rotationsdetektoren, når fotodioden får lyssignal fra LED'en. Her ses det, at der igen er 0,4V, der hvor det var meningen, der skulle være 0,5V, så det er fint. Det ses også, at der er et PWM-signal på udgangen af PSoC'ens operationsforstærker og forstærkeren forstærker signalet, som ønsket.

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

Tabel 17: Modultest af rotationsdetektor, når fotodioden modtager lyssignal fra LED

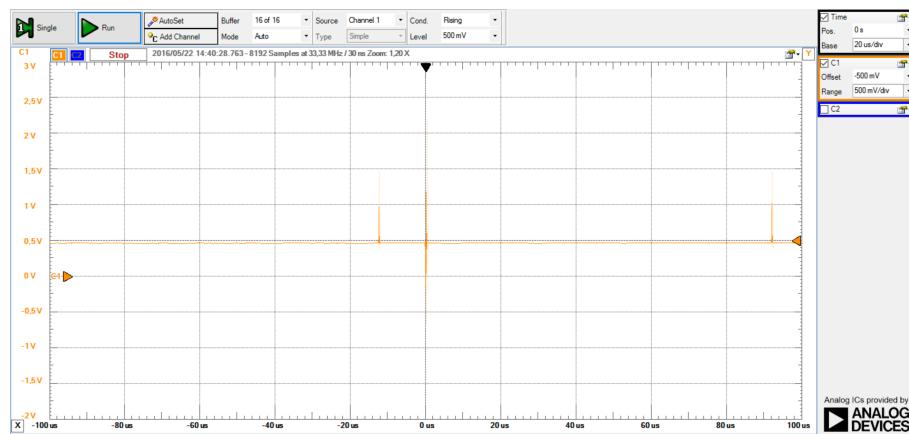
I det følgende ses screenshots af resultater fra modultesten.

På figur 108 ses, at det virtuelle nulpunkt virker, når fotodioden får signal fra LED'en, da spændningen ligger på 0,4V og den forventede værdi var 0,5V.



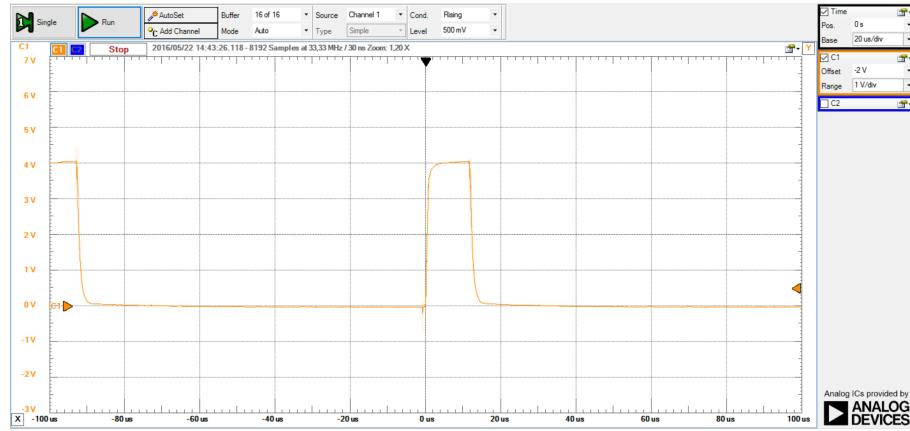
Figur 108: Modultest af det virtuelle nulpunkt, hvor fotodioden får signal fra LED'en

Det ses på figur 109, at spændingsdeleren virker, da referencespændingen er målt til 0,45V og den forventede værdi var 0,5V.



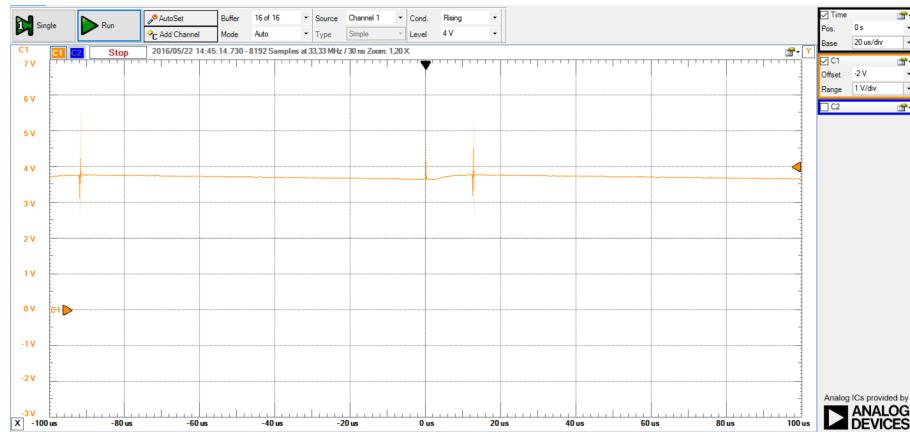
Figur 109: Modultest af spændingsdeleren i rotationsdetektorkredsløbet, hvor fotodioden får signal fra LED'en

Det ses på figur 110, at udgangen på forstærkeren giver et PWM-signal, som ønsket. Det ligger dog kun på 4V, hvor den forventede værdi var på 5V.



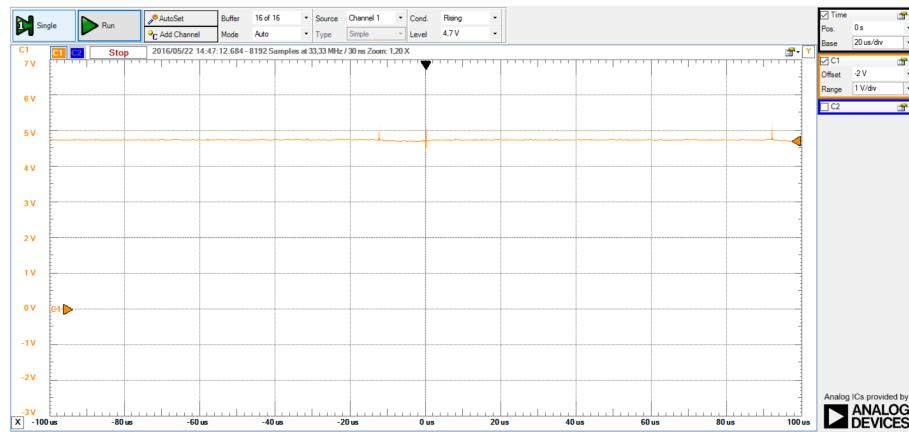
Figur 110: Modultest af udgangen på PSoC'ens operationsforstærker, hvor fotodioden får signal fra LED'en

Det ses på figur 111, at envelopedetektoren virker, som den skal. Denne kondensator når at lade op til cirka 3,7V, hvorefter den ikke når at aflade, før der kommer en ny puls fra PWM-signalet.



Figur 111: Modultest af rotationsdetektorens enveloppedetektor, hvor fotodioden får signal fra LED'en

På figur 112 ses det, at forstærkeren forstærker signalet, der kommer fra enveloppedetektoren. Den fordobler det ikke, hvilket var meningen, men signalet forstærkes, og det var det, der var meningen.



Figur 112: Modultest af udgang på forstærkeren i rotationsdetektorkredsløbet, hvor fotodioden får signal fra LED'en

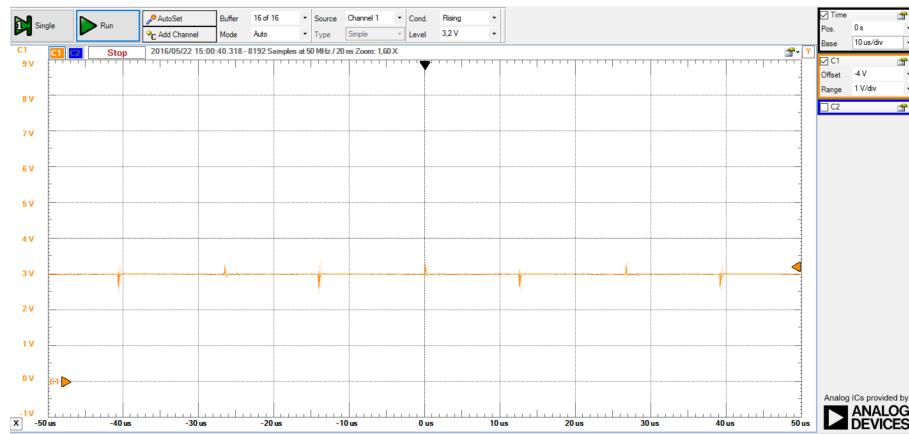
Test af båndpasfilter

I tabel 18 ses resultaterne fra modultesten af båndpasfilteret. Testen er foregået således, at værdier, der ligger tæt på afskæringsfrekvenserne, er testet. Derudover er centerværdien for filteret testet. Endelig er det testet om LED'en lyser, når der sendes en DC-værdi på 5V gennem filteret.

PWM-værdi	Forventet værdi	Målt værdi
40kHz		3V
29kHz		3,9V
10kHz		4,7V
1Hz		-
5V (lyser LED?)	Ja	Ja

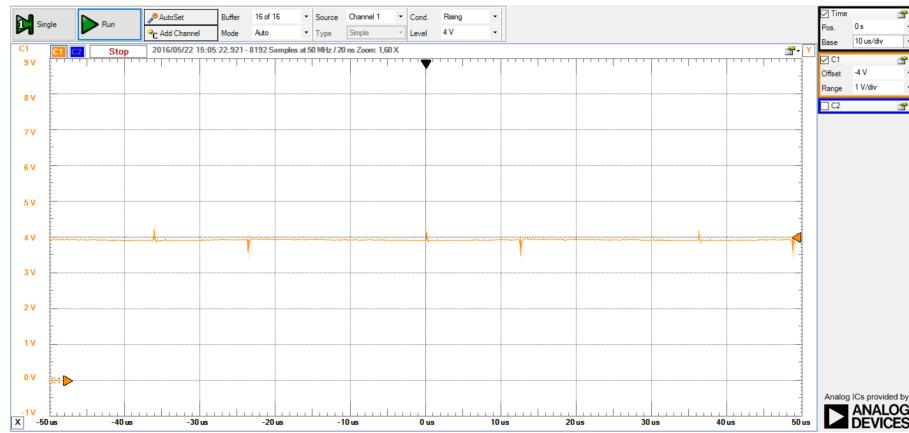
Tabel 18: Modultest af rotationsdetektorens båndpasfilter

På figur 113 ses det, at filteret dæmper spændingen til 3V, når der sendes en frekvens på 40kHz gennem det. Da 40kHz ligger forholdsvis tæt på lavpasfilterets afskæringsfrekvens på 31,2kHz, kunne der være valgt en højere frekvens at teste i stedet.



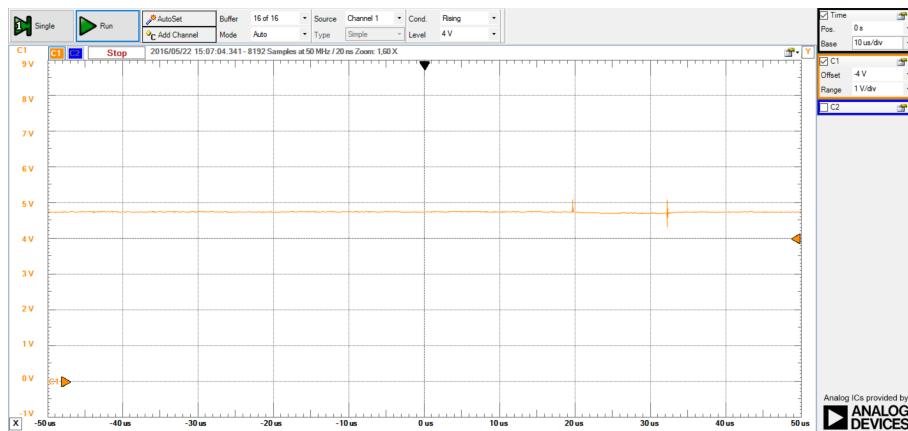
Figur 113: Modultest af båndpasfilter, hvor et signal med en frekvens på 40kHz sendes gennem filteret

Det ses på figur 114, at filteret dæmper spændingen til 3,9V, når der sendes en frekvens på 29 kHz gennem det. Da lavpasfilterets afskæringsfrekvens ligger på 31,2kHz, skulle filteret egentlig ikke dæmpe signalet her.



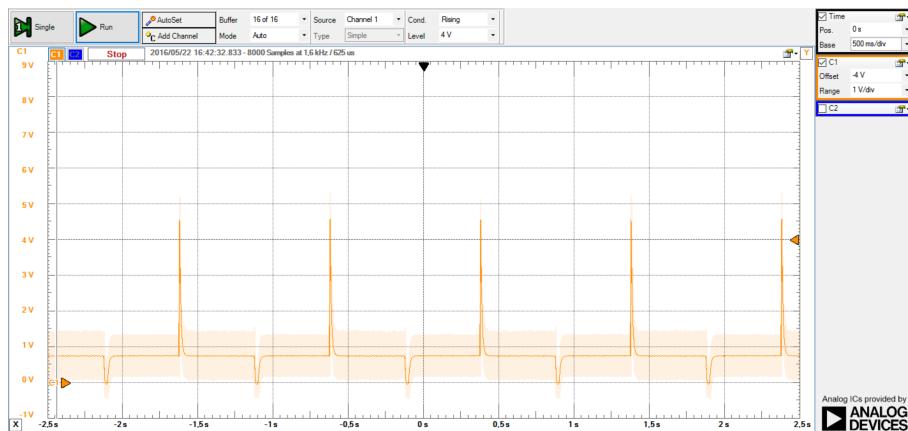
Figur 114: Modultest af båndpasfilter, hvor et signal med en frekvens på 29kHz sendes gennem filteret

Det ses på figur 115, at filteret ikke dæmper signalet ved 10kHz. Signalet ligger nemlig på 4,7V.



Figur 115: Modultest af båndpasfilter, hvor et signal med en frekvens på 10kHz sendes gennem filteret

Det ses på figur 116, at det er et atypisk signal, der kommer gennem filteret.

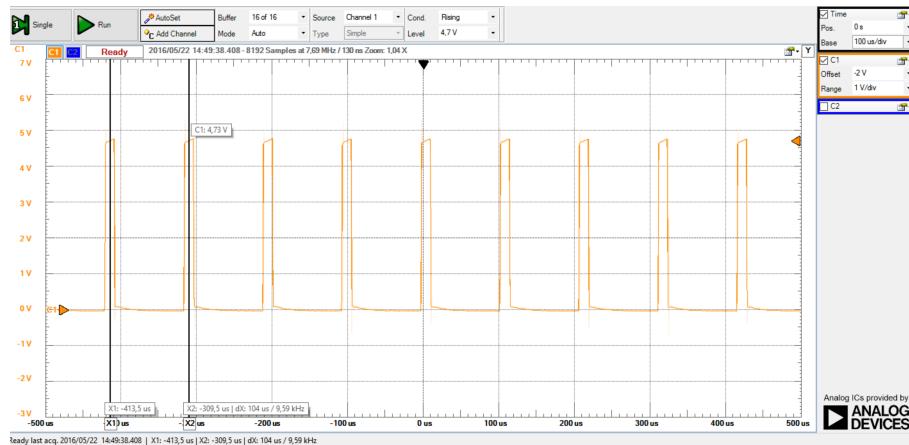


Figur 116: Modultest af båndpasfilter, hvor et signal med en frekvens på 1Hz sendes gennem filteret

Afskæringsfrekvenserne for båndpasfilteret ligger på 15,9Hz og 31,2kHz, så hvis båndpasfilteret skulle være helt optimalt skulle alt, der ligger udenfor disse frekvenser dæmpes fuldstændig. Det ses dog i tabel ??, at det ikke helt er tilfældet. Ved 40kHz er signalet dæmpet til 3V, og ved 10kHz lader filteret alt gå igennem. Ved 1Hz ses det, at signalet dæmpes, men der forekommer også mange spikes, som forstyrrer signalet. Derfor er der ikke indsattet et tal i tabellen.

Test af LED's PWM-signal

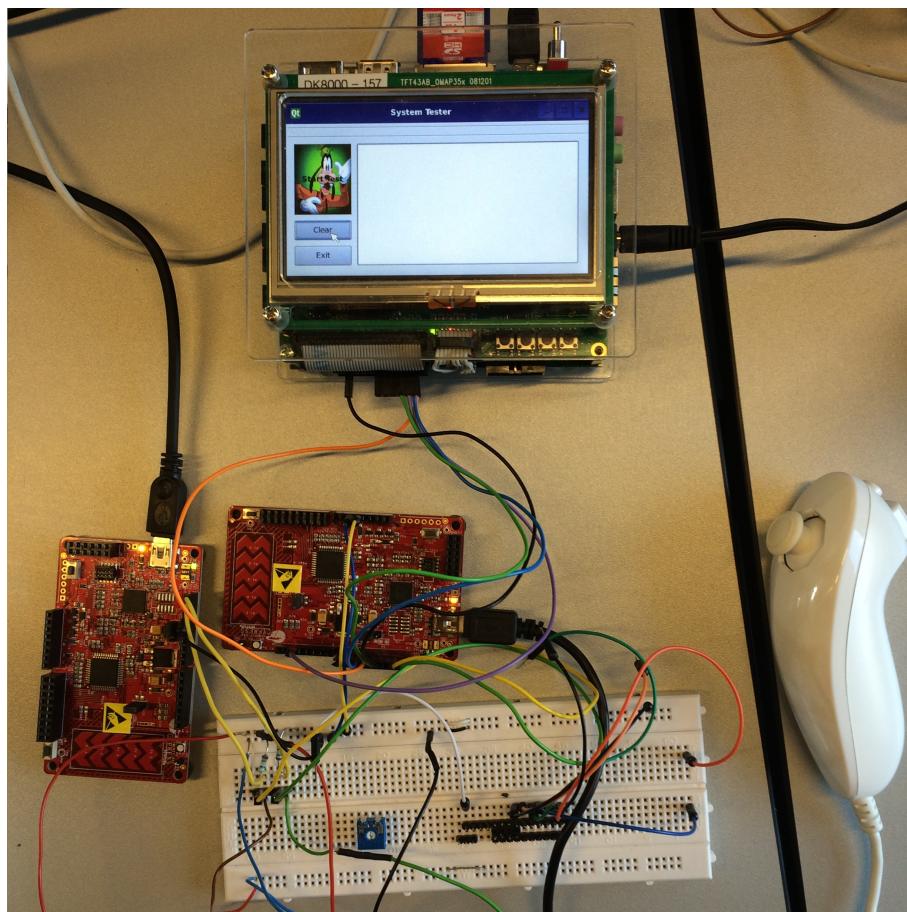
LED'ens PWM-signal blev testet ved at måle med et oscilloskop på LED'ens indgang. Resultatet af denne test ses på 117. Det ses, at signalet ikke når helt op på 5V, men kun 4,7V, hvilket er det, der i testen af forstærkerudgangen på figur 115 også er målt. Det ses også, at signalet har en frekvens på 9,59kHz, hvilket er tilnærmelsesvis det samme som for det indgående signal.



Figur 117: Modultest af LED'ens PWM-signal

5.3 Integrationstest - Use case 2

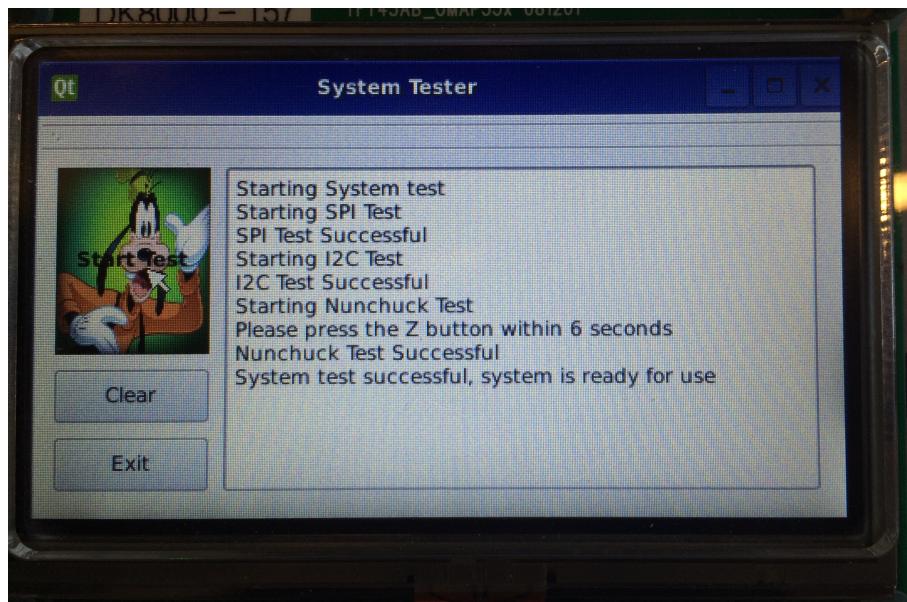
For at verificere at use case 2 fungerer når det sammensættes til en enkelt enhed, er der lavet en integrationstest. Testen er lavet ud fra et 'black-box' princip, hvilket vil sige, at der kun evalueres ud fra systemets funktionalitet, og ikke på den interne struktur. Testen udføres ved, at klikke på 'Start-test', og bagefter observeres udskriften på terminal-vinduet på brugergrænsefladen. På figur 118 ses opstillingen for integrationstesten.



Figur 118: Integrationstest opstilling

Opstillingen viser, at Nunchuck'en og de to PSoC's er forbundet til samme I2C-netværk via fumlebrættet. PSoC0(PSoC'en til højre) er også forbundet til DevKit8000 via SPI, ledt igennem fumlebrættet. På Devkittet ses brugergrænsefladen, hvor testen initieres ved at klikke på "Start test". Brugergrænsefladen har også et terminalvindue, hvor testens status bliver udprintet.

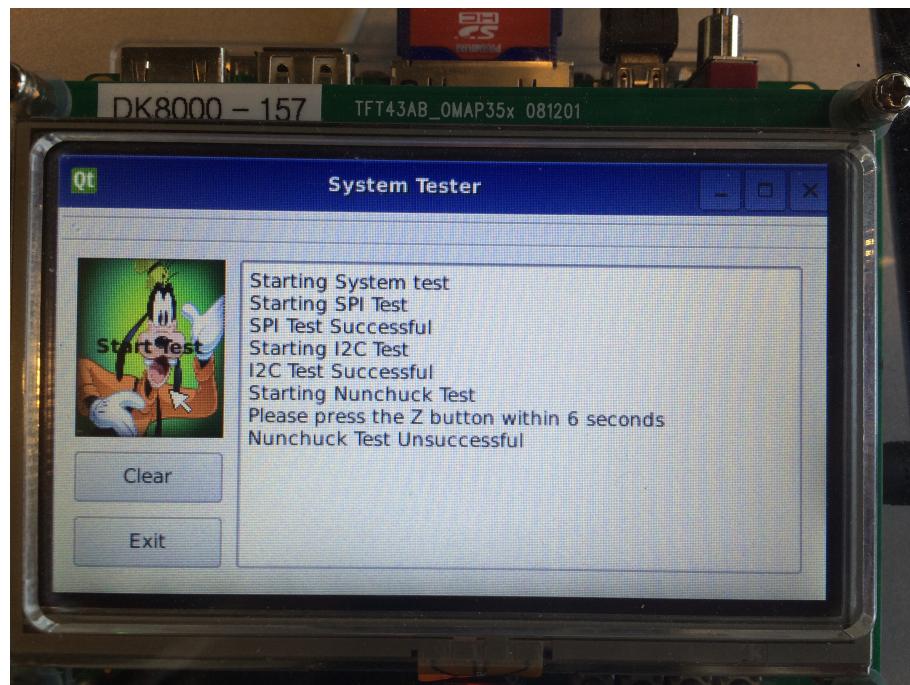
Selve testen gennemføres, ved at der klikkes "Start test" på brugergrænsefladen, og derefter følges evt. instruktioner der vises på brugergrænsefladen. Det forventes, at efter testen, vil brugergrænsefladen fortælle, at testen blev gennemført uden fejl, og systemet er klar til brug. Figur 119 viser brugergrænsefladen efter endt system test.



Figur 119: Resultat af integrationstesten

Som det ses, er resultatet af testen som forventet, og testen er gennemført. Under testen bliver brugeren bedt om at trykke på nunchucken's 'z' knap.

Hvis brugeren ikke klikker på knappen indenfor det angivne stykke tid, forventes det at testen vil fejle. Figur 120 viser netop dette.



Figur 120: Resultat hvis man ikke klikker på nunchuck

Som det ses, fejlede testen, hvilket stemte overens med det forventede resultat. Derved kan det konkluderes at systemet opfører sig som forventet.

6 Udfyldt Accepttest

Accepttesten i afsnit 2 er blevet udført i samarbejde med vejleder (Gunvor Elisabeth Kirkelund). Dette afsnit vil vise den udfyldte accepttest.

6.1 Use case 1 - Hovedscenarie

Step	Handling	Forventet observasjon/resultat	Faktisk observasjon/resultat	Vurdering (OK/FAIL)
1	Vælg single-player mode.	Brugergrænsefladen viser spilside for one-player mode og anmoder om, at der fyldes slik i magasinet og at kanon indstilles.		FAIL
3	Fyld slik i kanon. Indstil kanon til affyring med Wii-nunchuck.	Kanon indstiller sig svarende til Wii-nunchucks placering.		OK
4	Udløs kanon med trigger på wii-nunchuck.	Kanon udløses.	Slik "dispenses"	FAIL
5	Gentag punkt 4 og 5 to gange.	Punkt 4 og 5 gentages.	Virker lidt	FAIL
6	Tryk på knap for at vende tilbage til starttilstand.	Brugergrænseflade vender tilbage til startside.		OK

6.1.1 Use case 1 - Extension 1

Step	Handling	Forventet observasjon/resultat	Faktisk observasjon/resultat	Vurdering (OK/FAIL)
1	Vælg party mode.	Brugergrænsefladen viser spilside for two-player mode og anmoder om valg af antal skud.		OK
2	Fyld slik i kanon. Indstil kanon til affyring med Wii-nunchuck.	Kanon indstiller sig svarende til Wii-nunchucks placering.	Kører frem og tilbage	OK
3	Udløs kanon med trigger på Wii-nunchuck.	Kanon udløses.		OK
4	Giv Wii-nunchuck til den anden spiller.	Den anden spiller modtager Wii-nunchuck.		OK
5	Gentag punkt 5 til 6 indtil skud er opbrugt.	Punkt 5 til 6 gentages.		OK
6	Tryk på knap for at vende tilbage til starttilstand.	Brugergrænseflade vender tilbage til startside.		OK

6.1.2 Use case 1 - Extension 2

Step	Handling	Forventet observasjon/resultat	Faktisk observasjon/resultat	Vurdering (OK/FAIL)
1	Vælg single-player mode.	Brugergrænsefladen viser spilside for one-player mode og anmoder om, at kanon indstilles.	Går til singleplay menu	OK
2	Tryk på knap for afslutning af spil.	Brugergrænseflade vender tilbage til startside.	Går ud af single player menu	OK

6.2 Use case 2 - Hovedscenarie

Step	Handling	Forventet observation/resultat	Faktisk observation/resultat	Vurdering (OK/FAIL)
1.	Tryk på "Systemtest" på GUI	Systemtest brugergrænsefladen vises på Devkittet.		OK
2.	Tryk på "Start Test" på GUI"	Brugergrænsefladen udskriver at SPI og I2C testen er godkendt. Brugergrænsefladen anmoder brugereren om tryk på Z på Wii-nunchuck		OK
3.	Tryk 'Z' knappen på Wii-nunchucken	Brugergrænsefladen udskriver at Wii-testen er godkendt		OK

6.2.1 Use case 2 - Exception 1

Step	Handling	Forventet observation/resultat	Faktisk observation/resultat	Vurdering (OK/FAIL)
1.	Fjern SPI-kablet fra Devkittet		Fjerner hele SPI kablet	OK
2.	Tryk på "Systemtest" på GUI	Systemtest brugergrænsefladen vises på Devkittet.		OK
3.	Tryk på "Start Test" på GUI	Brugergrænsefladen udskriver at SPI forbindelsen mislykkedes	SPI Unsuccessful	OK

6.2.2 Use case 2 - Exception 2

Step	Handling	Forventet observation/resultat	Faktisk observation/resultat	Vurdering (OK/FAIL)
1.	Fjern I2C-kabel fra PSoC0			OK
2.	Tryk på "Systemtest" på GUI	Systemtest brugergrænsefladen vises på Devkittet.		OK
3.	Tryk på "Start Test" på GUI	Brugergrænsefladen udskriver at SPI forbindelsen lykkedes og I2C forbindelsen mislykkedes	I2C Unsuccessful	OK

6.2.3 Use case 2 - Exception 3

Step	Handling	Forventet observation/resultat	Faktisk observation/resultat	Vurdering (OK/FAIL)
1.	Tryk på "Systemtest" på GUI	Systemtest brugergrænsefladen vises på Devkittet.		OK
2.	Tryk på "Start Test" på GUI	Brugergrænsefladen udskriver at SPI og I2C forbindelsen lykkedes	Er succesfuld	OK
3.	Afvent timeout	Brugergrænsefladen udskriver at Nunchuck forbindelsen mislykkedes	Nunchuck unsuccessful	OK

6.3 Ikke-funktionelle krav

Krav	Test	Forventet observation/resultat	Faktisk observation/resultat	Vurdering (OK/FAIL)
1	Bruger drejer kanonen så lang til venstre og højre som muligt.	Det observeres at kanonen ikke har roteret 360 °	Drejer under 90 °	OK
2	Et projektil på 1.25 cm i diameter \pm 5mm affyres fra kanonen.	Projektilet bliver affyret	Falder ud	FAIL
3	Mål produktet dimensioner med en lineal.	Dimensionerne overstiger ikke 50cm x 60cm x 50cm.	Målt til 40x56x32	OK
4	Tryk på 'Z' knappen på Wii Nunchuck, og mål med et stopur hvor lang tid der går fra tryk, til kanonen bliver affyret.	Den målte tid er mindre end 10 sekunder.	Målt til 1.36 sek.	OK
5	Kanonen affyres 3 gange, og et stopur startes ved første skud, og stoppes ved det tredje skud.	Den målte tid er mindre end 60 sekunder.	Målt til 2.21 sek.	OK

7 Udviklingsværktøjer

Under udarbejdelsen af Goofy Candygun 3000 er der blevet gjort brug af forskellige udviklingsværktøjer. De anvendte værktøjer beskrives i følgende afsnit.

7.1 QT Creator

QT Creator [1] er blevet brugt til at designe brugergrænsefladerne. QT's design widget suite/API er blevet brugt som den primære del af QT frameworkt. Det fungerede godt til at designe en EDP-baseret grænseflade. Frameworkt's indbyggede beskedssystem 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 kålerhjelmen.

7.2 PSoC Creator

Der er blevet anvendt værktøjet PSoC Creator til at udvikle softwaren til PSoC [2]. Det er smart, da PSoC Creator har en blok for hver komponent der findes i PSoC'en. Disse blokke kan trækkes ind i Topdesignet, hvor det så er muligt at udarbejde designet for det PSoC-program, der ønskes. Desuden findes der i PSoC Creator datablade, hvor alle funktioner for de forskellige PSoC-blokke er beskrevet samt der er en beskrivelse af, hvilken funktion selve blokken har.

7.3 Analog Discovery

Analog Discovery er et USB oscilloskop der er brugt hyppigt til udviklingen af systemet [3]. Det er anvendt til modultest, hvor værdier for systemets I2C og SPI busser, samt kredsløb, skal verificeres for korrekt funktionalitet.

8 Referencer

Litteratur

- [1] The QT Company. Qt creator. <http://www.wiki.qt.io/Main> , <http://www.qt.io/ide/>. Accessed: 2016-05-25.
- [2] Cypress. Psoc creator ide. <http://www.cypress.com/products/psoc-creator-integrated-design-environment-ide>. Accessed: 2016-05-26.
- [3] Digilent. Analog discovery. <http://store.digilentinc.com/analog-discovery-100msps-usb-oscilloscope-logic-analyzer/>. Accessed: 2016-05-26.
- [4] I2C-bus.org. I2c-what's that? <http://www.i2c-bus.org/>. Accessed: 2016-05-26.
- [5] Microchip. Spi - overview and use of the picmicro serial periperal interfase. <http://ww1.microchip.com/downloads/en/devicedoc/spi.pdf>. Accessed: 2016-05-26.
- [6] Philo. Lego 9v technic motors compared characteristics. <http://www.philohome.com/motors/motorcomp.htm>. Accessed: 2016-05-26.
- [7] Kuphaldt Tony R. All about circuits - logic signal voltage levels. <http://www.allaboutcircuits.com/textbook/digital/chpt-3/logic-signal-voltage-levels>. Accessed: 2016-03-25.
- [8] NXP Semiconductors. I2c - bus specification and user manual. *UM10204*, apr 2014.
- [9] CSE325 Embedded Microprocessor Systems. *I2C Interface with Wii Nunchuck*, chapter Assignment 6 - I2C Interface with Wii Nunchuck. CSE325 Embedded Microprocessor Systems, 2010.
- [10] Wikipedia. Wii-remote nunchuk. https://en.wikipedia.org/wiki/Wii_Remote#Nunchuk. Accessed: 2016-05-25.