# 3. Semesterprojekt - Goofy Candy Gun
# Dokumentation - Gruppe 3


Rieder, Kasper
201310514

Jensen, Daniel V.
201500152

Nielsen, Mikkel
201402530

Kjeldgaard, Pernille L.
PK94398

Konstmann, Mia
201500157

Kloock, Michael
201370537

Rasmussen, Tenna
201406382


8. april 2016

# Indhold

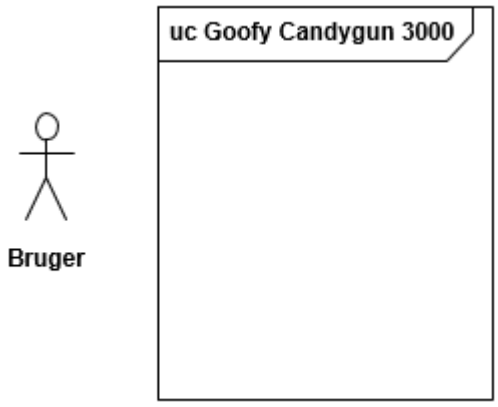
## Figurer

# 1 Kravspecifikation

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

## 1.1 Aktør kontekst diagram

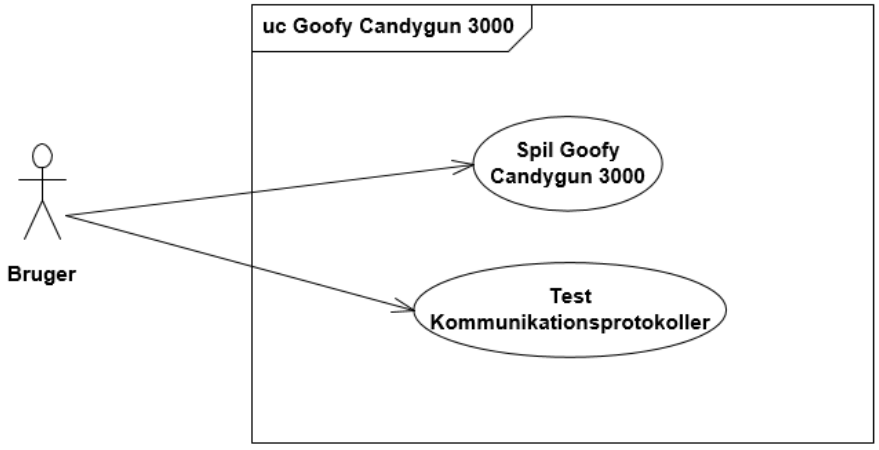
Figur 1 viser et kontekst diagram for Goofy Candygun 3000.




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.<br>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 kan findes under afsnittet **Use Case Diagram**.

### 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** | 1. Bruger vælger spiltype på brugergrænseflade<br>2. Bruger vælger antal skud til runde<br>3. Bruger fylder magasin med slik tilsvarende antal skud<br>4. Bruger indstiller kanon med analogstick på Wii-nunchuck<br>5. Bruger udløser kanonen med Wii-nunchucks trigger<br>6. System lader et nyt skud<br>7. Brugergrænseflade opdateres med spillets statistikker<br>8. Punkt 4 til 7 gentages indtil skud er opbrugt<br>[Extension 1: Bruger vælger 2 player mode]<br>[Extension 2: Bruger afslutter det igangværende spil]<br>9. Brugergrænseflade viser afslutningsinfo for runden<br>10. Bruger afslutter runde<br>11. Brugergrænseflade vender tilbage til starttilstand |
| **Udvidelser/ undtagelser** | [**Extension 1: Brugeren vælger 2 player mode**]<br>1. Bruger overdrager Wii-nunchuck til den anden bruger<br>2. Punkt 4 til 7 gentages indtil skud er opbrugt<br>3. Use case genoptages fra punkt 8<br>[**Extension 2: Bruger afslutter det igangværende spil**]<br>1. Brugergrænseflade vender tilbage til starttilstand<br>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** | 1. Bruger vælger test system på brugergrænseflade<br>2. Devkit sender start SPI test til PSoC0 via SPI<br>3. PSoC0 sender acknowledge til Devkit via SPI<br>[Exception 1: PSoC0 sender ikke acknowledge]<br>4. Brugergrænseflade meddeler om gennemført SPI test<br>5. Devkit sender start I2C test til PSoC0 via SPI<br>6. PSoC0 sender start I2C test til PSoC slaver via I2C<br>7. PSoC slaver sender acknowledge til PSoC0 via I2C<br>[Exception 2: PSoC slaver sender ikke acknowledge]<br>8. PSoC0 meddeler om gennemført I2C test til Devkit via SPI<br>9. Brugergrænseflade meddeler om gennemført I2C test<br>10. Brugergrænseflade anmoder bruger om at trykke på knap 'Z' på Wii-nunchuck<br>11. Wii-nunchuck sender besked "Knap Z trykket"til PSoC0 via I2C<br>[Exception 3: Wii-nunchuck sender ikke "Knap Z trykket"]<br>12. PSoC0 videresender besked om "Knap Z trykket"til Devkit via SPI<br>13. Brugergrænseflade meddeler om gennemført Wii-nunchuck test<br>14. Brugergrænseflade meddeler at test af kommunikationsprotokoller er gennemført |

| | |
|---|---|
| **Udvidelser/ undtagelser** | [**Exception 1: PSoC0 sender ikke acknowledge**]<br>1. Brugergrænseflade meddeler fejl i SPI kommunikation<br>2. UC2 afsluttes<br>[**Exception 2: PSoC slaver sender ikke acknowledge**]<br>1. PSoC0 sender fejlmeddelse til Devkit<br>2. Brugergrænseflade meddeler fejl i I2C kommunikation<br>3. UC2 afsluttes<br>[**Exception 3: Wii-nunchuck sender ikke "Knap Z trykket"**]<br>1. PSoC0 sender fejlmeddelse til Devkit<br>2. Brugergrænseflade meddeler fejl i I2C kommunikation med Wii-nunchuck<br>3. UC2 afsluttes |

## 1.5 Ikke funktionelle krav

1. Kanonen skal kunne drejes med en nøjagtighed på ± 5 °
   1.1. Vertikalt gælder dette for intervallet fra 0 til 70°
   1.2. Horizontalt gælder dette for intervallet fra -45 til 45°
2. Kanonen skal kunne affyre projektiler med en diameter på 1,25 cm ± 2 mm
3. Kanonen skal kunne affyre sit projektil minimum 1 meter
4. Kanonens størrelse må maksimalt være 40cm høj, bred og dyb
5. Fra aftryk på trigger til affyring må der maksimalt gå ti sekunder
6. Affyring af kanonen skal kunne afvikles minimum tre gange pr. minut
7. Figur 3 viser en skitse af hvordan den grafiskbrugergrænseflade kommer til at se ud


Figur 3: Skitse af brugergrænsefladen

# 2 Accepttestspecifikation

## 2.1 Use case 1 - Hovedscenarie

| Step | Handling | Forventet observation/resultat | Faktisk observation/resultat | Vurdering (OK/FAIL) |
|---|---|---|---|---|
| 1 | Vælg one-player mode. | Brugergrænsefladen viser spilside for one-player mode og anmoder om valg af antal skud. | | |
| 2 | Vælg ti skud. | Brugergrænseflade anmoder om, at der fyldes ti stykker slik i magasin. | | |
| 3 | Fyld ti stykker slik i magasinet og tryk på knap for at starte spil. | Brugergrænseflade går til spilside og anmoder om, at kanon indstilles. | | |
| 4 | Indstil kanon til affyring med Wii-nunchuck. | Kanon indstiller sig svarende til Wii-nunchucks placering. | | |
| 5 | Udløs kanon med trigger på wii-nunchuck. | Kanon udløses. | | |
| 6 | Gentag punkt 4 og 5 ti gange. | Punkt 4 og 5 gentages. | | |
| 7 | Kig på brugergrænsefladen. | Brugergrænsefladen viser info om spillet. | | |
| 8 | 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 observation/resultat | Faktisk observation/resultat | Vurdering (OK/FAIL) |
|---|---|---|---|---|
| 1 | Vælg two-player mode. | Brugergrænsefladen viser spilside for two-player mode og anmoder om valg af antal skud. | | |
| 2 | Vælg ti skud på brugergrænseflade. | Brugergrænseflade anmoder om, at der fyldes ti stykker slik i magasin. | | |
| 3 | Fyld ti stykker slik i magasinet og tryk på knap for at starte spil. | Brugergrænseflade går til spilside og anmoder om, at kanon indstilles. | | |
| 4 | Indstil kanon til affyring med Wii-nunchuck. | Kanon indstiller sig svarende til Wii-nunchucks placering. | | |
| 5 | Udløs kanon med trigger på wii-nunchuck. | Kanon udløses. | | |
| 6 | Giv Wii-nunchuck til den anden spiller. | Den anden spiller modtager Wii-nunchuck. | | |
| 7 | Gentag punkt 4 til 6 indtil skud er opbrugt. | Punkt 4 til 6 gentages. | | |
| 8 | Kig på brugergrænseflade. | Brugergrænseflade viser info om spil. | | |
| 9 | 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 observation/resultat | Faktisk observation/resultat | Vurdering (OK/FAIL) |
|---|---|---|---|---|
| 1 | Vælg one-player mode. | Brugergrænsefladen viser spilside for one-player mode og anmoder om valg af antal skud. | | |
| 2 | Vælg ti skud på brugergrænseflade. | Brugergrænseflade anmoder om, at der fyldes ti stykker slik i magasin. | | |
| 3 | Fyld ti stykker slik i magasinet og tryk på knap for at starte spil. | Brugergrænseflade går til spilside og anmoder om, at kanon indstilles. | | |
| 4 | 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 start test på brugergrænseflade | Brugergrænsefladen udskriver at SPI og I2C testen er godkendt. Brugergrænsefladen anmoder bruger om tryk på Z på Wii-nunchuck | | |
| 2 | Tryk Z på Wii- nunchuck | 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 DevKittet. | | | |
| 2 | Tryk på start test på brugergrænseflade | Brugergrænsefladen udskriver SPI forbindelses fejlmeddelelse. | | |

### 2.2.2 Use case 2 - Exception 2

| Step | Handling | Forventet observation/resultat | Faktisk observation/resultat | Vurdering (OK/FAIL) |
|---|---|---|---|---|
| 1 | Fjern I2C-kabler fra alle I2C slaver. | | | |
| 2 | Tryk på start test på brugergrænseflade | Brugergrænsefladen udskriver I2C forbindelses fejlmeddelelse. | | |

### 2.2.3 Use case 2 - Exception 3

| Step | Handling | Forventet observation/resultat | Faktisk observation/resultat | Vurdering (OK/FAIL) |
|---|---|---|---|---|
| 1 | Disconnect Wii nunchucken fra systemet. | | | |
| 2 | Tryk på start test på brugergrænseflade | | | |
| 3 | Vent på timeout. | Brugergrænsefladen udskriver Wii Nunchuck forbindelses fejlmeddelelse | | |

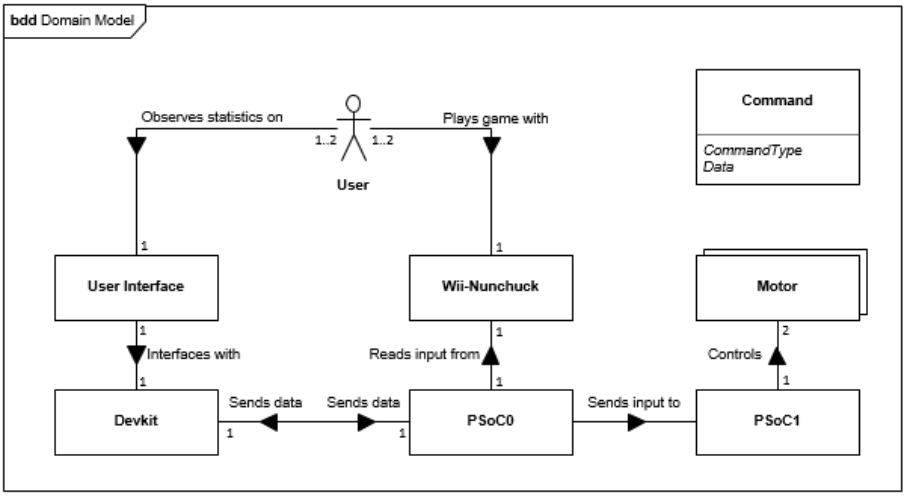
## 2.3 Ikke-funktionelle krav

| Krav | Test | Forventet observation/resultat | Faktisk observation/resultat | Vurdering (OK/FAIL) |
|---|---|---|---|---|
| 1.1 | Bruger styrer kanon fra "top"position til "bund"posiion, og måler vinkelforskellen. | Den afmålte vinkelforskel må være 70 °± 5 ° | | |
| 1.2 | Bruger drejer kanonen fra længst til højre til længst til venstre og måler vinkelforskellen. | Den afmålte vinkelforskel ligger indenfor 70 °± 5 ° | | |
| 2 | Et projektil på 1.25 cm i diameter ± 5mm affyres fra kanonen. | Projektilet bliver affyret | | |
| 3 | Et projektil affyres, og distancen mellem kanonen og stedet hvor projektilet lander måles. | Distancen er blevet målt til at være større end 1 meter. | | |
| 4 | Mål kanonens dimensioner med en lineal. | Dimensionerne overstiger ikke 40cm x 40cm x 40cm. | | |
| 5 | Tryk på "triggeren"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. | | |
| 6 | 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

### 3.0.1 Domæne model

På figur 4 ses domæne modellen for systemet. Denne er udarbejdet ved at lave en navneordsanalyse fra de use cases der er specificerede i afsnit 1.




Figur 4: Domæne model for systemet

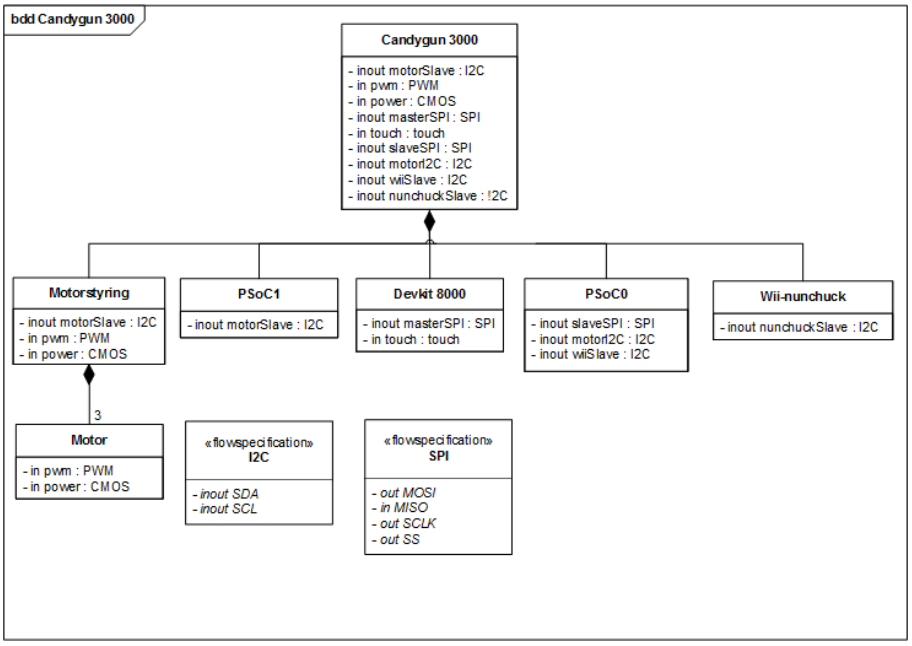
I domæne modellen ses det, at brugeren interagerer med både Wii-nunchucken og med brugergrænsefladen. Brugergrænsefladen er en grænseflade til devkittet. Devkittet kommunikerer med PSoC0, som læser den analoge data der kommer fra Wii-nunchucken. Denne data bliver derefter afkodet og videresendt til PSoC1, som ud fra denne data styrer de forskellige motorer.

## 3.1 Hardware Arkitektur

I hardwarearkitekturen brydes systemet ned i dele, som senere gør det muligt at uddele arbejdsopgaver, og specificere grænseflader. Hardwarearkitekturen består at BDD og IBD for systemet.

### 3.1.1 BDD for Candygun 3000

I BDD-diagrammet på figur 5 er Candygun 3000 brudt ned i blokkene PSoC0, PSoC1, PSoC2 og Devkit 8000. Devkit 8000 er brugergrænsefladen, som brugeren kan interagere med via touchskærmen. Den er forbundet via SPI til PSoC0, som er SPI-slave. PSoC0 er desuden også I2C-master. PSoC0 kommunikerer via I2C til PSoC1 og PSoC2. PSoC1 står for motorstyring, som via et PWM-signal styrer de 3 motorer. PSoC2 har til opgave at aflæse brugerinput fra Wii-nunchucken, som også kommunikerer via I2C. På figur 5 ses de forskellige blokke og deres porte. Desuden er der en flowspecification for I2C og SPI, hvor forbindelserne er beskrevet mere detaljeret (set fra master-synspunkt).




Figur 5: Overordnet BDD for Candygun 3000.

### Blokbeskrivelse

**DevKit 8000**
*DevKit 8000* er en embedded Linux platform med touch-skærm der bruges til brugergrænsefladen for produktet. Det er her hvor brugeren interagerer med systemet og ser status for spillet.

**Motorstyring**
*Motorstyring* er blokken som består af Candy Gun 3000's motorerer - brugt til at styre den - samt *PSoC1*, som bruges til styring af disse motorer.

**Wii-Nunchuck-Styring**
*Wii-Nunchuck-Styring* er blokken som består af den fysiske Wii-Nunchuck controller der bruges af brugeren til at styre kanonen, samt *PSoC2*, som bruges til at videresende I2C dataen fra controlleren.

**Wii-Nunchuck**
*Wii-Nunchuck* er controlleren brugeren styrer kanonen med.

**Motor**
*Motor* blokken er Candy Gun 3000's motorerer der bruges til styring af kanonen i forskellige retninger.

**PSoC0**
*PSoC0* er PSoC hardware der både er I2C master og SPI slave. Denne PSoC fungerer som bindeled mellem resten af systemets hardware, så kommunikation er muligt.

**PSoC1**
*PSoC1* er PSoC hardware der bruges til softwarestyring af Candy Gun 3000's motorerer samt affyringsmekanisme.

**PSoC2**

*PSoC2* er PSoC hardware der bruges til at videresende input data fra Wii-Nunchuck controlleren.

**SPI (FlowSpecification)**

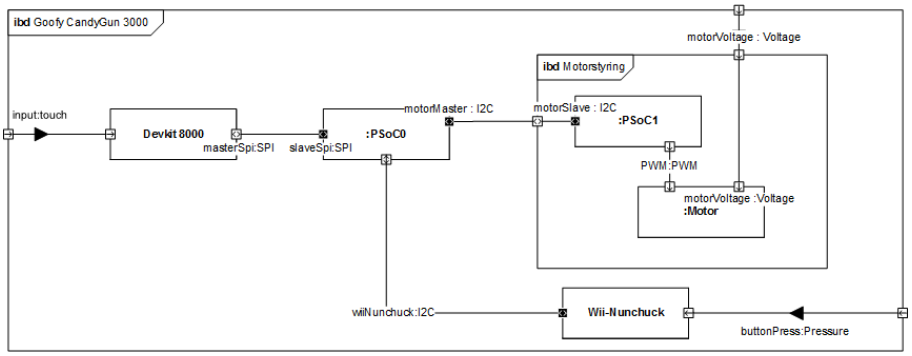
*SPI (FlowSpecification)* beskriver signalerne der indgår i *SPI* kommunikation.

**I2C (FlowSpecification)**

*I2C (FlowSpecification)* beskriver signalerne der indgår i *I2C* kommunikation.

### 3.1.2 IBD for Candygun 3000

I IBD'et på figur 6 er forbindelserne mellem de forskellige blokke overskueliggjort. Det er dermed let at få et overblik over, hvilke grænseflader der skal tages højde for i den videre udvikling.




Figur 6: IBD for Candygun 3000

#### Signalbeskrivelse

Generelt for signalbeskrivelsen gælder, at når et signal beskrives som 'højt' menes der i et spændingsområde på 3.5V til 5 V, som er defineret for CMOS kredse [3]. På samme måde er signaler beskrevet som 'lav' defineret som spændinger indenfor 0 V til 1.5 V.

| **Blok-navn** | **Funktionsbeskrivelse** | **Signaler** | **Signalbeskrivelse** |
|---|---|---|---|
| Devkit8000 | Fungerer som grænseflade mellem bruger og systemet. | masterSPI | Type: SPI<br>Spændingsniveau: 0-5V<br>Hastighed: ?? |
| | | touch | Type: touch<br>Tryk på DevKit8000 display. |
| PSoC0 | Fungerer som I2C master for systemet samt SPI slave til DevKit8000. | slaveSPI | Type: SPI<br>Spændingsniveau: 0-5V<br>Hastighed: ?? |

| | | | |
|---|---|---|---|
| | | masterI2C | Type: I2C<br>Spændingsniveau: 0-5V<br>Hastighed: 100kbit/sekund |
| Motorstyring | Modtager input fra Wii-Nunchuck og omsætter det til PWM signaler. | motorSlave | Type: I2C<br>Spændingsniveau: 0-5V<br>Hastighed: 100kbit/sekund<br>Beskrivelse: Indeholder Wii-Nunchuck data der skal bruges til motorstyring. |
| | | power | Type: $V_{CC}$<br>Spændingsniveau: 5V<br>Beskrivelse: Strømforsyning til motorstyringen. |
| PSoC1 | Modtager input fra Wii-Nunchuck og omsætter det til PWM signaler. | MotorI2C | Type: I2C<br>Spændingsniveau: 0-5V<br>Hastighed: 100kbit/sekund<br>Beskrivelse: Indeholder formatteret Wii-Nunchuck data som skal bruges til styring af motorens PWM signal. |
| | | PWM | Type: PWM<br>Frekvens: 22kHz<br>PWM %: 0-100%<br>Spændingsniveau: 0-5V<br>Beskrivelse: PWM signal til styring af motorens hastighed. Udregnet ud fra MotorI2C signalet. |

| | | | |
|---|---|---|---|
| Motor | Motorerne der skal styre kanonen | PWM | Type: PWM<br>Frekvens: 22kHz<br>PWM%: 0-100%<br>Spædingsniveau: 0-5V<br>Beskrivelse: PWM signal til styring af motorens hastighed. |
| | | power | Type: $V_{CC}$<br>Spændingsniveau: 12V<br>Beskrivelse: Strømforsyning til motorstyringen |
| PSoC2 | Modtager input data fra Wii-Nunchuk og videresender det i behandlet format. | wiiSlave | Type: I2C<br>Spændingsniveau: 0-5V<br>Hastighed: 100kbit/sekund<br>Beskrivelse: Sender input data fra Wii-Nunchuck til PSoC2. |
| | | WiiI2C | Type: I2C<br>Spændingsniveau: 0-5V<br>Hastighed: 100kbit/sekund<br>Beskrivelse: Videresender behandlet Wii-Nunchuk data til andre dele af systemet. |
| Wii-nunchuck | Den fysiske controller som brugeren styrer kanonen med. | WiiSlave | Type: I2C<br>Spændingsniveau: 0-5V<br>Hastighed: 100kbit/sekund<br>Beskrivelse: Denne I2C linje bruges til kommunikation mellem PSoC 2 og Wii-Nunchuck. |

| | | | |
|---|---|---|---|
| | | buttonPress | Type: I2C<br>Det fysiske tryk når brugeren trykker på Wii-Nunchuck knapper. |
| SPI | Denne blok beskriver den ikke-atomiske SPI forbindelse. | MOSI | Type: CMOS<br>Spændingsniveau: 0-5V<br>Hastighed: ??<br>Beskrivelse: Binært data som sendes fra master til slave. |
| | | MISO | Type: CMOS<br>Spændingsniveau: 0-5V<br>Hastighed: ??<br>Beskrivelse: Binært data som sendes fra slave til master. |
| | | SCLK | Type: CMOS<br>Spændingsniveau: 0-5V<br>Hastighed: ??<br>Beskrivelse: Clock signalet fra master til slave, som bruges til at synkronisere den serielle kommunikation. |
| | | SS | Type: CMOS<br>Spændingsniveau: 0-5V<br>Hastighed: ??<br>Beskrivelse: Slave-Select, som bruges til at vælge slaven der skal modtage og sende data. |
| I2C | Denne blok beskriver den ikke-atomiske I2C forbindelse. | SDA | Type: CMOS<br>Spændingsniveau: 0-5V<br>Hastighed: ??<br>Beskrivelse: Databussen mellem I2C masteren og I2C slaver. |

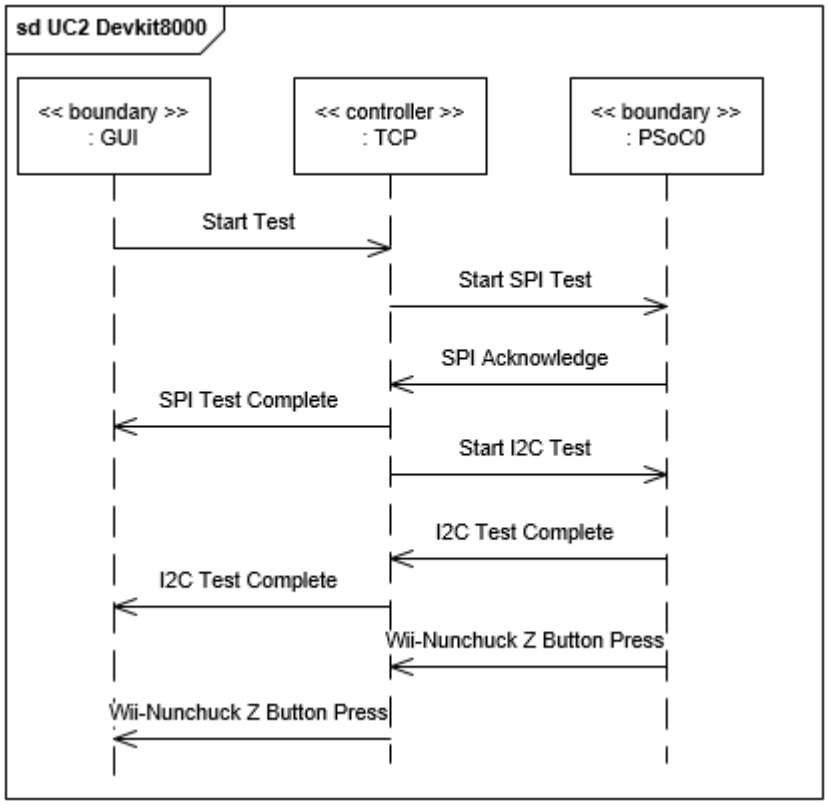
| | | | |
|---|---|---|---|
| | | SCL | Type: CMOS<br>Spændingsniveau: 0-5V<br>Hastighed: ??<br>Beskrivelse: Clock signalet fra master til lyttende I2C slaver, som bruges til at synkronisere den serielle kommunikation. |

## 3.2 Software Arkitektur

I softwarearkitekturen udarbejdes der applikationsmodeller bestående af sekvensdiagrammer og klassediagrammer for hvert delsystem. Denne arkitektur overskueliggører kravene til de boundaryklasser, der muliggør kommunikation mellem delsystemerne. Derudover bliver der gennem analyse af use case og sekvensdiagrammer udledt grundlæggende metoder i klasserne.

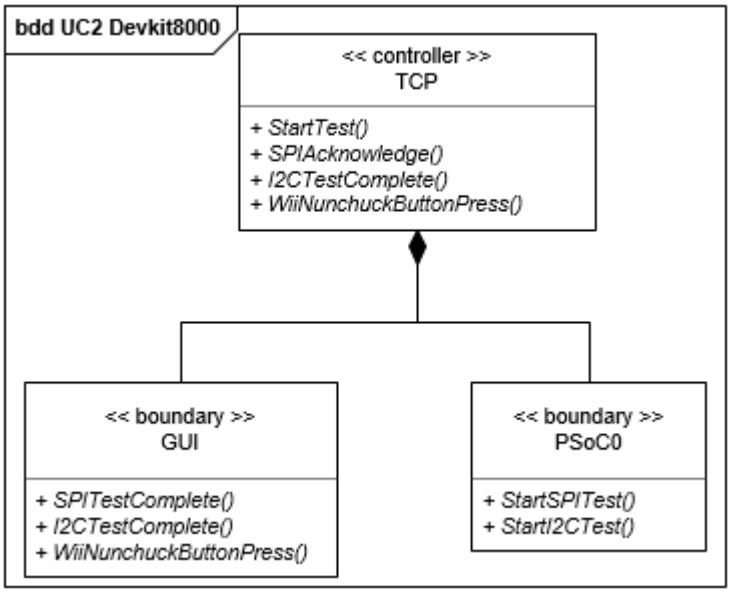
### 3.2.1 Applikationsmodel for Devkit 8000

Sekvensdiagrammet for Devkit 8000 ses på figur 7. Der tages udgangspunkt i use case 2. Der er to boundaryklasser, da brugergrænsefladen skal håndtere kommunikationen mellem brugeren og PSoC0. Brugeren interagerer via en grafisk brugergrænseflade (GUI). Boundaryklassen ComProtocol, skal håndtere SPI-kommunikationen til PSoC0. Som det ses af diagrammet initieres testen af brugeren og derefter er det kontrolklassen, der sørger for, at de forskellige tests bliver sat i gang. Når en test er færdiggjort meldes resultatet ud til brugeren via GUI'en.




Figur 7: Sekvensdiagram for Devkit 8000

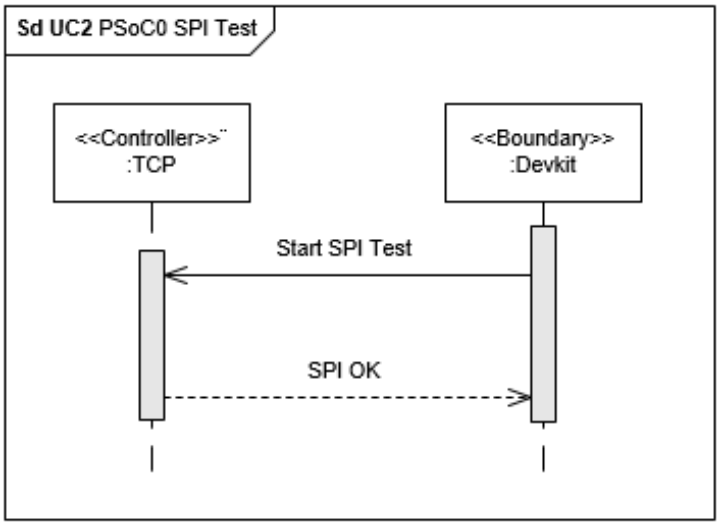
Ud fra sekvensdiagrammet for Devkit 8000 er der udledt disse metoder til klasserne. De ses i klassediagrammet på figur 8.




Figur 8: Klassediagram for Devkit 8000

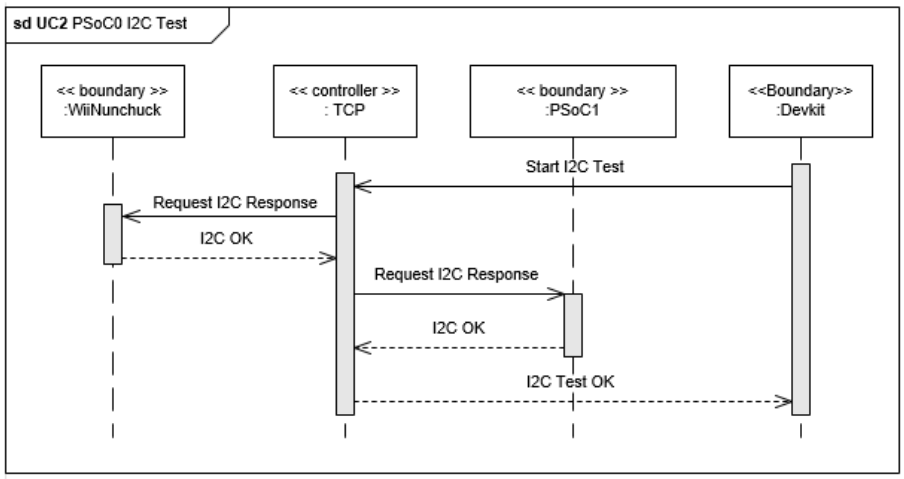
### 3.2.2 Applikationsmodel for PSoC0

På figur 9, 10 og 11 ses sekvensdiagrammer for PSoC0 med udgangspunkt i use case 2. Sekvensdiagrammerne er blevet opdelt i de 3 tests der gennemføres i use casen - nemlig I2C, Nunchuck og SPI kommunikations tests. Kontrolklassen er Test Communication Protocol, hvilket på figurerne er forkortet til TCP. Derudover er der tre boundaryklasser, da PSoC0 skal kommunikere både med Devkit 8000, Nunchucken og PSoC1. På figur 12 ses klasse diagrammet for PSoC0, der udledes af de tre sekvensdiagrammer.



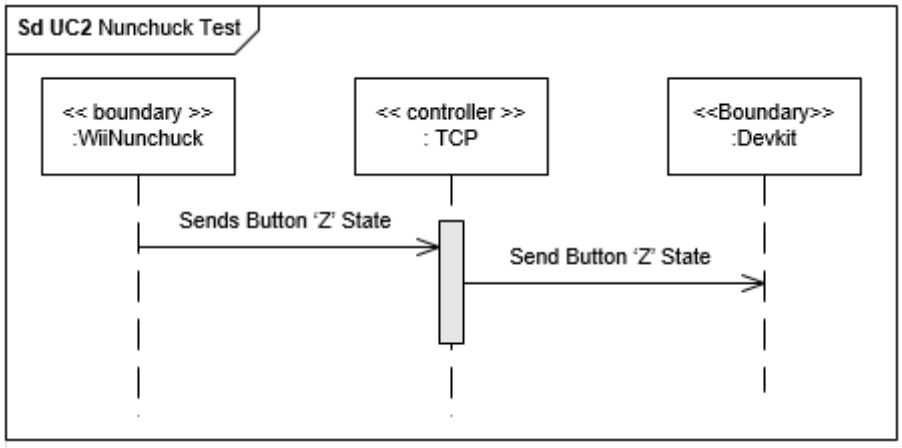

Figur 9: Sekvensdiagram for PSoC0 SPI test

På figur 9 ses, at Devkittet sender en besked til kontrolklassen for at påbegynde SPI testen. Når testen er udført, svarer denne med en SPI OK og derved er SPI testen gennemført.



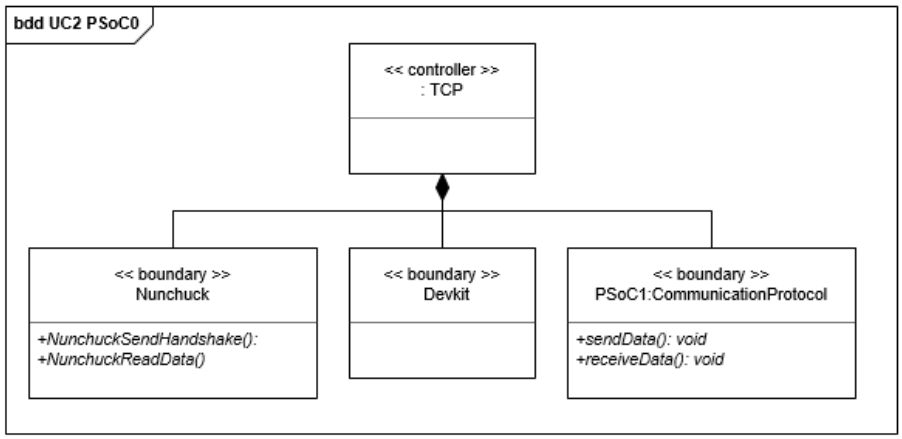

Figur 10: Sekvensdiagram for PSoC0 I2C test

På figur 10 ses, at Devkittet starter I2C testen, ved at sende en besked til kontroller klassen. Kontrolklassen anmoder herefter om svar via I2C nettet fra Nuchucken og PSoC1. Når disse enheder har svaret med et I2C OK er testen gennemført og der sendes besked til Devkittet med en I2C Test OK.




Figur 11: Sekvensdiagram for PSoC0 Nunchuck test

På figur 11 ses, at Nunchucken sender 'Z' knappens status (tryket eller ikke-trykket) til kontroller klassen. Denne videresender knappens status til Devkittet, og derved er testen færdig. **Devkit skal anmode om knaptryk på display. Ændres når SPI inkluderes**

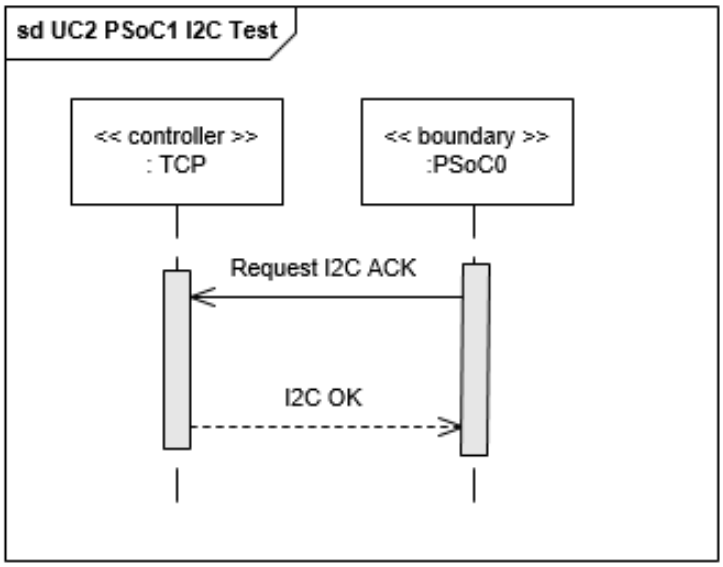



Figur 12: Klassediagram for PSoC0.

I klassediagrammet på figur 12 ses kontrolklassen og de tre boundaryklasser, som hører til PSoC0. I klasserne er der tilføjet metoder, som er udledt ud fra sekvensdiagrammerne.

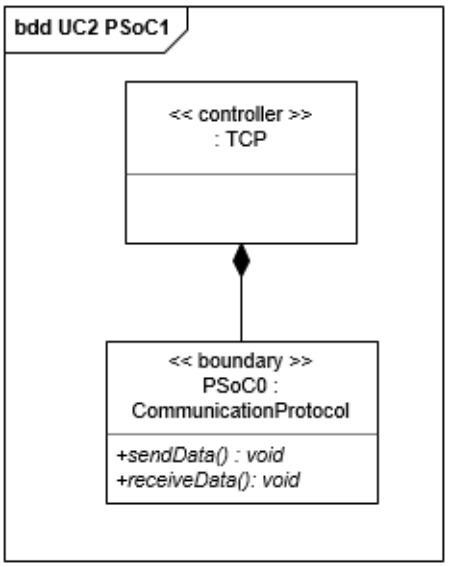
### 3.2.3 Applikationsmodel for PSoC1

Sekvensdiagrammet for PSoC1 ses på figur 13. Som forrig afsnit er kontrolklassen Test Communication Protocols, hvilket i diagrammet er forkortet til TCP. I dette tilfælde er der kun én boundaryklasse, da PSoC1, i denne use case, kun anvendes til I2C testen og derfor kun skal den kommunikere med PSoC0.




Figur 13: Sekvensdiagram for PSoC1

På figur 13 ses, at boundaryklassen anmoder om respons fra kontrolklassen til at bekræfte om at I2C slaven kan kommunikeres med. Hvis dette er tilfældet sendes der I2C OK tilbage til boundaryklassen.



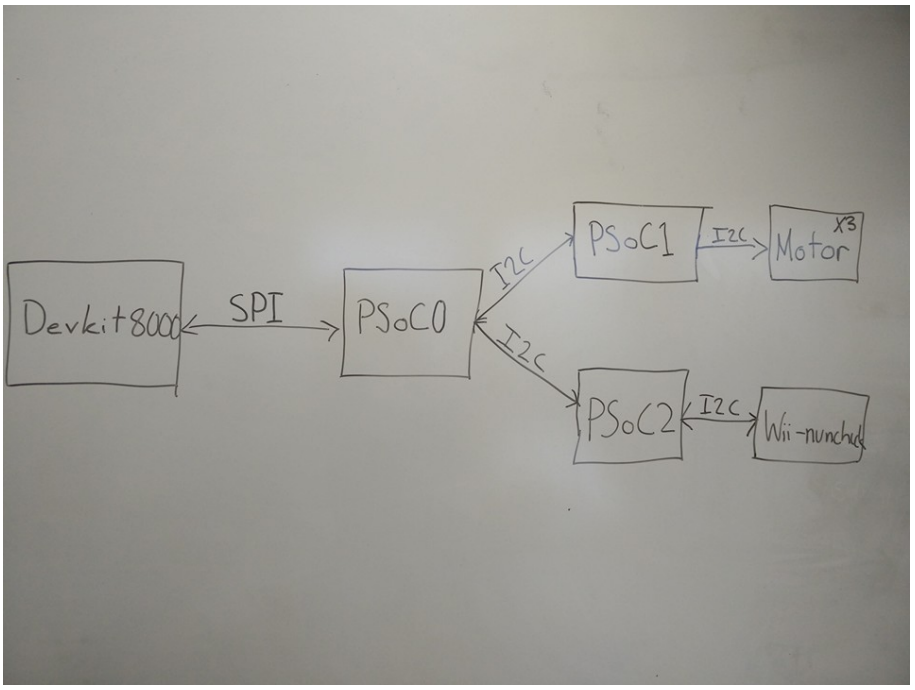

Figur 14: Klassediagram for PSoC1

Fra sekvensdiagrammet på figur 13 udledes et klassediagram som ses foroven i figur 14.

### 3.2.4 Kommunikationsprotokoller

I dette afsnit beskrives de kommunikationsprotokoller der anvendes til at sende data mellem systemets komponenter på de brugte bustyper - I2C og SPI.

På figur 15 gives et overblik over forbindelserne mellem dets embedded linux platform og microcontrollers. Ved hver forbindelse ses typen af bus der bruges.




Figur 15: Forbindelser mellem systemets komponenter

#### SPI Protokol

#### I2C Protokol

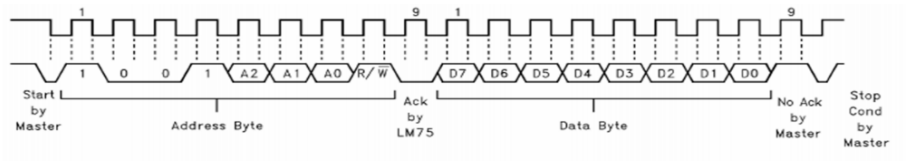
I2C[1] er en bus bestående af to ledninger. Den ene ledning bruges som databus og navngives *Serial Data Line* (SDA). Den anden ledning bruges til clock signalet, til synkronisering kommunikationen, og navngives *Serial Clock Line* (SCL). Enheder på I2C bussen gør brug af et master-slave forhold til at sende og læse data. En fordel ved I2C bussen er at netværket kan bestå af multiple masters og slaver, hvilket gavner sig godt for dette system da fire 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 3 ses addresserne tildelt systemets PSoCs.

| I2C Adresse bits | 7 | 6 | 5 | 4 | 3 | 2 | 1 | LSB er read/write indikator |
|---|---|---|---|---|---|---|---|---|
| PSoC0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0/1 |
| PSoC1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0/1 |
| PSoC2 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0/1 |

Tabel 3: Adresser brugt på systemets I2C bus

Den integrerede I2C protokol sender data serielt i pakker af 8-bit (1 byte). På figur 16 ses et timing-diagram for aflæsning af 1 byte. Her ses at processen begynder med en addresse-byte, efterfulgt af en data-byte.

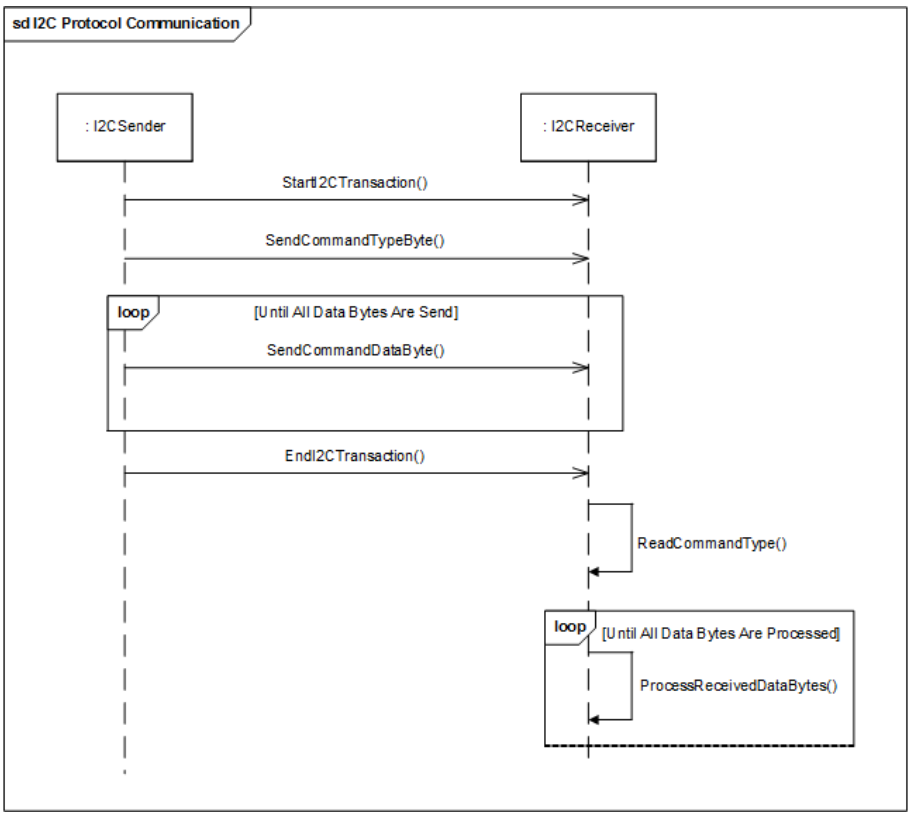



Figur 16: Timing Diagram af 1-byte I2C aflæsning

Systemet gør brug af denne integrerede I2C protokol via en højere abstraheret *Application Programming Interface* (API). Ved brug af denne API er en brugerdefineret protokol udviklet, som gør det muligt at sende kommandoer og data mellem systemets PSoC's.

Da I2C data udveksling - som beskrevet før - underliggende sker bytevist, er den brugerdefinerede I2C protokol opbygget ved at første modtagede byte indikerer typen af kommando. Herefter følger $N$ bytes som 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.

Modtagere bruger kommandoens type til at vide hvordan de efterfølgende data-bytes fortolkes. På figur 17 ses et sekvensdiagram der demonstrerer forløbet mellem en I2C afsender og modtager ved brug af I2C protokollen via pseudo-kommandoer.




Figur 17: Eksempel af I2C Protokol Forløb

Det kan på figur 17 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 frit kan fortolke $N$ antal modtagne bytes afhængig af den modtagne kommandotype.

På tabel 4 ses de definerede kommandoer der gøres brug af.

| Kommando Type | Beskrivelse | Binær værdi | Hex værdi | Data bytes |
|---|---|---|---|---|
| NunchuckData | Indeholder aflæst data fra Wii Nunchuck controlleren | 0010 1010 | 0xA2 | Byte #1 Analog X-værdi<br>Byte #2 Analog Y-værdi<br>Byte #3 Analog ButtonState |
| I2CTestRequest | Beder PSoC om at starte en I2C kommunikations test | 0010 1001 | 0x29 | Ingen Databytes |
| I2CTestACK | Beder om at få en I2C OK besked tilbage fra I2C enhed | 0010 1000 | 0x28 | Ingen Databytes |

Tabel 4: De forskellige I2C Kommandoer der bruges

Kolonnerne "Binær Værdi"og "Hex Værdi"i tabel 4 viser kommandotypens unikke tal-ID i både binær- og hexadecimalform. Det er denne værdi der sendes som den første byte, for at identificere kommandotypen.

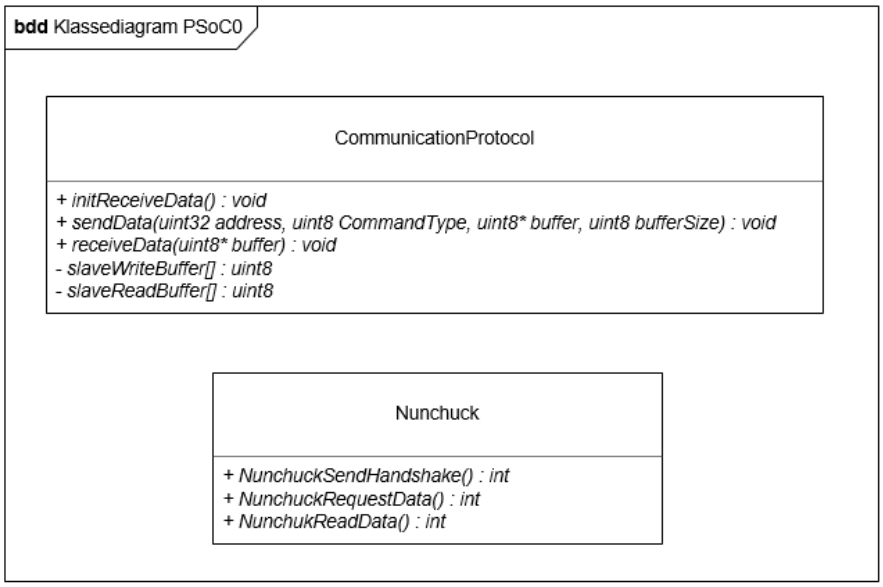
# 4 Design og implementering

## 4.1 Software Design

### 4.1.1 Klassediagrammer

#### PSoC0

På figur 18 ses klassediagrammet indeholdende klasserne som bruges af softwaren allokeret på CPU'en PSoC0.




Figur 18: Klassediagram for CPU'en PSoC0

På figur 18 kan det ses at softwaren på PSoC0 CPU'en gør brug af klasserne *CommunicationProtocol* samt *Nunchuck*.

Klassebeskrivelser kan findes i **!INDSÆT BILAG TIL DOXYGEN!**

### 4.1.2 Afkodning af Wii-Nunchuck Data Bytes

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

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

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

### 4.1.3 Kalibrering af Wii-Nunchuck Analog Stick

De afkodede bytes for Wii-Nunchuck's analoge stick har definerede standardværdier for dets forskellige fysiske positioner. Disse værdier findes i tabel 5

| | |
|---|---|
| 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 5: 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*

## 4.2 Hardware Design

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

- Motorstyring
- Tre motorer

### 4.2.1 Motorstyring

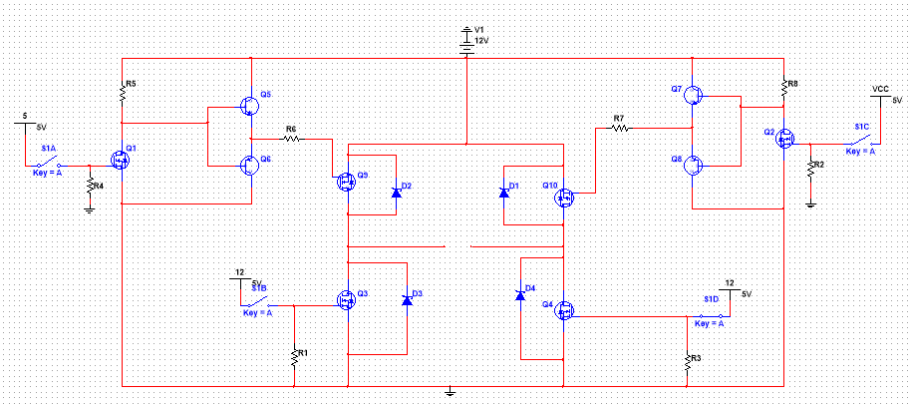
Til at styre de tre motorer er der bygget en H-bro, der skal bruges i tre eksemplarer. To af disse motorer skal kunne styre kanonen, så den kan køre op og ned og frem og tilbage. Den tredje skal bruges til at styre affyringsekanismen.

#### H-bro

Først blev der designet en H-bro, som bestod af to N-MOSFET's af typen IRLZ44 og to P-MOSFET's af typen ZVP3306. Denne kan ses på figur... Det viste sig dog, at den P-MOSFET der var brugt, var for svag til at kunne trække den strøm, som motoren skulle bruge, hvilket betød, at den blev brændt af.

Tabel 6: Komponentbetegnelser på H-bro

| Betegnelse | Komponent |
|---|---|
| VCC | 5V |
| Q1 | IRLZ44(mosfet N-Channel) |
| Q2 | IRLZ44(mosfet N-Channel) |
| Q3 | IRLZ44(mosfet N-Channel) |
| Q4 | IRLZ44(mosfet N-Channel) |
| Q5 | BC547 |
| Q6 | BC557 |
| Q7 | BC547 |
| Q8 | BC557 |
| Q9 | IRF9Z34N(mosfet P-Channel) |
| Q10 | IRF9Z34N(mosfet P-Channel) |
| R1 | 10kΩ |
| R2 | 10kΩ |
| R3 | 10kΩ |
| R4 | 10kΩ |
| R5 | 10kΩ |
| R6 | 100Ω |
| R7 | 100Ω |
| R8 | 10kΩ |
| D1 | IN5819 |
| D2 | IN5819 |
| D3 | IN5819 |
| D4 | IN5819 |




Figur 19: H-bro kredsløb

### Mosfet

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 kredsløbstegningen) og de to andre mosfet er af typen IRLZ44 (mosfet N-Channel, som er Q3 og Q4 kredsløbstegningen). 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-channel:

- Mosfet P-channel:
  Den kan klare en strøm på 6,7A ifølge databladet. Det vil altså ikke komme til at påvirke motoren, som kan trække en strøm på 0,35A.

  For at der kan løbe spænding igennem IRF9Z34N(mosfet P-channel), så skal den have en negativ spænding for at åbne og en spænding på over 0V for at lukke.

### Diode

Over fire af mosfetene (Q9, Q10, Q3 og Q4) er der sat en diode af typen IN5819. Den skal fungere som beskyttelse af de fire mosfet (Q9, Q10, Q3 og Q4). Det, de gør, er, at de sikrer, at den spænding, som er tilbage i motoren, når man lukker for mosfetene, ikke løber tilbage ind i mosfetene og brænder dem af.

### Modstande

- Pull down modstande:
  Der er blevet brugt fire modstande (R1, R2, R3 og R4), som pull down modstande, som sørger for, at signalet vil blive holdt lavt, når der ikke er trykket, så det ben ikke står og flyver, så det kan komme til at åbne en mosfet, ved fejl og derved kommer til at brænde en mosfet eller motoren af. Der er valgt en modstand på 10kOhm, som er lille nok til at trække de små spændinger ned, når der ikke er trykket og den stor nok til at spændingen ikke løber der ned, når der er trykket.

- modstande
  - R6 og R7
  - R5 og R8

### Motor

# 5 Test

## 5.1 Modultest

### 5.1.1 Software

**Modultest af Wii-Nunchuck**

På PSoC0 er der software til aflæsning af Wii-Nunchuck input data. Følgende afsnit beskriver test af dette software.

Aflæsning af Wii-Nunchuck sker i to skridt, som begge verificeres ved modul test. 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.

**Test af Wii-Nunchuck Handshake**

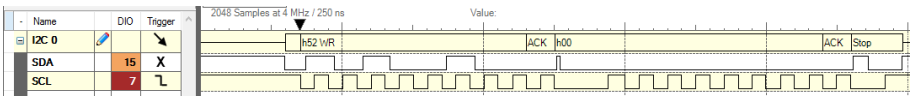
**Test af data udveksling mellem PSoC0 og Wii-Nunchuck**

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.

**Afsendelse af 0x00 byte**

Den første forventede I2C besked er en *0x00* byte fra PSoC0 for at starte en ny aflæsning. På figur 20 ses aflæsningen af I2C bussen på tidspunktet hvor anmodningen til Wii-Nunchuck bliver udført. Dette er en tidslinje læst fra ventre til højre.




Figur 20:

Det kan på figur 20 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 at ACK fra Wii-Nunchuck. Til sidst afsluttes I2C transaktionen ved "Stop".

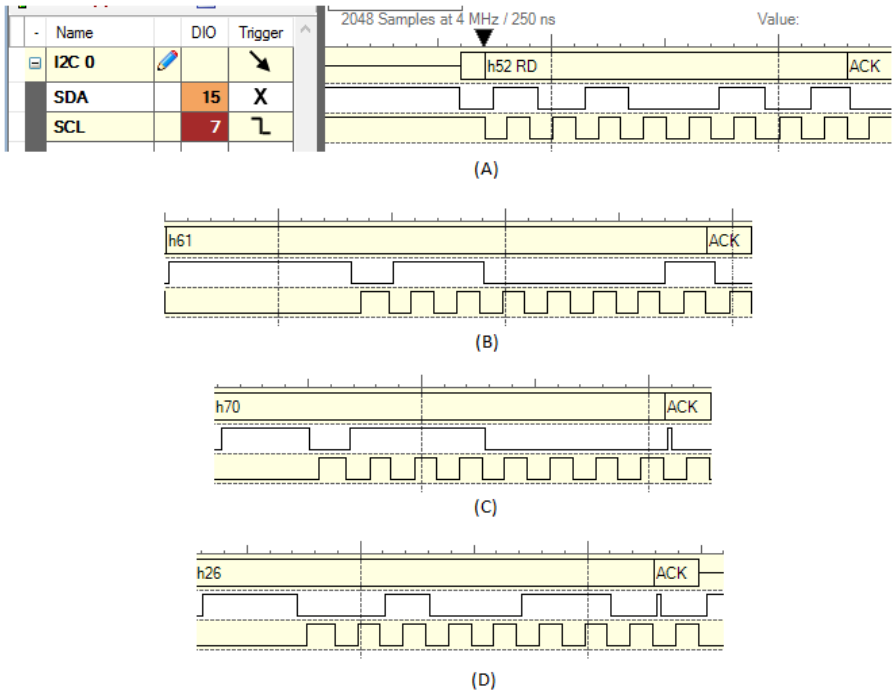
Det kan altså konkluderes at målingen er i overensstemmelse med forventningen om at en 0x00 byte skal sendes til Wii-Nunchuck for opstart af dataudveksling.

**Aflæsning af Wii-Nunchuck**

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

Her forventes en række beskeder indeholdende

På figur 21 ses I2C beskederne der bliver udvekslet mellem PSoC0 og Wii-Nunchuck efter vellykket Wii-Nunchuck Handshake.




Figur 21: Tidslinje af aflæste I2C beskeder af PSoC0 fra Wii-Nunchuck

### I2C Protokol

PSoC0 og PSoC1 kommunikerer over en I2C bus via I2C protokollen beskrevet i afsnit 3.2.4. Dette afsnit beskriver test af denne protokol. Følgende test tager udgangspunkt i kommandotypen *NunchuckData* beskrevet i tabel 4).

### Test af NunchuckData kommandotype

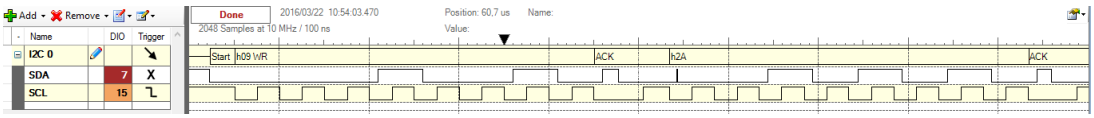
Testen blev udført i to dele. I første del måles I2C bussen ved brug af Analog Discovery's Logic Analyzer; for at verificere at den forventede kommandotype bliver overført via bussen. Anden del verificerer at den overførte data er modtaget korrekt via PSoC Creator's debugger.

### NunchuckData kommandotype test del 1

I testen afsendes, som nævnt i afsnittets indledning, kommandotypen NunchuckData. Som vist i tabel 4 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 22.




Figur 22: Tidslinje af målt I2C kommandotype

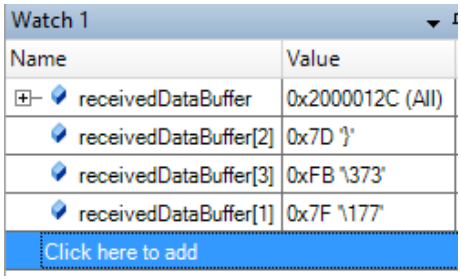
Det kan ses på figur 22 at I2C overførslen starter med en I2C *write*, som får et successfuldt acknowledge fra slaven PSoC1. Herefter kan det ses at den næste byte der sendes har værdien 0x2A. Denne byte er kommandoentypens ID, og er altså som forventet 0x2A.

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

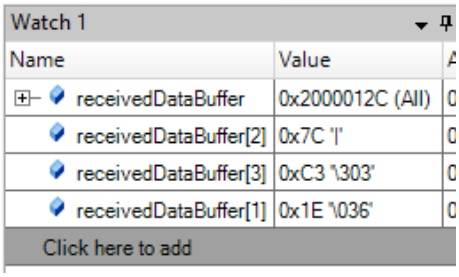
### NunchuckData kommandotype test del 2

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

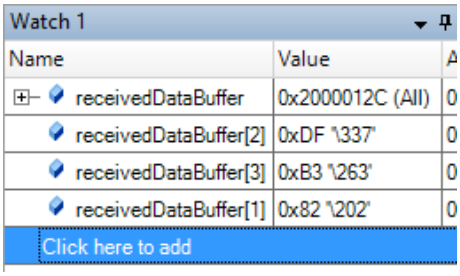
Testen gennemføres ved at fortolke den modtagne data tre gange, hvor nunchucken er i forskellige tilstande (hvilken retning det analoge stik er trykket) i hver test. Værdierne sammenlignes de forventede standardværdier som ses i tabellen på side 3 i [2, I2C Interface with Wii Nunchuck]. Da testene kun 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 5 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 23, 24 og 25.




Figur 23: Afmåling af modtager-buffer på PSoC1 efter at have modtaget "NunchuckData"kommando typen. Intet input på Nunchuck'en




Figur 24: Afmåling af modtager-buffer på PSoC1 efter at have modtaget "NunchuckData"kommando typen. Den analoge stick er presset til venstre på Nunchuck'en




Figur 25: Afmåling af modtager-buffer på PSoC1 efter at have modtaget "NunchuckData"kommando typen. Den analoge stick er presset frem på Nunchuck'en

På figur 24 ses afmålingen af modtager-bufferen når Nunchuckens 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 25 ses afmålingen af modtager-bufferen når Nunchuckens 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 23 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 vi konkludere at implementeringen af I2C protokollen fungerer efter hensigten.

### 5.1.2 Hardware

## 5.2 Integration

## 5.3 Accepttest

# 6 Referencer

## Litteratur

[1] UM10204, *I2C Bus Specification and user manual*, 4 April 2014

[2] Assignment 6 - I2C Interface with Wii Nunchuck, *I2C Interface with Wii Nunchuck*, CSE325 Embedded Microprocessor Systems

[3] Logic Signal Voltage Levels, *All about circuits*, Kuphaldt Tony R., `http://www.allaboutcircuits.com/textbook/digital/chpt-3/logic-signal-voltage-levels/` Besøgt 22 Marts 2016