**E3 PRJ3**

**Efterår 2014**

**BODY ROCK 3000**

**Rapport**

**Gruppe 9**

**Deltagere:**

|  |
| --- |
| ***#1*** Stud.nr.: 201370738 Navn: Kristian Boye Jakobsen |
| ***#2*** Stud.nr.: 201205998 Navn: Lasse Fisker |
| ***#3*** Stud.nr.: 201270810 Navn: Mathias Siig Nøregaard |
| ***#4*** Stud.nr.: 201370768 Navn: Lukas Hedegaard Jensen |
| ***#5*** Stud.nr.: 201370914 Navn: Jonas Evers Nikolajsen |
| ***#6*** Stud.nr.: 201370801 Navn: Jeppe Hofni Hansen |
| ***#7*** Stud.nr.: 201371008 Navn: Felix Blix |
| ***#8*** Stud.nr.: 201370952 Navn: Kristoffer Lerbæk Pedersen |

**Vejleder: Peter Høgh**

16. december 2014

Indhold

[Projektformulering 5](#_Toc406434451)

[Termliste 6](#_Toc406434452)

[Body 6](#_Toc406434453)

[Rock 6](#_Toc406434454)

[BodyRock3000 6](#_Toc406434455)

[Lydpakke: 6](#_Toc406434456)

[MIDI-parameter 6](#_Toc406434457)

[Lydmodul 6](#_Toc406434458)

[Mapping scheme 6](#_Toc406434459)

[Sensorkonfiguration 6](#_Toc406434460)

[Preset 6](#_Toc406434461)

[Kravspecifikation 7](#_Toc406434462)

[Use Case diagram 7](#_Toc406434463)

[Aktørbeskrivelse 8](#_Toc406434464)

[Bruger 8](#_Toc406434465)

[MIDI-modtager 8](#_Toc406434466)

[1.1.1 Højtalersystem 8](#_Toc406434467)

[1.4 Use Case beskrivelser 9](#_Toc406434468)

[1.4.1 Forbind Body og Rock 9](#_Toc406434469)

[1.4.2 Installér lydpakker 11](#_Toc406434470)

[1.4.3 Konfigurer sensorer 12](#_Toc406434471)

[1.4.4 Konfigurer preset 14](#_Toc406434472)

[1.4.5 Vælg preset 16](#_Toc406434473)

[1.4.6 Indsamle sensordata 17](#_Toc406434474)

[1.4.7 Generér MIDI 18](#_Toc406434475)

[1.4.8 Afspil lyd 19](#_Toc406434476)

[Krav til Mapping Scheme 21](#_Toc406434477)

[Ikke funktionelle krav 22](#_Toc406434478)

[Generelt 22](#_Toc406434479)

[Body 22](#_Toc406434480)

[Rock 22](#_Toc406434481)

[Systemarkitektur Hardware 23](#_Toc406434482)

[Overordnet arkitektur 23](#_Toc406434483)

[Domæne model BodyRock3000 23](#_Toc406434484)

[Body HW Arkitektur 24](#_Toc406434485)

[BDD Body 24](#_Toc406434486)

[Allokeringsdiagram Body 25](#_Toc406434487)

[IBD Body 26](#_Toc406434488)

[Sensor IBD (Generisk) 27](#_Toc406434489)

[Rock HW Arkitektur 27](#_Toc406434490)

[BDD Rock 27](#_Toc406434491)

[Allokeringsdiagram Rock 28](#_Toc406434492)

[Grænsefladebeskrivelse 29](#_Toc406434493)

[Trådløs kommunikations protokol 30](#_Toc406434494)

[Systemarkitektur Software 32](#_Toc406434495)

[Detaljeret domænemodel for software-moduler 33](#_Toc406434496)

[Klasse-identifikation 34](#_Toc406434497)

[ALSA 36](#_Toc406434498)

[Anvendte ALSA-elementer 36](#_Toc406434499)

[RawMidi 36](#_Toc406434500)

[Virmidi 36](#_Toc406434501)

[Aconnect 36](#_Toc406434502)

[Body 37](#_Toc406434503)

[Klassediagram Body 37](#_Toc406434504)

[Rock 42](#_Toc406434505)

[Tråd kommunikation 42](#_Toc406434506)

[Slow Lane 43](#_Toc406434507)

[Fast Lane 45](#_Toc406434508)

[Overordnet Klassediagram Rock 51](#_Toc406434509)

[Design og implementering (HW) 52](#_Toc406434510)

[Trådløs kommunikation 52](#_Toc406434511)

[Rock - trådløs kommunikation 52](#_Toc406434512)

[Body - trådløs kommunikation 54](#_Toc406434513)

[Test (HW) 55](#_Toc406434514)

[Enhedstests 55](#_Toc406434515)

[Trådløs kommunikation moduler 55](#_Toc406434516)

[Integrationstests 59](#_Toc406434517)

[Design og implementering (SW) 59](#_Toc406434518)

[Implementering af Body Control 59](#_Toc406434519)

[Body - Main 59](#_Toc406434520)

[Body - SerialUnit 62](#_Toc406434521)

[Body - Sensor 64](#_Toc406434522)

[FlexiForce A301 Sensor 67](#_Toc406434523)

[FlexiForce A301 Main 67](#_Toc406434524)

[Test (SW) 68](#_Toc406434525)

[Enhedstests 68](#_Toc406434526)

[Sensor software 68](#_Toc406434527)

[Integrationstests 71](#_Toc406434528)

[Integrationstest af fast lane (Fra sensor til ekstern MIDI output) 71](#_Toc406434529)

[Integrationstest af sensordataforsendelse på Body 73](#_Toc406434530)

[Accepttest 74](#_Toc406434531)

[Referencer 74](#_Toc406434532)

[Bilag 75](#_Toc406434533)

# Projektformulering



Opgaven i dette projekt er at udvikle et intuitivt musikinstrument, baseret på kropslige bevægelser, der kan implementeres på diverse synthesizere, drum machines og DAW's (Digital Audio Workstation).

Instrumentet skal kunne afspille lydsamples, samt generere MIDI-tone- og CC-signaler, på baggrund af data fra

* Accelerationer
* Tilt
* Afstande
* Taktile tryk

Dette resulterer i at systemet skal indeholde accelerometer-, gyroskop-, proksimitets- og taktile tryk-sensorer. Systemet skal på baggrund af disse:

* Afspille samples
* Generere MIDI-signaler

Systemet består af to dele, som kommunikerer trådløst:

1. En bærbar enhed med sensorer til aflæsning af kropslige bevægelser og positioner
2. En stationær enhed til processering af aflæst rådata fra sensorer, afspilning af samples fra downloadede lydpakker, generering af MIDI-signaler og styring af systemindstillinger

Visionen bag projektet er at skabe et nyt udtryks-medie for musikere, foruden at inkludere hidtil ulærte musiktalenter i den kreative og musikalske glæde, systemet vil medføre.

Systemets fleksibilitet og muligheder for udvidelse, samt den lette integration med eksisterende synthesizer- og sample-systemer, gør det til en kærkommen udvidelse af repertoiret for midler til musikalsk udtryk.

# Termliste

### Body

Body er den bærbare enhed for systemet BodyRock3000. Den består af en controller-enhed, som kan sende sensordata til den stationære enhed Rock. På controller-enheden kan der tilsluttes et vilkårligt antal sensorer, der kan foretage accelerometer-, gyroskop-, proksimitets- og taktile trykmålinger. Da det er et krav at anvende PSoC4-boardet i projektet, vil denne blive brugt som controller-enhed.

### Rock

Rock er den stationære enhed for systemet BodyRock3000. Denne enhed modtager sensordata fra Body-enheden via trådløs kommunikation. Sensordataene konverteres herefter til MIDI, som fastlægger lydoutputtet. Gruppen har opnået dispensation til at erstatte DevKit8000 med Raspberry Pi i projektet. Denne vil blive brugt som stationær enhed.

### BodyRock3000

BodyRock3000 er en samlet betegnelse for systemet. Denne bruges bl.a. som initiator ved Use Case 6, og indikerer at Use Casen startes af interne processer i systemet.

### Lydpakke:

Bestemmer hvilke lyde der skal syntetiseres, på baggrund af de modtagne MIDI-signaler.

### MIDI-parameter

En MIDI parameter er en type besked, der kan sendes over MIDI. Der er generelt tre typer MIDI-parametre:

* **Note**: Angiver hvilken tone, der skal afspilles i intervallet [0-127]
* **Control Change (CC):** Angiver hvilken parameter, der ønskes ændret i intervallet [0-127]
* **Program Change**: Angiver hvilket preset (på eksterne MIDI-enheder), der ønskes i intervallet [0-127]

### Lydmodul

Et digitalt datamodul, der agerer som bindeled mellem de valgte lydpakker og MIDI-parameter-inputs, således at den korrekte lyd afspilles analogt.

### Mapping scheme

En agenda for hvorledes et givet sensor-input mappes til en given MIDI-parameter.

### Sensorkonfiguration

En sensorkonfiguration indeholder:

* ID for en specifik sensor-enhed
* Mapping scheme
* Lydpakke

### Preset

Et sæt af sensorkonfigurationer, der simultant[[1]](#footnote-1) kan indlæses på systemet

# Kravspecifikation

Kravspecifikationen er udfærdiget med basis i en Use Case-baseret tilgang. Den indeholder en beskrivelse over relevante aktører for systemet, samt Fully Dressed Use Cases.

## Use Case diagram

Herunder ses et UC diagram for systemet BodyRock3000. Bemærk at UC7 og UC8 er systeminitierede via hhv. UC6 og UC7.



## Aktørbeskrivelse

### Bruger

|  |  |
| --- | --- |
| **Aktørnavn:** | **Bruger** |
| **Type:** | Primær |
| **Beskrivelse:** | Ønsker at benytte systemet, herunder til at indstille diverse konfigurationer for systemet ved hjælp af Rock. |

### MIDI-modtager

|  |  |
| --- | --- |
| **Aktørnavn:** | **MIDI-modtager** |
| **Type:** | Sekundær |
| **Beskrivelse:** | Ekstern MIDI-enhed som overholder MIDI-standarden[[2]](#footnote-2). Denne vil i praksis kunne være alt fra synthesizere, drum machines, pedal-board loopere, samt nogle lys-systemer. |

### Højtalersystem

|  |  |
| --- | --- |
| **Aktørnavn:** | **Højtalersystem** |
| **Type:** | Sekundær |
| **Beskrivelse:** | Det eksterne højtalersystem modtager lydsignal fra audio-outputtet, og afspiller den ønskede lyd |

## Use Case beskrivelser

For alle use cases gælder det, at der i enhver undermenu, altid findes mulighed for at annullere og vende tilbage til hovedmenuen.

### Forbind Body og Rock

|  |  |
| --- | --- |
| **Navn** | **Forbind Body og Rock** |
| **Use Case ID** | 1 |
| **Samtidige forekomster** | 1 (inklusiv denne) |
| **Primær aktør** | Bruger |
| **Sekundær aktør** | Ingen |
| **Initiator** | Bruger |
| **Prækondition** | Bruger har adgang til Body og Rock |
| **Formål** | Body og Rock er forbundne og kan anvendes |
| **Postkondition** | Body og Rock er klar til brug |

|  |
| --- |
| **Hovedscenarie** |
| 1. Bruger tænder Body og Rock 2. Systemet viser hovedmenuen på Rocks display 3. Systemet forbinder automatisk Body og Rock   [Undtagelse 3.a: Rock kunne ikke finde Body] |

|  |
| --- |
| **Undtagelser** |
| [Undtagelse 3.a: Rock kunne ikke finde Body]   1. 1. Systemet giver besked om at der ikke er forbindelse til Body   2. Bruger genetablere forbindelse til Body |

### Installér lydpakker

|  |  |
| --- | --- |
| **Navn** | **Installér lydpakker** |
| **Use Case ID** | 2 |
| **Samtidige forekomster** | 1 (inklusiv denne) |
| **Primær aktør** | Bruger |
| **Sekundær aktør** | Ingen |
| **Initiator** | Bruger |
| **Prækondition** | Der er fysisk forbundet en USB-hukommelsesenhed til Rock |
| **Formål** | Der installeres en lydpakke |
| **Postkondition** | En lydpakke er installeret |

|  |
| --- |
| **Hovedscenarie** |
| 1. Bruger vælger ”Installér lydpakke” i Rocks hovedmenu 2. Bruger vælger ”Importer lydpakke” i menu   [Undtagelse 2.a: Bruger vælger ”Slet lydpakke”]   1. System opretter forbindelse til USB-hukommelsesenhed   [Undtagelse 3.a: Der kan ikke oprettes forbindelse]   1. Bruger vælger lydpakke fra liste over kildens lydpakker 2. System installerer lydpakke, og UC2 afsluttes |

|  |
| --- |
| **Undtagelser** |
| [Undtagelse 2.a: Bruger vælger ”Slet lydpakke”]  1. Bruger vælger lydpakke fra liste over systemets lydpakker  2. Bruger bekræfter sletning af lydpakke  3. System sletter lydpakke, og UC2 afsluttes  [Undtagelse 3.a: Der kan ikke oprettes forbindelse]   1. 1. Der vises en fejlmeddelelse, og UC2 afsluttes |

### Konfigurer sensorer

|  |  |
| --- | --- |
| **Navn** | **Konfigurer sensorer** |
| **Use Case ID** | 3 |
| **Samtidige forekomster** | 1 (inklusiv denne) |
| **Primær aktør** | Bruger |
| **Sekundær aktør** | Ingen |
| **Initiator** | Bruger |
| **Prækondition** | Rock er tændt, og mindst én lydpakke er installeret |
| **Formål** | Der oprettes, nedlægges eller ændres i en sensorkonfiguration |
| **Postkondition** | En sensorkonfiguration er oprettet, nedlagt eller ændret |

|  |
| --- |
| **Hovedscenarie** |
| 1. Bruger vælger ”Konfigurer sensor” i Rocks hovedmenu 2. Bruger vælger ”Opret ny” i menuen   [Undtagelse 2.a: Bruger vælger ”Rediger eksisterende”]  [Undtagelse 2.b: Bruger vælger ”Slet”]   1. Bruger indtaster konfigurationens navn   [Undtagelse 3.a: Det indtastede navn er optaget]   1. Bruger vælger sensor på liste over sensorer 2. Bruger vælger MIDI-parameter på liste over MIDI-parametre 3. Bruger vælger mapping scheme på liste over mapping schemes 4. Bruger vælger lydpakke på liste over installerede lydpakker 5. Bruger vælger at tilføje sensor til et eller flere presets på liste over eksisterende presets 6. Bruger præsenteres for sine opsætningsvalg, og UC3 afsluttes |

|  |
| --- |
| **Undtagelser** |
| [Undtagelse 2.a: Bruger vælger ”Rediger eksisterende”]  1. Bruger vælger sensorkonfiguration fra liste over sensorkonfigurationer  2. Der fortsættes fra punkt 4 i hovedscenariet, hvor alle menuer herefter er udfyldt med  data fra den valgte opsætning  [Undtagelse 2.b: Bruger vælger ”Slet”]  1. Bruger vælger sensorkonfiguration fra liste over sensorkonfigurationer  2. Bruger bekræfter valg   1. 3. Sensorkonfiguration slettes, og UC3 afsluttes   [Undtagelse 3.a: Det indtastede navn er optaget]  1. Systemet viser fejlmeddelelse og der fortsættes fra hovedscenariets punkt 3 |

### Konfigurer preset

|  |  |
| --- | --- |
| **Navn** | **Konfigurer preset** |
| **Use Case ID** | 4 |
| **Samtidige forekomster** | 1 (inklusiv denne) |
| **Primær aktør** | Bruger |
| **Sekundær aktør** | Ingen |
| **Initiator** | Bruger |
| **Prækondition** | Der findes mindst én sensorkonfiguration i systemet |
| **Formål** | At oprette, redigere eller slette et preset |
| **Postkondition** | Der er foretaget en ændring i et preset |

|  |
| --- |
| **Hovedscenarie** |
| 1. Bruger vælger ”Konfigurer preset” i Rocks hovedmenu 2. Bruger vælger ”Opret nyt” i menuen   [Undtagelse 2.a: Bruger vælger ”Rediger eksisterende”]  [Undtagelse 2.b: Bruger vælger ”Slet”]   1. Bruger indtaster ønsket navn på preset 2. Bruger markerer de ønskede konfigurationer på liste over sensorkonfiguration, og tilføjer disse til det nuværende preset 3. Bruger trykker på ”Fortsæt” 4. Systemet præsenterer de valgte indstillinger på displayet, og anmoder om bekræftelse 5. Bruger accepterer ændringer, og UC4 afsluttes   [Undtagelse 7.a: Bruger vælger ”Fortryd”] |

|  |
| --- |
| **Undtagelser** |
| [Undtagelse 2.a: Bruger vælger ”Rediger eksisterende”]   * + - * 1. Bruger vælger preset fra liste over eksisterende presets         2. Der fortsættes fra punkt 4 i hovedscenariet, hvor alle menuer herefter er udfyldt med data fra det valgte preset   [Undtagelse 2.b: Bruger vælger ”Slet”]   * + - * 1. Bruger vælger preset fra liste over eksisterende presets         2. Bruger bekræfter valg   [Undtagelse 2.b.2.a: Bruger vælger ”Fortryd”]  [Undtagelse 2.b.2.a: Bruger vælger ”Fortryd”]   * + - * 1. Der fortsættes fra punkt 2.b.1 i Undtagelser   [Undtagelse 7.a: Bruger vælger ”Fortryd”]   * + - * 1. Der fortsættes fra punkt 4 i hovedscenariet |

### Vælg preset

|  |  |
| --- | --- |
| **Navn** | **Vælg preset** |
| **Use Case ID** | 5 |
| **Samtidige forekomster** | 1 (inklusiv denne) |
| **Primær aktør** | Bruger |
| **Sekundær aktør** | Ingen |
| **Initiator** | Bruger |
| **Prækondition** | Mindst ét preset er konfigureret |
| **Formål** | Bruger vælger et preset |
| **Postkondition** | Der er skiftet til et nyt preset |

|  |
| --- |
| **Hovedscenarie** |
| * 1. Bruger vælger preset på Body   2. Body sender et signal til Rock med det valgte preset   [Undtagelse 2.a. Afsendelse mislykkedes]   * 1. Rock opdaterer det aktive sæt af sensorkonfigurationer på baggrund af det valgte preset, og UC5 afsluttes |

|  |
| --- |
| **Undtagelser** |
| [Undtagelse 2.a: Afsendelse mislykkedes]  1. Body afgiver fejlmeddelelse  2. Bruger afvikler UC1 ”Forbind Body og Rock”, og der fortsættes fra punkt 1 i hovedscenariet. |

### Indsamle sensordata

|  |  |
| --- | --- |
| **Navn** | **Indsaml sensordata** |
| **Use Case ID** | 6 |
| **Samtidige forekomster** | 1..\* (inklusiv denne) |
| **Primær aktør** | BodyRock3000 |
| **Sekundær aktør** | Ingen |
| **Initiator** | BodyRock3000 |
| **Prækondition** | Body og Rock er forbundet, og der er forbundet mindst én sensor |
| **Formål** | At sende rådata fra Body til Rock |
| **Postkondition** | Systemet har indsamlet og gemt sensordata i buffer på Rock |

|  |
| --- |
| **Hovedscenarie** |
| 1. Sensor genererer rådata 2. Body sender rådata trådløst til Rock   [Undtagelse 2.a: Afsendelse mislykkedes]   1. Rock indlæser data i buffer 2. Rock er klar til at modtage nye data og UC6 afsluttes |

|  |
| --- |
| **Undtagelser** |
| [Undtagelse 2.a: Afsendelse mislykkedes]   1. 1. Body afgiver fejlmeddelelse |

### Generér MIDI

|  |  |
| --- | --- |
| **Navn** | **Generér MIDI** |
| **Use Case ID** | 7 |
| **Samtidige forekomster** | 1..\* (inklusiv denne) |
| **Primær aktør** | Ingen |
| **Sekundær aktør** | MIDI-Modtager |
| **Initiator** | Ingen |
| **Prækondition** | Mindst ét preset er konfigureret for den aktuelle sensor |
| **Formål** | At konvertere rådata til MIDI-signaler som sendes til hhv. MIDI-afspiller og MIDI-udgang |
| **Postkondition** | MIDI-signaler, baseret på rådata, er sendt til hhv. MIDI-afspiller og MIDI-udgang |

|  |
| --- |
| **Hovedscenarie** |
| 1. Konverteringsmodul læser data fra buffer   [Undtagelse 1.a: Aflæsning af buffer mislykkedes]   1. Konverteringsmodul[[3]](#footnote-3) omdanner rådata til MIDI-meddelelser 2. Signalet fra konverteringsmodul sendes til hhv. MIDI-afspiller og MIDI-udgang, og UC7 afsluttes |

|  |
| --- |
| **Undtagelser** |
| [Undtagelse 1.a: Aflæsning af buffer mislykkedes]   1. 1. Rock afgiver en fejlmeddelelse 2. 2. Der fortsættes fra punkt 1 i hovedscenariet. |

### Afspil lyd

|  |  |
| --- | --- |
| **Navn** | **Afspil lyd** |
| **Use Case ID** | 8 |
| **Samtidige forekomster** | 1 (inklusiv denne) |
| **Primær aktør** | Ingen |
| **Sekundær aktør** | Højtalersystem |
| **Initiator** | Ingen |
| **Prækondition** | Mindst én lydpakke er installeret, mindst ét preset et konfigureret, og MIDI-signal modtages fra konverteringsmodul |
| **Formål** | At generere lyd via BodyRock3000 |
| **Postkondition** | Et lydsignal, der repræsenterer valgt preset, lydpakke og MIDI-signal, er genereret og sendt til højtalersystem |

|  |
| --- |
| **Hovedscenarie** |
| 1. Lydmodul aflæser valgt lydpakke   [Undtagelse 1.a: Aflæsning af lydpakke mislykkedes]   1. Lydmodulet aflæser valgte preset   [Undtagelse 2.a: Aflæsning af preset mislykkedes]   1. Lydmodulet modtager MIDI-signaler   [Undtagelse 3.a: Modtagelse af MIDI-signaler mislykkedes]   1. Lydfil der skal manipuleres vælges ud fra MIDI-signal 2. Valgt lydfil manipuleres i overensstemmelse med indkommende MIDI-signal (LFO, pitch, volumen, tænd, sluk, eller lignende) 3. Lydsignal udsendes analogt på en port til højtalersystem, via en DAC, og UC8 afsluttes |

|  |
| --- |
| **Undtagelser** |
| [Undtagelse 1.a: Aflæsning af lydpakke mislykkedes]   1. 1. Rock afgiver fejlmeddelelse   2. Bruger trykker på ”Forsøg igen”, og der fortsættes fra punkt 1 i hovedscenariet.  [Undtagelse 1.a.2.a: Bruger trykker på ”Afbryd”]  1. Der returneres til Hovedmenu, og UC8 afsluttes  [Undtagelse 1.a.2.a: Bruger trykker på ”Afbryd”]  1. Der returneres til Hovedmenu, og UC8 afsluttes  [Undtagelse 2.a: Aflæsning af preset mislykkedes]   1. 1. Rock afgiver fejlmeddelelse   2. Bruger trykker på ”Forsøg igen”, og der fortsættes fra punkt 2 i hovedscenariet.  [Undtagelse 2.a.2.a: Bruger trykker på ”Afbryd”]  1. Der returneres til Hovedmenu, og UC8 afsluttes  [Undtagelse 2.a.2.a: Bruger trykker på ”Afbryd”]  1. Der returneres til Hovedmenu, og UC8 afsluttes  [Undtagelse 3.a: Modtagelse af MIDI-signal mislykkedes]  1. Der fortsættes fra punkt 3 i hovedscenariet |

## Krav til Mapping Scheme

Følgende features skal kunne indstilles for et givent Mapping Scheme

**Note:**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Skala** | Kromatisk | | | Dur | | | | Mol | | | |
| **Grundtone** | c | c# | d | d# | e | f | f# | g | g# | a | a# |
| **Retning** | Opadgående | | | | | Nedadgående | | | | | |

**Velocity:**

|  |  |
| --- | --- |
| **Lower Threshold** | 0-127 |

**Control Change:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | Absolute | Relative | |
| **Channel num** | 0-127 | | |
| **Min value (for absoulte)** | 0-127 | | |
| **Max value (for absoulte)** | 0-127 | | |
| **Følsomhed (for relative)** | Slow | Medium | Fast |

**Midi channel:**

|  |  |  |
| --- | --- | --- |
| **Type** | Omni | Specific channel |
| **Channel no (for specific)** | 1-16 | |

## Ikke funktionelle krav

### Generelt

* + Den trådløse forbindelse mellem Body og Rock skal have en rækkevidde på minimum 10 meter

### Body

* + Skal ved aktiv brug kunne holde strøm i min. 2 timer
  + Skal forsynes af et 9V batteri
  + Skal have en power-indikator i form af en diode, der lyser grønt når enheden er tændt
  + Skal have en fejl-indikator i form af en diode, der lyser rødt ved fejl
  + Skal kunne styre op til 16 sensorer på én gang

### Rock

* + Skal have en strømforsyning, der tilsluttes et 220V 50 Hz el-net
  + Skal have en power indikator i form af en diode, der lyser grønt når enheden er tændt
  + Skal have en fejlindikator i form af en diode, der lyser rødt ved fejl
  + Skal have en brugergrænsefladen der er en GUI baseret på QT
  + Skal have et MIDI-output af typen ”5-pin DIN hun”
  + Skal have et lydoutput af typen ”stereo mini-jack hun”

# Systemarkitektur Hardware

## Overordnet arkitektur

Dette afsnit beskriver systemarkitekturen for ”BodyRock3000”. Yderligere beskrivelse kan ses i projektformuleringen og kravspecifikationen.

Formålet er at identificere:

* De overordnede komponenter, samt at bestemme deres grænseflader
* De eksterne komponenter, samt at beskrive dem
* Arbejdsopgaver for projektets design- og implementeringsfase

### Domæne model BodyRock3000



På figuren herover ses en overordnet domæne model med konteksten for systemet ”BodyRock3000”.

Bemærk at Body og Rock har hver sit bluetooth-modul, der benyttes som interface til deres indbyrdes kommunikation.

## Body HW Arkitektur

### BDD Body



**Body** består af Controller, Spændingsforsyning, Bluetooth-modul og Preset-knapper.

**Controller** er hjernen i Body. Denne står for indsamling, pakning og videresendelse af data fra sensorBus til Bluetooth-modul. Den afkoder og videresender desuden information fra Preset-knapper.

**Spændingsforsyning** leverer spænding til Body.

**Bluetooth-modul** videresender data fra Controller til Rocks Bluetooth-modul.

**Preset-knapper** bruges af brugeren til at vælge preset.

### Allokeringsdiagram Body



**Controller** implementeres på en **PSoC4**

**Spændingsforsyningen** implementeres som hhv. en **Reguleringskreds** og et **Batteri**

**Bluetooth-modul** implementeres med komponenten **HC-05**

**Preset-knapper** implementeres med en **3x4 knapmatrix**

### IBD Body

Herunder ses et internt block-diagram for Body. Dette illustrerer interne forbindelser i Body:



**Batteri** genererer strømmen til systemet.

**Reguleringskreds** bruges til at transformere den indgående batterispænding til 3,3V.

**PSoC4** er systemets kontrol-enhed. Den benyttes til at indsamle, bearbejde og pakke data fra sensorbus og 3x4 knapmatrix, samt sender disse til bluetooth-modulet HC-05. Den kontrollerer desuden StatusLEDer.

**StatusLEDer** består af en grøn og en rød LED, der indikerer hhv. power status (hvorvidt enheden er tændt) og error status (hvorvidt der opstår fejl på enheden).

**3x4 knapmatrix** benyttes af brugeren til at skifte mellem forskellige systempresets.

**HC-05** forbinder Body og Rock trådløst via bluetooth.

**Sensorbus** er en I²C bus, der består af hhv. GND, VDD, SDA og SCL forbindelserne.[[4]](#footnote-4)

## Sensor IBD (Generisk)

Alle sensor-enheder kobles på sensorbus som følger:



Sensorenhederne er alle koblet parallelt op med en strømforsyning (GND og VCC), samt I²C forbindelserne SCL (serial clock) og SDA (serial data). SDA-forbindelsen er non-atomic, da både master (Body) og slave (den pågældende sensorenhed) kan tage styring over linjen. Sensorenhedernes interfaces er lavet således, at de kan kobles i forlængelse af hinanden.

## Rock HW Arkitektur

### BDD Rock



**Rock** består af Functionalities, Bluetooth-modul og Touchscreen.

**Functionalities** er en abstraktion for alle Rocks logiske funktionaliteter.

**Bluetooth-modul** modtager data fra Bodys Bluetooth-modul.

**Touchscreen** benyttes af bruger til at operere Rocks GUI og ændre indstillinger for Rock.

### Allokeringsdiagram Rock



**Functionalities** implementeres på et **Raspberry Pi** samt en **ESI MIDI Mate** til output af MIDI.

Som **Touchscreen** benyttes en **Adafruit PiTFT.**

**Bluetooth-modul** implementeres med komponenten **RN42**

## Grænsefladebeskrivelse

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Navn** | **Beskrivelse** | **Signaler** | **Område** | **Kommentar** |
| sensorbus | I²C [[5]](#footnote-5) bus forbundet med RJ11-stik | GND | 0 V |  |
| VCC | 3,3 V |  |
| SCL | 0-3,3 V | I²C seriel clock signal.  Følger 3,3V CMOS standard |
| SDA | 0-3,3 V | I²C seriel data signal Følger 3,3V CMOS standard |
| BluetoothCon | Forbindelse fra Controller til RN42-modul | GND | 0 V |  |
| VCC | 3,3 V |  |
| RXD | 0-3,3 V | RS232 seriel recieve data signal Følger 3,3V CMOS standard |
| TXD | 0-3,3 V | RS232 seriel transmit data signal Følger 3,3V CMOS standard |
| Reconnect | 0-3,3 V | 0V =deaktiveret, 3,3 V aktiveret. Ved aktiveret genetablerer RN-42 modulet bluetoothforbindelsen mellem Body og Rock[[6]](#footnote-6) |
| lineOut | Ubalanceret signal med 3.5 mm (1/8") 3-conductor TRS minijack stik. | Left channel (tip) | -0.447-0.447 V | Consumer audio. (Wikipedia, u.d.)  Nominal level: -10 dBV |
| Right channel (ring) | -0.447-0.447 V |
| Ground (sleeve) | 0V |  |

**MIDI out / ekstern MIDI**

MIDI beskeder, der overholder MIDI protokollen som defineret af Midi Manufacturers Association (MIDI Manufacturers Association, u.d.), sendes via ALSA (Alsa-Project, u.d.) til et ESI MIDI Mate USB til MIDI lydkort. (ESI, u.d.) Denne har både et input-kabel og et output-kabel. Det er kun output-kablet der benyttes. Det anvender følgende pin layout:

|  |  |
| --- | --- |
| Pin | Signal |
| 1 | Ikke anvendt |
| 2 | Ground |
| 3 | Ikke anvendt |
| 4 | Nuværende Sync |
| 5 | Nuværende Kilde |



### Trådløs kommunikations protokol

Den trådløse kommunikationsprotokol er til for at Rock kan tolke hvilke sensordata og presets der modtages fra Body. Fra Body sendes der to typer char-arrays.

Type 1:   
Et sensordata-array der indeholder seks pladser. Der er oprettet ét sensordata-array pr. tilsluttet sensor.

Type 2:  
Et preset-array der indeholder tre pladser.

Sensordata array:

|  |  |  |
| --- | --- | --- |
| Plads: | Indhold: | Beskrivelsen: |
| 0 | 0x0F | Definition på sensordata forsendelsen |
| 1 | 0-15 | Sensor ID |
| 2 | 0-127 | Sensorkoordinat x værdi |
| 3 | 0-127 | Sensorkoordinat y værdi |
| 4 | 0-127 | Sensorkoordinat z værdi |
| 5 | 0 | Nul-terminering |

Preset array:

|  |  |  |
| --- | --- | --- |
| Plads: | Indhold: | Beskrivelsen: |
| 0 | 0xF0 | Definition på preset-data forsendelsen |
| 1 | 1-9 | Preset værdi |
| 2 | 0 | Nul-terminering |

Forsendelsen af sensordata arrayene skal ske 50 gange i sekundet[[7]](#footnote-7). Presetdata forsendelsen skal kun ske når der er ændret preset ved at trykke på knap-matrix’en.

**Oversigt over ID på Body samt sammenhæng mellem type, navn og parametre.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SensorID | Definition | Sensor navn | Type | Parametre |
| 0 | ACC1\_ID | ADXL345 | Accelerometer | (X,Y,Z) |
| 1 | GYR1\_ID | MPU6050 | Gyroskop | (X,Y,Z) |
| 2 | FLE1\_ID | FLEXIFORCE A301 | Tryksensor | (X) |
| 3 | PRX1\_ID | HC-SR04 | Proximitysensor | (X) |
| 4 | ACC2\_ID | ADXL345 | Accelerometer | (X,Y,Z) |
| 5 | GYR2\_ID | MPU6050 | Gyroskop | (X,Y,Z) |
| 6 | FLE2\_ID | FLEXIFORCE A301 | Tryksensor | (X) |
| 7 | PRX2\_ID | HC-SR04 | Proximitysensor | (X) |
| 8 | ACC3\_ID | ADXL345 | Accelerometer | (X,Y,Z) |
| 9 | GYR3\_ID | MPU6050 | Gyroskop | (X,Y,Z) |
| 10 | FLE3\_ID | FLEXIFORCE A301 | Tryksensor | (X) |
| 11 | PRX3\_ID | HC-SR04 | Proximitysensor | (X) |
| 12 | ACC4\_ID | ADXL345 | Accelerometer | (X,Y,Z) |
| 13 | GYR4\_ID | MPU6050 | Gyroskop | (X,Y,Z) |
| 14 | FLE4\_ID | FLEXIFORCE A301 | Tryksensor | (X) |
| 15 | PRX4\_ID | HC-SR04 | Proximitysensor | (X) |

# Systemarkitektur Software

I følgende afsnit beskrives softwarearkitekturen for systemet BodyRock3000. Softwarearkitekturen er opbygget på baggrund af projektformuleringen og kravspecifikationen.

Formålet er at skabe en ramme for softwaren-udviklingen, hvorfra delopgaver kan uddelegeres til udviklere.

Softwarearkitekturen består af:

* Identifikation af problemer, klasser og metoder med udgangspunkt i en detaljeret domænemodel for system og dets software-moduler
* Oprettelse af skelet for videre implementering af Bodys SW vha. et simpelt sekvensdiagram for program-flow
* Oprettelse af skelet for videre implementering af Rocks SW vha. applikationsmodeller for systemets moduler

Det er valgt at lave sidstnævnte applikationsmodeller for modulerne/modulgrupperne internt frem for direkte Use Case baseret. Applikationsmodellerne indeholder således funktionaliteter for op til flere Use Cases, men kun afgrænset til det/den pågældende modul(-gruppe). Denne tilgang leder op til en uddelegering af design og implementering af systemet til mindre teams, der står for hver deres modul.

## Detaljeret domænemodel for software-moduler

Herunder ses en domænemodel for kommunikationen mellem systemets interne SW-moduler.



## Klasse-identifikation

På baggrund af domænemodellen for softwaremoduler, identificeres følgende klasser for hhv. Body og Rock:

**Body** generer data ud fra brugerens påvirkning af sensorer og tryk på knapmatrix. Disse data pakkes i Bodys Serial Unit og videresendes via Bluetooth-modul.

* **Main** er Bodys central for indsamling af sensordata og håndtering af knap-tryk
* **SerialUnit** står for at pakke de, fra Body, genererede data og adaptere dem til Bluetooth-moduls interface
* **Sensor** står for at aflæse alle tilkoblede sensorer, standardisere sensordata, samt sende dem til SerialUnit
* **Knap** er 4x3 knapmatrix afkoderen. Den meddeler Main at der er ønsket nyt preset
* **Bluetooth-modul** er det fysiske Bluetooth-modul HC-05, der muliggør afsendelse af datapakker fra Body til Rock

**Rock** afkoder data fra Body og generer MIDI og lyd på baggrund af de indstillede værdier i dataStorage. Sidstnævnte indstilles fra GUI.

* **GUI** er systemets bruger-interface, der benyttes til bruger-indstillinger for systemet
* **Controller** er Rocks styrings-hub. Den konfigurerer dataStorage på baggrund af brugerens interaktion med GUI, og konfigurerer Receiver (dataAdapter), MIDI-module (MIDI-mapper) og Sound-module (Linuxsampler) ved modtagelse af besked om preset-skift fra Receiver
* **DataStorage** opbevarer data på systemet. Denne består af Preset-bank, Sensorkonfiguration og Sample Bibliotek
  + **Preset-bank** indeholder systemets samlede presetindstillinger. Den består af op til 9 presets, der hver kan tilknyttes op til 16 Sensorkonfigurationer
  + **SensorConfiguration** indeholder systemets brugerindstillinger for hver sensor-enhed, der kan tilkobles
    - **MappingScheme** indeholder policy for, hvordan der, for den givne sensorkonfiguration, genereres MIDI
  + **Samplebibliotek** indeholder de lyd-samples, som Linuxsampler kan afspille
* **Receiver** står for modtagelse og udpakning af data fra Body via Bluetooth, bestemmelse af datatype, pakning af data og videre forsendelse i systemet. Receiver består af følgende sub-moduler:
  + **Bluetooth-modul** er det fysiske Bluetooth-modul RN-42, der modtagelse afsendelse af datapakker fra Body til Rock
  + **dataAdapter** står for genkendelse af typen for de udpakkede data (hhv. sensor-data eller preset-valg) samt adaptering og afsendelse af disse til det relevante modul i systemet. Adaptering kan i fremtidige systemudgaver indeholde implementering af velocity-kurve for sensorData. Presetvalg og systeminfo videresendes til Controller, og sensordata videresendes til MIDI-module
* **MidiModule** står for at klargøre MIDI-signaler til videre afsendelse på baggrund af modtagne sensordata og nuværende preset. Disse videresendes til software-frameworket ALSA, der står for videre processering af data. MidiModule består af følgende sub-moduler:
  + **MIDI Mapper** holder den funktionalitet, der lader systemet konvertere sensordata til MIDI-signaler. Jf. det nuværende aktive preset, består denne af en liste af SensorConfiguration, der hver har et tilknyttet sensorID til aflæsning af sensordata og et tilknyttet plads i MIDI Signal Bufferen, der mappes til
  + **MIDI Signal Buffer** indeholder de MIDI signaler, der skal videresendes til ALSA
  + **ALSA Adapter** står for at interface til ALSA, så værdierne fra MIDI Signal Buffer aflæses korrekt
* **ALSA-functions** er de i systemet benyttede funktionaliteter fra ALSA. ALSAer et software-framework, der leverer et API for systemets lydkort drivere, og har i dette tilfælde til opgave at generere MIDI-output
  + **Linux Sampler** afspiller lyd-samples fra Sample Bibliotek på baggrund af modtaget MIDI Signal og givne indstillinger i Sensorkonfiguration
  + **MIDI-generator** er driveren, der genererer den MIDI på Rocks MIDI output

## ALSA

Følgende afsnit redegør for de elementer af Advanced Linux Sound Architecture, forkortet ALSA, der er anvendt i projektet.

ALSA er et software-framework, der bl.a. håndterer konfiguration af device drivers til lydkort, og routing af lyd i Linux, både internt og til eksterne enheder. Frameworket er en del af Linuxkernen og stilles således til rådighed som standard af langt de fleste distributioner. I systemet anvendes ALSA til at modtage genererede MIDI-beskeder og sende disse videre til enten LinuxSampler, et ekstern lydkort eller begge.

### Anvendte ALSA-elementer

I nedenstående præsenteres de anvendte dele af ALSA-frameworket.

### RawMidi

De generede MIDI-data sendes som rå MIDI-data. Derunder forstået, at det sendes på byteform[[8]](#footnote-8). Derfor anvendes ALSA’s RawMidi interface. Dette interface er indbygget i ALSA og indeholder metoder, der muliggør afsendelse og modtagelse af rå MIDI-data mellem applikationer og lydkort via ALSA-porte.

Disse porte specificeres vha. ASCII-identifiers på formen:

hw: CARD, DEVICE, SUBDEVICE

CARD refererer til navnet eller nummeret på det lydkort, hvor porten befinder sig. Dette kort kan enten være fysiske lydkort eller interne virtuelle lydkort. Da et lydkort kan indeholde flere porte, anvendes DEVICE til at referere til et specifikt device på lydkortet. SUBDEVICE specificerer på samme måde et device under DEVICE.

### Virmidi

Snd\_virmidi er et kernemodul, der opretter et virtuelt MIDI-lydkort. Dette anvendes som bindeled mellem AlsaAdapter og LinuxSampler.

### Aconnect

Command line værktøjet aconnect er en del af ALSA-utils, der gør det muligt at forbinde en ALSA-port med en sequencer. Dette anvendes i projektet til at forbinde Virmidi lydkortet til LinuxSampleren.

## Body

Body Control er styresystemet på Body-delen af BodyRock3000. Systemet står for at samle sensordata fra de fire[[9]](#footnote-9) forskellige sensorer via I²C, konverterer disse data og videresende dem vha. en UART forbundet til et Bluetooth modul.

### Klassediagram Body



Figur 1 UML af Body Control

#### Main

Main er Bodys control klasse. Main står for at initerer I²C, UART og de tilkoblede sensorer samt at kalde funktioner der kan læse sensorer, standardisere sensordata og sende sensor- og presetdata via SerialUnit. På nuværende tidspunkt skal initieringsfunktionerne til sensorene stå i Main klassen for at funktionerne kan implementeres.

#### Sensor

Sensor står for at klargøre I²C, læse sensordata via I²C, gemme sensordata i en midlertidig buffer, konvertere sensordata til et generisk interval fra 0-127. Sensor sender også de generiske sensordata til SerialUnit. Sensor indeholder også I²C fejlhåndtering.

#### SerialUnit

Står for at klargøre sensordata til afsendelse via UART og videresende sensordata’en over uart.

#### Knap

Ikke implementeret da denne klasse omhandler preset.

#### Funktioner

|  |  |
| --- | --- |
| Funktioner \* | Beskrivelse |
| initSensors() | Kalder de forskellige initieringsfunktioner for hver sensor. |
| initADXL345() | Initialiserer accelerometeret. |
| initMPU6050() | Initialiserer gyroskopet. |
| initUART() | Starter UART komponenten. |
| setupI²C() | Starter I²C komponenten. Returnerer master status og clearer status flaget. |
| readAllSensors() | Kalder funktionen readI²C for hvert sensorregister der er, at læse fra. |
| readI²C(int numOfReg) | Modtager det register nummer der skal læses fra og følger herefter I²C protokollen for læsning og gemmer det læste data. |
| convSensData() | Samler sensoraksernes most-significant og least-significant byte til en.  Konverterer herefter til MIDI-skalaen 0-127.  Kalder funktionen setdataArray(int, int, int, int). |
| setdataArray(int, int, int, int) | Fylder array med sensor ID og data fra convSensData() |
| sendDataArray() | Sender et array med data for hver sensor. |
| handleI²CError() | Funktion som kaldes ved fejl i I²C forbindelsen. Genererer en stop condition. |
| setSensArray(int, int, int) | Fylder det multidimensionelle array. |

\*Alle preset funktioner er ikke skrevet da man har valgt ikke at implementere preset I den iteration.

#### Uddybende funktionsbeskrivelser

##### void initSensors(void)

|  |  |
| --- | --- |
| **Beskrivelse** | Kalder de initieringsfunktioner til de specifikke sensorer som er sat op. |
| **Parametre** | Ingen |
| **Returværdi** | Ingen |
| **Påvirkninger** | Ingen |

##### void initADXL345(void)

|  |  |
| --- | --- |
| **Beskrivelse** | Sætter de første 6 pladser i sensArray, plads [numOfSens][0] med ADXL345 ID, plads [numOfSens][1] med registeret som der skal læses fra og plads [numOfSens][2] med en default data værdi -1.  Opsætter sensoren til I²C jf. I²C-protokollen. |
| **Parametre** | Ingen |
| **Returværdi** | Ingen |
| **Påvirkninger** | Ingen |

##### void initMPU6050(void)

|  |  |
| --- | --- |
| **Beskrivelse** | Sætter de første 6 pladser i sensArray, plads [numOfSens][0] med ADXL345 ID, plads [numOfSens][1] med registeret som der skal læses fra og plads [numOfSens][2] med en default data værdi -1.  Opsætter sensoren til I²C jf. I²C-protokollen. |
| **Parametre** | Ingen |
| **Returværdi** | Ingen |
| **Påvirkninger** | Ingen |

##### void initUART(void)

|  |  |
| --- | --- |
| **Beskrivelse** | Starter SCB\_init() og SCB\_Enable(). Efter denne funktion er kaldt er UART komponenten enabled og klar til at operere.  Når konfigurationen er sat til ”Unconfigured SCB”, skal komponenten først initialiseres til at operere i en af følgende konfigurationer: I²C, SPI, UART eller EZ I²C. Ellers kan denne funktion ikke enable komponenten  Returnerer master status og clearer status flaget. |
| **Parametre** | Ingen |
| **Returværdi** | Ingen |
| **Påvirkninger** | Ingen |

##### void setupI²C(void)

|  |  |
| --- | --- |
| **Beskrivelse** | Starter SCB\_init() og SCB\_Enable(). Efter denne funktion er kaldt er UART komponenten enabled og klar til at operere.  Når konfigurationen er sat til ”Unconfigured SCB”, skal komponenten først initialiseres til at operere i en af følgende konfigurationer: I²C, SPI, UART eller EZ I²C. Ellers kan denne funktion ikke enable komponenten |
| **Parametre** | Ingen |
| **Returværdi** | Ingen |
| **Påvirkninger** | Ingen |

##### void readAllSensors(void)

|  |  |
| --- | --- |
| **Beskrivelse** | Kalder funktionen readI²C for hvert sensorregister der er, at læse fra. |
| **Parametre** | Ingen |
| **Returværdi** | Ingen |
| **Påvirkninger** | Ingen |
|  |  |

##### void readI²C(int)

|  |  |
| --- | --- |
| **Beskrivelse** | Følger I²C protokollen for at læse fra et register. Protokollen er generisk ved at bruge pladserne i det multidimensionelle array, sensArray.  Eks. Sendes der en start til sensArray[numOfReq][0], som er adressen på sensoren.  For yderligere beskrivelse se I²C protokol.[[10]](#footnote-10)  Dataen som læses gemmes i sensArray[numOfReq][2]. |
| **Parametre** | En int indeholdende den række der ønskes læst fra i sensArray. |
| **Returværdi** | Ingen |
| **Påvirkninger** | Ingen |

##### void convSensData(void)

|  |  |
| --- | --- |
| **Beskrivelse** | Tager data fra sensArray, og samler most significant og least significant for de givne akser: x, y og z.  Konverterer herefter til MIDI-skalaen 0-127.  Kalder funktionen setdataArray(int, int, int, int). |
| **Parametre** | Ingen |
| **Returværdi** | Ingen |
| **Påvirkninger** | Ingen |

##### void setdataArray(int, int, int, int)

|  |  |
| --- | --- |
| **Beskrivelse** | Fylder dataArray plads 0 med sensor ID fra 0-15 og de næste 3 pladser med data fra x, y og z aksen fra sensoren. De resterende pladser i dataArray sættes til 1 |
| **Parametre** | Fire int: ID, x\_data, y\_data, z\_data |
| **Returværdi** | Ingen |
| **Påvirkninger** | Ingen |

##### void sendDataArray(void)

|  |  |
| --- | --- |
| **Beskrivelse** | Opretter et array for hver sensor. Funktionen fylder array plads 0 med START\_SENSDATA, de næste 4 pladser sættes med værdierne fra dataArray i samme rækkefølge som de er sat i setdataArray(). Den sidste plads i arrayet sættes til NUL. Herefter sendes array over BT vha. UART\_UartPutString() |
| **Parametre** | ingen |
| **Returværdi** | Ingen |
| **Påvirkninger** | Ingen |

##### void handleI²CError(void)

|  |  |
| --- | --- |
| **Beskrivelse** | Sender en stopkondition |
| **Parametre** | ingen |
| **Returværdi** | Ingen |
| **Påvirkninger** | Ingen |

##### void setSensArray(int, int, int)

|  |  |
| --- | --- |
| **Beskrivelse** | Fylder det multidimensionelle array, sensArray på følgende vis:  Den første int fyldes i plads [numOfReqToRead[0]. Den næste int fyldes i plads [numOfRegToRead[1] og den sidste int i plads [numOfReqToRead[2].  Den globale variabel numOfRegToRead inkrementeres hver gang funktionen kaldes. |
| **Parametre** | Tre ints: device adressen, registeret som ønskes læst fra og data. |
| **Returværdi** | Ingen |
| **Påvirkninger** | Ingen |

## Rock

Softwaren på Rock implementeres event-baseret og indeholder således en række selvstændige tråde. De vil være opdelt i to hovedkategorier, bestemt ud fra hvor kritisk afviklingshastigheden er for den musikalske oplevelse ved brug af systemet. Disse er benævnt som følger:

* **Slow Lane**
* **Fast Lane.**

Visse software-moduler vil gå igen for begge baner, men hastigheden af disse (prioriteten i systemets scheduler), vil afhænge af hvilken tråd, de indgår i.

Der vil i følgende afsnit være udfærdiget applikationsmodeller for de individuelle moduler. Hver af disse moduler har sin egen tråd, og applikationsmodellerne rummer dermed de enkelte trådes funktionalitet.

### Tråd kommunikation

I dette afsnit skitseres overordnet hvordan kommunikation trådene imellem vil foregå.

### Slow Lane

Denne bane håndterer de software-moduler i systemet, hvis afvikling ikke er kritisk for den musikalske oplevelse ved benyttelse af BodyRock3000.

#### SW-moduler i Slow Lane

Bruger-indstillinger af systemet (UC 2-4) tildeles en lav prioritet i systemets scheduler, da disse kun anvendes sjældent, og ikke er latens-kritiske. Følgende softwaremoduler tager sig af disse indstillinger:

* GUI
* DataStorage
* Controller

Controller og DataStorage vil blive brugt i hhv. Slow- og Fast Lane afhængigt af hvilken sammenhæng de benyttes i.

#### Applikationsmodel for Slow Lane

*Slow Lane* benyttes i systemet som betegnelse for trådene ”GUI” og ”Controller”. Denne del af systemet står for at modtage input fra brugeren, samt for at administrere *Databank* i henhold til disse.

Herudover administrerer Controlleren ligeledes trådene i *Fast Lane*, hvilket bl.a. indebærer at modtage presets fra *Receiver*, samt at sende opdaterede data fra *Databank* til *MidiModule* og *ALSA-Functions*.

##### Sekvensdiagram

##### 

I dette diagram ses en beskrivelse af kommunikation mellem GUI og Controller tråden når brugeren skal oprette en ny sensorkonfiguration.  
Denne kan betragtes som repræsentativ for den gennerelle kommunikation mellem de to tråde.

GUI delen i diagrammet er en samlet betegnelse for diverse klasser i vores User Interface. Disse klasser kommunikere alle sammen igennem MsgHandler og ned til Controller klassen.

### Fast Lane

Denne bane håndterer de software-moduler i systemet, hvis afvikling er kritisk for den musikalske oplevelse ved benyttelse af BodyRock3000.

#### SW-moduler i Fast Lane

Benyttelse af systemet som instrument (UC 5-8) tildeles en høj prioritet i systemets scheduler, da disse er latens-kritiske. Følgende softwaremoduler benyttes ved brug af Rocks MIDI- og lydgenerations funktion:

* Receiver
* MidiModule
* ALSA-functions
* (Controller)
* (DataStorage)

Controller og DataStorage indgår i UC 5, ”Vælg preset”, der er en afstikker-funktionalitet i Fast Lane, idet den kan anses som en (re-)initieringsfunktion for banen Fast Lane, hvis hovedfunktion er at omdanne sensor data til hhv. MIDI-signaler og lyd.

#### Receiver

##### Sekvensdiagram



**Figur 2** Sekvensdiagram over operationerne udført af klassen Receiver

##### Klassediagram



**Figur 3** Klassediagram over klassen Receive

##### Funktionsbeskrivelser

|  |  |
| --- | --- |
| **Funktion** | Constructor |
| **Parametre** | Pointer til køen fra klassen *MIDI Module* |
| **Returværdi** | Ingen |
| **Beskrivelse** | Kalder funktionen *connect* |

|  |  |
| --- | --- |
| **Funktion** | Destructor |
| **Parametre** | Ingen |
| **Returværdi** | Ingen |
| **Beskrivelse** | Afbryder den eventuelt kørende tråd, og kalder funktionen *disconnect* |

|  |  |
| --- | --- |
| **Funktion** | Start |
| **Parametre** | Ingen |
| **Returværdi** | Ingen |
| **Beskrivelse** | Starter en tråd som kalder funktionen *receive* i en uendelig løkke |

|  |  |
| --- | --- |
| **Funktion** | Receive |
| **Parametre** | Ingen |
| **Returværdi** | Ingen |
| **Beskrivelse** | Aflæser og tømmer Rx-bufferen fra UART-noden *ttyAMA0*, som indeholder data modtaget fra Body over Bluetooth. Reagerer herefter på den først modtagne byte:  Startbyte = 0x0F: Modtaget sensordata pakkes. Registreres det at en ny omgang sensordata er påbegyndt, sendes den sidste pakker videre til *MIDI Module*.  Startbyte = 0xF0: Modtaget preset videresendes til *Controller*.  Ellers: Modtaget data ignoreres. |

|  |  |
| --- | --- |
| **Funktion** | Connect |
| **Parametre** | Ingen |
| **Returværdi** | Ingen |
| **Beskrivelse** | Opretter *read only*, blocking forbindelse til noden *ttyAMA0*, som håndterer UART-forbindelsen på de benyttede ben på Rock. |

|  |  |
| --- | --- |
| **Funktion** | Disconnect |
| **Parametre** | Ingen |
| **Returværdi** | Ingen |
| **Beskrivelse** | Afbryder forbindelsen til og frigiver noden *ttyAMA0*. |

#### Applikationsmodel for MidiModule

I dette afsnit findes en applikationsmode for MidiModule tråden.

##### Klasseidentifikation

MidiModulet opdeles i underklasser for at vise systemets interne virkemåde.

**Boundary klasser:**

* ***MsgQueue***: Inderholder messages fra hhv. Receiver (data) og Controller (preset)
* ***AlsaAdapter***: Interface til ALSA-funktionerne

**Domæne klasser:**

* ***Timer***: Systemet interne clock, der holder takten for omdannelse af data til MidiSignaler.
* ***SensorConfig***: Et objekt af klassen SensorConfiguration i det aktive preset, der indeholder konfiguration for hvilket datapunkt et specifikt MidiSignal skal generers ud fra.
* ***MappingScheme:*** Et objekt af klassen MappingScheme i et specifikt SensorConfig objekt. Denne indeholder konfiguration- of funktionaliteten for hvordan et bestemt MidiSignal genereres.

**Controller klasse:**

* ***MidiModule:*** Det overordnede objekt, de indeholderden samlede funktionalitet for hvordan en mængde af datapunkter omdannes til en række MidiSignaler jf. det nuværende aktive preset.

##### Sekvens diagram

Herunder ses et sekvensdiagram for programflowet internt i MidiModule. Ved hvert taktslag, tjekkes MsgQueue. Hernæst håndteres den modtagne message jf. dens type.

For data messages loopes gennem de aktive sensorkonfigurationer, hvor SensorConfig tilgår et specifikt datapunkt, og MappingScheme dernæst opdaterer det givne SensorConfig objekts MidiSignal.

For preset messages nedlægges først de aktive sensorkonfigurationer, hvorefter et preset svarende til den modtagne message findes og initieres.



##### State Machine Diagram

MidiModule har to hovedstates: ”Idle” og ”Handling”. Handling igangsættes ved hvert taktslag og igangsættes med substate ”Handling Queue”, hvori msgQueue tømmes. Afhængigt af beskedens type, igangsættes så enten ”Handling Data” eller ”Handling Preset”. Når msgQueue er håndteret, går MidiModule i state ”Idle”.



##### Klassediagram

Herunder er angivet et UML klassediagram for MidiModule og dens bestanddele.



### Overordnet Klassediagram Rock

# Design og implementering (HW)

## Trådløs kommunikation

### Rock - trådløs kommunikation

#### Montering af modul RN-42(Pmod BT2 module)

RN-42 kræver en forsyningsspænding på 3.3V og trækker 30mA når den transmitterer, hvilket giver en samlet effekt på:

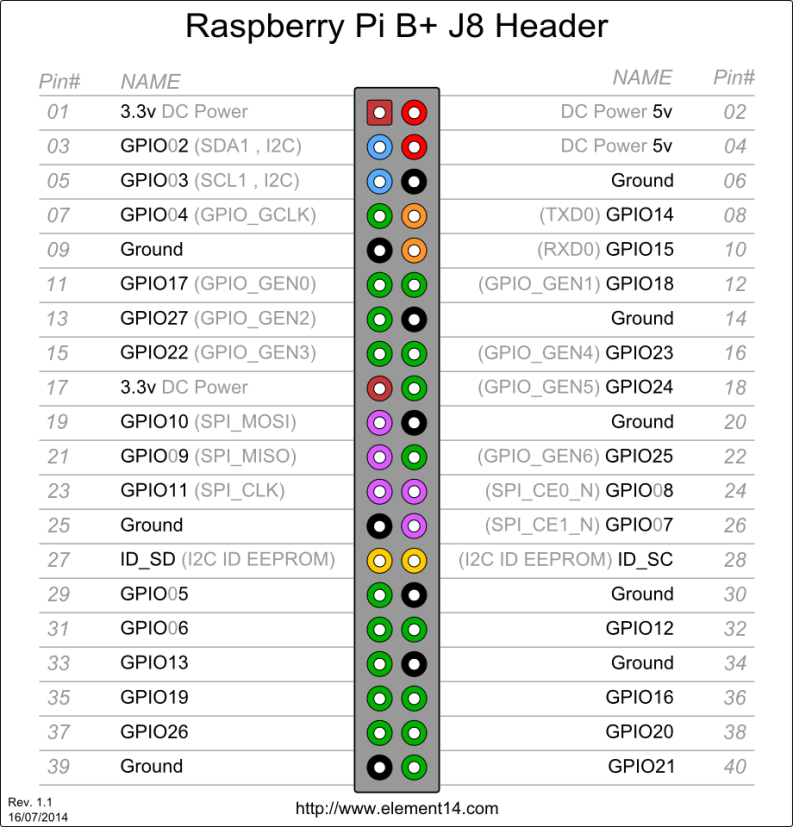


Dette lægger inden for en Raspberry Pi B+s tolerance. Raspberry Pi B+’ samlede effekt kapacitet:



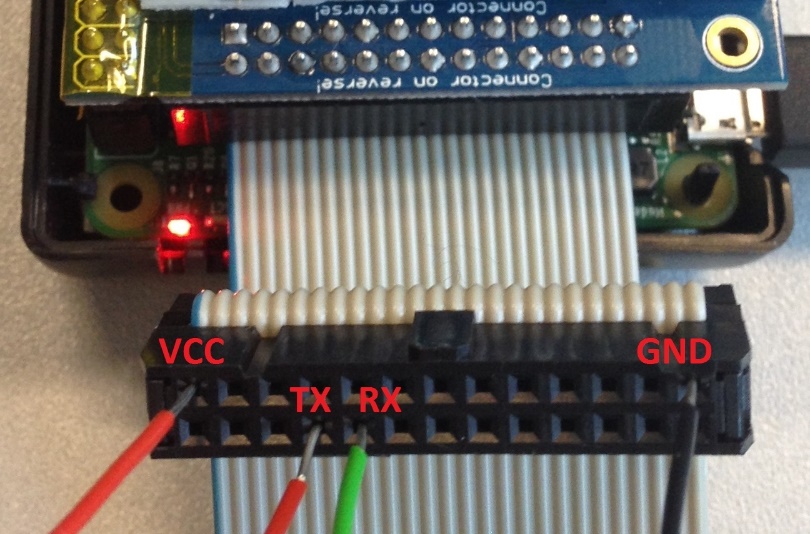
100mW ud af 6 W:



Modulet påmonteres derfor direkte på minicomputeren på pin#:

|  |  |
| --- | --- |
| RN-42 Pins  (REFERENCE) | Raspberry Pi B+ Pins (REFERENCE)[[11]](#footnote-11) |
| Pin 06 VCC | Pin 01 3.3v DC Power |
| Pin 05 GND | Pin 06 GND |
| Pin 02 RXD | Pin 08 GPIO14(TXD0) |
| Pin 03 TXD | Pin 10 GPIO15(RXD0) |
| JP3 fungerer som reconnect |  |

For at omgå den påmonterede Adafruit Pi TFT-skærm har vi sat et fladkabel til Raspberry Pi B+’ J8-Header. Pins er forbundet efter samme fremgangsmåde bare igennem fladkablet, dog spejlvendt. Se billede:



#### Opsætning af modul RN-42 (Pmod BT2 module)

RN-42 er sat op som master. Specifikt er den indstillet til Master Mode 4 – Auto-Connect DTR Mode.(REFERENCE til side 11 i bluetooth\_cr\_UG-v1.0r (RN 42).pdf) Dette er den fordi at man derigennem kan styre dens autoforbindefunktion ved at holde JP3 på Pmod BT2 høj (3.3V). Denne autoforbindefunktion er ikke blevet implementeret fuldt i denne iteration. Autoforbinde styres ved hjælp af en ekstern jumper sat på JP3. Når autoforbindefunktionen aktiveres forbinder RN-42 automatisk til den MAC-adresse, der ligger inde i Remote Adress registret.(REFERENCE til side 25 i bluetooth\_cr\_UG-v1.0r (RN 42).pdf).

Der blev valgt at kører med en dataoverførselsstørrelse på 8 bit – ingen paritet.

For at Rock fik nok sensordata og processere bestemte vi at Body skulle sende alle sensordata 50 gange i sekundet. Alle disse data fylder:



I denne iteration er der valgt kun at bruge fire sensorer. Den endelige udgave skal have plads til 16 sensorer. Derfor valgte man at kører med en baudrate på 115200 for at være sikker på at have nok båndbrede til senere iterationer.

For at opsætte Rock skal man ind i Command Mode(REFERENCE til side 16 i bluetooth\_cr\_UG-v1.0r (RN 42).pdf). Det gør man ved at forbinde til RN-42 modulet trådløst og skrive kommandoen:

”$$$”

Inde i Command Mode sættes først navnet til ROCK, da dette modul skal repræsentere Rock delen af systemet. Kommando:

”SN,ROCK”

Hvis en kommando er godtaget respondere RN-42 med ”ACK”

Herefter sættes Remote Adress registret til HC-05s MAC adresse vha. følgende kommando:

”SR, 98D331B07C6B”

RN-42 sættes til Master mode 4:

”SM,4”

Efter dette er det vigtigt at opsætte Command Mode timeren, da man ellers ikke vil kunne få adgang til modulet igen uden at resette tilbage til fabriksindstillinger. Set timer kommando:

”ST,60”

I dette tilfælde er den sat til 60 sekunder.

Baudraten behøver man ikke at sætte da den default er sat til 115200.

Modulet er nu klar til dataoverførelse for Rockenheden.

### Body - trådløs kommunikation

#### Montering af modul HC-05

HC-05 er forbundet til PSoC4 boardet igennem Body shield(REFERENCE).

|  |  |
| --- | --- |
| HC-05 Pins  (REFERENCE) | PSoC 4 Board Pins (REFERENCE)[[12]](#footnote-12) |
| Pin VCC 3.3V | 3.3v DC Power |
| Pin GND | GND |
| Pin TXD | P0[4] RXD |
| Pin RXD | P0[5] TXD |

#### Opsætning af modul HC-05

Dette modul skal fungere som slave i forhold til RN-42 masteren. Derfor skal det sættes til de samme parametre.

* Baudrate 115200
* Dataoverførelsesstørrelse: 8 bit, no pararity

For at komme i AT command mode skal man slukke for modulet og drive KEY-pin/PIO11(REFERENCE til side 5 i DS\_BluetoothHC-05.pdf) høj (3.3V) og tænde modulet igen. RXD og TXD skal sættes op som beskrevet i *Montering af modul HC-05* og baudraten man skriver fra i terminalen(REFERENCE til Tera Term) skal være sat til 38400. I terminalen skrives kommandoen:

”AT”

Efter hver godkendt kommando svarer HC-05 med ”OK”

Først sættes navnet på HC-05 modulet til BODY:

AT+NAME=BODY

Herefter sættes baudrate og dataoverførelsesstørrelse:

”AT+UART=115200, 0,0”

Modulet er som default sat op i Slave Mode.

Modulet er nu klar til dataoverførelse for Bodyenheden.

# Test (HW)

## Enhedstests

### Trådløs kommunikation moduler

#### Modultest af datamodtagelse og -forsendelse på Body

**Formål**

At teste om Body modtager data fra Bluetooth-modulet HC-05.

Til dette formål er Body programmeret med en FPGA-UART. UART’en er sat op til en baud rate på 115.200 med 8 bit, no parity. HC-05 er forbundet korrekt til Body (PSoC4). Der etbleres forbindelse til Body fra computer via UART-terminalen Tera Term. Der etableres forbindelse til HC-05-modulet vha. af en ekstern Bluetooth-enhed (i dette tilfælde en Bluetooth-USB-adapter til PC).

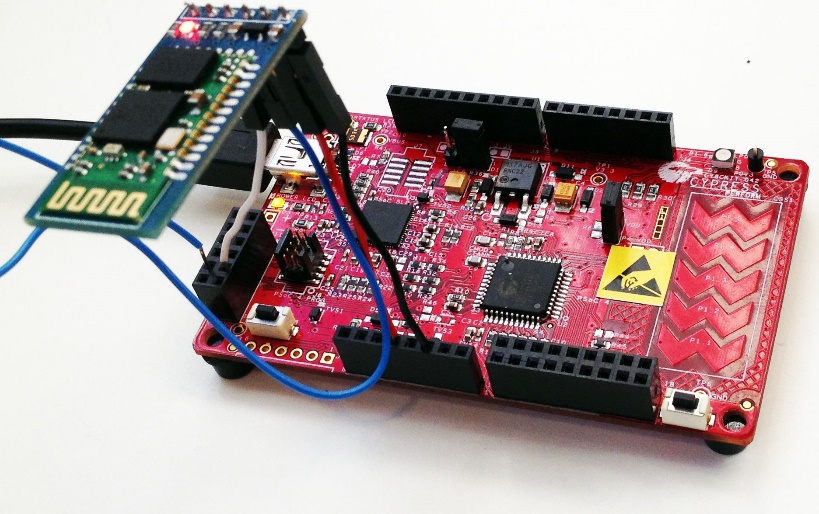
Testen optages, og der tages billeder som dokumentation.

Der er ét testscenarie i denne modultest:

1. Test af forsendelsesteksten ”Hello world” kan sendes og modtages både i Bodys terminal og på den eksterne Bluetooth-enheds terminal.

**Overordnet opstilling**

1. Body forbindes til en af computerens USB-porte.
2. HC-05-Bluetooth-modul monteres på Body med følgende forbindelser (HC-05 Body)
   * 3.3 V3.3\_EXT
   * GND GND
   * RXD P12\_7
   * TXD P12\_6
3. PC parres med HC-05 over Bluetooth
4. Et terminalprogram startes på PC med Bluetooth-USB-adapter
5. Et terminalprogram startes på PC med UART-forbindelse til Body

****

**Figur 4** Testopstilling til modultest af forsendelse

**Figur XX** viser testopstillingen. Den røde cirkel markerer HC-05-Bluetooth-modulet.

**Test**

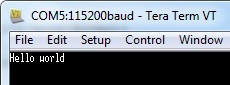
1. Opsæt systemet som beskrevet ovenfor
2. Skriv ”Hello world” i den eksterne Bluetooth-enhedsterminal
3. Der observeres om det bliver udskrevet på Tera Terms terminal
4. Herefter skrives der ligeledes ”Hello world” i Tera Terms terminal
5. Der observeres om det bliver udskrevet på den eksterne Bluetooth-enhedsterminal

**Forventet resultat**

Det forventes at der er udskrevet ”Hello world” i begge terminaler efter overstående handlinger er udført.

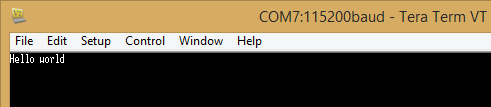
**Resultat**

**Figur XX** viser observationen fra testens punkt 3.



**Figur 5** Udskrift af besked fra PC til Body

**Figur XX** viser observationen fra testens punkt 5.



**Figur 6** Udskrift af besked sendt fra Body til PC

Der blev udskrevet i terminalerne som forventet.

Testen er godkendt.

#### Modultest af datamodtagelse og -forsendelse på Rock

**Formål**

At teste om Rock modtager data fra Bluetooth-modulet RN-42.

Til dette formål er programmet Minicom (REFERENCE til Minicom) installeret i Rocks styresystem, og sat op til en baud rate på 115.200, med 8 bit no parity. RN-42 er forbundet korrekt til Rock (Raspberry Pi). Der etableres forbindelse til RN-42-modulet vha. af en ekstern Bluetooth-enhed (i dette tilfælde en USB-Bluetooth-adapter til PC).

Testen optages, og der tages billeder som dokumentation.

Der er ét testscenarie i denne modultest:

1. Test af hvorvidt en forsendelse med teksten ”Hello world!” kan sendes og modtages, både i Rocks terminal, og på den eksterne Bluetooth-enheds terminal.

**Overordnet opstilling**

1. Rock forbindes til computerens Ethernet-port
2. RN-42 er monteret på Rock jf. montering af modul RN-42 (REFERENCE i produktdokumentation)
3. Der etableres en SSH-forbindelse til Rock
4. Minicom installeres og startes på Rock
5. PC parres med RN-42 over Bluetooth
6. Et terminalprogram startes på PC



**Figur 7** Testopstilling til modultest af forsendelse

**Figur XX** viser testopstillingen. Den røde cirkel markerer Bluetooth-USB-adapteren, mens den grønne cirkel marker RN-42-Bluetooth-modulet.

**Test**

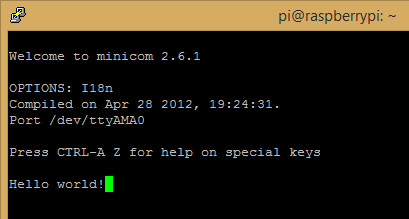
1. Opsæt systemet som beskrevet ovenfor.
2. Skriv ”Hello world!” i den eksterne Bluetooth-enhedsterminal
3. Der observeres om det bliver udskrevet på Rocks Minicom-terminal
4. Herefter skrives der ligeledes ”Hello world!” i Rocks Minicom terminal
5. Der observeres om det bliver udskrevet på den eksterne Bluetooth-enhedsterminal

**Forventet resultat**

Det forventes at der er udskrevet ”Hello world!” i begge terminaler efter overstående handlinger er udført.

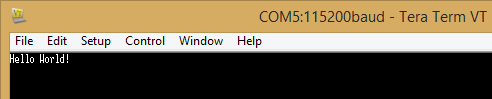
**Resultat**

**Figur XX** viser observationen fra testens punkt 3.



**Figur 8** Udskrift af besked sendt fra PC til Rock

**Figur XX** viser observationen fra testens punkt 5.



**Figur 9** Udskrift af besked sendt fra Rock til PC

Der blev udskrevet i terminalerne som forventet.

Testen er godkendt.

## Integrationstests

# Design og implementering (SW)

## Implementering af Body Control

### Body - Main

For beskrivelse af mains funktion se system arkitektur (REFERENCE)

|  |
| --- |
| Main  Tænd power-indikator  Initier I²C, sensorer og UART  Loop  Hvis nyt preset\*  Send preset  Læs alle sensorer via I²C  Konverter sensordata til rette format og placer i dataarray  Send dataarray via UART  Vent så forsendelsesraten er på 50Hz |

\*Da preset ikke er med i denne iteration er den heller ikke implementeret i den endelige udgave af mainkoden for projektet.

#### Main kode

int main()

{

//Initialisering

initUART();

initSensors();

// Tænd Power indikator

POWERPIN\_Write(1);

while(1)

{

// Klargøre I²C før hver aflæsning sensorer

setupI²C();

// Læs sensorer

readAllSensors();

// Standartiser sensordata

convSensData();

// Send sensordata

sendDataArray();

// Tilpas afsending til 50 Hz

CyDelay(19); // 19 ms

}

return 0;

}

Det skal nævnes at tilpasning til 50Hz i første udgave af Main koden var lavet med et timer interrupt der satte et flag højt hvert 20 ms (= 50Hz). Men at dette interrupt lavede fejl i I²C-aflæsningen af sensorerne.

#### initSensors

|  |
| --- |
| Skal kalde init-funktionerne for de ønskede sensorer |

#### 

#### Kode:

void initSensors()

{

/\*\*\*\*\* INIT INDIVIDUAL SENSORS \*\*\*\*\*/

initADXL345(); //Accellerometer

//initMPU6050(); //Gyroscope

// ..osv

}

#### initADXK345

|  |
| --- |
| Skal kalde funktionen setSensArray indtil alle seks registre fra accelerometeret er læst ind i funktionen.  Skal opsætte accelerometeret op til I²C |

#### Kode:

//Accelerometer ADXL345 defines

#define ACCEL\_ADDRESS 0x53 // jumperen sættes til GND

#define PWR\_CTRL\_REG 0x2D //Power control registeret Datablad side 26

#define PWR\_MODE (1<<3) // Sættes i measurement mode, standby til den skal måle (Strømbesparende)

#define DATAX0 0x32 //least significant

#define DATAX1 0x33 //most significant

#define DATAY0 0x34

#define DATAY1 0x35

#define DATAZ0 0x36

#define DATAZ1 0x37

//init accelerometer

void initADXL345(void)

{

// Fylder ADXL345 ind i sensor array'et.

setSensArray(ACCEL\_ADDRESS, DATAX0, DEFAULT\_DATA);

setSensArray(ACCEL\_ADDRESS, DATAX1, DEFAULT\_DATA);

setSensArray(ACCEL\_ADDRESS, DATAY0, DEFAULT\_DATA);

setSensArray(ACCEL\_ADDRESS, DATAY1, DEFAULT\_DATA);

setSensArray(ACCEL\_ADDRESS, DATAZ0, DEFAULT\_DATA);

setSensArray(ACCEL\_ADDRESS, DATAZ1, DEFAULT\_DATA);

//Opsætter til I²C

if(I²C\_1\_I²CMasterSendStart(ACCEL\_ADDRESS, I²C\_1\_I²C\_WRITE\_XFER\_MODE) == I²C\_1\_I²C\_MSTR\_NO\_ERROR //Tjekker om transfer er fuldført uden fejl

&& I²C\_1\_I²CMasterWriteByte(PWR\_CTRL\_REG) == I²C\_1\_I²C\_MSTR\_NO\_ERROR // POWER CONTROL REGISTER

&& I²C\_1\_I²CMasterWriteByte(PWR\_MODE) == I²C\_1\_I²C\_MSTR\_NO\_ERROR) // Sættes i measurement mode. Standby til den skal måle (Strømbesparende)

handleI²CError();

#### }

#### initMPU6050

|  |
| --- |
| Skal kalde funktionen setSensArray indtil alle seks registre fra gyroskopet er læst ind i funktionen.  Skal opsætte gyroskopet op til I²C |

#### Kode:

//Gyroscope MPU6050 defines

#define GYRO\_ADDRESS 0x68 //Jumperen sættes til GND

#define GYRO\_XOUT\_H 0x43 // Most significant

#define GYRO\_XOUT\_L 0x44 // Least significant

#define GYRO\_YOUT\_H 0x45

#define GYRO\_YOUT\_L 0x46

#define GYRO\_ZOUT\_H 0x47

#define GYRO\_ZOUT\_L 0x48

//init gyroscope

void initMPU6050(void)

{

// Fylder ADXL345 ind i sensor array'et.

setSensArray(GYRO\_ADDRESS, GYRO\_XOUT\_H, DEFAULT\_DATA);

setSensArray(GYRO\_ADDRESS, GYRO\_XOUT\_L, DEFAULT\_DATA);

setSensArray(GYRO\_ADDRESS, GYRO\_YOUT\_H, DEFAULT\_DATA);

setSensArray(GYRO\_ADDRESS, GYRO\_YOUT\_L, DEFAULT\_DATA);

setSensArray(GYRO\_ADDRESS, GYRO\_ZOUT\_H, DEFAULT\_DATA);

setSensArray(GYRO\_ADDRESS, GYRO\_ZOUT\_L, DEFAULT\_DATA);

//Opsætter til I²C

if(I²C\_1\_I²CMasterSendStart(GYRO\_ADDRESS, I²C\_1\_I²C\_WRITE\_XFER\_MODE) == I²C\_1\_I²C\_MSTR\_NO\_ERROR) //Tjekker om transfer er fuldført uden fejl

handleI²CError();

}

### Body - SerialUnit

Beskrivelser af SerialUnits funktioner kan ses i system arkitektur (REFERENCE)

|  |
| --- |
| Opret klasse-lokalt dataarray[16] // Med plads til fire sensorer\* |

\*I projektet var der oprindeligt afgrænset til fire sensorer i den endelige udgave er der afgrænset til én (REFERENCE).

#### Kode:

// Klasse-lokalt array

char dataArray[MAXSENSDATA]; // Array with id-number and data to send via BT

#### initUART

|  |
| --- |
| initUART  Start og initer UART komponenten |

#### Kode:

void initUART()

{

// Enable og initier UART komponenten

UART\_1\_Start();

}

#### setDataArray

|  |
| --- |
| setDataArray(ID, X-sensordata, Y-sensordata, Z-sensordata)  Sæt klasse-lokale dataarray til ID, X-sensordata, Y-sensordata,  Z-sensordata  Sæt ubrugte pladser til kendt værdi = 1 |

#### Kode:

void setdataArray(int ID, int x\_data, int y\_data, int z\_data)

{

// Klasse-lokale dataarray til ID, x\_data, y\_data, z\_data

dataArray[0] = ID+1;

dataArray[1] = x\_data+1;

dataArray[2] = y\_data+1;

dataArray[3] = z\_data+1;

//Sætter de resterende pladser i array'et til 1.

int i;

for(i = 4; i<MAXSENSDATA; i++)

{

dataArray[i] = 1;

}

}

#### sendDataArray

|  |
| --- |
| sendDataArray  Opret et array med seks pladser pr sensor jf. Trådløs kommunikations  protokol(REFERENCE)  Sæt hvert sensorarray med de tilsvarende pladser fra det  klasse-lokale dataarray\* //Alle data der sendes bliver +1 pga.  //datavalidering i receiver klassen på Rock  Valider sensordata // Hvis der er fejl tænd rød Error diode  Send sensordata  Reset dataarray til kendt værdi // ASCII ’0’ = 48 |

\*Da der er afgrænset til en sensor i denne udgave af projektet vil alle andre arrays end accelerometer arrayet blive sat til 1.

#### Kode:

void sendDataArray()

{

// Opret et array med seks pladser pr sensor

char accelArray[6];

char gyroArray[6];

char proxArray[6];

char flexArray[6];

// Sætter accelerometer array.

accelArray[0] = START\_SENSDATA;

accelArray[1] = dataArray[0]; // dataArray[0] - ID 0

accelArray[2] = dataArray[1]; // dataArray[1] - x

accelArray[3] = dataArray[2]; // dataArray[2] - y

accelArray[4] = dataArray[3]; // dataArray[3] - z

accelArray[5] = 0; // Nul terminering

// Sætter gyroskop array

gyroArray[0] = START\_SENSDATA;

gyroArray[1] = 1+1; // dataArray[4] - ID 1

gyroArray[2] = 1+1; // dataArray[5] - x

gyroArray[3] = 1+1; // dataArray[6] - y

gyroArray[4] = 1+1; // dataArray[7] - z

gyroArray[5] = 0; // Nul terminering

// Sætter Proximity array

proxArray[0] = START\_SENSDATA;

proxArray[1] = 2+1; // dataArray[8] - ID 2

proxArray[2] = 1+1; // dataArray[9] - x

proxArray[3] = 1+1; // dataArray[10] - y

proxArray[4] = 1+1; // dataArray[11] - z

proxArray[5] = 0; // Nul terminering

// Sætter flex array

flexArray[0] = START\_SENSDATA;

flexArray[1] = 3+1; // dataArray[12] - ID 3

flexArray[2] = 1+1; // dataArray[13] - x

flexArray[3] = 1+1; // dataArray[14] - y

flexArray[4] = 1+1; // dataArray[15] - z

flexArray[5] = 0; // Nul terminering

//Error Check om I²C sensordata er valid

if((dataArray[1] == 65)&&(dataArray[2] == 65)&&(dataArray[3] == 65))

{

ERRORPIN\_Write(1);

}

//Send sensordata

UART\_1\_UartPutString(accelArray);

UART\_1\_UartPutString(gyroArray);

UART\_1\_UartPutString(proxArray);

UART\_1\_UartPutString(flexArray);

//Reset dataArray til kendt acii værdi: 48 = '0';

int i;

for(i = 0; i < MAXSENSDATA; i++)

{

dataArray[i] = 48;

}

}

### Body - Sensor

Beskrivelser af Sensors funktioner kan ses i system arkitektur (REFERENCE)

#### setupI²C

|  |
| --- |
| Skal kalde de to initieringsfunktioner fra I²C komponenten. |

#### Kode:

void setupI²C()

{

I²C\_1\_Start(); // Funktion fra I²C komponent datablad

I²C\_1\_I²CMasterClearStatus(); // Funktion fra I²C komponent datablad

}

#### setSensArray

|  |
| --- |
| setSensArray(deviceAddress, registerToRead, data)  Sæt plads [i][0] med sensorens device adresse  Sæt plads [i][1] med det sensor register som skal læses fra  Sæt plads [i][2] med data fra register som skal læses fra |

#### Kode:

void setSensArray(int deviceAddress, int registerToRead, int dataTarget)

{

sensArray[numOfRegToRead][0]=deviceAddress; //set device address

sensArray[numOfRegToRead][1]=registerToRead;//set register for future read

sensArray[numOfRegToRead][2]=dataTarget; //set where to save read data

++numOfRegToRead; //increment current number of places in array that are set

}

#### readAllSensors

|  |
| --- |
| Skal kalde funktionen readI²C i det antal af regsitre det ønskes læst fra |

#### Kode:

void readAllSensors()

{

int numOfReg; // Variabel til at inkrementere

for (numOfReg = 0; numOfReg < numOfRegToRead; numOfReg++)

{

readI²C(numOfReg); // Kalder readI²C for hver inkrementering

}

}

#### convSensdata\*

|  |
| --- |
| convSensdata  Opret variabler  Saml most significant og least significant for hver sensorkoordinat  X, Y, Z i variabler  Konverter fra værdier fra [-512..511] til værdier [0-127] og læg i  8bit variabel  Kald setdataArray fra SerialUnit og sæt accelerometer ID samt  konverteret X, Y og Z værdier |

\*Denne funktion er kun lavet til at standardiser accelerometerets(ADXL345) sensordata da de andre er afgrænset væk.

#### Kode:

void convSensData(void)

{

//ADXL345

// Opret variabler

int16 x, y, z; // Da det er to 8 bit registre som skal lægges sammen

unsigned char x1, y1, z1;

//Samler most significant og least significant for X

x = sensArray[1][2]<< 8;

x+= sensArray[0][2];

//Samler most significant og least significant for Y

y = sensArray[3][2]<< 8;

y+= sensArray[2][2];

//Samler most significant og least significant for Z

z = sensArray[5][2]<< 8;

z+= sensArray[4][2];

//Konvertering fra int16 til uint8

x1 = (x+512)/8;

y1 = (y+512)/8;

z1 = (z+512)/8;

// Sæt dataarray i SerialUnit

setdataArray(ACC1\_ID, x1, y1, z1);

}

#### readI²C

|  |
| --- |
| Skal generisk kunne læse fra en sensor vha. |

#### Kode:

void readI²C(int numOfReg)

{

uint8 rawData;

if (I²C\_1\_I²CMasterSendStart(sensArray[numOfReg][0],

I²C\_1\_I²C\_WRITE\_XFER\_MODE) == I²C\_1\_I²C\_MSTR\_NO\_ERROR)

/\* Check if transfer completed without errors \*/

{

if(I²C\_1\_I²CMasterWriteByte(sensArray[numOfReg][1]) ==

I²C\_1\_I²C\_MSTR\_NO\_ERROR)

{

if(I²C\_1\_I²CMasterSendRestart(sensArray[numOfReg][0],

I²C\_1\_I²C\_READ\_XFER\_MODE) == I²C\_1\_I²C\_MSTR\_NO\_ERROR)

{

rawData = I²C\_1\_I²CMasterReadByte(I²C\_1\_I²C\_NAK\_DATA);

I²C\_1\_I²CMasterSendStop();

sensArray[numOfReg][2] = rawData;

// gemmer rawData på den tilhørende plads i sensArray.

}

}

}

}

#### handleI²CError

|  |
| --- |
| Skal håndtere hvis der kommer fejl på I²C ved at sende en stop |

#### Kode:

void handleI²CError(void)

{

I²C\_1\_I²CMasterSendStop();

}

## FlexiForce A301 Sensor

### FlexiForce A301 Main



I²C delen af styresystemet på ATtiny26L til FlexiForce A301 er lavet ud fra open source koden usiTwiSlave. (REFERENCE). Koden er oprindelig lavet ud fra et officielt dokument fra Atmel der beskriver opsætning af I²C på en AVR310 IC(REFERENCE).

Koden er herefter tilpasset projektet således at det passer til en ATtiny26L samt den måde I²C er valgt at kører i dette projekt.

ADC’en er indstillet til at reagere med eksternt interrupt. Når der sker en ændring i spændingen i FlexiForce A301 aflæses spændingsværdien relativt i forhold til reference spænding på 1.65V og sættes ind i et array.

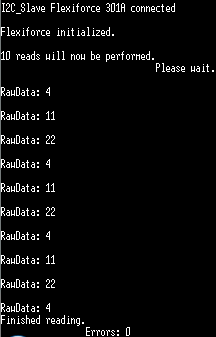
Når I²C masteren (Body) beder om en forsendelse af sensordata sendes en pointer værdi med. Alt efter denne pointer værdi tilbage sendes plads 1 eller 2 i ADC data arrayet.

**ATtiny26L Main beskrevet i pseudo kode:**

Main  
 Initier I²C, ADC og global interupt  
 Sæt og initier adcArray  
 LOOP  
 Vent på master  
 Læs pointer værdi fra master  
 Vent på master  
 Afsend adcArray udfra pointerværdi  
 Hvis ny ADC data flag er høj  
 Sæt ADC data ind i adcArray  
 Sæt ADC data flag lav

ADC ISR  
 Sæt ADC data flag høj

Implementeringen af styresystemet for ATtiny26L er ikke færdig i denne iteration af projektet. Styresystemet mangler at få I²C- og ADCdelen til at fungere sammen.   
I²C forbindelsen fra Body til ATtiny26L med afsendelse af dummy ADC(REFERENCE) fungere på nuværende tidspunkt.



Figur 10 Udklip af test - 10 Reads med udskrift

# Rock SW

## Databank dokumentation

Databanken er et API der agerer som bindeled mellem Controlleren (og dermed GUI) og Databankens underklasser.

Databankens funktioner er alle wrapper-funktioner, der ikke laver andet end at kalde funktioner i dens underklasser som er beskrevet nedenunder:

## Sensorkonfigurationsbankens funktioner

Sensorkonfigurationsbanken inkluderer headerfilen *SensorConfiguration.h* der indeholder implementeringen af en sensorkonfiguration. Denne klasse har kun get/set-metoder. For alle attributter (navn, sensor ID, akse, mapping scheme, lydpakke) i en sensorkonfiguration findes der én set-metode og én get-metode. Sensorkonfigurationsbanken har en container (*std::map*) med disse sensorkonfigurationer.

|  |  |
| --- | --- |
| **Funktion** | SensorConfigurationBank - *Constructor* |
| **Parametre** |  |
| **Returværdi** |  |
| **Beskrivelse** | ’Loader’ sensorkonfigurationerne, ved at bruge Boost:   * Læser XML-filen ”SensorConfigurationBank.xml” og gemmer hver *configuration*-knude som en *SensorConfiguration*-klasse * XML-filen indeholder en knude *mappingscheme* som er referencen (navnet) til et mapping scheme * Mapping schemet er en underklasse til sensorkonfigurationen. Vi skal have fat i disse tilhørende attributter, og derfor køres *getMappingScheme*   Disse sensorkonfigurationer tilføjes til mappet *SensorConfigurations*, så de kan bruges omgående. |

|  |  |
| --- | --- |
| **Funktion** | getMappingScheme |
| **Parametre** | string id – navnet på mapping schemet vi skal have hentet de tilhørende attributter til. |
| **Returværdi** | MappingScheme – en mapping scheme klasse der skal bruges til en sensorkonfiguration. |
| **Beskrivelse** | Funktionen virker lidt ligesom constructoren *SensorConfigurationBank*.   * Læser XML-filen ”MappingSchemes.xml” * Finder den specifikke mapping scheme med samme navn som den *id* man sendte med som parameter * Gemmer attributterne fra denne *scheme*-knude til en *MappingScheme* klasse * Returnér denne |

|  |  |
| --- | --- |
| **Funktion** | getMappingSchemes – *ej at forveklse med overstående* |
| **Parametre** |  |
| **Returværdi** | Returnerer en *vector<string>* med navnene på alle mapping schemes. |
| **Beskrivelse** | Minder lidt om constructoren og *getMappingScheme*.   * Læser XML-filen ”MappingSchemes.xml” * Finder alle *id­*-knuder og tilføjer indholdet af disse til en *vector* *myMappingSchemeList* * Returnér denne   Bruges i GUI’en for at få et overblik over hvilke mapping schemes der er til rådighed. |

|  |  |
| --- | --- |
| **Funktion** | getSensorIDs |
| **Parametre** |  |
| **Returværdi** | Returnerer en *vector<string>* med ID på alle sensorer der er i brug. |
| **Beskrivelse** | Samme som overstående, bare med en anden XML-fil og anden knude.   * Læser XML-filen ”SensorConfiguration.xml” * Finder alle *sensorid*-knuder i alle *configuration*-knuder og tilføjer indholdet af disse til en *vector mySensorIDs* * Returnér denne |

|  |  |
| --- | --- |
| **Funktion** | edit |
| **Parametre** | En sensorkonfiguration man gerne vil ændre |
| **Returværdi** | void |
| **Beskrivelse** | * Sletter sensorkonfiguration med samme navn * Tilføjer den ’nye’ sensorkonfiguration vha. funktion *add* |

|  |  |
| --- | --- |
| **Funktion** | add |
| **Parametre** | En sensorkonfiguration man gerne vil tilføje |
| **Returværdi** | void |
| **Beskrivelse** | * Indsætter sensorkonfiguration i *sensorConfigurations*-mappet, hvor data er sensorkonfigurationen man gerne vil tilføje, og *key*’en er navnet på sensorkonfigurationen * Kalder metoden *save* |

|  |  |
| --- | --- |
| **Funktion** | removeSensorConf |
| **Parametre** | Navnet (*key*’en) på sensorkonfigurationen man gerne vil fjerne. |
| **Returværdi** | void |
| **Beskrivelse** | * Fjerner sensorkonfigurationen fra *sensorConfigurations*-mappet * Kalder metoden *save* |

|  |  |
| --- | --- |
| **Funktion** | save |
| **Parametre** |  |
| **Returværdi** | void |
| **Beskrivelse** | Gemmer (serialiserer) sensorkonfigurationerne i *sensorConfigurations*-mappet. Disse gemmes til ”SensorConfigurationBank.xml”   * Opret ’træ’ (*boost::property\_tree::ptree)* vi kalder for ’*pt*’ der fungerer som roden til vores XML-dokument der skal gemmes om lidt * For hver sensorkonfiguration der findes i *sensorConfigurations*-mappet:   + Opret endnu et træ *myTree* der vil fungerer som en gren i XML-dokumentet   + Tilføjer alle attributter fra netop den sensorkonfiguration vi kigger på nu som knuder til dette nye træ   + *myTree* sættes som child (gren) til roden/parent-træet *pt* * Gem træet *pt* som XML-filen med navnet “SensorConfigurationBank.xml” |

|  |  |
| --- | --- |
| **Funktion** | list |
| **Parametre** |  |
| **Returværdi** | Returnerer en *vector<string>* med navnene på alle sensorkonfigurationer. |
| **Beskrivelse** | * Opret ny *vector* der hedder *sensorConfigs* * For alle sensorkonfigurationer i *sensorConfigurations*: * Læg navnet på sensorkonfigurationen ind i *sensorConfigs* * Returnér *sensorConfigs* |

|  |  |
| --- | --- |
| **Funktion** | get |
| **Parametre** | Navnet på sensorkonfiguration der ønskes returneret |
| **Returværdi** | Sensorkonfigurationen man ønsker |
| **Beskrivelse** | * Da navnet på sensorkonfigurationen man gerne vil returnere er det samme som *key*’en i *sensorConfigs*, skal man blot returnere det element der binder sig til *key’*en som er input parameteren. |

## Presetbankens funktioner

|  |  |
| --- | --- |
| **Funktion** | PresetBank – *Constructor* |
| **Parametre** |  |
| **Returværdi** |  |
| **Beskrivelse** | ’Loader’ presets, ved at bruge Boost:   * Læser XML-filen ”PresetBank.xml” og gemmer hver *preset*-knude som en *Preset*-klasse   Disse presets tilføjes til vectoren *presets\_*, så de kan bruges omgående. |

|  |  |
| --- | --- |
| **Funktion** | get |
| **Parametre** | Et tal der repræsenterer pladsen i *presets­\_* som presettet man gerne vil ’gette’ ligger på. |
| **Returværdi** | Presettet man gerne vil have |
| **Beskrivelse** | * Returnér preset på input-parameter pladsen fra *presets\_* |

|  |  |
| --- | --- |
| **Funktion** | edit |
| **Parametre** | Et preset man gerne vil ændre |
| **Returværdi** | void |
| **Beskrivelse** | * Assign det nye preset i vectoren *presets\_* |

|  |  |
| --- | --- |
| **Funktion** | save |
| **Parametre** |  |
| **Returværdi** | void |
| **Beskrivelse** | Gemmer (serialiserer) presets der ligger i *presets\_*. Disse gemmes til ”PresetBank.xml”   * Opret ’træ’ (*boost::property\_tree::ptree)* vi kalder for ’*pt*’ der fungerer som roden til vores XML-dokument der skal gemmes om lidt * For hver preset der findes i *presets­\_*-vectoren:   + Tilføj en ’gren’ (knude) *preset* der indeholder de sensorkonfigurationer der skal bruges. * Gem træet *pt* som XML-filen med navnet “PresetBank.xml” |

## Samplebankens funktioner

|  |  |
| --- | --- |
| **Funktion** | SampleBank – *Constructor* |
| **Parametre** |  |
| **Returværdi** |  |
| **Beskrivelse** | ’Loader’ lydpakkerne, ved at bruge Boost:   * Læser XML-filen ”SampleBank.xml” og gemmer hver *sample*-knude som en *SoundPack*-klasse   Disse lydpakker tilføjes til mappet *soundPacks*, så de kan bruges omgående. |

|  |  |
| --- | --- |
| **Funktion** | list |
| **Parametre** |  |
| **Returværdi** | Returnerer en *vector<string>* med navnene på alle lydpakker |
| **Beskrivelse** | * Opret ny *vector* med navnet *soundsVector* * For hver lydpakke i mappet *soundPacks*: * Tilføj navnet på lydpakken til *soundsVector* * Returnér denne |

|  |  |
| --- | --- |
| **Funktion** | add |
| **Parametre** | Navn på lydpakken |
| **Returværdi** | void |
| **Beskrivelse** | * En ny lydpakke oprettes, hvor stien er defaultstien konkateneret med input parameteren (navnet) på lydpakken * Indsæt denne lydpakke i mappet *soundPacks*, hvor *key*’en er input parameteren, og data’en er selve lydpakken. * Kald funktionen *save* |

|  |  |
| --- | --- |
| **Funktion** | remove |
| **Parametre** | Navn på lydpakke |
| **Returværdi** | void |
| **Beskrivelse** | * Slet lydpakken med input-parameteren (navnet) *key* i mappet *soundPacks* * Kald funktionen *save* |

|  |  |
| --- | --- |
| **Funktion** | save |
| **Parametre** |  |
| **Returværdi** | void |
| **Beskrivelse** | Gemmer (serialiserer) lydpakkerne i *soundPacks*-mappet. Disse gemmes til ”SampleBank.xml”   * Opret ’træ’ (*boost::property\_tree::ptree)* vi kalder for ’*pt*’ der fungerer som roden til vores XML-dokument der skal gemmes om lidt * For hver sensorkonfiguration der findes i *soundPacks*-mappet:   + Tilføj en ’gren’ (knude) *sample* der indeholder stien på lydpakken * Gem træet *pt* som XML-filen med navnet “SampleBank.xml” |

# Test (SW)

## Enhedstests

### Sensor software

#### Test af I²C forbindelse mellem Body og BR3K FlexiForce A301 V1\_0 med dummy ADC data

**Formål**

At teste om Body modtager data fra Flexiforce sensoren.

Til dette formål er Body programmeret med Sensorforbindelse\_body\_v1. UART’en er sat op til en baud rate på 9600 med 8 bit, no parity. Body shield er forbundet korrekt til Body (PSoC4). Der etableres forbindelse til Body fra computer via UART-terminalen Tera Term. BR3K FlexiForcer A301 V1.0 er forbundet til Body shield via RJ11 kabler.

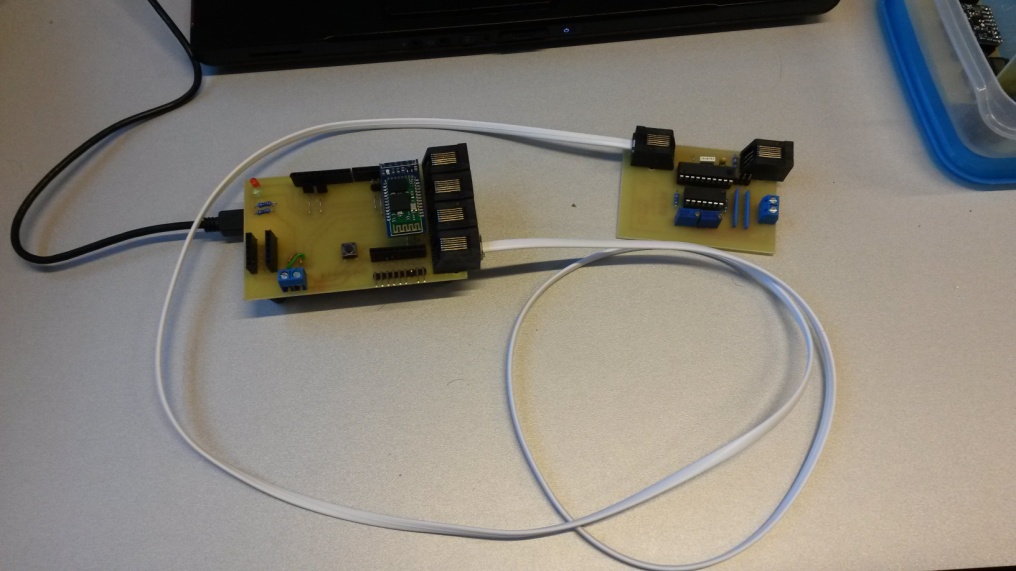
Testen dokumenteres med billeder fra Tera Term.

Der er to testscenarier i denne modultest:

1. Test af 1000 reads af sensoren
2. Test af 10 reads med udskrift af sensor ID og dummy data

**Overordnet opstilling**

1. Body shield monteres på PSoC4 boardet
2. Body forbindes til en af computerens USB-porte
3. Et terminalprogram startes på PC med UART-forbindelse til Body
4. BR3K FlexiForcer A301 V1.0 er forbundet til Body shield via RJ11 kabler



**Figur 11** Testopstilling til test af I²C forbindelse mellem Body og BR3K FlexiForcer A301 V1.0 med dummy ADC data

**Test 1 - 1000 reads af sensoren**

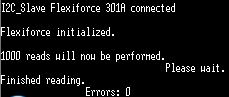
1. Opsæt systemet som beskrevet ovenfor
2. Programmer Body(PSoC) med Sensorforbindelse\_body\_v1.
3. Der observeres om testen bliver udskrevet på Tera Term

**Forventet resultat**

Det forventes at testen gennemføres med 1000 reads og 0 errors

**Resultat**

**Figur XX** viser observationen fra testens punkt 3.



**Figur 12** Test 1 udskrift fra Tera Term

**Test 2 - 10 reads med udskrift af sensor ID og dummy data**

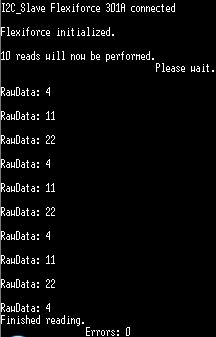
1. Opsæt systemet som beskrevet ovenfor
2. Programmer Body(PSoC) med Sensorforbindelse\_body\_v1.
3. Der observeres om testen bliver udskrevet på Tera Term

**Forventet resultat**

Det forventes at testen gennemføres med 10 reads og 0 errors samt der er udskrift af sensordata

**Resultat**

**Figur XX** viser observationen fra testens punkt 3.



Figur 13 Test 2 udskrift fra Tera Term

Testen er godkendt.

## Integrationstests

### Integrationstest af fast lane (Fra sensor til ekstern MIDI output)

**Formål**

At teste om ekstern MIDI output reagere på sensormanipulation.

Til dette formål er Bodys styresystem installeret og startet på PSoC4. PSoC4 er udstyret med Body Shield. En tilfældig sensor er monteret via. RJ11 kable til Body Shield I²C port. Rocks GUI er installeret og startet på Raspberry Pi. Rock er forbundet til det ekserne lydkort MIDI Mate(REFERENCE MIDI Mate) som er forbundet til det eksterne MIDI output Nord Rack 2X(REFERENCE til Nord Rack 2X).

Testen optages som dokumentation.

Der er ét testscenarie i denne integrationstest:

1. Test af ekstern MIDI outputs reaktion ved sensormanipulation

**Overordnet opstilling**

1. PSoC4 forbindes til en af computer 1s USB-indgange.
2. PSoC shield sættes nedover PSoC4.
3. HC05 bluetoothmodul monteres på PSoC4 jf. Montering af modul HC05(REFERNCE i produktdokumentation)
4. PSoC4 programmeres med Body\_Main\_Version\_4
5. Der åbens en Tera Term forbindelse fra computer 1 til Bodys UART med baud rate 115200, 8 bit data, 1 stopbit, ingen paritet
6. Raspberry Pi forbindes til computer 2s Ethernet indgang.
7. Der etableres en SSH forbindelse til Raspberry Pi’ styresystem
8. RN-42 er monteret på Raspberry Pi jf. Montering af modul RN-42(REFERNCE i produktdokumentation)
9. Raspberry Pi installeres med Rock GUI og startes
10. MIDI Mate sættes i en af Raspberry Pis fire USB porte
11. MIDI Mate forbindes til ekstern Nord Rack 2X porte med Input til MIDI out og output til MIDI IN.
12. Nord Rack 2X startes med standart program 110
13. Højtaler monteres i Nord Racks 2X headphone stik
14. PSoC4 initieres vha. reset knappen

**Test opstilling**

****

Figur 1 Test opstilling Body

Blå: USB forbindelse, GUL: Sensor, Grøn: HC-05, Rød: Body



Figur 2 Test opstilling Rock

Lyseblå: Raspberry Pi og RN-42, Rød: MIDI Mate, Lyserød: Nord Rack 2X, GUL: Højtaler, Lilla: Ethernet forbindelse.

**Test**

1. Opsæt systemet som beskrevet ovenfor.
2. Tryk på resetknappen placeret på PSoC shield.

**Forventet resultat**

Det forventes at der at der afspilles lyd ved sensormanipulation, computer 1s Tera Term viser sensormanipulationen tolket som ASCII værdier og computer2 viser modtagende værdier ved sensor manipulation.

**Resultat**

Se video(REFERENCE)

Testen er godkendt.

### Integrationstest af sensordataforsendelse på Body

**Formål**

At teste om Body afsender sensordata fra en tilfældig sensor, via Bodys Bluetooth-modul, til Rocks Bluetooth-modul, 50 gange i sekundet, dvs. en forsendelse pr 20 ms.

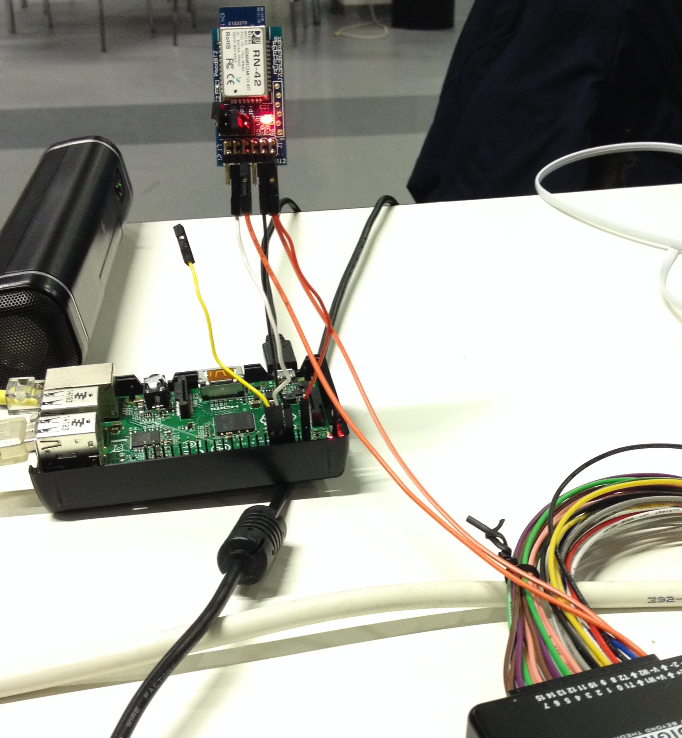
Til dette formål er et accelerometer (ADXL345) forbundet til Body. Bodys main-program er startet, og et Bluetooth-modul (HC-05) er forbundet til PSoC-shield’et, som er påsat Body. På Rock er et Bluetooth-modul (RN-42) forbundet til 3,3V og GND, og et oscilloskop er forbundet til Bluetooth-modulets Tx-ben, samt 3,3V som referencespænding. Integrationstesten dokumenteres med et oscilloskopbillede, samt et billede af opstillingen.

Der er ét testscenarie i denne integrationstest:

1. Test af forsendelsesfrekvens (50 Hz)

**Overordnet opstilling**

1. Body forbindes til en af computerens USB-porte
2. PSoC-shield, med HC-05 påsat, sættes ned over Body
3. RJ11-kablets ene ende forbindes til en af de fire connectors på PSoC-shieldet
4. RJ11-kablets anden ende forbindes til den ene af de to connectors på ADXL345-printet
5. Jumperen på ADXL345 printet sættes til GND
6. Body programmeres med Body\_Main\_Version\_4
7. RN-42 tilsluttes 3,3V og GND på Rock
8. Oscilloskopet forbindes til TXD og 3,3V på RN-42
9. Jumper J3 på RN-42 forbindes til GND

****

**Figur 1** Testopstilling til integrationstest af forsendelse

**Figur XX** viser testopstillingen.

**Test**

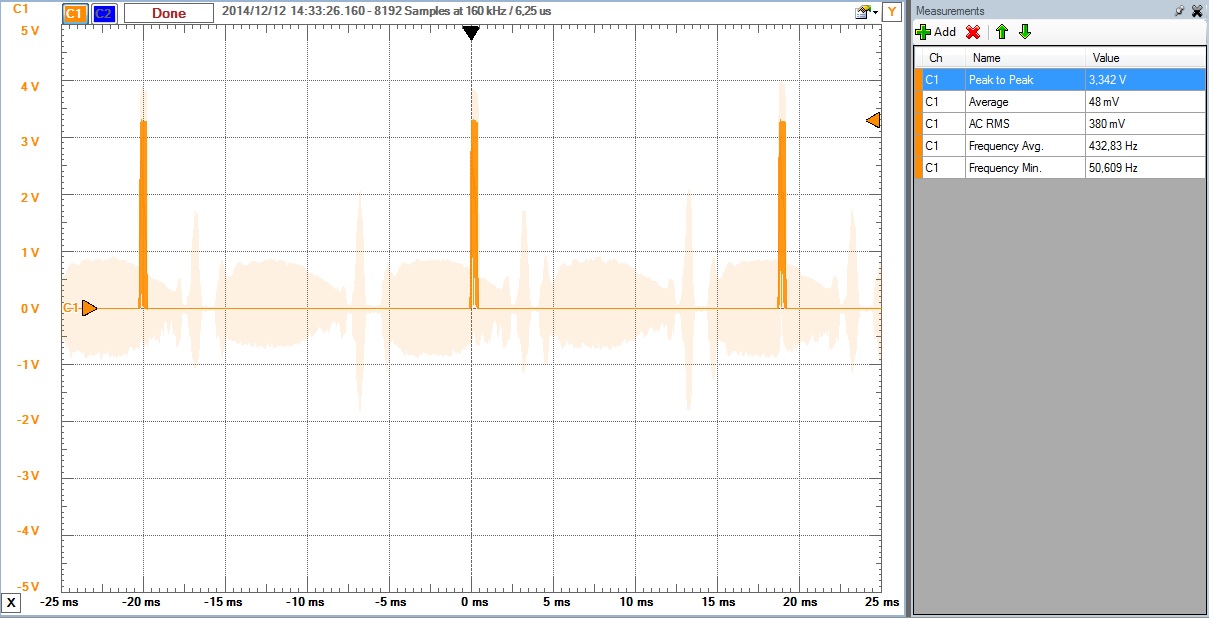
1. Opsæt systemet som beskrevet ovenfor.
2. Foretag måling med oscilloskopet, og observer afstand mellem forsendelserne

**Forventet resultat**

Det forventes at der at der er ca. 20 ms mellem hver forsendelse.

**Resultat**

Som det fremgår af **figur XX**, er der ca. 20 ms afstand mellem hver forsendelse af data. Dette screenshot er dog et af de ”pænere”, da der både kan være kortere og længere mellem forsendelserne, men der foretages ca. 50 forsendelser i sekundet, med en acceptabel tolerancegrænse.



**Figur 2** Oscilloskop billede

Testen er godkendt.

# Accepttest

# Referencer

MIDI Manufacturers Organization. (n.d.). *The Complete MIDI 1.0 Detailed Specification*. Retrieved from midi.org: http://www.midi.org/techspecs/midispec.php

# Bilag

Bilag forefindes på CD-rom

Oversigt noteres her

1. Teknisk set ikke simultant, men virtuelt simultant, da mennesket vil opfatte det sådan [↑](#footnote-ref-1)
2. (MIDI Manufacturers Organization, u.d.) [↑](#footnote-ref-2)
3. Konverteringsmodulet indeholder én til flere konverteringstråde, afhængig af antal sensorer i systemet [↑](#footnote-ref-3)
4. For beskrivelse af I²C, se Bilag 15 [↑](#footnote-ref-4)
5. For en detaljeret beskrivelse af I²C se Bilag 15 [↑](#footnote-ref-5)
6. For en detaljeret beskrivelse af funktionen se Bilag 13 s. 11 [↑](#footnote-ref-6)
7. Se *Opsætning af modul RN-42 (Pmod BT2 module)* [↑](#footnote-ref-7)
8. Bilag 5 [↑](#footnote-ref-8)
9. Denne iteration samler kun data fra en sensor. [↑](#footnote-ref-9)
10. REFERENCE [↑](#footnote-ref-10)
11. http://www.raspberrypi.org/forums/viewtopic.php?f=78&t=82397 [↑](#footnote-ref-11)
12. http://www.raspberrypi.org/forums/viewtopic.php?f=78&t=82397 [↑](#footnote-ref-12)