**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**

15. december 2014

# Abstract

Resuméet skal udarbejdes **både** på dansk og på engelsk (abstract). Resuméet skal være en kortfattet sammenfatning af projektrapporten og bør indeholde følgende:

* Projektets formål
* Projektets problemstilling
* Valgte løsninger
* Anvendte metoder
* De væsentligste resultater

Resuméet skal beskrive arbejdet mere præcist, end det er muligt i den korte titel, således at læseren herudfra kan afgøre, om det er umagen værd at læse hele rapporten. Det skal kunne læses selvstændigt og må ikke indeholde henvisninger til afsnit i rapporten. Resuméets omfang må ikke overstige 2-300 ord og under **ingen** omstændigheder 500 ord. Skriv så konkret som muligt, og undgå vage udtryk eller fyldord.

Indhold

[Abstract 2](#_Toc406410840)

[Indledning 7](#_Toc406410841)

[Opgaveformulering 7](#_Toc406410842)

[Projektafgrænsning 8](#_Toc406410843)

[Systembeskrivelse 9](#_Toc406410844)

[Krav 10](#_Toc406410845)

[Aktørbeskrivelse 10](#_Toc406410846)

[Use case-beskrivelse 11](#_Toc406410847)

[Forbind Body og Rock 11](#_Toc406410848)

[Installér lydpakker 11](#_Toc406410849)

[Konfigurer sensorer 11](#_Toc406410850)

[Konfigurer presets 11](#_Toc406410851)

[Vælg preset 11](#_Toc406410852)

[Indsaml sensordata 11](#_Toc406410853)

[Generér MIDI 11](#_Toc406410854)

[Afspil lyd 11](#_Toc406410855)

[Projektgennemførelse 12](#_Toc406410856)

[Udviklingsmodel 12](#_Toc406410857)

[Projektstyring 13](#_Toc406410858)

[Scrum 13](#_Toc406410859)

[Tidsplan 14](#_Toc406410860)

[Mødestruktur 15](#_Toc406410861)

[Dokumentorganisering og log 15](#_Toc406410862)

[Metoder 15](#_Toc406410863)

[SysML 15](#_Toc406410864)

[Stucture Diagram 16](#_Toc406410865)

[Behavior Diagram 16](#_Toc406410866)

[Applikationsmodel 16](#_Toc406410867)

[Scrum 16](#_Toc406410868)

[Nøglebegreber i Scrum 16](#_Toc406410869)

[Afvigelser i brugen af Scrum i dette projekt 16](#_Toc406410870)

[Scrum Sprints 17](#_Toc406410871)

[Specifikation og analyse 17](#_Toc406410872)

[Sensortyper 17](#_Toc406410873)

[Bus-teknologier 17](#_Toc406410874)

[Ekstern instrument interface 17](#_Toc406410875)

[LydSampler 18](#_Toc406410876)

[Trådløse teknologier: 19](#_Toc406410877)

[Kunde undersøgelse 19](#_Toc406410878)

[Systemarkitektur 21](#_Toc406410879)

[Overordnet arkitektur 21](#_Toc406410880)

[Design, implementering og test af HW 22](#_Toc406410881)

[Indledende designovervejelser 22](#_Toc406410882)

[Overvejelser omkring sensorer 22](#_Toc406410883)

[Sensorer 22](#_Toc406410884)

[Accelerometer 23](#_Toc406410885)

[Gyroskop 23](#_Toc406410886)

[Proximity sensor 24](#_Toc406410887)

[Tryksensor 24](#_Toc406410888)

[I²C-bus 24](#_Toc406410889)

[Trådløs kommunikation 24](#_Toc406410890)

[PSoC shield 24](#_Toc406410891)

[Spændingsforsyning 25](#_Toc406410892)

[Reguleringskreds 25](#_Toc406410893)

[Batteri 25](#_Toc406410894)

[Design, implementering og test af SW 25](#_Toc406410895)

[Body 26](#_Toc406410896)

[Body - Main 26](#_Toc406410897)

[Sensorer 27](#_Toc406410898)

[SerialUnit 27](#_Toc406410899)

[Slow Lane 28](#_Toc406410900)

[Controller 28](#_Toc406410901)

[Databank 28](#_Toc406410902)

[GUI 29](#_Toc406410903)

[Hovedmenu 29](#_Toc406410904)

[Sensorer 30](#_Toc406410905)

[Ny sensorkonfiguration 30](#_Toc406410906)

[Fast Lane 30](#_Toc406410907)

[Receiver 30](#_Toc406410908)

[MidiModule 32](#_Toc406410909)

[Mapping Scheme 33](#_Toc406410910)

[ALSA 35](#_Toc406410911)

[Integrationstest 36](#_Toc406410912)

[Integrationstest af fast lane (Fra sensor til ekstern MIDI output) 36](#_Toc406410913)

[GUI 🡪 Controller 🡪 DataStorage 38](#_Toc406410914)

[Sensor 🡪 Receiver (Bluetooth) 38](#_Toc406410915)

[Receiver 🡪 MidiModule 38](#_Toc406410916)

[MidiModule 🡪 ALSA 38](#_Toc406410917)

[Udviklingsværktøjer 38](#_Toc406410918)

[PSoC Creator 38](#_Toc406410919)

[Atmel Studio 38](#_Toc406410920)

[Linux sampler 38](#_Toc406410921)

[Multisim 38](#_Toc406410922)

[Eagle CAD 38](#_Toc406410923)

[QT Creator 39](#_Toc406410924)

[Git 39](#_Toc406410925)

[Andre software biblioteker 39](#_Toc406410926)

[Resultater og diskussion 39](#_Toc406410927)

[Opnåede erfaringer 39](#_Toc406410928)

[Fælles 39](#_Toc406410929)

[Individuelt 39](#_Toc406410930)

[Jonas 39](#_Toc406410931)

[Kristoffer 40](#_Toc406410932)

[Jeppe 40](#_Toc406410933)

[Kristian 41](#_Toc406410934)

[Lukas 41](#_Toc406410935)

[Fremtidigt arbejde 41](#_Toc406410936)

[Konklusion 42](#_Toc406410937)

[Referencer 42](#_Toc406410938)

# Indledning

Denne rapport er skrevet på baggrund af et projektoplæg, som stiller visse krav til hvad projektet skal indeholde, men selve emnet er frit.

Dette projekt omhandler hvorledes sensorer, PSoC og Rasberri Pi kan benyttes til at opbygge et avanceret musikinstrument, som på sigt bl.a. kan skifte mellem flere lydpakker, etc.. Systemet skal bringe glæde og leg til musikken, og er blevet døbt ”BodyRock3000”.

Emnet for rapporten er blevet valgt på baggrund af et ønske om at se gruppens egne originale idéer blive realiseret fra bunden. Flere idéer blev overvejet, og BodyRock3000 blev valgt ud fra kriterier som:

* Integration af flere forskellige sensorer
* Mulighed for diverse software-databanke
* Brug af flere forskellige lydpakker
* Mulighed for at have en fungerende prototype ved slut

Opgaven udføres ved hjælp af de forskellige fag, som både 1., 2. og 3. semester på Ingeniørhøjskolen Aarhus Universitet har budt på, med særligt udgangspunkt i fagene på 3. semester. Først vil der blive udarbejdet en kravspecifikation, hvorefter gruppen arbejder med elementer af Scrum, hvor gruppemedlemmerne arbejder i iterationer.

# Opgaveformulering

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

Instrumentet skal kunne afspille lydsamples, og generere MIDI-tone- og CC-signaler på baggrund af data fra accelerometer-, gyroskop-, proksimitets- og taktile trykmålinger.

Der er stillet følgende krav i den udleverede opgaveformulering:

* Systemet *skal* via sensorer/aktuatorer interagere med omverdenen
* Systemet *skal* have brugerinteraktion
* Systemet *skal* indeholde faglige elementer fra semesterets andre fag
* Systemet *skal* anvende Devkit 8000 og PSoC-teknologi

Dette medfører at det færdige produkt skal kunne opfange:

* Accelerationer
* Tilt
* Afstande
* Taktile tryk

Og på baggrund af disse:

* Afspille samples
* Generere MIDI-signaler

Efter konsultation med, og tilladelse fra, vejleder, er det besluttet at udskifte Devkit 8000 med en Raspberry Pi model B+. Denne beslutning er taget på baggrund af problemer med implementering af ALSA-biblioteker, det eksterne MIDI-lydkort og Linux-sampleren på Devkit 8000.

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.

# Projektafgrænsning

Ingeniørhøjskolen Aarhus Universitet har opstillet følgende obligatoriske krav til 3. semesterprojekt:

* *Systemet skal via sensorer/aktuatorer interagere med omverdenen*
* *Systemet skal have brugerinteraktion*
* *Systemet skal indeholde faglige elementer fra semesterets andre fag*
* *Systemet skal anvende Devkit 8000- og PSoC-teknologi.(REFERENCE – CITAT)*

Ud fra kravene udvikles et elektronisk musikinstrument, der bruger sensorteknologi til at frembringe lyd. For yderlige uddybning, se projektbeskrivelsen (REFERENCE). Det skal nævnes at der for gruppe 9 er givet dispensation til at benytte en Raspberry Pi B+(REFERENCE) i stedet for Devkit 8000.

Projektet afgrænses til en at bestå af en prototype. Prototypen overholder kravene fra kravspecifikationen(REFERENCE) med undtagelse af:

* Antal sensorer er begrænset til én sensor af typen accelerometer
* Body vil ikke være forsynet fra ekstern spændingsforsyning
* Hverken Body- eller Rock-enheden indeholder ”Preset” funktionalitet(REFERENCE)
* Det vil ikke være muligt gøre Rock-enheden lydløs
* Det vil ikke være muligt at tilføje andre lydpakker end standard-lydpakken

# Systembeskrivelse

BodyRock3000



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

Instrumentet skal kunne afspille lyd-samples og generere MIDI tone- og CC-signaler, på baggrund af data fra accelerometer-, gyroskop-, proksimitets- og taktile trykmålinger.

Dette medfører at det færdige produkt skal kunne opfange:

* Accelerationer
* Tilt
* Afstande
* Taktile tryk

Og på baggrund af disse:

* Afspille samples
* Generere MIDI-signaler

Systemet består af to dele, der kommunikerer trådløst:

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

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

Systemets fleksibilitet og mulighed 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.

# Krav

Ud fra opgaveformuleringen, er der udarbejdet en række use cases, som beskriver aktørernes interaktion med systemet. Disse use cases fungerer som kravspecifikation, og bruges i den tidlige del af udviklingsfasen, til at bestemme systemets funktionalitet. For at se de fully dressed use cases, henvises til projektdokumentationen.[[1]](#footnote-1)

## Aktørbeskrivelse

På use case-diagrammet på **figur XX** ses en række aktører. Disse er beskrevet i følgende aktørbeskrivelse:

**Figur 1** Use case-diagram over BodyRock3000

**Bruger** er en primær aktør, som ønsker at benytte systemet BodyRock3000, ved at indstille diverse konfigurationer.[[2]](#footnote-2)

**MIDI-modtager** er en sekundær aktør, som kan transformere de generede MIDI-signaler.[[3]](#footnote-3)

**Højtalersystem** er en sekundær aktør, som afspiller den ønskede lyd.[[4]](#footnote-4)

## Use case-beskrivelse

Use cases fra use case-diagrammet er beskrevet i følgende afsnit. Hver use case beskriver et scenarie, hvor en aktør interagerer med systemet.

### Forbind Body og Rock

Brugeren tænder for hhv. Body og Rock, og benytter herefter hovedmenuen til at forbinde Body til Rock.[[5]](#footnote-5)

### Installér lydpakker

Brugeren benytter Rocks hovedmenu til at importere og installere en lydpakke.[[6]](#footnote-6)

### Konfigurer sensorer

Brugeren benytter Rocks hovedmenu til at oprette en ny sensorkonfiguration.[[7]](#footnote-7)

### Konfigurer presets

Brugeren benytter Rocks hovedmenu til at oprette en ny preset-konfiguration.[[8]](#footnote-8)

### Vælg preset

Brugeren vælger preset, ved at trykke på knapmatricen placeret på Body.[[9]](#footnote-9)

### Indsaml sensordata

Sensor genererer rådata, hvilket sendes trådløst til Rock, hvor det gemmes i en buffer.[[10]](#footnote-10)

### Generér MIDI

Læser data fra buffer og omdanner det til MIDI-signaler.[[11]](#footnote-11)

### Afspil lyd

Der bliver afspillet den lyd, som er genereret af BodyRock3000.[[12]](#footnote-12)

# Projektgennemførelse

## Udviklingsmodel

Projektet er gennemført med en modificeret udgave af *ASE*-modellen(REFERENCE) som den primære udviklingsmodel. ASE-modellen er som udgangspunkt en vandfaldsmodel. Man arbejder sig systematisk frem med at lave opgaveformulering, kravspecifikation og systemarkitektur, for så at designe og implementere modulerne i hardware og software hver for sig og til sidst laver man en samlet accepttest.



**Figur 2** ASE-proces

I dette projekt er en ren vandfaldsmodel forsøgt undgået. Derfor er elementer fra *Verification* and *Validation-*modellen (V-modellen) indført. Alle trin i ASE-modellen er blevet parret med et tilsvarende afsluttende trin. Se **figur XX**.



**Figur 3** Verification and Validation-model

Kravspecifikationen er udarbejdet sammen med accepttesten, systemarkitekturen sammen integrationstestene, og til designfasen/implementeringen er modultesten til hver enhed lavet.

V-modellen lægger op til at hvert et trin i modellen afsluttes, før et nyt påbegyndes. Dette er det undervejs blevet valgt at ændre i projektgennemførelsen, da forudsætningen for at dette kan lade sig gøre, kræver at man har et overblik over hvilke teknologier, der skal bruges i projektet, og hvordan disse kan implementeres. V-modellen er derfor blevet brugt, for at sammenholde de forskellige trin, mens projektet, ved siden af, har haft iterative processer til hvert enkelt delsystem.

Typisk har disse processer bestået af teknologiundersøgelser og forsøg med at udvikle fungerende funktionelle udgaver af delsystemerne (REFERENCE til teknologiundersøgelserne). Iterationerne er blevet styret ved hjælp af Scrum-sprints, hvilket i de senere sprints har resulteret i at visse delsystemer har været færdige til fremvisning for kunden/vejlederen.

## Projektstyring

### Scrum

Fra starten af semesteret, blev det besluttet at projektet, for så vidt muligt, skulle styres vha. Scrum, som er en agil udviklingsmetode, hvilket var en af årsagerne til at en ren vandfaldsudviklingsmodel blev forsøgt undgået. Scrum har fordelene:

* Det er ikke nødvendigt at alle krav er kendte, før udviklingen påbegyndes
* Projektet opdeles i iterationer (sprints), hvilket øger overskueligheden
* Hvert sprint resulterer ideelt i et produkt, som kan demonstres for kunden/produktejeren
* Daglige møder (eller hver anden dag, for dette projekt) giver status for opgaver og eventuelle problemer
* Det er nemt og hurtigt at reallokere ressourcer til der, hvor der er mest brug for dem

Scrum er primært et softwareudviklingsværktøj, men er til dette projekt ligeledes benyttet til udvikling af hardware.

De benyttede nøglebegreber fra Scrum er:

* ’Stå op’-møde
* Gruppen har, mandag, onsdag og fredag, holdt stående møde, hvor hvert gruppemedlem har fortalt hvad personen har lavet sidst, skal lave som det næste, og hvilke eventuelle problemer, der skal imødekommes.
* Roller
* Gruppen består af 8 medlemmer, hvor et par af medlemmerne har delt rollen som *Scrum master*. Rolen har været opdelt således et gruppemedlem har fungeret som primær kontaktperson til *product owner*/vejleder, mens en anden har haft til opgave at holde ’stå op’-møderne på sporet, korte og relevante.
* Product Backlog
* *Backlog*’en består af alle opgaver, som skal laves før projektet er færdigt. I dette projekt, er den løbende blevet udfyldt i fællesskab. *Backlog*’ens opgaver deles op i *stories* når det laves. Når et sprint er blevet planlagt, er opgaverne lagt over på *task board*’et.
* Task Board og Sprint Backlog
* *Task board*’et har fungeret som organiseringsværktøj for listen over opgaver i det pågældende sprint. Opgaverne er blevet defineret med beskrivelser, prioritering, estimeret tid og uddelegering. *Task board*’et har også fungeret som kontrakt for hvilke opgaver, gruppen har valgt at forpligte sig til i sprintet.

Kort beskrivelse af hvert sprint

* Sprint 1
* Længde: En uge.
* Formål: At lave kravspecifikation, accepttest og organisering af projektet.
* Sprint 2
* Længde: Tre uger.
* Formål: At lave teknologiundersøgelse af forbindelser, sensorer, MIDI og GUI, samt at afslutte kravspecifikation og accepttest, og påbegynde systemarkitektur.
* Sprint 3
* Længde: To uger
* Formål: At lave de første implementeringer af delsystemer, samt videre arbejde på systemarkitektur.
* Sprint 4
* Længde: Tre uger.
* Formål: At få lavet en stor del af hardwareimplementeringen, få overblik over Rock-delen af systemet, forsætte med implementering af delsystemer, samt videre arbejde på systemarkitektur.
* Sprint 5
* Længde: To uger
* Formål: At få overblik over Body-delen af systemet, videreudvikle delsystemer, samt implementering af software på Rock-delen af systemet.
* Sprint 6
* Længde: En uge
* Formål: At få systemet til at generere lyd fra lydafgiveren på Rock, baseret på sensordata genereret på Body.
* Sprint 7
* Længde: To uger
* Formål: Færdiggørelse af projektrapport og –dokumentation, samt sammenkobling af alle færdige dele af systemet.

### Tidsplan

Den overordnede tidsplan (REFERENCE), der strækker sig igennem hele projektet, er blevet udarbejdet i Microsoft Projekt. Tidsplanen er blevet brugt til at holde overblik over tidsrammen for projektet, og er blevet opdateret efter hvert gruppe/vejledermøde.

### Mødestruktur

Gruppe- og vejledermøder er blevet styret ved hjælp af en mødeindkaldelse, efterfulgt af et møde med dagsorden, dirigent og referent. De administrative roller er blevet fastlagt vha. en turnusordning (REFERENCE TIL TURNUSORDNING), hvor de forskellige roller som mødeindkalder, referent og dirigent skifter fra møde til møde. Dette er gjort for at sikre at alle gruppemedlemmer får et indblik i det administrative arbejde. For at sikre konsensus i dokumenter, er der udarbejdet skabeloner til mødeindkaldelser og referater. Møderne er blevet afholdt efter behov, med udgangspunkt i et møde ved et sprints begyndelse og afslutning. Referatet fra forrige møde er blevet gennemgået og godkendt ved hvert møde.

## Dokumentorganisering og log

Gruppen har benyttet Git til organisering af dokumenter, kode og generel dokumentation for projektet. Git består af et repository, som kan rumme alle disse data. Det har den fordel, at alt hvad sendes til repository’et skal have en medfølgende beskrivelse, og bliver stemplet med både dato/tidspunkt og præcise informationer om hvilke data, der bliver sendt med. Git’en har derfor også fungeret som gruppens fælles log for projektet. Gits funktion med at *merge* dokumenter er kun blevet brugt til dels, grundet tekniske udfordringer med denne funktion.

(REFERENCE TIL GITloggen, Github)

# Metoder

## SysML

Til formidling af kravspecifikation og systemarkitektur, har projektgruppen valgt at anvende SysML (Systems Modeling Language). Dette er valgt, for at formidle systemet bedst og bredest muligt, da SysML er industristandard, og simpel og intuitiv at gå til for omverdenen.

SysML udspringer af UML (Unified Modeling Language), men er ulig UML, der centrerer sig om udvikling af SW, tiltænkt hele systemer (både SW og HW).

De af gruppen anvendte SysML-diagrammer, kategoriserer sig i grupperne ”structure” og ”behavior” (adfærd).



**Figur 4** Strukturen i SysML

### Stucture Diagram

**Block Definition Diagram (BDD)** er benyttet til at vise nedbrydningen af systemet i delelementer (blokke).

**Internal Block Diagram (IBD)** er benyttet til at vise grænseflader mellem systemets blokke.

### Behavior Diagram

**Use Case Diagram (UC)** er benyttet til at vise systemets krav til funktionalitet, på baggrund af hvordan det benyttes af eksterne aktører til at opnå nogle mål.

**Sequence Diagram (SD)** og **State Machine Diagram (STM)** er benyttet til at beskrivesystemets logiske funktionalitet.

### Applikationsmodel

Der er for enkelte, mere komplekse software-moduler gjort brug af applikationsmodeller. Disse tager udgangspunkt en udfærdiget domænemodel, og viser det pågældende moduls funktionalitet ved forskellige brugssituationer. Produktet af en applikationsmodel er et klassediagram, samt SD og STM, der lægger et solidt fundament for efterfølgende software-udvikling.

For videre beskrivelse af SysML, se <http://www.sysmlforum.com/>.

## Scrum

Følgende afsnit er en tilrettet version af det lignende afsnit i ”Projektgennemførelse” (det hører nemlig mere til i metoder).

Til projektstyring har gruppen har valgt at inddrage elementer fra den agile udviklingsmetode Scrum. Dette er gjort for at drage sig flg. fordele:

* **En hurtig igangsættelse af systemudvikling:** Det er ikke nødvendig at kende alle krav inden.
* **Overskuelighed:** Der vælges for hvert *sprint* et specifikt fokuspunkt.
* **Konstant værdiskabelse:** Hvert sprint ender ideelt ud i et produkt, som kan demonstreres for kunden*/*produktejeren.
* **Hyppig kommunikation:** Hver dag (hver anden dag i netop dette projekt) mødes gruppen til ’*stå op’-møde*, og får overblik over opgaver og eventuelle problemer.
* **Fleksibilitet:** Reallokerering af ressourcer kan ske hurtigt, som følge af hyppige *’stå op’-møder*.

### Nøglebegreber i Scrum

Se bilag: ”*The Scrum Guide.pdf*”.

### Afvigelser i brugen af Scrum i dette projekt

**Hardwareudvikling**: Selv om Scrum i klassisk forstand er et softwareudviklingsværktøj, har gruppen valgt også at benytte den til hardwareudvikling.

**Frekvens af *’stå op’-møder***: Grundet projektmedlemmernes sideløbende skolegang, har det ikke været muligt at arbejde nok til at retfærdiggøre daglige møder. I stedet har der i projektets begyndelse været to ugentlige møder, og mod projektets afslutning 3-4 ugentlige møder.

***Scrum Master*-rollen**: Denne har været fordelt på flere af gruppens medlemmer; en kontaktperson til product owner (i dette tilfælde projektvejleder), samt en opretholder af Scrums værdisæt og ’stå op’-mødernes fremgang. Sidstnævnte rolle blev, som projektet skred frem, mere kollektiv, idet gruppen fik en fælles forståelse for arbejdsmetoden Scrum.

### Scrum Sprints

**Tabel *XX*** viser en oversigt over gruppens Scrum-sprints og varigheden af disse:

|  |  |  |
| --- | --- | --- |
| Sprint # | Varighed | Mål |
| 1 | 1 uge | Påbegyndelse af kravspecifikation og accepttest  Projektorganisering |
| 2 | 3 uger | Teknologiundersøgelser af brugerønsker, forbindelser, sensorer, MIDI og GUI Færdiggørelse af kravspecifikation og accepttest.  Påbegyndelse af systemarkitektur |
| 3 | 2 uger | Videre arbejde med systemarkitektur  Første implementeringer af delsystemer |
| 4 | 3 uger | Fortsat implementering af delsystemer, herunder hardwareimplementering  Overordnet design af SW på Rock |
| 5 | 2 uger | Overordnet design af SW på Body  Udvikling af SW-moduler på Rock |
| 6 | 1 uge | Integration af moduler med slutmålet at generere lyd ud fra sensormåling |
| 7 | 2 uger | Færdiggørelse af projektdokumentation og -rapport  Videre arbejde med modulintegration |

**Tabel 1** Projektforløbets Scrum-sprints

# Specifikation og analyse

I dette afsnit vil vi komme med en kort beskrivelse over vores valgte løsninger. For yderligere begrundelse og analyse for hvorfor disse er valgt henvises til dokumentationen (REFERENCE)

Der er fortaget Teknologiundersøgelser for følgende teknologier:

## Sensortyper

## Bus-teknologier

### Ekstern instrument interface

Hvad angår interfacing mellem elektriske musikinstrumenter, synthesizere og DAWs (Digital Audio Workstation), findes én interface standard, der uhyre sjældent undlades: MIDI (Musical Instrument Digital Interface).MIDI blev i 1983 standardiseret af MIDI Manufacturers Association og har siden da været en ubestridt del af elektroniske musikinstrumenter.

**HW interface**

MIDI transporteres typisk over et 5-pin DIN stik, men kan også transporteres via bl.a. USB, FireWire og Ethernet. MIDI opkobles serielt i et link i stil med I2C og kan bære op til 16 kanalers information.

**MIDI indhold**

Informationer båret over en MIDI protokol kan bestå af:

* Node notation
* Tone
* Anslagskraft
* Vibrato
* Tempo
* Preset

Og meget mere.

**MIDI konkurrenter**

Af konkurrenter kan nævnes OSC (Open Sound Control), der har større overførselshastighed, men hvis beskeder dog er længere og mere komplekse, hvilket gør protokollen upraktisk for mindre enheder. OSC er desuden mindre vidtspedt end MIDI, hvilket gør det til en upraktisk løsning.

**MIDI i fremtiden**

Siden år 2005 har en ny protokol kaldt HD-procol eller HD-MIDI været under diskussion. Denne indeholder en række forbedringer såsom flere kanaler, større opløsning og større hastighed. HD-MIDI har fuld backward kompatibilitet, hvilket i fremtiden også gør MIDI 1.0 brugbart. Det er endnu uvist om industrien tager HD-MIDI til sig, og for netop vores system vil MIDI 1.0 derfor være det sikre valg.

Kilde: <http://en.wikipedia.org/wiki/MIDI>

## LydSampler

Til afvikling af lydsamples på baggrund af generede midisignaler blev det besluttet at anvende en tredjeparts applikation, da udviklingen af denne lå uden for dette projekts omfang. Udvalget af Linuxkompatible software samplere blev derfor undersøgt.

Valget faldt på programmet LinuxSampler[[13]](#footnote-13), da dette tilbød en stabil og letvægts engine, der er i stand til at køre både med og uden frontend, hvilket er attraktivt på en embedded platform med begrænsede resourser. Udover dette understøtter LinuxSampler også en række forskellige formater, derunder .sfz formatet, der ligeledes er mindre resoursekrævende end mange andre sampleformater.

Selvom LinuxSampler er i stend til at køre uden frontend, er det dog stadig praktisk at kunne indstille en konfiguration vha. en grafisk flade. Til dette anvendes LinuxSamplers egen letvægts frontend Qsampler. En, ikke ubetydelig, fordel ved Qsampler er, at konfigurationer gemt i dette program i realiteten er scripts i LinuxSamplers egen protokol LinuxSampler Command Protocol, LSCP. Dette gør, at en konfiguration kan indstilles og gemmes i Qsampler, og derefter åbnes som en LinuxSampler engine uden frontend.

## Trådløse teknologier:

Der var taget en beslutning om at kommunikationen mellem Body og Rock enhederne skulle foregå trådløst. Der blev taget højde for følgende krav i kravspecifikationen: Den trådløse forbindelse skulle kunne række minimum 10 meter. Ud fra dette blev forskellige hardwaremoduler undersøgt, og ud fra disse undersøgelser valgt to moduler til trådløs kommunikation.

For Body enheden blev valgt modulet: **Generic HC-05 Bluetooth RF Transceiver Module RS232**

For yderligere beskrivelse se dokumentationen (REFERENCE)

For Rock enheden blev valgt modulet: **RN42-I/RM Bluetooth 2.1 Module, -86dBm Receive Sensitivity, 4dBm Output**

For yderligere beskrivelse se dokumentationen (REFERENCE)

* **Grafisk user interface (GUI)**
* **Lydsamppler**
* **MIDI teknologi**

## Kunde undersøgelse

Eftersom projektgruppen ikke kun agerer udviklere, men også projektinitiativtagere og –indehavere, er der lavet en undersøgelse af ønskede features fra en musikers perspektiv. Undersøgelsen er foretaget af projektgruppemedlem Lukas Hedegaard, der selv er udøvende musiker og dermed har brugererfaring inden for feltet.

**Ønskede features**:

* Styring af toner (Note)
  + Retning: hhv. opadgående, hvor lav sensorværdi giver dyb tone, høj sensorværdi giver lys tone, samt nedadgående (modsat opadgående).
  + Skala valg, herunder kromatik, dur og mol.
  + Ændring af toneart
* Styring af anslagskraft (Velocity) med justerbar sensitivitet
* Styring af expression-parametre (Control Change) med hhv. absolut- og relativ ændring af parameter
* Mulighed for benyttelse af en enkelt sensor til justering af flere parametre
* Integration med eksisterende instrumenter/synthesizere over MIDI.
* System presets med let adgang, så systemets indstillinger kan skifte momentant, for at lette systemets brug i et live-set med flere på hinanden følgende sange med forskellige lyde.
* Intern samplebank med justerbare lyde (sekundært ønske).

Kundeundersøgelsen udmunder sig i kravspecifikationens afsnit ”Krav til Mapping Scheme” på s. 20 i projektdokumentatioen. (REFERENCE TJEK)

# Systemarkitektur

## Overordnet arkitektur

Den overordnede systemarkitektur ses på nedenstående domænemodel. Denne model giver et godt overblik over det samlede system og hvordan det agere med hinanden.



Figur

Systemet består bland andet af vores to ”hovedenheder” Body (PSoC4)(REFERENCE) og Rock (Raspberry Pi)(REFERENCE), disse to enheder er vores arbejdsmaskiner i systemet. Body modtagere data input fra vores sensorer, pakker dette og sender det via sit bluetooth-modul(REFERENCE) til Rock enhedens bluetooth-modul(REFERENCE). Rock udpakker så denne data og omdanner denne til et lydoutput. (MIDI/ALSA beksrivelse?????????????????????)(REFERENCE)

Brugeren har muligheden for at opsætte diverse konfigurationer via den grafiske brugeroverflade (GUI)(REFERENCE) på Rock enheden. På denne måde kan der konfigureres hvordan Rock enheden skal håndtere de data input den modtager.

# Design, implementering og test af HW

I det følgende beskrives det detaljerede hardwaredesign for projektet. Det detaljerede design tager udgangspunkt i kravsspecifikationen og systemarkitekturen. Designprocessen samt hvilke overvejeler og valg der er taget vil ligeledes blive beskrevet i dette afsnit.

## Indledende designovervejelser

I domain model BodyRock3000(REFERENCE TIL DOKUMENTATION), ses hvilke blokke det samlede system består af og hvilken funktionalitet de skal udfylde.

Systemet kan nedbrydes til to hardwareblokke en Body blok og en sensor blok.

Body blokken kan nedbrydes til følgende:



Figur : BDD for body

**Body** består af hardwareblokkene: Spændingsforsyning, Bluetooth-modul og Presetknapper. For yderlige beskrivelse af blokkene henvises til projektdokumentationen(REFERENCE TIL DOKUMENTATION).

## Overvejelser omkring sensorer

I projektformuleringen(REFERENCE TIL DOKUMENTATION) blev det fastlagt, at det endelige produkt skulle kunne opsamle data fra bevægelser herunder, accelerationer, tilt, afstande og taktil tryk.

Ud fra ovenstående blev det besluttet, at følgende sensortyper skulle anvendes:

* Accelerometer
* Gyroskop
* Proximity sensor
* Tryksensor

## Sensorer

Da alle sensorer kobles til I²C-bussen gennem et 4-polet RJ11-stik, kan et generisk IBD for en sensorenhed tegnes.

Af **figur XX** fremgår det at alle sensorenheder er koblet til den eksterne 3.3V spændingsforsyning med tilhørende GND, via to af polerne fra RJ11 stikket. Herudover er sensorerne koblet til henholdsvis SCL og SDA.

Det ses ligeledes heraf at sensorenheden er koblet til I²C to steder, hvilket giver mulighed for at serieforbinde flere sensorer.

**Figur 7** IBD for generisk sensorenhed

I det følgende afsnit beskrives de forskellige sensorenheder. Kun accelerometeret er fuldt beskrevet. For fuld beskrivelse af de øvrige sensorer, henvises til projektdokumentationen (REFERENCE TIL DOKUMENTATION).

### Accelerometer

Sensoren som benyttes i dette projekt, er et 3-akset accelerometer af typen **ADXL345**[[14]](#footnote-14). Denne model er *ultralow power*, den kan gå så lavt som til i *measure mode*, og kun i *standby mode*. Disse værdier er fundet i databladet[[15]](#footnote-15). ADXL345 understøtter i forvejen I²C[[16]](#footnote-16), og er derfor at foretrække, da der ikke skal tilføjes noget ekstra til enheden for at muliggøre benyttelsen af I²C.

Det fremgår desuden af databladet at ADXL345 opererer ved 2V til 3.6V, hvilket passer til den benyttede spændingsforsyning som leverer 3.3V[[17]](#footnote-17).

#### I²C

Når der skal oprettes forbindelse til en sensorenhed via I²C, er det vigtigt at kende komponentens I²C-adresse. ADXL345 har et ben kaldet **ALT ADDRESS**[[18]](#footnote-18), som bruges til at styre ADXL345’s to forskellige I²C-adresser.

|  |  |
| --- | --- |
| **I²C adresse (hex)** | **ALT ADDRESS PIN** |
| 0x1D | Koblet til VCC |
| 0x53 | Koblet til GND |

**Tabel 2** I²C adresser og kobling til ALT ADDRESS pin

Ud fra denne viden benyttes en 1x3 Harwin pin med tilhørende jumper, således at brugeren hurtig og nemt kan skifte mellem de to alternative I²C adresser.

For at se breakout board og forbindelser, henvises til projektdokumentationen (REFERENCE TIL DOKUMENTATION).

### Gyroskop

Gyroskopet har til formål at generere data på baggrund af *tilt*. For fuld beskrivelse af gyroskopets design og implementering, henvises til projektdokumentationen (REFERENCE TIL DOKUMENTATION).

### Proximity sensor

Proximity-sensoren har til formål at generere data på baggrund af afstand. For fuld beskrivelse af proximity-sensorens design og implementering, henvises til projektdokumentationen (REFERENCE TIL DOKUMENTATION).

### Tryksensor

Tryksensoren har til formål at generere data på baggrund af taktilt tryk. For fuld beskrivelse af tryksensorens design og implementering henvises til projektdokumentationen (REFERENCE TIL DOKUMENTATION).

### I²C-bus

### Trådløs kommunikation

#### Body – Trådløs modul

Afsendingen af sensordata fra Body til Rock sker via Bluetooth-modulet HC-05. Modulet er indstillet ved hjælp af AT kommandoer til en baud rate på 115200, 8 data bit, 1 stopbit, ingen paritet samt Slave mode. Modulet er koblet til Body Shield og har igennem dette TXD-pin koblet til P0[4] RXD og RXD-pin koblet til P0[5] TXD. Body Shield forsyner også HC-05 med 3.3V DC og GND. For yderligere forklaring se Projekt dokumentation(REFERENCE til Montering af HC-05)

#### Rock – Trådløs modul

Modtagelsen af sensordata på Rock fra Body behandles af Bluetooth-modulet RN-42. Modulet er indstillet vha. Command mode (REFERENCE til datablad bluetooth\_cr\_UG-v1.0r (RN 42) s. 16) til en baud rate på 115200, 8 data bit, 1 stopbit, ingen paritet samt Master Mode 4, Auto-Connet DTR Mode. Dette mode gør at RN-42 afsætter 6 bytes hukommelse til at gemme en MAC-adresse i. Ved at fører JP3(GPIO6) på Pmod BT2 høj vil modulet forsøge at auto-genetablere til denne MAC—adresse. Modulet er derfor programmeret med HC-05s(Bodys Bluetooth-moduls) adresse. Modulet er forbundet til Rock(Raspberry Pi B+) på følgende vis:

|  |  |
| --- | --- |
| RN-42 Pins  (REFERENCE) | Raspberry Pi B+ Pins (REFERENCE)[[19]](#footnote-19) |
| 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 re-connect |  |

## PSoC shield

## Spændingsforsyning

Body-delen af systemet BodyRock3000, opererer på spændingen 3,3V.

Til at forsyne systemet med 3,3V designes en spændingsforsyning, som består af et 9V batteri og en reguleringskreds.

### Reguleringskreds

Til reguleringskreds benyttes **LM317** (REFERENCE). LM317 som er en 3-terminals, justerbar regulator med et *output range* fra 1,2V til 25V. En typisk opsætning fremgår af **figur XX**.



**Figur 8** Typisk opsætning af LM317 (fra datablad LM317)

Modstanden **R2** justeres for at få den ønskede udgangsspænding. Størrelsen af R2 findes vha. følgende formel fra databladet:

Som det fremgår af ovenstående formel, er ikke afhængig af inputtet, hvilket er en fordel, da man i så fald kan bruge batterier med forskellig spændingsstørrelser. Det gælder dog at indgangsspændingen som minimum skal være 1,5V større end den ønskede output-spænding (REFERENCE TIL DATABLAD).

For beregning og simulering af reguleringskredsen, henvises til projektdokumentationen (REFERENCE TIL DOKUMENTATION).

### Batteri

Vælges efter det er bestemt hvorledes reguleringskredsen skal laves. Da reguleringskredsen kræver at inputspændingen som minimum skal være 1,5V større end den ønskede outputspænding, er et batteri på 9V blevet valgt.

# Design, implementering og test af SW

I det følgende afsnit beskrives det detaljerede softwaredesign for projektet. Designprocessen, samt hvilke overvejelser og valg der er blevet truffet, vil ligeledes blive beskrevet.

For uddybende beskrivelser af design og implementering af software, henvises til dokumentationen (REFERENCE)

Identifikation af ønskede softwaredele:

I prototypen af vores produkt er der kun implementeret vejen fra hovedmenuen og ned til oprettelse af en ny sensorkonfiguration

**Grafisk brugeroverflade (GUI) – Rock**, bestående af grafisk platform med følgende afgrænset indhold:

* Hovedmenu: Generelt overblik over de mulige konfigurationer  
   Herunder: Sensorer, Presets og Lydpakker
* Sensorer: Menuen for mulige konfigurationer af sensorer   
   Herunder: Ny sensorkonfiguration, Rediger sensorkonfiguration og Slet sensorkonfiguration
* Ny sensorkonfiguration: Muligheden for at konfigurere indstillingerne for en ny sensorkonfiguration
* messageHandler: Denne klasse håndtere beskederne mellem GUI klasserne og Controller klassen

**Controller klassen – Rock,**

kontrollerer trådkommunikation for systemet, fungere som bindeled mellem GUI’s messageHandler og Datastorage.

## Body

Implementering af Bodys styresystem er beskrevet vha. psudokode. For hele beskrivelsen af implementeringen se Projekt Dokumentation(REFERENCE til Design og implementering SW Body)

### 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.

### Sensorer

#### 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.

### 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).

#### initUART

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

#### 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 |

#### 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.

## Slow Lane

### Controller

### Databank

Databank har til formål at gemme og hente konfigurationer. Konfigurationerne kan være sensorkonfigurationer (REFERENCE), presets (REFERENCE) eller lydpakker (REFERENCE). Hvilke attributter, konfigurationerne indeholder, kan ses i referencerne.

Databank er implementeret som en klasse med tre underliggende klasser; sensorkonfigurationsbanken, presetbanken og samplebanken. Disse indeholder konfigurationerne nævnt tidligere.

Controller-klassen har en instans af Databank, således at den blot kalder funktioner i Databank, så den kan få de relevante data og sende dem videre til GUI.

For at gemme og indlæse konfigurationerne, benyttes *Boost*-biblioteket (REFERENCE). Boost indeholder funktioner til serialisere konfigurationerne til XML-filer. De samme filer omdannes tilbage til sensorkonfigurationer, presets og samples vha. *Boost*.

**Figur XX** viser en overordnet skitse for Databank.



**Figur 9** Skitse over Databank

For detaljeret beskrivelse af interaktion henvises til hhv. sekvensdiagrammet (REFERENCE) og klassediagrammet (REFERENCE) over Databank.

### GUI

### Hovedmenu

Hovedmenuen er implementeret som klassen *mainWindow*. Der oprettes en instans af denne klasse i *main*-funktionen. I *mainWindow*’s constructor oprettes den grafiske UI (brugergrænseflade) som danner rammerne for designet af diverse elementer i GUI’en, såsom knapper, dropdown-menuer og lignende.

Brugeren bliver i hovedmenuen præsenteret for følgende tre valgmuligheder:

1. Sensorer Danner en instans af *sensorWindow*
2. Lydpakker Danner en instans af *lydpakkeWindow*
3. Presets Danner en instans af *presetWindow*

Den grafiske UI for den tilhørende klasse dannes på baggrund af brugerens valg.

For yderligere beskrivelse af hovedmenuens funktionalitet, henvises til dokumentationen (REFERENCE).

### Sensorer

Efter brugeren har valgt menupunktet *Sensorer*, dannes der en instans af klassen *sensorWindow*, der har samme funktionalitet i constructoren som *mainWindow*.

Brugeren bliver i *Sensorkonfigurationer* (instansen af *sensorWindow*) præsenteret for følgende valgmuligheder:

1. Ny Sensorkonfiguration
2. Rediger Sensorkonfiguration
3. Slet Sensorkonfiguration

Den grafiske UI for den tilhørende klasse dannes igen på baggrund af brugerens valg, og der sendes en forespørgsel, gennem *Message Handler*-klassen, efter den nødvendige info. Da der tages udgangspunkt i prototypen, er det *Ny sensorkonfiguration*, der er implementeret.

### Ny sensorkonfiguration

Efter brugeren har valgt *Ny Sensorkonfiguration*, dannes der en instans af klassen *nySensorkonf*, der har samme funktionalitet i constructoren som de ovenstående klasser. Derudover sendes en forespørgsel, gennem *Message Handler*-klassen, efter den nødvendige information, der skal bruges til at oprette en ny sensorkonfiguration. Denne information er følgende:

1. Sensortype
2. Mapping scheme
3. Akse
4. Lydpakke

Sensortype og akse er ”hardcoded” i prototypen, da der ikke er implementeret mere end én type sensor, og der kun er tre mulige akser (x, y, z). De øvrige informationer hentes gennem *Message Handler*’s funktion *getSensorKonfInfo*, der henter en struct med lister af strings, med navne på de Mapping schemes og lydpakker, der ligger i *Data Storage* (klassen, der håndtere lagring af data). Derved dannes der et UI med mulighed for at vælge den information, som skal bruges til at oprette en ny sensorkonfiguration.

BILLEDE!!!!!!!!

Efter brugeren har valgt sine ønskede sensorkonfigurationer, er det muligt at gemme disse ved anvendelse af en *gem*-knap. Når denne anvendes, lagres de valgte information i *Data Storage* via *Message Handler*

Den nyoprettede sensorkonfiguration er nu klar til at blive benyttet af *MIDI Module*.

For yderligere information om GUI-klasserne, henvises til dokumentationen (REFERENCE).

## Fast Lane

### Receiver

Receiver-klassen har til opgave at modtage, sortere og videresende alle meddelelser fra Body, modtaget over Bluetooth.

Når et objekt af klassen oprettes, oprettes en forbindelse der læser på noden *ttyAMA0*, som håndterer UART-forbindelser på en Raspberry Pi. Denne forbindelse lukkes igen ved nedlæggelse af objektet.

**Figur XX** viser et sd-diagram over operationerne som udføres af klassen Receiver.



**Figur 10** Sekvensdiagram over operationerne udført af klassen Receiver

Receiver kan udføre to typer af operationer, som begge initieres af indkommende beskeder over bluetooth. Operationerne er indrammet i de to yderste **opt**-kasser, og initieres af indholdet af den først byte i forsendelsen, startbyten.

Operationen i **opt1** initieres af brugeren, når der på Body-enheden skiftes preset. Der modtages en startbyte, som indikerer at den følgende byte er det ønskede preset, og preset-valget sendes hernæst videre til Controller-klassen.

Operationen i **opt2** initieres for hver aktive sensor 50 gange i sekundet af Body-enheden, når outputtet fra den aktuelle sensor er blevet aflæst. Der modtages en startbyte, som indikerer at den følgende byte er sensorens ID, mens de efterfølgende tre bytes er sensor-outputtet i op til tre dimensioner. Hvis ID’et er større end sidst modtagede ID, tilføjes dataene til en datapakke. Er ID’et derimod mindre end eller lig med sidst modtagede ID, sendes datapakken til MIDImodule-klassen, og en ny datapakke påbegyndes med de medsendte data.

For funtionsbeskrivelser og UML-diagram, se projektdokumentation (REFERENCE).

### MidiModule

#### MidiModule control

#### SensorConfiguration

### Mapping Scheme

#### Design

*Mapping Scheme*-klassen har til funktion at løse to opgaver:

1. At lagre brugerindstillinger for hvorledes data fra en given sensor omdannes til et MIDI-signal
2. At syntetisere et MIDI-signal ud fra en given sensormåling

##### Klassediagram

**Figur XX** viser et klassediagram for *MappingScheme* og dets lagringsstrukturer[[20]](#footnote-20), samt for *MidiSignal*, som *MappingScheme* også benytter i sin *map()* –funktion. Bemærk at der, for klassediagrammet herunder, er udeladt *set*- og *get*-metoder.



**Figur 11** Klassediagram over Mapping Scheme

For funktionsbeskrivelser, se projektdokumentation s. XXX.

#### Implementering

Herunder foreligger et udsnit af beskrivelsen af implementeringsfremgangsmåden for klassens funktioner. For funktionerne *mapVelocity*, *mapCCAbs* og *mapCCRel*, se projektrapport s. XXX.

Se bilagene *MappingScheme.h* og *MappingScheme.c* for den endelige implementering af *Mapping Scheme*.

##### map

*map()* har til ansvar at kalde den underfunktion, der svarer til den mapping-parameter, brugeren har indstillet.

##### mapKey

**Figur** XX viser programflowet i *mapKey*



**Figur 12** Flowchart over programflowet i mapKey

**Tilrettelse af data:** Systemet kan generere toner fra oktaverne -2 til 9 = 12 oktaver. I hver oktav er der 12 toner. Dette giver mulighed for at generere i alt 120 forskellige toner (12 gange 12 er 144?). Derfor benyttes kun sensorData i intervallet 3-122.

**quantizeDiatonic:** Beskrives i det følgende afsnit. Dataforskydning muliggør korrekt kvantisering ved forskellige grundtoner.

**Sæt Midisignal:** For at muliggøre brug af flere *note*-inputs på samme polyfoniske MIDI-instrument, skal den forrige tone for sensoren slukkes, inden en ny igangsættes. Der tjekkes derfor først om tonen er forskellig fra den gamle, inden den sættes. Er den ny, sættes kommandoen til ”note off”. Dermed kræves to gennemløb med samme modtagne data før tonen ændres.

##### quantizeDiatonic

Herunder vises et udsnit af dokumentationen for *quantizeDiatonic*. Foruden nedenstående, forefindes en beskrivelse af mol-kvantiseringen i projektrapporten s. XXX.

Under ét, betegner man dur- og mol-skalaer som diatoniske skalaer – deraf navnet.

Når de modtagne data (som indeholder alle oktavens toner) skal kvantiseres, skiftes de toner, der ikke ligger i skalaen op eller ned til en tone i skalaen. Kvantiseringen er designet således, at de mest brugte toner favoriseres: Grundtone (1. trin), terts (3. trin) og kvint (5. trin), herunder markeret som hhv. 1, 3 og 5.

Dur-kvantiseringen foregår som på illustrationen af en oktav på **figur XX**.



**Figur 13** Ilustration af en oktav

Påskrevet er dur-skalaens trin (1-7) for en c-dur-skala, samt hvilken vej det ønskes at tonerne uden for skala skal kvantiseres til (røde pile).

##### Test

Test af *Mapping Scheme* er for de fleste funktioner testet for et begrænset, men nøje udvalgt sæt input-data. De er som udført af designer, med udgangspunkt i en *white box*-tilgang, hvor alle signalveje testes, om end ikke alle mulige udfald afprøves; dette ville for funktionen *mapKey* være *12 grundtoner x 3 skalaer x 2 retninger x 127 mulige datainput = 9144* muligheder for testens udfald. Netop denne funktion er således testet for et begrænset antal inputs for en specifik grundtone-skala-kombination. Der henvises til endelig integrationstest for auditiv validering af signalvej, for forskellige grundtoner og skalaer.

Alle tests er at finde i projektdokumentationen s. XXX.

#### AlsaAdapter

### ALSA

# Integrationstest

## 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 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.

## GUI 🡪 Controller 🡪 DataStorage

## Sensor 🡪 Receiver (Bluetooth)

## Receiver 🡪 MidiModule

## MidiModule 🡪 ALSA

# Udviklingsværktøjer

Herunder gives en kort beskrivelse af relevante udviklingsværktøjer benyttet i projektet.

## PSoC Creator

Givet at brugen af PSoC var et projekt krav, har Cypress’ PSoC Creator været væsentlig for projektet. Demme IDE tillader udvikling af hele systemer til PSoC chippen fra et sammenhængende miljø. Dette dækker over hardware mapping af de digitale og analoge subsystemer, automatisk generering af software API ‘er til indbyggede komponenter samt udvikling af programkode.

## Atmel Studio

Til at forsyne vores sensor kredsløb med I²C interfaces er der blevet benyttet to ATtiny microcontrollere, til hvilke programmerne er blevet skrevet i Atmel Studio. Dette tillader at programmerne kan bygges med de relevante Atmel biblioteker og passende compiler alene ved at fortælle IDE’en hvilken chip man skriver til. Derudover har AS også et værktøj til at programmere chippen gennem vores STK500 board.

## Linux sampler

For at kunne omsætte de genererede midi signaler til lyd skal der benyttes en midi sampler. Til vores projekt har vi valgt at benytte Linux sampler, da denne er en letvægts sampler lavet til at køre på Linux. Dette gjorde den ideel til vores projekt da dette benytter sig af en indlejret Linux platform med begrænsede system ressourcer.

## Multisim

Til kredsløbne for afstands- og tryksensoren benyttes der en række hardware komponenter da disse er adskilt fra PSoC ’en af en I²C bus. Disse er simuleret i Multisim før implementeringen for at forhåndsteste designet.

## Eagle CAD

Til PCB layout er der i projektet blevet benyttet Eagle. Dette er gratis at bruge til nonkommercielle formål og har rigeligt med features til dette projekt. Programmet tillader udarbejdelse af skematiske diagrammer, samt print layout med automatisk kontrol af overensstemmelse med skematisk design.

## QT Creator

Til udvikling af den grafiske brugerflade på Rock enheden blev det valgt at benytte QT biblioteket, da dette er forholdsvis udbredt til GUI udvikling på indlejrede Linux systemer. QT Creator er et IDE der tillader at man kan designe sin QT brugerflade gennem et grafisk værktøj, hvorefter koden til dette genereres automatisk.

## Git

Til organisation af filer, både kode og dokumentation, er der blevet benyttet et Git repository fra GitHub. Dette giver et system til fildeling hvor der samtidig tages højde for filhistorik. Da dokumentationen ikke er skrevet i LaTeX eller lign. er der ikke blevet draget nytte af Gits potentiale ift. at tillade flere at arbejde på den samme fil på samme tid. Dette fungerer kun i plaintext formater, og ikke i formater som f.eks. .docx.

## Andre software biblioteker

Til udviklingen af softwaren på Rock enheden er der gjort stor nytte af C++’s Standard Template Library, specielt til brogrammets datastrukturer. Da programmet også har behov for at kunne gemme data ved system genstart er der ud over standard bibliotekerne også benyttet Boost’s serialisations bibliotek til lagring af objekter som XML kode.

# Resultater og diskussion

Beskrivelse af projektets resultater i kort form bl.a. ved anvendelse af tabeller, grafer eller billeder. Det er vigtigt, at man her klart og nøgternt præsenterer sine resultater. Det er vigtigt at udpege og diskutere relevante dele af de opnåede resultaterne og deres betydning. Bl.a. en samlet vurdering af resultaterne i lyset af problemstillingen og formålet med – eller hypotesen for projektet. Der må også gerne være en beskrivelse af de dele af projekt man er specielt stolt af.

Hvad er lykkedes hvad er ikke?

Fælles.

# Opnåede erfaringer

## Fælles

## Individuelt

### Jonas

Dette projekt har for mig været et af de sværere. Dette skyldes at jeg har haft svært ved meget af undervisningen, og har derfor haft vanskeligheder med at overføre det faglige stof til projektet. Derfor har jeg været meget afhængig af at have en sparringspartner, når jeg skulle arbejde.

Projektet som helhed, tror jeg at vi har fået op i en lidt for ambitiøs størrelse, hvilket også har medført at vi har haft svært ved at færdigøre en virkende prototype.

Selv om det har været en svær del at komme igennem, føler jeg stadig jeg har fået noget lærerigt ud af det. Jeg har anvendt udviklingsværktøjet QT, og lært en masse om dette program.

Jeg har også kunnet mærke, at struktureringen omkring rapport og dokumentation ikke har været på samme niveau som sidste semester, hvor dette netop også var fokuspunktet. Dette har været lidt forvirrende.

Vi har anvendt Scrum til den daglige strukturering af opgaver og møder, dette har været lærerigt og en god arbejdsmetode.

Personligt synes jeg det har været knap så godt et projekt fra min side af.

### Kristoffer

Jeg synes at det har været et spændende projekt, hvor jeg i særdeles har lært meget om Bluetooth-håndtering, datapakning og trådkommunikation, ligesom jeg føler at jeg er blevet langt mere komfortabel ved projektstyringsmetoden Scrum.

Jeg syntes i starten at det var et svært system at overskue, men efter at have arbejdet med det i flere måneder, hvor jeg har haft fingrene i bl.a. trådløs kommunikation, databehandling og -pakning, kommunikation på tværs af tråde og generel opsætning af et Linux-baseret system (Raspberry Pi’en), har jeg fået tilegnet mig vigtige erfaringer med at overskue et mere omfattende system, end jeg tidligere har arbejdet med. Jeg kunne også mærke at jeg havde mere og mere at bidrage med, i takt med at jeg begyndte at få overblik over systemet. Selv om det i nogle sprints viste sig, at jeg sad med lidt for mange højprioritets-opgaver på én gang, til at jeg kunne fordybe mig ordentligt, har bidraget til at jeg har lært, hvad jeg bør påtage mig på én gang (og hvad jeg bør lade andre påtage sig, da jeg ikke var den eneste som oplevede dette).

Ydermere var det rigtig dejligt med fleksibiliteten fra vejlederens side, da vi var nødt til at skifte devkit8000 ud med en Raspberry Pi, ikke mindste fordi jeg også meget gerne ville lære at rode med sådan en. Alt i alt har det været rigtig fedt, denne gang at få mere eller mindre frie tøjler, til hvad vores projekt har skullet handle om, og jeg synes at har været det projekt, jeg har haft det sjovest med at arbejde på.

### Jeppe

For mig, har den store udfordring i dette projekt været at tilpasse en skoleuge sideløbende med projektudviklingsmetoden Scrum.

Scrum har på mange måder været udbytterig for mig. Identificeringen og uddelegering af arbejdsopgaver, samt at hvert sprint havde et endeligt mål, effektiviserede samarbejdet, og hjalp mig til at holde styr prioriteringen af SW- og HW-opgaverne.

Der hvor det haltede for mig, var med fordybelsen i opgaverne. Sprints på to til tre uger, hvor skolearbejde var en del af hverdagen, gjorde at projektopgaverne ofte føltes forvirrende og til tider frustrerende, da man simpelthen havde mistet fokusset i udviklingsprocessen af projektet. Man skulle, så at sige, sætte sig ind i konteksten af opgaverne forfra hver gang.

Min konklusion er, at jeg skal have lavet et personligt værktøj, der hurtigt og effektivt kan sætte mig ind i en udviklingsopgaves kontekst. Dette kunne være en metode, hvor man på få linjer skitserede en analyse, fortolkning og perspektivering af opgaven som man læste hver gang før man begyndte, samt at holde en mere udførlig, personlig log, som holdte styr på hvor lang man var kommet i udviklingsprocessen.

Alt i alt har dette udviklingsprojekt givet mig endnu et ingeniørfagligt løft, som jeg vil bruge i mit fremtidige arbejde som ingeniør.

### Kristian

3. semesterprojektet har givet mig større selvstændighed som ingeniør, og har udviklet mine evner inden for tilegnelse af viden. Dette er opnået, da vi har udtaget store elementer fra Scrum, og integreret dem i vores projekt. Med dette har vi haft mulighed for at identificere og uddelegere opgaver, hvilket har hjulpet mig med at holde styr på hvorhenne i projektet vi er, og hvad der mangler at blive lavet.

Jeg har primært brugt min viden fra semesterfagene I3GFV og E3MSE til at udfører mine opgaver.

Jeg har til tider taget lidt for mange opgaver på mine skuldre, og jeg har fundet ud af at i stedet for at tage opgaverne selv, er der andre i gruppen som kan klare dem. I og med at vi har kørt Scrum, har jeg lært hvor afhængig man er af at folk klarer de opgaver som de bliver tildelt. I gruppen har vi til tider været dårlige til hurtigt at søge hjælp, når folk er i problemer med deres opgaver. I næste projekt vil jeg være bedre til at sige fra, så jeg ikke påtager mig for mange opgaver. Herudover vil jeg medtage mange af de gode elementer fra projektstyringen, og de ting som jeg har lært, i forbindelse med de opgaver, jeg har løst i løbet af projektet.

### Lukas

I projektgruppen aftaltes det fra start, at fokus for dette projekt skulle ligge i realiseringen frem for dokumentationen. Dette har for mig været en bittersød affære: At komme tidligt i gang med moduldesign har været motiverende og sjovt, men det har desværre også resulteret i at der ved senere integrationstests er opstået mange uforudsete problemer; for mit vedkommende i integrationen af *MappingScheme*, *SensorConfiguration* og ALSA-adapter i *MidiModule*. Set i bakspejlet, kunne disse problemer have været løst vha. to ting:

1. Mere præcis specifikation af modulære grænseflader på SW-siden: Tiden brugt på at lave en applikationsmodel fra start ville eksempelvis være givet godt ud.
2. Bedre testprocedure i forbindelse med godkendelse af et modul: At skrive testen først som kravspecifikation for modulet er en effektiv metode for kvalitetssikring.

Det har for projektet hovedsageligt været mit ansvar at udfærdige systemarkitekturen, og dette ikke før implementeringerne påbegyndtes, men sideløbende med. Dette har desværre resulteret i at gruppen ikke i optimal grad høstede fordelene af netop denne, men at mange resurser er blevet lagt i hen ad vejen at opfinde sammenhængen og tilrette allerede implementerede moduler.

Når dette så er sagt, har projektemnet for mig som musiker været utroligt spændende. Fagligt har jeg ikke i udpræget grad arbejdet med de på semestret tilegnede elektro-kernefagligheder, såsom sensor- og busteknologi, men derimod mere med programmering. Jeg føler heraf jeg har tilegnet mig ekstra viden i IKT-faglig retning og meget bedre forståelse for problematikkerne i udviklingen af større softwaresystemer.

### Lasse

Dette semesters projekt har budt på mange udfordringer og muligheder. På godt og ondt, har vi i gruppen, i langt højere grad end før, været ansvarlige for projektets udformning og indhold. Det, at emnet i dette semester er selvvalg har, i min optik, medvirket til en højere grad af entusiasme omkring projektet, da man føler, at projektet i større grad er ens eget. Processen med helt selv at skulle definere et produkt, der både er realistisk, ambitiøst og vigtigst af alt relevant og anvendeligt, var spændende. Man var nødt til, på tværs af gruppen, at forventningsafstemme og gå på kompromis, uden at nogen mistede følelsen af, at de kunne stå inde for produktet.

Selv om vi tidligt i forløbet besluttede ikke at følge en waterfall-udviklingsmodel, stod det halvvejs inde i projektet klart, at vi i høj grad havde fulgt den proces, som vi var blevet præsenteret for på sidste semester. Det var derfor først relativt sent i projektet, at vi rent faktisk fik ”hænderne beskidte”, og kom i gang med at implementere designet. Havde vi i højere grad implementeret og afprøvet sideløbende med designprocesserne havde vi givetvist undgået nogle af de uforudsete problemer, vi løb ind i, såsom problemet med at anvende devkit8000 sammen med linux sampler og ALSA. Vi blev dog klar over problemet, og ændrede vores fokus til at komme i gang med implementeringen før det var for sent.

Alt i alt har processen været positiv. Trods problemer, udfordringer og pressede tidsplaner undervejs, er det at se vores design udført i virkeligheden en fantastisk tilfredsstillende oplevelse, der gør det hele værd.

### Mathias

Projektet har givet mig erfaring inde for QT, og C++ generelt, da dette er det første semester hvor jeg anvender C++ frem for eksempelvis Java eller C#. Derfor har det været en udfordring at kaste sig ud i vanskelige opgaver, når man fra start af ikke kender godt til C++-syntaks. Dermed sagt har jeg også lært en del, da det er vigtigt at vide hvordan IDE’en (QT Creator), styresystemet (Linux/Rasparian) og sproget (C++) hænger sammen – og det finder man hurtigt ud af gennem nogle timers debugging. I projektet (og de fag, der ligger op af projektet) finder man hurtigt ud af at man må acceptere at ting ser vanskeligere ud end de er – så bliver man meget mere produktiv.

Jeg har ikke prøvet at køre Scrum så gennemført som vi gjorde under dette projekt, og det synes jeg egentligt fungerede meget godt. Jeg har følelsen af at der er blevet lagt flere kræfter i implementeringen end i systemarkitekturen. Det synes jeg er fint nok, for jeg har selv haft 3 tidligere projekter hvor der blev lagt for mange mandetimer i at lave systemarkitekturen perfekt. Jeg synes at det har været et meget ambitiøst projekt, men jeg tror at man lærer mere af at have høje forventninger og ikke nå alting, end det omvendte. Samlet set ville jeg ønske at jeg havde lavet mere, men jeg har svært ved at sige hvor jeg skulle have hjulpet til præcist. Derudover synes jeg at det var et ganske udmærket projekt, og jeg vil tage udviklingserfaringerne med videre.

# Fremtidigt arbejde

Første mål for fremtidigt arbejde vil være at tilfredsstille alle de i kravspecifikationen stillede krav. Disse er:

* Implementere preset-skift fra Rocks GUI
* Live preset-skift fra Body over systemets bluetooth-forbindelse
* Implementering af forskellige lydpakker i ALSA, og at være i stand til at skifte disse med presetvalg

Næste målsætning er en fuld implementering af MIDI med eksempelvis ”pitch bend” (mulighed for at ”bøje” tonerne) og ”patch change”, så det ved preset-skift også er muligt at ændre preset på eksterne MIDI-instrumenter.

En mulig implementering i fremtidige systemiterationer kunne endvidere være opkobling af flere Body-enheder per Rock-enhed, for at facilitetere brugen af systemet til sociale aktiviteter.

Derudover er der mulighed for udvidelse af justeringsmuligheder i Mapping Scheme:

* **Skalaer:** Heltone, pentaton, harmonisk og melodisk mol
* **Velocitetskurver:** Lineær, logaritmisk, eksponentiel, fuld

# Konklusion

Her gives en samlet konklusion på projektarbejdet. Hvad er lykkedes, hvad er evt. ikke lykkedes og årsagen til dette. Konklusionen skal gerne indeholde et klart budskab og forholde sig objektivt til de opstillede krav og opnået resultater.

Desuden sammenfattes de slutninger, der kan drages af de resultater, som er omtalt i rapportens tidligere afsnit. Konklusioner kan være såvel positive som negative. Man skal tage sig i agt for ikke at undertrykke de negative fund (hvis f.eks. en metode har vist sig uegnet, bør det opfattes som et bidrag til ens erfaringsmateriale, ikke som et personligt nederlag).

I konklusionen trækkes desuden de store linier op. Væsentlige kvantitative resultater kan nævnes, hvorimod den detaljerede redegørelse og argumentationen henvises til diskussionen i rapportens hoveddel.

# Referencer

Brug words funktion

Harward citation:

<http://guides.is.uwa.edu.au/harvard>

1. REFERENCE [↑](#footnote-ref-1)
2. REFERENCE [↑](#footnote-ref-2)
3. REFERENCE [↑](#footnote-ref-3)
4. REFERENCE [↑](#footnote-ref-4)
5. REFERENCE [↑](#footnote-ref-5)
6. REFERENCE [↑](#footnote-ref-6)
7. REFERENCE [↑](#footnote-ref-7)
8. REFERENCE [↑](#footnote-ref-8)
9. REFERENCE [↑](#footnote-ref-9)
10. REFERENCE [↑](#footnote-ref-10)
11. REFERENCE [↑](#footnote-ref-11)
12. REFERENCE [↑](#footnote-ref-12)
13. http://www.linuxsampler.org/about.html [↑](#footnote-ref-13)
14. REFERENCE [↑](#footnote-ref-14)
15. REFERENCE [↑](#footnote-ref-15)
16. REFERENCE [↑](#footnote-ref-16)
17. RERERENCE [↑](#footnote-ref-17)
18. REFERENCE [↑](#footnote-ref-18)
19. http://www.raspberrypi.org/forums/viewtopic.php?f=78&t=82397 [↑](#footnote-ref-19)
20. Som specificeret i afsnittet ”Krav til Mapping Scheme” [↑](#footnote-ref-20)