AARHUS UNIVERSITET
INGENIØRHØJSKOLEN

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

Resumé

Denne rapport beskriver et semesterprojekt for 3. semester på Ingeniørhøjskolen Aarhus. Gruppens otte medlemmer er elektro- og IKT-ingeniørstuderende.

Problemstilling

Produktet har til formål at skabe et digitalt, intuitivt og moderne musikinstrument der kan anvendes af professionelle musikanter samt nybegyndere.

Formål

Formålet med projektet er, ifølge introduktionsoplægget for 3. semesterprojekt:

- *Udarbejdelse af et abstract rettet mod eksterne folk om projektet.*
- *Implementering og test af et udviklingsprojekt med både HW og SW, der integrerer semesterets kurser.*
- *Definition af en kravspecifikation for projektet.*
- *Samarbejde i grupper med både HW og SW udviklerroller*
- *Arbejdsmetode orienteret mod at udvikle nye produkter baseret på HW og SW.*

Opstillede krav og valgte løsninger

Krav fra introduktionsoplægget:

- *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.*

Da Devkit 8000 ikke kunne bruges til lydarkitekturdelen af produktet blev det valgt at skifte til Raspberry Pi, hvilket vores vejleder godkendte.

Anvendte metoder

Under udarbejdelse af projektet blev der anvendt Scrum til overskueliggørelse af opgaver og møder. Der blev holdt stand-up møder tre gange om ugen. Sprint-planlægningsmøder og retrospektmøder blev holdt én gang hver 2.-3. uge. Dette gjorde det mindre nødvendigt at dele gruppen op i hhv. hardware- og software-gruppe.

Resultater

Der er implementeret en fuldt fungerende signalvej fra sensor til hhv. et midi output og et audio output. Endvidere er der opbygget en fungerende brugerflade, samt bagvedliggende datastrukturer med understøttelse af lagring til disk.

Abstract

This report describes a project made by eight 3rd semester students from the Engineering School of Aarhus. The group participants are electrical- and ICT-engineering students.

Problem summary

The idea behind the product is to design a digital, intuitive, and modern musical instrument which can be used by professional musicians as well as amateurs.

Goal

Our project goal is according to the introductory presentation:

- *Write an abstract directed towards external readers*
- *Implementing and testing a development project including hardware and software which integrates the courses of the current semester*
- *Defining a requirements specification for the project*
- *Cooperation in groups with both hardware and software developer-roles.*
- *Work methods meant to develop new products based on hardware and software.*

Requirements and selected solutions

Requirements from the introductory presentation:

- *The system must have sensors/actuators which interact with the environment*
- *The system must have user interaction*
- *The system must include academic elements from other courses of the semester*
- *The system must use a DevKit 8000 and PSoC technology*

Since we could not use a DevKit 8000 for the sound-part of the product it was chosen to switch our platform to Raspberry Pi, which our instructor accepted.

Applied methods

During project execution we used Scrum to manage tasks and meetings. Stand-up meetings were organized three times a week. Sprint-meetings and retrospect-meetings were held once every two to three weeks. By doing this it was not as vital to divide the group into hardware developers and software developers.

Results

A functioning signal pathway from sensor to either a MIDI or an audio output has been implemented. Furthermore, a working user interface has been implemented, as well as underlying data structures, which support storing the data on disk.

Indhold

Arbejdsfordeling

Felix Blix Everberg	• **Sensor –** HW • **Body** – HW • **MessageHandler** • **MidiMudule**
Jeppe Hofni	• **Trådløsforbindelse** – HW og SW på Body og Rock • **Sensor** – SW • **Body** – SW
Jonas Evers Nikolajsen	• **GUI** – SW (På Rock) • **MessageHandler**
Kristian Boye Jakobsen	• **Sensor** – HW • **Body** – HW og SW
Kristoffer Lerbæk Pedersen	• **Trådløskommunikation** – HW og SW på Rockdelen • **Datapakning –** Pakning og videresendelse af sensordata fra Bluetooth • **MessageHandler** – SW • **Controller –** Kommunikation mellem GUI og DataBank
Lasse Fisker	• **MidiMudule** - SW • **Alsa** - SW • **LinuxSampler** - SW
Lukas Hedegaard Jensen	• **MidiModule** - SW • **MappenScheme** - SW • **Sensorkonfiguration** - SW
Mathias Siig Nørregaard	• **GUI** - SW • **DataBank** - SW

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 Raspberry Pi kan benyttes til at opbygge et avanceret musikinstrument. 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
- 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

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[1]:

- *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.*

[1] Bilag 1

Ud fra kravene udvikles et elektronisk musikinstrument, der bruger sensorteknologi til at frembringe lyd. For yderlige uddybning, se projektbeskrivelsen[2]. Det skal nævnes at der for gruppe 9 er givet dispensation til at benytte en Raspberry Pi B+[3] i stedet for Devkit 8000.

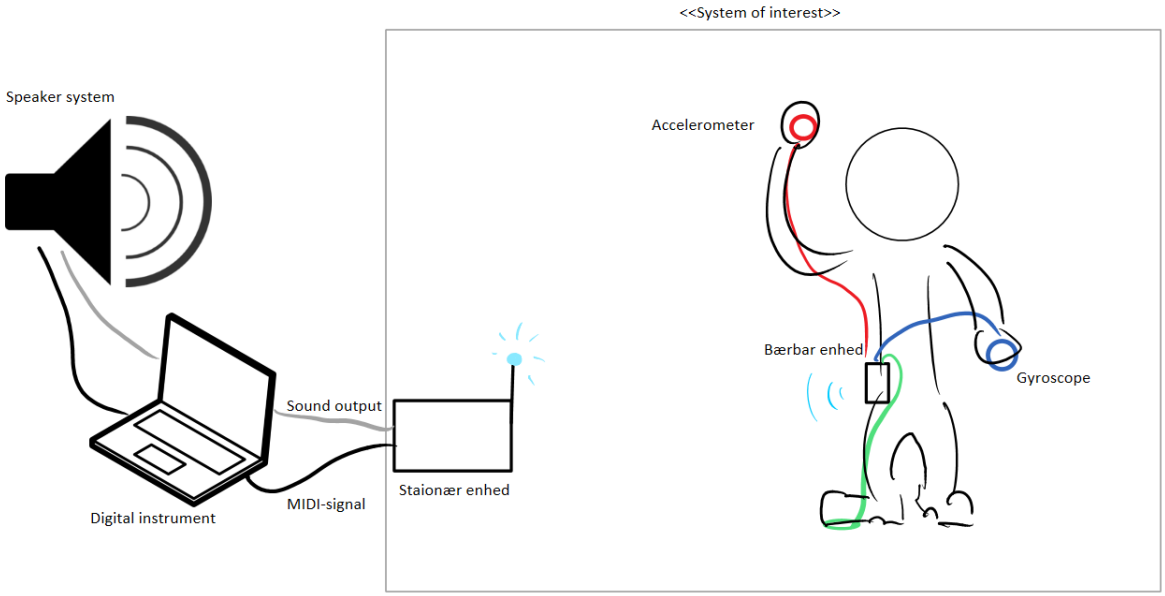
Projektet afgrænses til at bestå af en prototype. Prototypen overholder kravene fra kravspecifikationen[4] med undtagelse af:

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

[2] Projektdokumentation s. 6
[3] Bilag 2
[4] Projektdokumentation s. 8

Systembeskrivelse

BodyRock3000



Figur 1 Systemtegning

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.

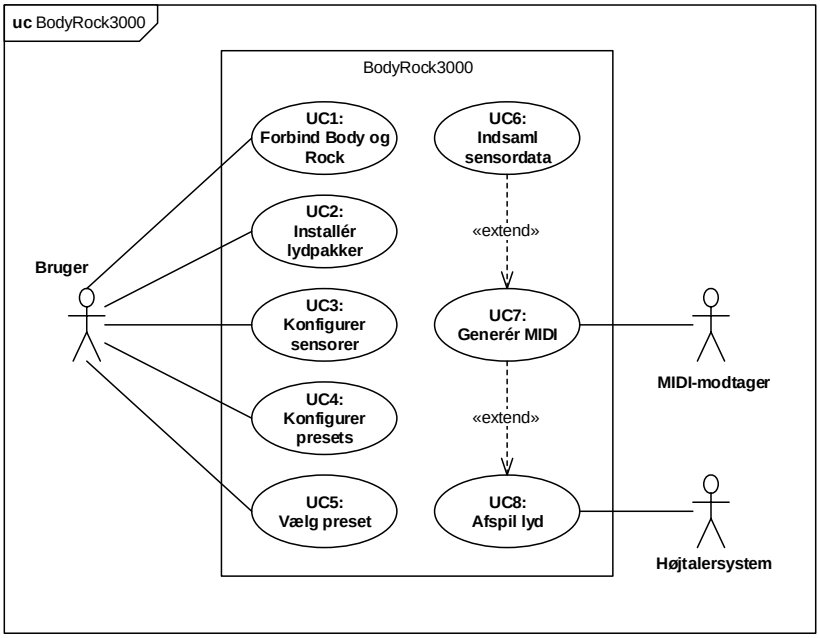
Systemets fleksibilitet og mulighed for udvidelse, samt den lette integration med eksisterende synthesizer- og sample-systemer, gør det til en kærkommen udvidelse af repertoiret for midler til musikalsk udtryk.

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 fully dressed use cases, henvises til projektdokumentationen.[5]

Aktørbeskrivelse

På use case-diagrammet på **Figur 2** ses en række aktører. Disse er beskrevet i følgende aktørbeskrivelse[6]:



***Figur 2** Use case-diagram over BodyRock3000*

Bruger er en primær aktør, som ønsker at benytte systemet BodyRock3000, ved at indstille diverse konfigurationer.

MIDI-modtager er en sekundær aktør, som kan transformere de generede MIDI-signaler til lyd.

Højtalersystem er en sekundær aktør, som afspiller den ønskede lyd.

[5] Projektdokumentation s. 8
[6] Projektdokumentation s. 9

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, hvorefter der automatisk oprettes forbindelse.

Installér lydpakker

Brugeren benytter Rocks hovedmenu til at importere og installere en lydpakke.

Konfigurer sensorer

Brugeren benytter Rocks hovedmenu til at oprette en ny sensorkonfiguration.

Konfigurer presets

Brugeren benytter Rocks hovedmenu til at oprette en ny preset-konfiguration.

Vælg preset

Brugeren vælger preset ved at trykke på knapmatricen placeret på Body.

Indsaml sensordata

Sensor genererer rådata, hvilket sendes trådløst til Rock, hvor det gemmes i en buffer.

Generér MIDI

Data læses fra bufferen og omdannes til MIDI-signaler.

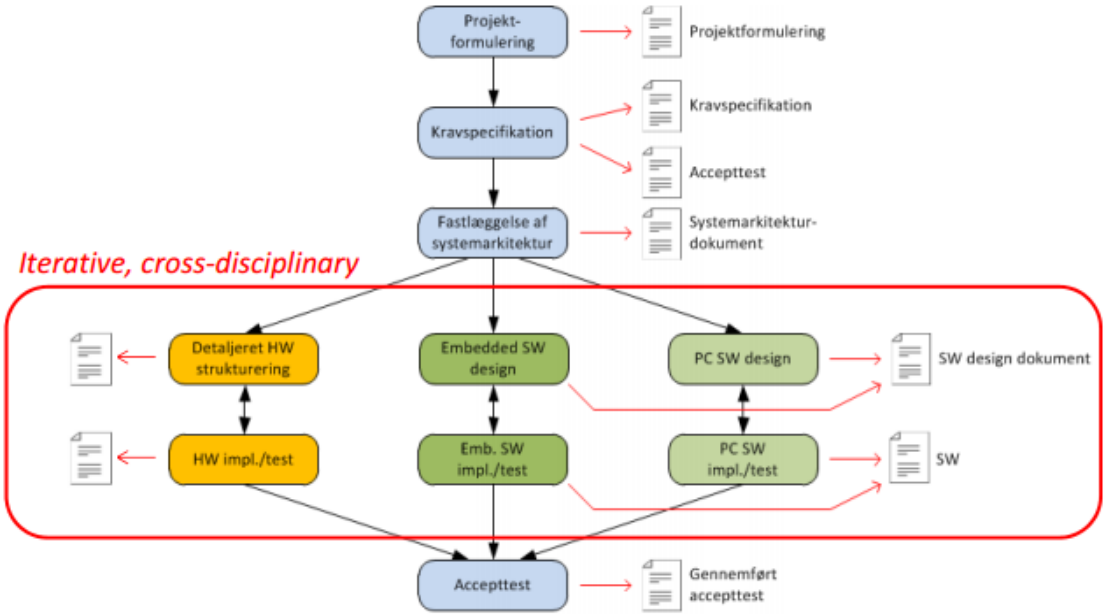
Afspil lyd

Det genererede MIDI-signal omdannes til lyd.

Projektgennemførelse

Udviklingsmodel

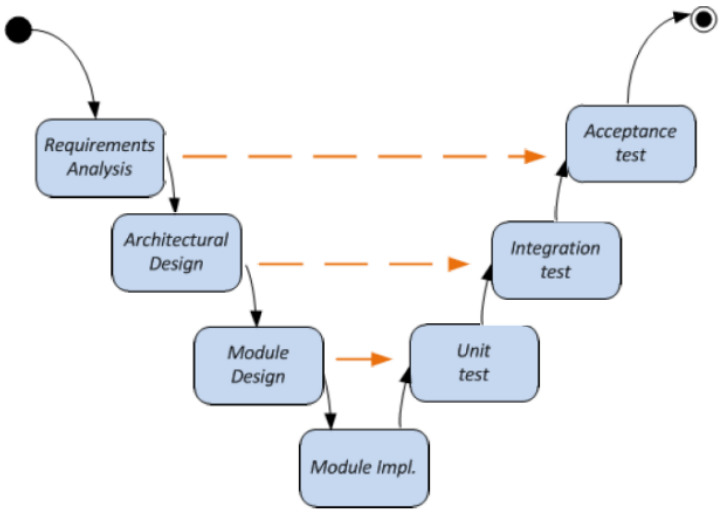
Projektet er gennemført med en modificeret udgave af *ASE*-modellen[7] som den primære udviklingsmodel. ASE-modellen er som udgangspunkt en vandfaldsmodel. Man arbejder sig systematisk frem med at lave opgaveformulering, kravspecifikation og systemarkitektur. Derefter designes og implementeres modulerne i hardware og software hver for sig og til sidst laver man en samlet accepttest.



***Figur 3** 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 4**.

[7] Bilag 35 s. 39



Figur 4 *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 sideløbende har haft iterative processer til hvert enkelt delsystem.

Iterationerne er blevet styret ved hjælp af Scrum, 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. 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 demonstreres for kunden/produktejeren
- Hyppige 'stå op'-møder 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 møder, 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. Rollen som *Scrum master* har været delt ud på flere medlemmer i projektgruppen. Rollen har været opdelt således, at 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[8]
  - *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 et sprint er blevet planlagt, overføres opgaverne til *Taskboardet*.
- Taskboard og Sprint Backlog[9]
  - *Taskboard*et har fungeret som organiseringsværktøj for listen over opgaver i det pågældende sprint. Opgaverne er blevet defineret med beskrivelser, prioritering, estimeret tid og uddelegering. *Taskboard*et har derudover fungeret som kontrakt for hvilke opgaver, gruppen har valgt at forpligte sig til i det pågældende sprint.

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.

[8] Bilag 33 (Alle tasks gennem projektet)
[9] Bilag 33

- Sprint 7
  - Længde: To uger
  - Formål: Færdiggørelse af projektrapport og –dokumentation, samt sammenkobling af alle færdige dele af systemet.

Tidsplan

Den overordnede tidsplan[10], der strækker sig igennem hele projektet, er blevet udarbejdet i *Microsoft Project*. Tidsplanen er blevet brugt til at holde overblik over tidsrammen for projektet, og er blevet opdateret efter hvert vejledermøde.

Mødestruktur[11]

Gruppe- og vejledermøder er blevet styret ved hjælp af en mødeindkaldelse[12], efterfulgt af et møde med dagsorden, dirigent og referent. De administrative roller er blevet fastlagt vha. en turnusordning[13], 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 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.

I dette projekt er GitHub blevet benyttet som git host[14].

[10] Bilag 34
[11] Bilag 32
[12] Bilag 31
[13] Bilag 36
[14] Bilag 37

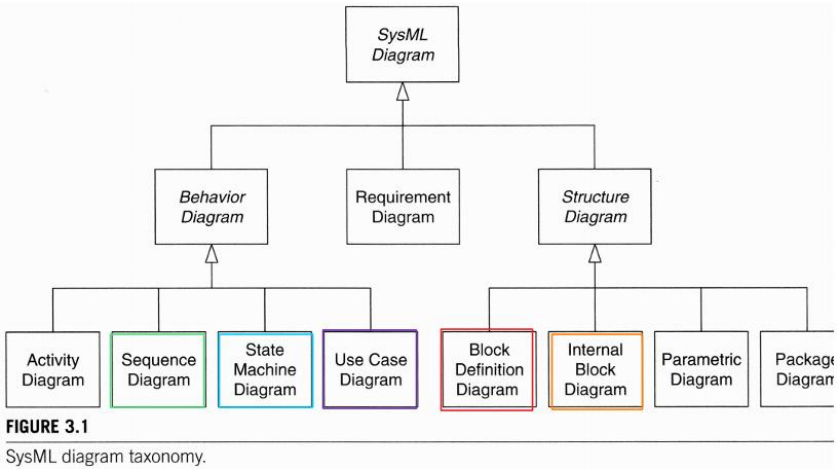
Metoder

SysML

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



Figur 5 Strukturen i SysML

Stucture Diagrammer

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

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

Behavior Diagrammer

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 veldefinerede mål.

Sequence Diagram (SD) og **State Machine Diagram (STM)** er benyttet til at beskrive systemets 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 eventuelt STM, der lægger et solidt fundament for efterfølgende software-udvikling.

For videre beskrivelse af SysML, se (PivotPoint Technology Corp. , u.d.).

Specifikation og analyse

I dette afsnit følger en kort beskrivelse over de valgte løsninger. For yderligere begrundelse og analyse omkring, hvorfor disse er valgt, henvises til projektdokumentationen[15].

Der er fortaget teknologiundersøgelser for følgende teknologier:

Sensortyper

Systemet BodyRock3000 anvender fire forskellige sensortyper. Sensortyperne er følgende:

1. Accelerometer
2. Gyroskop
3. Proximity-sensor
4. Tryksensor

Sensorerne står for at generere rådata, som behandles og sendes, og i sidste ende benyttes til at generere MIDI-lyd.

Bus-teknologier

Da BodyRock3000 skal kunne konfigureres så frit som muligt efter brugerens ønsker, er det valgt at systemets sensorer er tilsluttet gennem en *bus*. Målet med dette er at sensorer kan forbindes i en hvilken som helst konfiguration.

Protokoller

Det er valgt at benytte I²C. For begrundelse af dette samt undersøgelse af flere protokoller se projektdokumentationen[16].

Kabeltype

Til at stå for selve overførslen af sensordata gennem I²C, benyttes i dette projekt *RJ11-kabler* som forbindes til sensorprints via *RJ12- connectors*. Dette er valgt, idet RJ11-kabler er let tilgængelige, består af det nødvendige antal ledere, og samtidig har låsemekanisme i stikkene. Sidstnævnte giver mulighed for at ledningerne bliver siddende, selv når brugeren udfører diverse bevægelser for at generere data fra sensorerne.

Ved at have to connectors på hver sensorenhed, samt fire connectors på Body, kan sensorerne kobles i fire kæder ud fra Body. Dette tillader brugeren at minimere ledningsproblemer ved brug af flere sensorer, og gør at sensorerne kun kan forbindes korrekt idet der er tale om en *bus*.

GUI

Til implementering af en grafisk brugerflade på Rock, er det besluttet at anvende applikations-frameworket QT.

QT er cross-platform, og kræver ikke specielt mange ressourcer af systemet, hvorpå det køres. QT er derudover blevet anvendt i et andet fag på semesteret, hvilket har gjort valget yderligere oplagt, da gruppen således har kunnet trække på ressourcer fra dette fag til implementeringen.

Til design af QT-applikationen, er det tilhørende program *QT Creator* blevet benyttet.

[15] Projektdokumentation fra s. 54
[16] Projektdokumentation s. 54

Eksternt instrument-interface

Hvad angår interfacing mellem elektriske musikinstrumenter, synthesizere og DAW's (Digital Audio Workstation), findes der 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-pins DIN-stik, men kan også transporteres via bl.a. USB, FireWire og Ethernet. MIDI opkobles serielt i et link i stil med I^2C og kan bære op til 16 kanalers information.

MIDI-indhold

Informationer sendt via MIDI kan bl.a. bestå af følgende:

- Node-notation
- Tone
- Anslagskraft
- Vibrato
- Tempo
- Preset

Alternativer til MIDI

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

MIDI i fremtiden

Siden år 2005 har en ny protokol, kaldet HD-protocol (MIDI Manufacturers Association, 2014), eller HD-MIDI, været under diskussion. Denne indeholder en række forbedringer, såsom flere kanaler, større opløsning og større hastighed. HD-MIDI har fuld bagudrettet kompatibilitet, hvilket i fremtiden også gør MIDI 1.0 brugbart. Det er endnu uvist om industrien tager HD-MIDI til sig, og for dette system vil MIDI 1.0 derfor være det sikre valg.

LydSampler

Til afvikling af lydsamples, på baggrund af genererede MIDI-signaler, er det besluttet at anvende en tredjeparts-applikation, da udviklingen af denne lå uden for dette projekts omfang. Udvalget af Linux-kompatible softwaresamplere er derfor blevet undersøgt.

Valget er faldet på programmet LinuxSampler (LinuxSampler, u.d.), da dette tilbyder en stabil og letvægts engine, der er i stand til at køre både med og uden front end, hvilket er attraktivt på en embedded platform med begrænsede ressourcer. Ud over dette understøtter LinuxSampler også en række forskellige formater, herunder .sfz-formatet, der ligeledes er mindre ressourcekrævende end mange andre sampleformater.

Selv om LinuxSampler er i stand til at køre uden front end, er det stadig praktisk at kunne indstille en konfiguration vha. en grafisk flade. Til dette anvendes LinuxSamplers egen letvægts front end, *Qsampler*. En betydelig 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 front end.

Trådløse teknologier:

Det er besluttet at kommunikationen mellem Body og Rock foregår trådløst. Der er taget højde for følgende krav i kravspecifikationen: *Den trådløse forbindelse skulle kunne række minimum 10 meter*. Ud fra dette er forskellige hardwaremoduler blevet undersøgt, og ud fra disse undersøgelser er to moduler valgt til trådløs kommunikation.

Body: **Generic HC-05 Bluetooth RF Transceiver Module RS232**

For yderligere information, henvises til projektdokumentationen[17].

Rock: **RN42-I/RM Bluetooth 2.1 Module, -86dBm Receive Sensitivity, 4dBm Output**

For yderligere information, henvises til projektdokumentationen[18].

Kundeundersøgelse

Idet 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 høj tone, og nedadgående (modsat opadgående)
    - Valg af skala, 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 uden betydelig forsinkelse, for at lette systemets brug i et live-set med flere, på hinanden følgende, numre med forskellige lyde
- Intern samplebank med justerbare lyde (sekundært ønske)

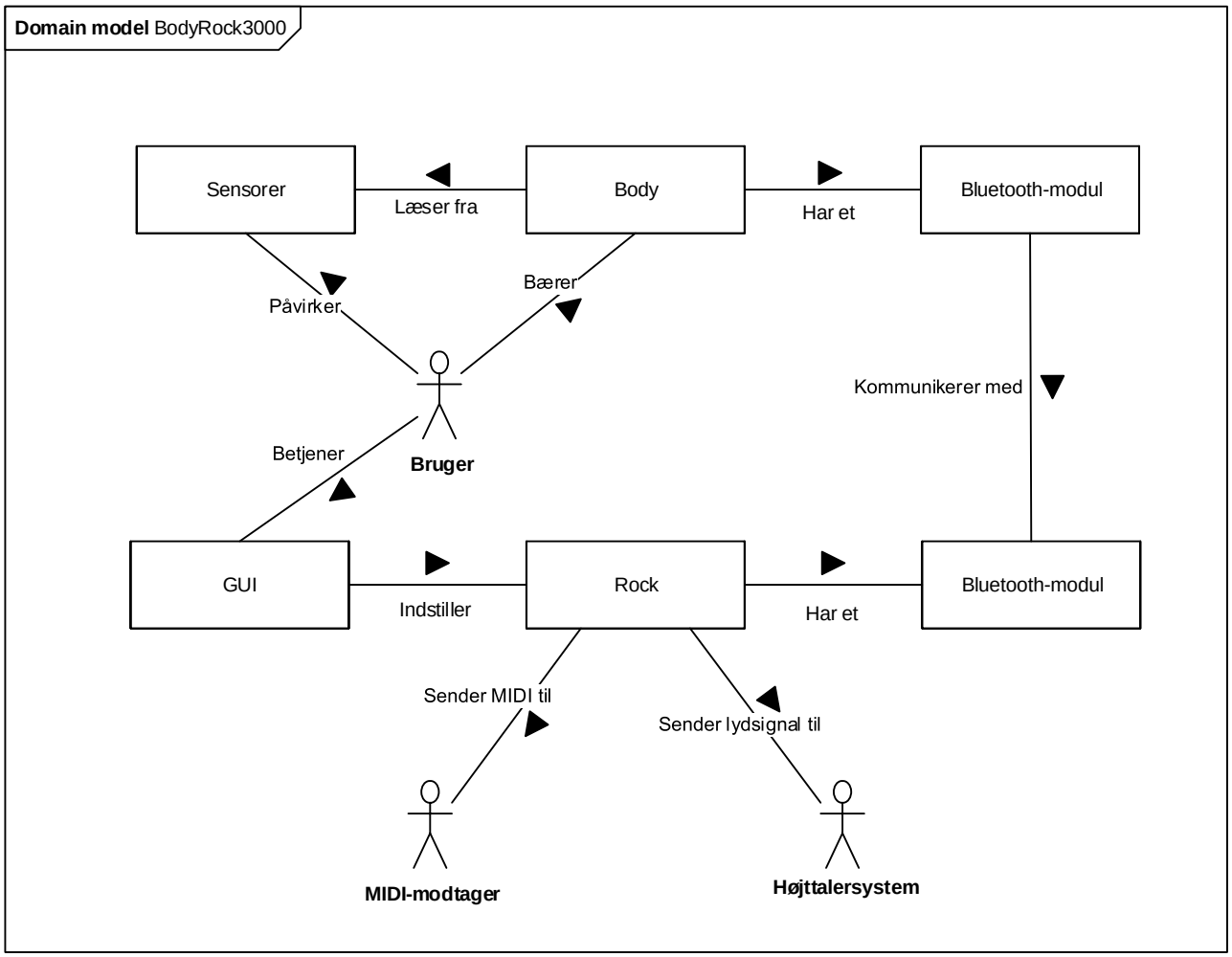
Kundeundersøgelsen udmønter sig i kravspecifikationens afsnit ”Krav til MappingScheme” i projektdokumentationen[19]

[17] Projektdokumentation fra s. 65
[18] Projektdokumentation fra s. 65
[19] Projektdokumentation s. 20

Systemarkitektur

Overordnet arkitektur

Den overordnede systemarkitektur ses i domænemodellen. Denne model giver et godt overblik over det samlede system og hvordan det interagerer med hinanden.



***Figur 6** Domænemodel for BodyRock3000*

Systemet består af de to "hovedenheder", Body og Rock[20], som er er arbejdsmaskinerne i systemet, samt et arbitrært antal sensorer. Body modtager datainput fra sensorerne, pakker dem og sender dem videre over Bluetooth til Rock, som omdanner disse data til lydoutput.

Brugeren har mulighed for at opsætte diverse konfigurationer via den grafiske brugergrænseflade (GUI) på Rock. På denne måde kan der konfigureres, hvordan Rock skal håndtere de datainput, den modtager.

[20] Projektdokumentation s. 7

Design, implementering og test af HW

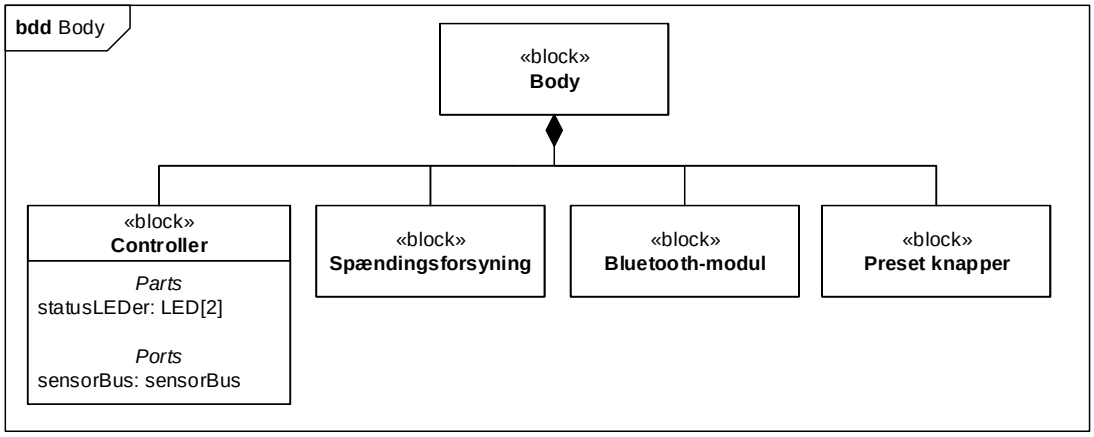
I det følgende beskrives det detaljerede hardwaredesign for projektet. Det detaljerede design tager udgangspunkt i kravspecifikationen og systemarkitekturen. Designprocessen, samt hvilke overvejeler og valg, der er taget, vil ligeledes blive beskrevet i dette afsnit.

Indledende designovervejelser

Domænemodellen for BodyRock3000[21] viser hvilke blokke, det samlede system består af, og hvilke funktionaliteter, de skal opfylde.

Systemet kan nedbrydes til tre hardwareblokke; en Body-blok, en sensor-blok og en Rock-blok.

Body-blokken kan nedbrydes til følgende:



***Figur 7** BDD for body*

Body består af følgende hardwareblokke:

- Spændingsforsyning
- Bluetooth-modul
- Preset-knapper

For yderlige beskrivelse af blokkene, henvises til projektdokumentationen[22].

Overvejelser omkring sensorer

I projektformuleringen blev det fastlagt, at det endelige produkt skulle kunne opsamle data fra bevægelser, herunder acceleration, tilt, afstand og taktilt tryk[23].

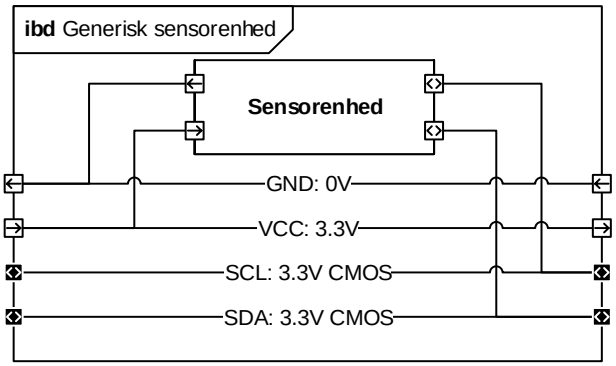
Ud fra ovenstående er det besluttet, at følgende sensortyper anvendes:

- Accelerometer
- Gyroskop
- Proximity-sensor
- Tryksensor

[21] Projektdokumentation s. 22
[22] Projektdokumentation s. 23
[23] Projektdokumentation s. 6

Sensorer

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



Figur 8 IBD for generisk sensorenhed

Af **Figur 8** 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.

I det følgende afsnit beskrives de forskellige sensorenheder. Kun accelerometeret er fuldt beskrevet. For fuld beskrivelse af de øvrige sensorer, henvises til projektdokumentationen[24]

Accelerometer

Sensoren som benyttes i dette projekt, er et 3-akset accelerometer af typen **ADXL345**[25]. Denne model er *ultralow power*, den kan gå så lavt som til $23\mu A$ i *measure mode*, og kun $0{,}1\mu A$ i *standby mode*. Disse værdier er fundet i databladet. ADXL345 understøtter i forvejen I²C, 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.

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**, 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 1 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[26].

[24] Projektdokumentation fra s. 54
[25] Bilag 10
[26] Projektdokumentation s. 57

Gyroskop

Gyroskopet har til formål at generere data på baggrund af *tilt*. For fuld beskrivelse af gyroskopets design og implementering, henvises til projektdokumentationen[27].

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[28].

Tryksensor

Tryksensoren har til formål at generere data på baggrund af taktilt tryk. For fuld beskrivelse af tryksensorens design og implementering, henvises til projektdokumentationen[29].

I^2C-bus

For at forbinde sensorerne til systemet benyttes en I^2C bus, for mere dybdegående beskrivelse af dette valg henvises til projektdokumentationen[30].

Trådløs kommunikation

Body – Trådløst modul

Afsendelsen af sensordata fra Body til Rock sker via Bluetooth-modulet HC-05. Modulet er, vha. AT-kommandoer, indstillet til en baud rate på 115.200, 8 data bit, 1 stopbit, ingen paritet, samt *slave mode*. Modulet er koblet til *Body Shield*, og har gennem dette forbindelse til benene på PSoC4 som i Tabel 2.

HC-05 pins	PSoC4 pins
VCC	3.3v DC Power
GND	GND
RXD	P0[5] TXD
TXD	P0[4] RXD

Tabel 2 Forbindelse mellem RN-42 og Raspberry Pi B+

For yderligere forklaring, henvises til projektdokumentationen[31]

Rock – Trådløst modul

Modtagelsen af sensordata til Rock fra Body sker via Bluetooth-modulet RN-42. Modulet er indstillet vha. Command mode[32] til en baud rate på 115.200, 8 data bit, 1 stopbit, ingen paritet, samt *Master Mode 4*, auto-connect DTR Mode. Denne tilstand gør at RN-42 afsætter 6 bytes hukommelse til at lagre en MAC-adresse. Ved at fører JP3 (GPIO6) på Pmod BT2 høj, vil modulet forsøge at auto-genetablere til denne MAC-adresse. Modulet er derfor programmeret med MAC-adressen på Bodys Bluetooth-modul (HC-05). Modulet er forbundet til Rock som i Tabel 3.

[27] Projektdokumentation s. 58
[28] Projektdokumentation s. 61
[29] Projektdokumentation s. 63
[30] Projektdokumentation s. 54
[31] Projektdokumentation fra s. 65
[32] Bilag 13 s. 16

RN-42 Pins (Digilent Inc., 2011)	Raspberry Pi B+ Pins (Raspberry Pi Foundation, 2014)
Pin 06 VCC	Pin 1 3.3v DC Power
Pin 05 GND	Pin 25 GND
Pin 02 RXD	Pin 08 GPIO14(TXD0)
Pin 03 TXD	Pin 10 GPIO15(RXD0)
JP3 fungerer som reconnect	

Tabel 3 Forbindelse mellem RN-42 og Raspberry Pi B+

For yderligere forklaring, henvises til projektdokumentationen[33]

Body Shield

Body shieldet skal sørge for en nem tilkobling af de ting som skal forbindes til PSoC:

- De fire RJ11 stik til sensorkæderne
- Tilkobling af ekstern spændingsforsyning
- Bluetooth modul HC-05
- Knapmatrix
- Status dioder
- Restart knap
- Adgang til ubrugte io pins

Shieldet er designet således at det passer ned over PSoC boardet som vist nedenfor

Figur 9: *Færdig print Body Shield*

[33] Projektdokumentation fra s. 65

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.

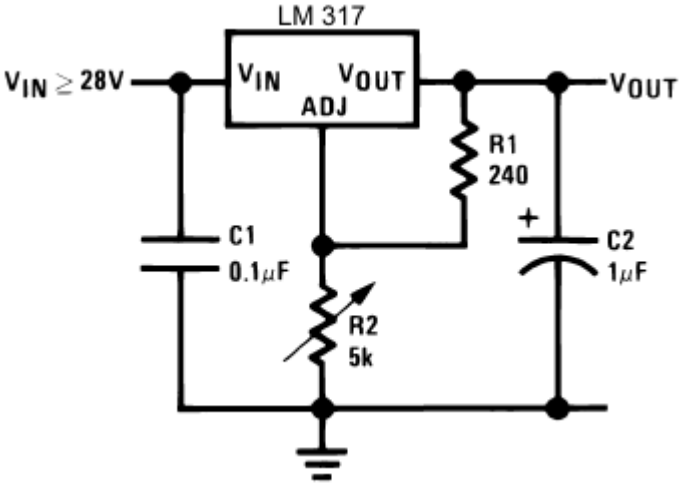
Reguleringskreds

Til reguleringskreds benyttes **LM317**[34], som er en 3-terminals, justerbar regulator med et *output range* fra 1,2V til 25V. En typisk opsætning fremgår af **Figur 10**.

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

$$V_{OUT} = 1{,}25V \cdot \left(1 + \frac{R2}{R1}\right) + I_{ADJ} \cdot R2$$

Som det fremgår af ovenstående formel, er V_{OUT} 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.



Figur 10 *Typisk opsætning af LM317 (fra datablad LM317)*

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 projektdokumentationen[35].

Body

Implementering af Bodys styresystem er beskrevet vha. pseudokode. For hele beskrivelsen af implementeringen, henvises til projektdokumentation[36].

[34] Bilag 03
[35] Projektdokumentation fra s. 78
[36] Projektdokumentation fra s. 78

Main

For beskrivelse af mains funktion, henvises til systemarkitekturen[37].

```
Main
  Tænd power-indikator

  Initier I²C, sensorer og UART
  Loop
    Hvis nyt preset[38]
      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
```

Sensorer

convSensdata

```
convSensData[39]
  Opret variabler
  Saml most significant og least significant for hvert sensorkoordinat
     X, Y, Z i variabler
  Konverter fra værdier i intervallet [-512..511] til værdier i intervallet [0-
..127], og gem i 8 bit variabel
  Kald setdataArray fra SerialUnit, og sæt accelerometer-ID samt konverterede
..X-, Y- og Z-værdier
```

[37] Projektdokumentation fra s. 36
[38] Da preset ikke er med i denne iteration, er det ikke implementeret i den endelige udgave af main-koden for projektet.
[39] Denne funktion er kun implementeret til at standardisere accelerometerets (ADXL345) sensordata, da de øvrige er afgrænset væk.

SerialUnit

For yderligere beskrivelse af SerialUnits funktioner, henvises til systemarkitekturen[40].

```
Opret klasse-lokalt dataarray[16] // Med plads til fire sensorer[41]
```

initUART

```
initUART
  Start og initier UART-komponenten
```

setDataArray

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

sendDataArray

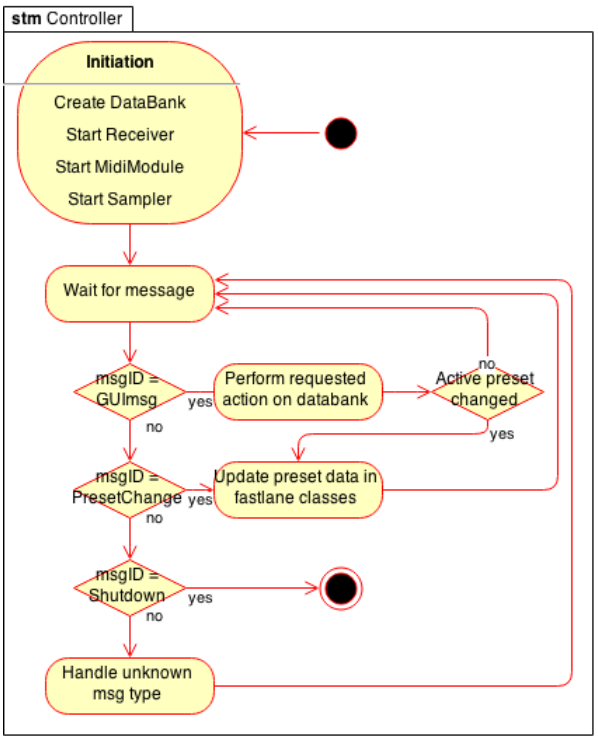
```
sendDataArray
  Opret et array med seks pladser pr. sensor
  Sæt hvert sensorarray[42] med de tilsvarende pladser fra det klasse-lokale
..dataarray + 1[43]
    Valider sensordata
    Hvis der er fejl
      Tænd rød error-LED
    Send sensordata
    Reset dataarray til '0' = 48
```

[40] Projektdokumentation fra s. 36
[41] I projektet var der oprindeligt afgrænset til fire sensorer i den endelige udgave er der afgrænset til én.
[42] Da der er afgrænset til en sensor i denne udgave af projektet vil akseværdierne i alle andre arrays end accelerometer-arrayet blive sat til 1.
[43] Alle data der sendes bliver inkrementeres én gang pga. datavalidering i receiver klassen på Rock.

Rock: Slow Lane

Controller

Systemets centrale klasse er Controller. Denne klasse står for at starte resten af systemets tråde, håndtere systemets DataBank og kontrol af fast lane klasserne ved skift af preset. **Figur 13** viser et stm diagram over klassens opførsel.



Figur 11 Controller stm diagram

DataBank

DataBank har til formål at gemme og hente konfigurationer. Disse kan være sensorkonfigurationer, presets eller lydpakker. Hvilke attributter konfigurationerne indeholder, kan ses i projektdokumentationen[44].

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

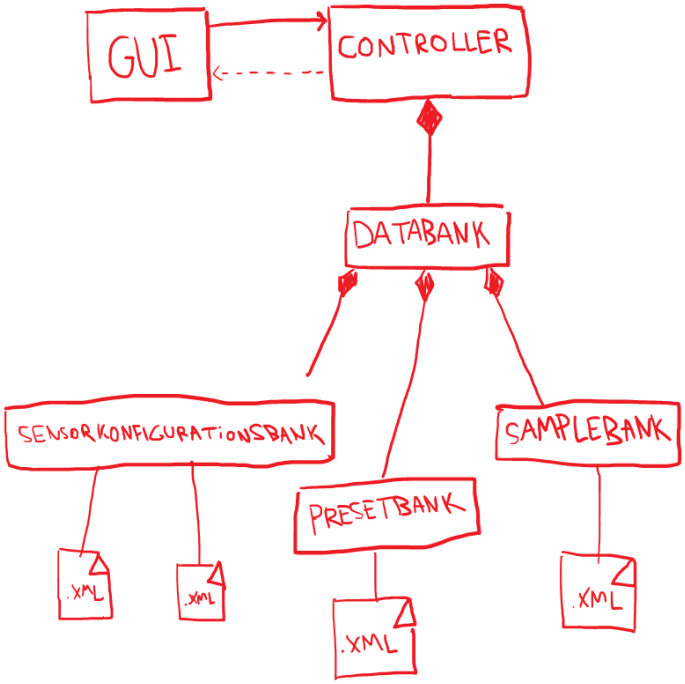
Controller-klassen har en instans af DataBank, således at den blot kalder funktioner i DataBank, så den kan få de relevante data og sende dem videre til GUI.

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

Figur 12 viser en overordnet skitse for DataBank.

For detaljeret beskrivelse af interaktion henvises til sekvensdiagrammet over DataBank[45].

For funktionsbeskrivelser, se projektdokumentation[46].



***Figur 12** Skitse 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[47].

[44] Projektdokumentation fra s. 94
[45] Projektdokumentation s. 44
[46] Projektdokumentation fra s. 94
[47] Projektdokumentation s. 94

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

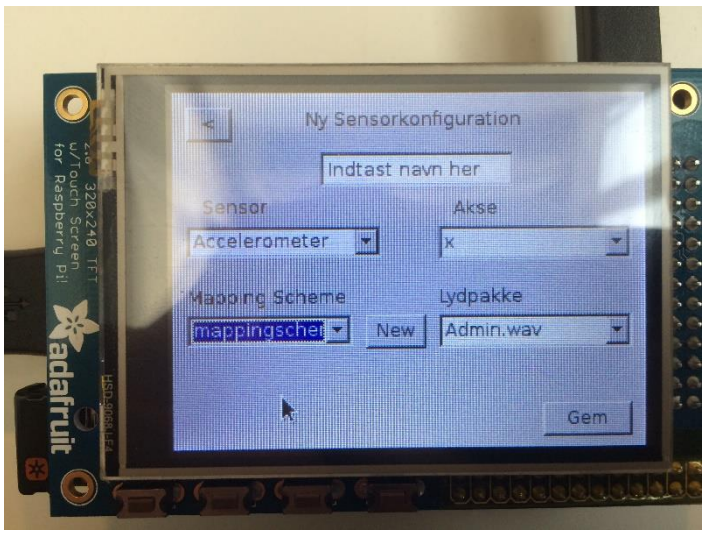
Den grafiske UI for den tilhørende klasse dannes igen på baggrund af brugerens valg, og der sendes en forespørgsel, gennem *Message Handler*-klassen, efter den nødvendige info. Da der tages udgangspunkt i prototypen, er det *Ny sensorkonfiguration*, der er implementeret.

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. MappingScheme
3. Akse
4. Lydpakke

Sensortype og akse er statiske 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 MappingSchemes og lydpakker, der ligger i *DataBank*. Derved dannes der et UI med mulighed for at vælge den information, som skal bruges til at oprette en ny sensorkonfiguration.



Figur 13: Eksempel på GUI

Efter brugeren har valgt de ønskede sensorkonfigurationer, er det muligt at gemme disse ved anvendelse af en *gem*-knap. Når denne anvendes, lagres de valgte informationer i *DataBank* via *Message Handler.*
Den nyoprettede sensorkonfiguration er nu klar til at blive benyttet af *MidiModule*.
For yderligere information om GUI-klasserne, henvises til dokumentationen[48].

Rock: Fast Lane

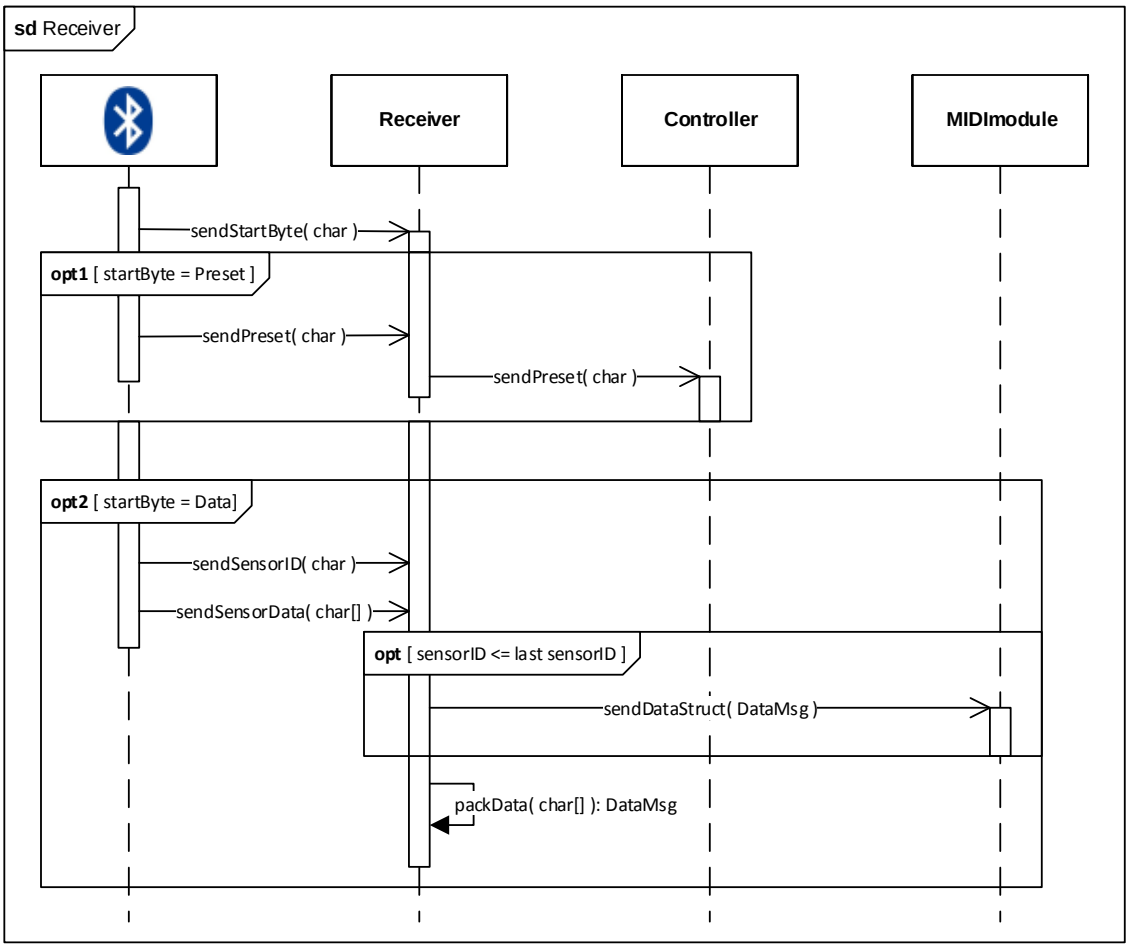
Receiver

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 Rock. Denne forbindelse lukkes igen ved nedlæggelse af objektet.

Efter oprettelsen af en instans af klassen, kaldes funktionen *start*, der opretter en ny tråd, som i en uendeligt løkke kalder den blokerende *receive*-funktion. Herfra vil klassen til enhver tid afvente ny data fra Body, og ved indkommende data behandle det som ønsket, for at gå tilbage til den ventende tilstand.

Figur 14 viser et sd-diagram over operationerne som udføres af klassen Receiver.



Figur 14 Sekvensdiagram over operationerne udført af klassen Receiver

[48] Projektdokumentation fra s. 89

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 skiftes preset[49]. 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, 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 akser. Hvis ID'et er større end sidst modtagede ID, tilføjes data 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 funktionsbeskrivelser og UML-diagram, henvises til projektdokumentation[50].

MidiModule

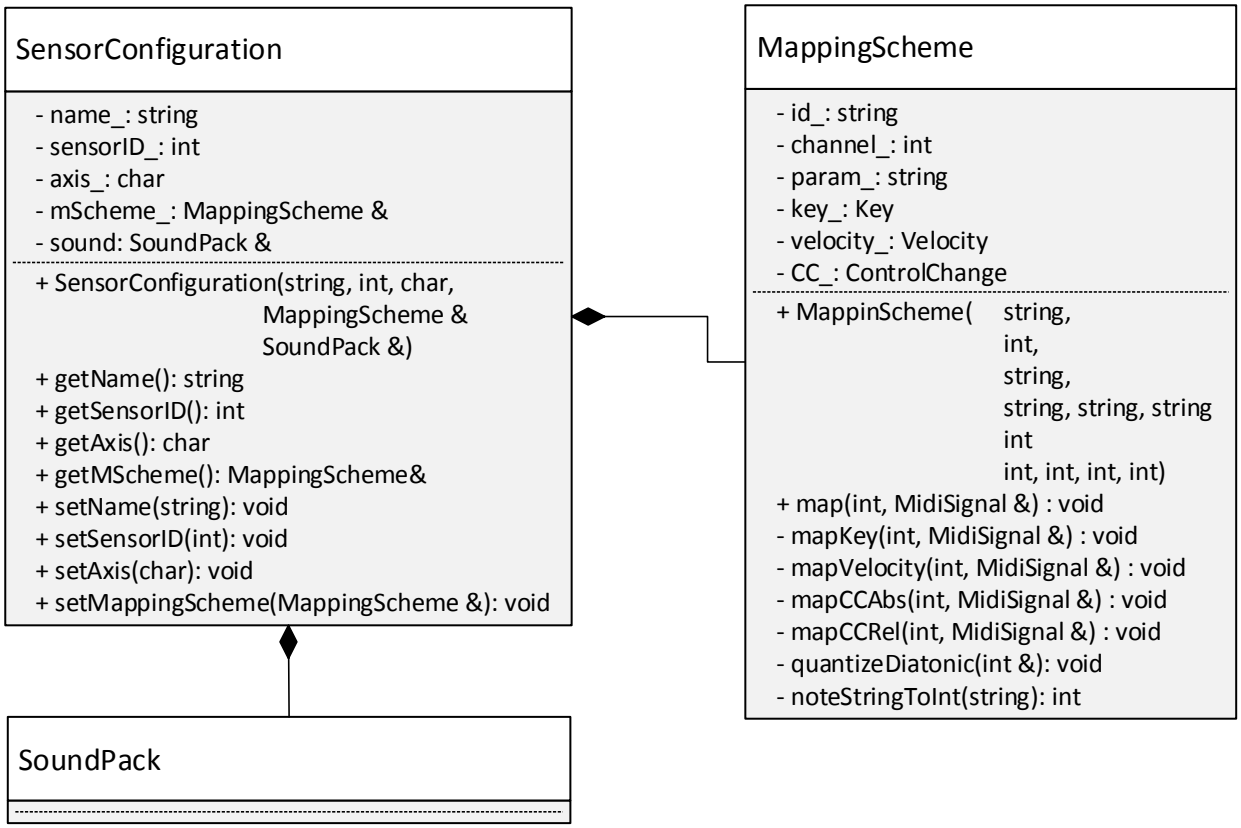
SensorConfiguration

SensorConfiguration har til opgave at lagre de samlede indstillinger for en enkelt sensorkonfiguration.

Denne består af:

- **name_**: Det navn, den givne indstilling gemmes under.
- **sensorID_**: Et ID, der repræsenterer en given sensor. Se projektdokumentationen for oversigt over sensorer og sensorID for disse[51].
- **axis_**: aksen for sensorer med flere akser (f.eks. gyroskop). Sensorer med kun én akse sættes til 'x' som default.
- **mScheme_**: Et givent MappingScheme. Se projektdokumentationen for en beskrivelse af denne[52].
- **soundPack**: En lydpakke indeholdende de samples, som LinuxSampler afspiller for den givne konfiguration. *Denne feature er ikke implementeret i nuværende system iteration*.

[49] Muligheden for preset-skift er blevet afgrænset, og dermed ikke implementeret i resten af systemet. Derfor er denne funktionalitet fjernet fra denne iteration af Receiver-klassen.
[50] Projektdokumentation fra s. 46
[51] Projektdokumentation fra s. 30
[52] Projektdokumentation fra s. 48



***Figur 15** Klassediagram over SensorConfiguration*

MappingScheme

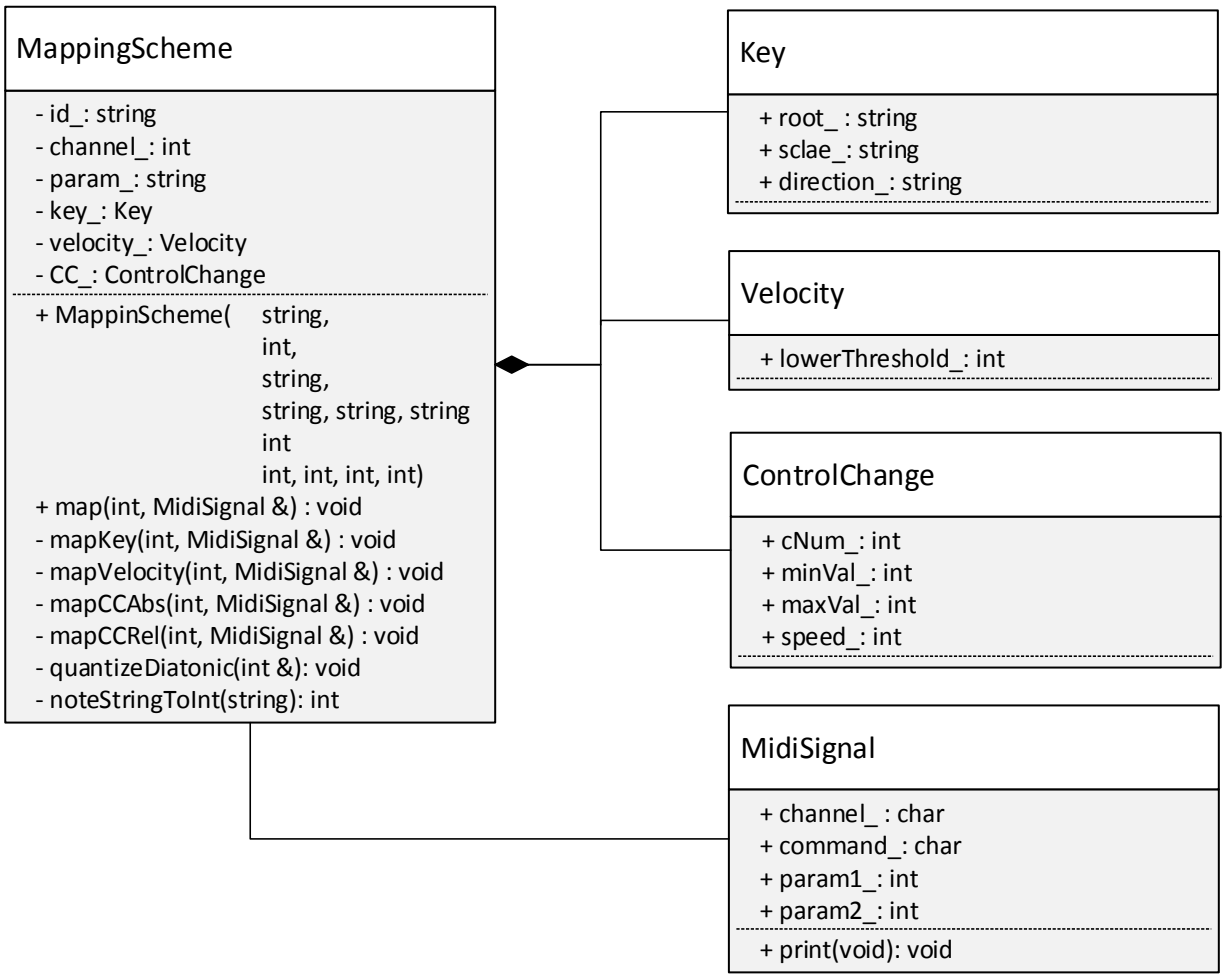
Design

MappingScheme-klassens funktion er 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 givne sensordata

Klassediagram

Figur 16Figur 16 viser et klassediagram for *MappingScheme* og dets lagringsstrukturer, samt for *MidiSignal*, som *MappingScheme* også benytter i sin *map*–funktion. Bemærk at der, for klassediagrammet herunder, er udeladt *set*- og *get*-metoder.



***Figur 16** Klassediagram over MappingScheme*

For funktionsbeskrivelser, se projektdokumentation[53].

[53] Projektdokumentation s. 102

Implementering

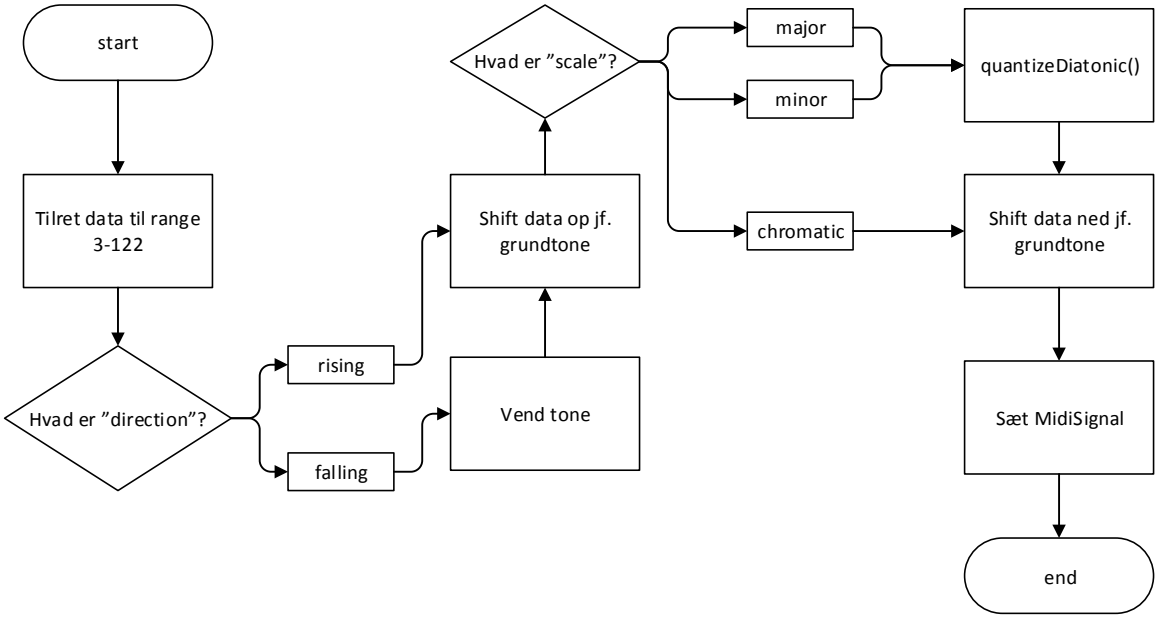
Herunder foreligger et udsnit af måden, hvorpå klassens funktioner er implementeret. For de resterende funktioner, *mapVelocity*, *mapCCAbs* og *mapCCRel*, se systemarkitektur[54].

map

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

mapKey

Figur 17 viser programflowet i *mapKey*



***Figur 17** Flowchart over programflowet i mapKey*

Tilrettelse af data: Systemet kan generere toner fra oktaverne -2 til 9, hvilket svarer til 12 oktaver. Dette giver mulighed for at generere i alt 120 forskellige toner. Derfor benyttes kun sensorData i intervallet 3-122.

quantizeDiatonic: Beskrives i det følgende afsnit. Dataforskydning muliggør korrekt kvantificering 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.

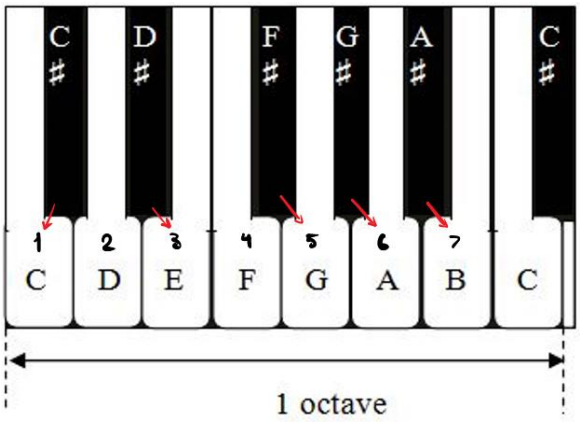
[54] Projektdokumentation s. 105

quantizeDiatonic

Herunder vises et udsnit af dokumentationen for *quantizeDiatonic*. Foruden nedenstående, forefindes en beskrivelse af mol-kvantificeringen i projektdokumentationen[55].

Når de modtagne data skal kvantificeres, skiftes de toner, der ikke ligger i skalaen op eller ned til en tone i skalaen. Kvantificeringen designes således, at de mest brugte toner favoriseres: Prim (1. trin), terts (3. trin) og kvint (5. trin), herunder markeret som hhv. 1, 3 og 5.

Dur-kvantificeringen foregår som vist på **Figur 18**.



Figur 18 *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 kvantificeres til (røde pile).

AlsaAdapter

Klassen AlsaAdapter modtager en vektor af MIDI-signaler fra MidiModule, og videresender disse til resten af systemet via Alsa.

For funktionsbeskrivelser for AlsaAdapter henvises til projektdokumentationen[56].

Alsa

Da projektet overordnet omhandler genereringen og manipulationen af MIDI-signaler, har det været oplagt at anvende det allerede indbyggede framework, Alsa. Anvendelsen af Alsa har gjort det muligt at fokusere på genereringen af MIDI-signaler, da dette framework har kunnet håndtere den videre afsendelse af disse.

I systemet anvendes specifikt RawMidi-API'et, der håndterer rå midi data. Dette bruges i AlsaAdapter-klassen, der agerer bro mellem MidiModule, som leverer de genererede MIDI-signaler, og Alsa. Beslutningen om at benytte et allerede etableret, og meget anvendt, framework, muliggjorde brugen af tredjepartskomponenter, såsom det eksterne lydkort og LinuxSampler. Skulle disse komponenters funktionalitet have været designet og implementeret fra bunden, havde det medført en betydelig forøgelse af projektets omfang og, i værste fald, have umuliggjort gennemførslen inden for den givne tidsramme. For yderligere information omkring projektets inddragelse af Alsa, henvises til projektdokumentationen[57].

[55] Projektdokumentation s. 104
[56] Projektdokumentation s. 35
[57] Projektdokumentation s. 35

Udviklingsværktøjer

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

PSoC Creator

Givet at brugen af PSoC var et projektkrav, har Cypress' PSoC Creator været væsentlig for projektet. Denne 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 sensorkredsløb 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 Atmel Studio også et værktøj til at programmere chippen gennem vores STK500 board.

LinuxSampler

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

Multisim

Til kredsløbene 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

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 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 organisering af filer, både kode og dokumentation, er der blevet benyttet et Git-repository fra GitHub. Dette er 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 er der gjort stor nytte af C++'s Standard Template Library, specielt til Rocks datastrukturer. Da programmet også har behov for at kunne gemme data ved system-genstart, er der

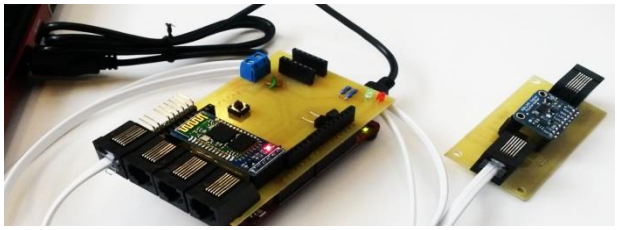
ud over standard bibliotekerne også benyttet Boost's serialisations bibliotek til lagring af objekter som XML kode.

Resultater og diskussion

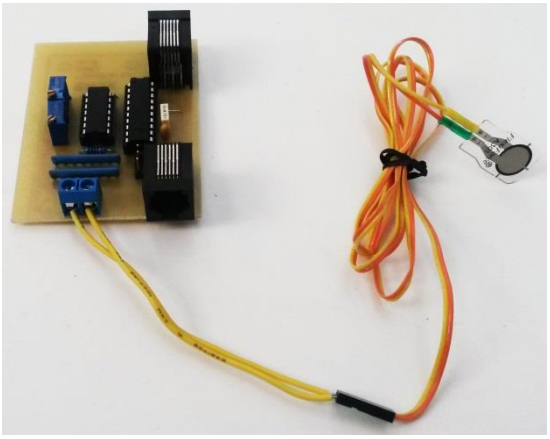
Størstedelen af projektet er implementeret. BodyRock3000 er defineret som et digitalt instrument, der skal afspille lydsamples og generere MIDI-toner på baggrund af, blandt andet, et accelerometer. Systemet skal fungere trådløst mellem de to enheder Body og Rock, for at brugeren får de rette udfoldelsesrammer, for at bevæge sensorerne. Indstillingen af systemet skal ske via en grafisk grænseflade på Rock.

På billederne herunder ses de forskellige hardware-dele, som er succesfuldt implementeret. På næste side ses der, hvad der blev færdigt overordnet.

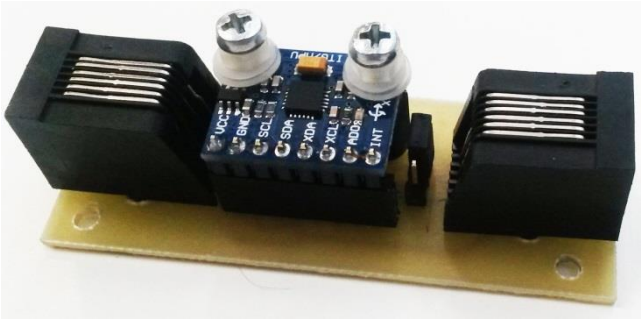
Figur 24 Rock: Øverst: Højtaler. Venstre: Raspberry Pi og skærm. Højre: Bluetooth receiver.



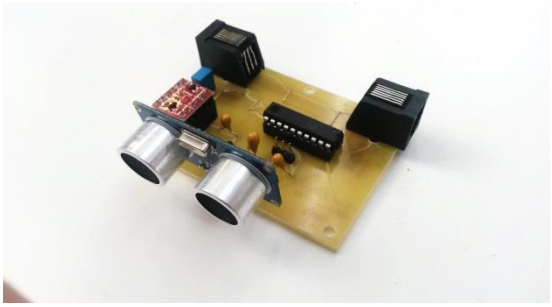
Figur 23 Body. PSoC'en med shield og bluetooth-modul til venstre og accelerometer til højre. Forbundet gennem I2C med ethernetkabel.



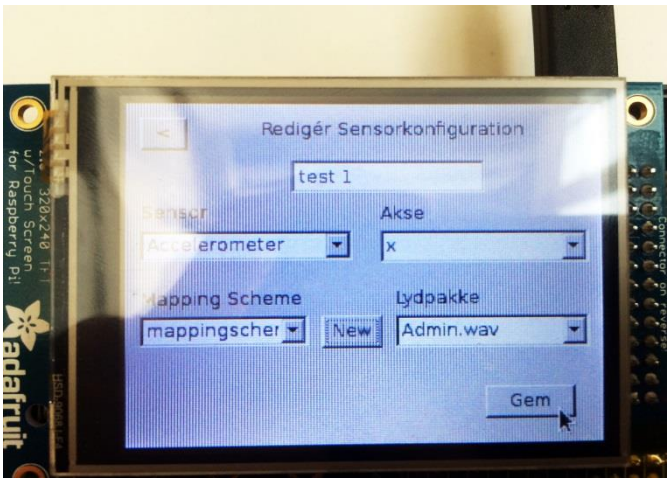
Figur 22 Tryksensoren



Figur 19 Gyroskop



Figur 21 Proximity-sensor



Figur 20 Rock. GUI på Raspberry Pi.

Nuværende iteration
GUI på Raspberry Pi
DataBank skelet
Sensorkonfiguration + Mapping Scheme
Rock Controller
Trådløs kommunikation
Accelerometer-sensor
Body Controller
Linux MIDI Sampler
Envejskommunikation mellem Body og Rock Afsendelse af sensordata

Fremtidige iterationer
Tovejskommunikation mellem Body og Rock
Presets generelt (manglende systemarkitektur på Body f.eks.)
Dele af DataBank/GUI (forbind/autoforbind-knap mangler blandt andet)
Implementering af yderligere sensorer – kun én kan sættes til, og kun et accelerometer kan tilsluttes. Der mangler: gyroskop, proximity-sensor, tryksensor
Forsyningsprint til Body (batteri)
Tænd/sluk funktion Body/Rock
'Dragten' Body skal sættes på
Lydpakker
Forbindelse mellem MIDI-adapter og Rock-Controller

Ved projektets begyndelse blev Devkit-8000 anvendt som platform for Rock. Det viste sig dog at dette board ikke understøttede de påkrævede ALSA- og MIDI-funktionaliteter. Dette førte til et skift af udviklingsplatform relativt langt inde i projektet. Konsekvensen af dette var, at en ikke ubetydelig del, af gruppens ressourcer blev reallokeret på et kritisk tidspunkt af projektet. Denne reallokering tog ressourcer fra andre vigtige dele af projektet og belastede gruppen yderligere. Havde udviklingsplatformen været Rasberry Pi fra start af, kunne processen i langt højere grad have været strømlinet.

Tages der udgangspunkt i projektformuleringen ses det, at størstedelen af målene for systemet er nået. Den, på nuværende tidspunkt, manglende funktionalitet ville med nok tid og ressourcer kunne blive implementeret uden større tekniske problemer.

Gruppen står, ved projektets udgang, stadig inde for det oprindelige design og føler, at det både er interessant og har potentiale til at bringe glæde til de personer som vil benytte systemet.

Individuelle opnåede erfaringer og konklusioner

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 LinuxSampler 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.

Felix

For mig har den største udfordring i dette projekt, såvel som i tidligere, været at opnå ro omkring mine opgaver. Dette har dog været lidt forbedret i dette projekt fordi vi har brugt Scrum. Dette har helt tydeligt hjulpet, idet ens opgaver ikke ændres undervejs. For at dette hjælper kræves der dog en hvis tilbageholdenhed ved Scrum planlægningen, samt en forholdsvis god ide om opgavernes omfang. Igennem projektet er jeg blevet betydeligt bedre til at vurdere opgavernes størrelse og min egen kapacitet, samt hvordan man bedst takler det hvis man får taget for mange opgaver.

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 MappingScheme:

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

Konklusion

Efter et slutsprint med mange timers hårdt arbejde, er det lykkedes at få produceret en fungerende prototype, som kan aflæse et flerdimensionelt output fra et accelerometer, forbundet til et PSoC4-board, som behandler og videresender dataene over Bluetooth til en Raspberry Pi, som pakker alle data fra samme aflæsningsrunde, og sender dem videre i systemet, til en klasse som på baggrund af dataene genererer lyd via MIDI, som sendes til afspilning gennem Pi’ens mini jack-stik. Dette mål er vi meget tilfredse med, da det demonstrerer sammenhængen i et system som er udviklet i moduler.

Vi er ligeledes godt tilfredse med at have fået udviklet en GUI på Pi’ens touchskærm, som kan oprette, lagre, indlæse og slette sensorkonfigurationer. Vi har desværre ikke nået at få indkorporeret GUI’en i det

øvrige systems funktionalitet, men det er ikke meget, der i skrivende stund mangler, før denne sammenkobling er mulig med GUI'ens nuværende funktionaliteter.

Valget om at erstatte devkit8000 med en Rasperry Pi må konkluderes at være det rigtige. Det gav os nogle markante udfordringer med at få udviklingsværktøjet QT (og dertilhørende toolchain) installeret, hvilket var kilde til megen frustration og forsinkelse. Fordelene mere end opvejede det dog, da det muliggjorde syntetisering af MIDI-lydene, og i det hele taget gjorde det meget nemmere at finde inspiration til løsning af problemer online.

Vi har, i forbindelse med de mange Scrum-sprints, draget os værdifulde erfaringer med iterativ projektstyring. Her har vi lært meget om vigtigheden af både at være ambitiøs, og på samme tid kende sine egne grænser. Førstnævnte bunder i at vi arbejdede langt mere produktivt, når der altid var mere at lave, i forhold til den sovepude det kunne blive, hvis man følte at man havde nået hvad man skulle inden næste sprint. En prioritering af opgaver, som giver mulighed for at påbegynde lavprioritets-opgaverne ved færdiggørelse af højprioritets-opgaverne, har derfor vist sig at være en god idé. Sidstnævnte bunder i at mange i gruppen har oplevet problemet ved at påtage sig for mange højprioritets-opgaver inden for samme sprint. Hvis ét gruppemedlem står med flere opgaver, som andre opgaver er afhængige af færdiggørelsen af, giver det både en flaskehals i udviklingsprocessen, og forhindrer det enkelte gruppemedlem i at fordybe sig i én enkelt opgave, hvilket medfører at man hele tiden kommer ind og ud af fokus på denne opgave.

Alt i alt må det konkluderes at have været et godt og succesfuldt projekt. Der er stadig meget udviklingspotentiale, og ting, vi ikke har nået, men vi kan stille op med en fungerende prototype, og har lært meget om processen omkring at udvikle et system, hvilket bidrager til at gøre os dygtigere og mere effektive ved fremtidige projekter.

Referencer

Boost Community. (n.d.). *Welcome to Boost.org!* Retrieved from Boost C++ libraries: http://www.boost.org/

Digilent Inc. (2011, august). *PmodBT2 - Bluetooth Interface*. Retrieved from Digilent: http://www.digilentinc.com/Products/Detail.cfm?Prod=PMOD-BT2

LinuxSampler. (n.d.). *About*. Retrieved from LinuxSampler: http://www.linuxsampler.org/about.html

MIDI Manufacturers Association. (2014, juni). *MIDI Manufacturers Investigate HD Protocol*. Retrieved from midi.org: http://www.midi.org/aboutus/news/hd.php

PivotPoint Technology Corp. . (n.d.). *SysML Forum*. Retrieved from SysML Forum: http://www.sysmlforum.com/

Raspberry Pi Foundation. (2014, juli). *MODEL B+*. Retrieved from RaspberryPi.org: http://www.raspberrypi.org/products/model-b-plus/

Bilag

Bilag forefindes på CD-rom.

Bilag 01	*Semesterprojekt 3 Oplæg 20140202.pdf*	*(Dokument)*
Bilag 02	*Raspberry Pi model b datasheet.pdf*	*(Dokument)*
Bilag 03	*LM317.pdf*	*(Dokument)*
Bilag 04	*atmel-2561-using-the-usi-module-as-a-i2c-master_ap-note_avr310.pdf*	*(Dokument)*
Bilag 05	*Essentials of the MIDI protocol.pdf*	*(Dokument)*
Bilag 06	*Getting Started with QT.pdf*	*(Dokument)*
Bilag 07	*RPi ssh setup.pdf*	*(Dokument)*
Bilag 08	*AT-tiny26P.pdf*	*(Dokument)*
Bilag 09	*The Scrum Guide.pdf*	*(Dokument)*
Bilag 10	*ADXL345.pdf*	*(Dokument)*
Bilag 11	*MUA08A.pdf*	*(Dokument)*
Bilag 12	*Pmod BT2 (Master Bluetooth module).pdf*	*(Dokument)*
Bilag 13	*bluetooth_cr_UG-v1.0r (RN 42).pdf*	*(Dokument)*
Bilag 14	*DS_BluetoothHC05.pdf*	*(Dokument)*
Bilag 15	*UM10204.pdf*	*(Dokument)*
Bilag 16	*MPU6050.pdf*	*(Dokument)*
Bilag 17	*HC-SR04.pdf*	*(Dokument)*
Bilag 18	*LM2750.pdf*	*(Dokument)*
Bilag 19	*Logic Level Converter.pdf*	*(Dokument)*
Bilag 20	*Flexiforce A301.pdf*	*(Dokument)*
Bilag 21	*Body_sensor_enhedstest_ACC*	*(Kode samling)*
Bilag 22	*I2C_Test_Program_Atmel_Studio*	*(Kode samling)*
Bilag 23	*Flexiforce_A301_Control_Program_Atmel_Studio*	*(Kode samling)*
Bilag 24	*Flexiforce_A301_Sensorforbindelse_body_v1.cydsn*	*(Kode samling)*
Bilag 25	*Nord Lead 2x English User Manual v1.0 Edition 1.1.pdf*	*(Dokument)*
Bilag 26	*Body_main_Version_4*	*(Kode samling)*
Bilag 27	*Rock Fast Lane source kode*	*(Kode samling)*
Bilag 28	*Rock Slow Lane source kode*	*(Kode samling)*
Bilag 29	*Intergrationstest af fast lane(Fra sensor til audio output).mp4*	*(Video)*
Bilag 30	*Integrationstest af fast lane(Fra sensor til ekstern MIDI output).mp4*	*(Video)*
Bilag 31	*Mødeindkaldelser*	*(Dokumenter)*
Bilag 32	*Referater*	*(Dokumenter)*
Bilag 33	*Taskboard*	*(Dokumenter)*
Bilag 34	*Tidsplaner*	*(Dokumenter)*
Bilag 35	*I2ISE Slide – Development Processes.pdf*	*(Dokument)*
Bilag 36	*Turnusordning.pdf*	*(Dokument)*
Bilag 37	*Git_log.pdf*	*(Dokument)*