

# 3. Semesterprojekt

## [Syng og Sammenlign]

### Projektrapport

Dato: 15/12 2023

Anslag: 70094

Gruppe 12

#### Deltagere i afleveringen

Studienummer	Navn	Studieretning
202207059	Emil Lydersen	SW3
202107339	Oliver Kok	SW3
202204902	Simon Dybdahl Damulog Andersen	SW3
202204919	Hanad Osman Hashi Aden	SW3
202204580	Bassam Salou	SW3
202206885	Mikail Ipek	SW3
202207724	Fardoos Kelival	SW3
202208488	Max Glintborg Dall	SW3

#### Vejleder

Vejleder	
Lars Mortensen	<a href="mailto:lamo@ece.au.dk">lamo@ece.au.dk</a>

## **Abstract**

In this project report, the development of a system named "Syng og Sammenlign" which mimics a karaoke system, is documented. The project utilizes the ASE Model in combination with the SCRUM working method. The system's architecture is designed using SysML and UML diagrams, which are used to define design choices for the system. The design includes, among other things, a graphical user interface (GUI), a server with a database, and SPI communication via a PSoC, which interacts with the main unit, a Raspberry Pi 4. Here, extensive stochastic data processing is carried out. Finally, the system's modules are tested individually, integrated, and then a final acceptance test is conducted.

## Resumé

I denne projektrapport dokumenteres udviklingen af et system kaldet "Syng og Sammenlign", der imiterer et karaokesystem. Projektet anvender ASE-Modellen i kombination med arbejdsmetoden SCRUM. Systemets arkitektur er designet ved hjælp af SysML- og UML-diagrammer, som er brugt til at definere designvalg for systemet. Designet omfatter blandt andet en grafisk brugerflade (GUI), en server med database og SPI-kommunikation via en PSoC, der interagerer med hovedenheden, en Raspberry Pi 4. Her udføres omfattende stokastisk databehandling. Til sidst testes systemets moduler individuelt, integreres, og en accepttest udføres.

## Forord

Denne rapport er udarbejdet af en gruppe studerende på 3.semester fra Aarhus Universitet for Elektro- og Computerteknologi. Rapporten er lavet på baggrund af de stillede minimumskrav til semester projekt 3, samt vejleder Lars Mortensens individuelle krav. Projektet inkluderer arbejde med SQL Database, Server Sockets, UI Design, Digital signal behandling fra PSoC og arbejde med .wav filer.

Rapportens fokus ligger på de mest interessante uddrag, som gruppen vurderer yderst vigtige for projektets forståelse og funktionalitet. Alle detaljerede og dybdegående dokumenter og referencer kan findes vedhæftet i projektets bilag. Alt i alt resulterer dette i en projektrapport, som afleveres d.15/12-2023.

## Ordliste

Her er en tabel over de forkortelser der bruges igennem hele rapporten.

<b>FORKORTELTSE/ORD:</b>	<b>FORKLARING:</b>
RPi4	Raspberry Pi 4
RPi0 /RPiZ	Raspberry Pi 0
PSoC	Portable System on Chip
GUI	Graphical User Interface
Db	Database
CD	Class Diagram
SD	Sequence Diagram
BDD	Block Definition Diagram
IBD	Internal Block Diagram
MoSCoW	Must have, should have, Could have, and Won't have
FURPS	Functionality, Usability, Reliability, Performance, Supportability
UC	Use Case
UML	Unified Modeling Language
SysML	Systems Modeling Language
SOS	Syng-Og-Sammenlign
OS	Operating System
ADC	Analog to Digital Converter
SPI	Serial Peripheral Interface
SW	Software
HW	Hardware
MISO	Master Input Slave Output
MOSI	Master Output Slave Input
WAV	Waveform Audio File
TCP	Transmission Control Protocol
DTW	Dynamic Time Warping

<b>ABSTRACT .....</b>	<b>2</b>
<b>RESUMÉ .....</b>	<b>3</b>
<b>FORORD .....</b>	<b>4</b>
<b>ORDLISTE .....</b>	<b>5</b>
<b>1. INDLEDNING .....</b>	<b>9</b>
1.1. PROBLEMFORMULERING .....	9
1.2. PROJEKTFORMULERING .....	9
1.2.1. FORMÅL .....	10
1.2.2. TEKNISKE KOMPONENTER OG FUNKTIONER .....	10
1.2.3. GRAFISK BRUGERGRÆNSEFLADE (GUI).....	11
1.2.4. ANSVARSOMRÅDER .....	12
<b>2 METODE OG ARBEJDSPROCES .....</b>	<b>14</b>
2.1 METODER.....	14
2.2 ARBEJDSPROCES .....	14
2.2.1 SAMARBEJDE.....	15
2.2.2 PROJEKTLEDELSE .....	15
<b>3 KRAV OG PRIORITERINGER.....</b>	<b>16</b>
3.1 FUNKTIONELLE KRAV.....	16
3.1.1 Aktør-kontekst diagram.....	16
3.1.2 Aktør beskrivelser .....	16
3.1.3 Use Case diagram .....	17
3.1.4 Use Case beskrivelser .....	17
3.1.5 Fully dressed Use Case 1: Brugeren vælger en sang .....	18
3.1.6 Fully dressed use Case 2: Syng og Sammenlign.....	19
3.2 FUNKTIONELLE KRAV .....	20
3.2.1 FURPS .....	20
3.2.2 MoSCoW .....	21
<b>4 INDLEDENDE ANALYSER .....</b>	<b>22</b>
4.1 OVERORDNEDE TEKNOLOGISKE VALG .....	22
4.2 RISIKOANALYSE FOR TEKNOLOGISKE VALG .....	23

<b>5</b>	<b>SYSTEMARKITEKTUR.....</b>	<b>25</b>
<b>5.1</b>	<b>OVERORDNET SYSTEM .....</b>	<b>25</b>
5.1.1	BDD.....	25
5.1.2	Blokbeskrivelse .....	25
5.1.3	IBD .....	26
5.1.4	Domænemodel.....	27
<b>5.2</b>	<b>HW ARKITEKTUR .....</b>	<b>28</b>
5.2.1	Signalbeskrivelser .....	29
<b>5.3</b>	<b>SW ARKITEKTUR .....</b>	<b>29</b>
<b>6</b>	<b>MODUL- OG KOMMUNIKATIONSDESIGN .....</b>	<b>35</b>
<b>6.1</b>	<b>PSoC.....</b>	<b>35</b>
<b>6.2</b>	<b>SERVER .....</b>	<b>41</b>
6.2.1	DATABASE DESIGN.....	42
6.2.2	KOMMUNIKATION MELLEM RPi4 & SERVER .....	42
6.2.3	KOMMUNIKATION MELLEM MODULER PÅ RPi4 .....	43
<b>6.3</b>	<b>LOGICUNIT .....</b>	<b>45</b>
6.3.1	Logik behandling .....	45
6.3.2	DESIGN AF MATLAB SKELET OG IMPLEMENTERINGSSTRATEGI .....	45
<b>6.4</b>	<b>BRUGERGRÆNSEFLADE (GUI) .....</b>	<b>50</b>
<b>7</b>	<b>REALISERING AF SYSTEMET .....</b>	<b>54</b>
<b>7.1</b>	<b>PSoC.....</b>	<b>54</b>
<b>7.2</b>	<b>SERVER .....</b>	<b>60</b>
<b>7.3</b>	<b>LOGICUNIT .....</b>	<b>62</b>
7.3.1	Python implementering af præstations vurdering.....	62
<b>7.4</b>	<b>BRUGERGRÆNSEFLADE .....</b>	<b>67</b>
<b>8</b>	<b>TEST OG RESULTATER .....</b>	<b>72</b>
<b>8.1</b>	<b>MODUL- OG INTEGRATIONSTEST .....</b>	<b>72</b>
8.1.1	PSoC.....	72
<b>8.2. SERVER .....</b>	<b>75</b>	
8.1.2	LOGICUNIT .....	77
8.1.3	BRUGERGRÆNSEFLADE.....	78
8.1.4	SERVER & UI.....	81
<b>8.2</b>	<b>ACCEPTTEST .....</b>	<b>82</b>
<b>9</b>	<b>DISKUSSION AF RESULTATER .....</b>	<b>84</b>

<b><u>10</u></b>	<b><u>KONKLUSION &amp; PERSPEKTIVERING .....</u></b>	<b><u>85</u></b>
	<b><u>REFERENCELISTE .....</u></b>	<b><u>87</u></b>
	<b><u>BILAGSOVERSIGT .....</u></b>	<b><u>89</u></b>



# 1. Indledning

## 1.1. Problemformulering

Karaoke er en populær og underholdende aktivitet, hvor folk synger med på deres yndlingssange og forsøger at efterligne de originale kunstneres præstationer. Men adgangen til karaokeoplevelsen kan være begrænset, og til tider kostelig, især når det kommer til at besøge karaokebarer eller bruge tungt og komplekst udstyr derhjemme.

Dette projekt, sigter mod at løse dette problem ved at udvikle et let-bærbart karaokesystem, der giver brugerne mulighed for at nyde karaokeoplevelsen hvor som helst og når som helst. Systemet er designet til nem transport og tilslutning til en hvilken som helst skærm og højttaler via et HDMI-kabel.

## 1.2. Projektformulering

“Syng og sammenlign” eller SOS er et projekt der forsøger at gøre den almindelige Karaoke brugers liv lettere ved at lave et let-bærbart karaokesystem, der er nemt at transportere og kan tilsluttes til en vilkårlig skærm og højttaler ved benyttelse af et HDMI-kabel. Dette lette bærbare karaokesystem vil være udviklet på en måde hvor den digitale signalbehandling sker med afsæt i tilgængelige data der ligger på en ekstern server, i form af en Raspberry Pi Zero på samme lokale netværk, og en bruger indspilning, der ved anvendelse af en mikrofon, indspilles i real tid. En repræsentativ score, for brugerens præstation, vil blive beregnet ved brug af digital signal behandling.

Dette karaoke-system vil blive udviklet på en måde, hvor digital signalbehandling udføres ved at hente data fra den ekstern server og modtage brugerens indspilning. Dataene fra den eksterne server og brugerens indspilning sammenlignes og der beregnes en score på flere faktorer. På denne måde får brugeren ved brug af dette system oplevelsen og udfordringen ved at tage på en karaokebar i en let kompakt løsning, der kan tilsluttes via den normalt tilgængelig HDMI-forbindelse, til lyd og billede.

### **1.2.1. Formål**

Formålet med dette projekt er at modtage et analogt signal via en mikrofon, der i en passende høj kvalitet konverteres til et digitalt signal, der kan sammenlignes med en passende datamængde i form af en sang. Beregningerne forgår i kernen af systemets logiske behandling og danner rammerne for den feedback, der skal fremvises på den grafiske brugergrænseflade (GUI). Brugergrænsefladen bruges også til valg af den ønskede sang, visuel repræsentation af sangens lyrik og til sidst scoringsfaktoren, der giver en repræsentativ score for brugerens optræden.

### **1.2.2. Tekniske komponenter og funktioner**

Dette projekt gør brug af en række hardware specifikke dele, disse er, men ikke nødvendigvis eksklusivt, en Raspberry Pi 4, Raspberry Pi Zero, PSoC, og en amplifier.

#### **Raspberry Pi's (2x):**

Raspberry Pi 4'eren kommer til at fungere som den centrale komponent for den logiske behandling og sammenligning af dataene, i form af digitale signaler. Yderligere vil en Rpi Zero stå for lagring af denne data, i form af en ekstern server med en database, hvor dataen skubbes og trækkes løbende igennem systemets løbetid.

#### **PsoC & Pre-amp:**

Portable System on Chip eller PSoC, kommer til at konvertere det analoge signal til et digitalt signal og fører det videre til den centrale del af systemet på en Raspberry Pi, gennem en kablet forbindelse. Signalet vil formodeligt være for lavt i forhold til Volten, så en pre-amp inkluderes for at forstærke vores rå signal.

#### **Interfaces:**

Systemet designes til at kunne blive benyttet af vilkårlige enheder som en dynamisk XLR-Mikrofon, et usb keyboard og mus, og et HDMI-stik til skærm og højttaler til video og lyd.

### **1.2.3. Grafisk brugergrænseflade (GUI)**

Den grafiske brugergrænseflade for systemet har tre hovedfunktioner, disse er:

#### **1. Valg af en ønsket sang**

Brugeren skal via den grafiske brugergrænseflade kunne lede efter og vælge en sang, der ønskes en sammenligning med. Dette gøres ved brug af mus eller keyboard inputs til at navigere og vælge.

#### **2. Visning af tekst i form af lyrik**

Efter at brugeren har valgt den ønsket sang, vil en grafisk indikation på brugergrænsefladen signalere hvad og hvornår en tone i form af ord skal “synges” ind i mikrofonen.

#### **3. Sammenligning af data**

Den samlede sammenlignings score vil blive præsenteret til brugeren efter indspilningen af brugerens real time indspilning. Denne score vil blive fremvist på sådan en måde, at der ingen tvivl er om brugerens præstation sammenlignet med sangen.

### 1.2.4 Ansvarsområder

## 2. INDLEDNING

Punkt:	Hovedansvarlig:	Sekundæransvarlig:
Problemformulering	Alle	Alle
Projektformulering	Alle	Alle
Ansvarsområder	Alle	Alle

## 3. METODE OG PROCES

Punkt:	Hovedansvarlig:	Sekundæransvarlig:

## 4. KRAV OG PRIORITERINGER

Punkt:	Hovedansvarlig:	Sekundæransvarlig:
Funktionelle krav	Alle	Alle
Funktionelle krav	Alle	Alle

## 5. INDLEDENDE ANALYSER

Punkt:	Hovedansvarlig:	Sekundæransvarlig:
PSoC	Mikail, Hannad	
Server & Database	Emil, Simon	
Logikenheden	Max, Oliver	
Brugergrænseflade	Bassam, Fardoos	
Risikoanalyse	Alle	Alle

## 6. SYSTEMARKITEKTUR

Punkt:	Hovedansvarlig:	Sekundæransvarlig:
BDD	Alle	Alle
Blokbeskrivelse	Alle	Alle
IBD	Alle	Alle
Domænemodel	Alle	Alle
HW-arkitektur	Ale	Alle
SW-arkitektur	Alle	Alle

## 7. MODUL- OG KOMMUNIKATIONSDESIGN

Punkt:	Hovedansvarlig:	Sekundæransvarlig:
PSoC	Mikail, Hanad	
Sever	Emil, Simon	
Database-design	Emil, Simon	
Kommunikation mellem RPI4 & server	Simon	
Kommunikation mellem moduler på RPI4	Alle	
LogicUnit	Max, Oliver	
Brugergrænseflade	Bassam, Fardoos	

## 8. REALISERING AF SYSTEMET

Punkt:	Hovedansvarlig:	Sekundæransvarlig:
PSoC	Mikail, Hanad	
Server & Database	Emil, Simon	
LogicUnit	Max, Oliver	
Brugergrænseflade	Bassam, Fardoos	

## 9. TEST OG RESULTATER

Punkt:	Hovedansvarlig:	Sekundæransvarlig:
Modul- og integrationstest	Alle	Alle
PSoC	Mikail, Hannad	

## 10.DISKUSSION AF RESULTATER

Punkt:	Hovedansvarlig:	Sekundæransvarlig:
	Alle	

## 11.KONKLUSION OG PERSPEKTIVERING

Punkt:	Hovedansvarlig:	Sekundæransvarlig:
	Alle	

## **2 Metode og arbejdsproces**

I dette afsnit vil der først gennemgås de metoder, som benyttes igennem projektet forløbet. Derefter vil der gennemgås arbejdsproces hvor under gruppens samarbejde og projektstyring formuleres.

### **2.1 Metoder**

I arbejdet med krav blev der anvendt use cases til at beskrive de funktionelle krav, samt FURPS for de ikke funktionelle krav. Disse krav blev prioriteret med MoSCoW for at kunne afgrænse projektet på metodisk vis, så de vigtigste elementer af projektets laves først.

I systemarkitekturfasen blev der anvendt SysML og UML. HW-Arkitekturen var minimal for dette projekt, men blev udformet med brug af IBD og BDD-diagrammer med tilhørende blok- & signalbeskrivelser. SW-Arkitekturen indledes med en overordnet domænemodel, hvorpå der tilbygges applikationsmodel for hver processor i systemet. Disse applikationsmodeller er lavet med generelt beskrivende funktionsnavne for hver processor, som burde give en ide om hvad der skal ske på de enkelte processorer. Dette er for at holde SW-Design åbent for hvert enkelte modul, men gør at alle projektets deltagere har en klar enighed om hvad systemet gør.

I udviklingsfasen blev systemet opdelt i moduler så udviklingen kunne forløbe uafhængigt. I begyndelsen analyserede de enkelte moduler deres domæne og delte ideer med resten af gruppen i research møder om hvad de havde fundet. I disse research møde blev der generelt benyttet flow charts og andre meget løse tegninger til at beskrive hvordan systemet kunne interageres såvel som de enkelte modulers funktionalitet. Da design & implementeringsfasen begyndte blev der igen benyttet UML til at lave klassediagrammer for bedre at formulere designet.

### **2.2 Arbejdsproces**

I dette afsnit beskrives den arbejdsproces gruppen har brugt igennem projektet. Der skrives om hvordan samarbejdet har været i gruppen og hvordan projektet er blevet ledet frem. Ønskes der en mere detaljeret gennemgang af de forskellige emner, henvises læser til Bilag A

### 2.2.1 Samarbejde

Denne gruppe er blevet sammensat udelukkende af softwareteknologi studerende. Dette er baseret på individuelle præferencer og ønsker og har resulteret i to firmands grupper der er blevet slået sammen til at danne denne gruppe. Centralt for projektet har det fra starten været gældende at alle ønskede at sidde med store mængder software og dette har været centralt i beslutningsprocessen for hvilket system vi kunne tænke os at designe og implementere.

Gruppen havde en vanskelig start i takt med semestrets nye fag og arbejdsbyrde der gjorde fremmøde i projektgruppen blev glemt eller fravalgt. En samarbejdskontrakt blev skrevet, blandt nogle af medlemmerne, men den fik aldrig ben at gå på, grundet gentagende brud fra samtlige gruppemedlemmer. Dette resulterer i en fælles dialog på gruppen om det dårlige arbejdsmiljø der var kommet og den generelle lad sig færre tilgang til arbejdet.

En fælles dialog var hvad der skulle til for at vende fokus på gruppe arbejdet, og halvvejs gennem semestret var gruppen tilbage på skinner og kommunikationen mellem alle medlemmer var i tiptop.

Det har overordnet været en ruchebane tur hvad angår samarbejdet i gruppen, men efter de indledende udfordringer var overstået har der været god plads til åben dialog, direkte kommunikation og fuldt fokus på at tackle projektets udfordringer fremfor de samarbejds-mæssig.

### 2.2.2 Projektledelse

Projektledelse i en gruppe er en af de afgørende faktorer for succes i et gruppeprojekt. Til dette er der gjort brug af SCRUM, så der har kunnet blive uddelt roller som: SCRUM-Master og en ”product owner”. Til dette projekt har der været brug for at tage en utraditionel fremgangsmetode til en produktejer, da dette normalt ikke er en af de medlemmer der arbejder på selve projektet. Men ejeren af projektet. Vi valgte til denne rolle at udpege Simon til ”produkt owner”. Da der på den måde kunne simuleres processen mellem teamet der arbejder på projektet og en produktejer der har en klar vision af hvordan produktet skulle ende ud. Udover dette blev der også udpeget en SCRUM-Master, hvor Max blev udnævnt til ”SCRUM-Master”. Hvis rolle var at holde teamet op på SCRUM-principperne, og sørge for at fjerne hindringer udefra så det ikke påvirkede teamet. Igennem værktøjer som SCRUM, Tidsplan og gruppens arbejdskontrakt, har gruppen formået at

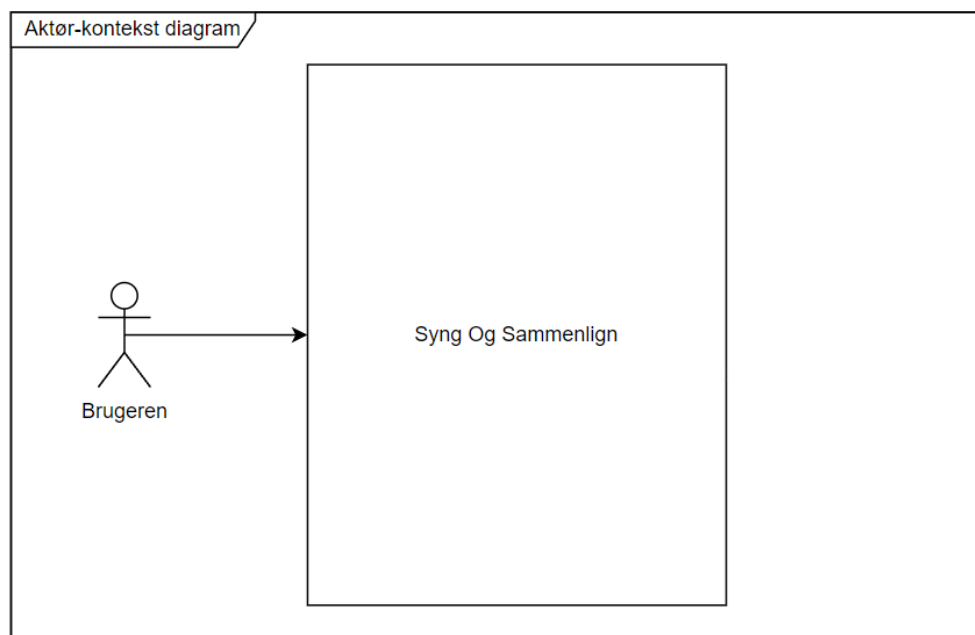
holde et godt tempo igennem projektprocessen. Håndtere konflikter når de opstod. Haft regelmæssige møder og statusjek. Og lede gruppen på rette vej mod vigtige deadlines.

### 3 Krav og prioriteringer

#### 3.1 Funktionelle Krav

##### 3.1.1 Aktør-kontekst diagram

Figur 1 herunder viser vores aktør kontekst diagram baseret på vores Use Case diagram.



Figur 1 Aktør kontekst diagram

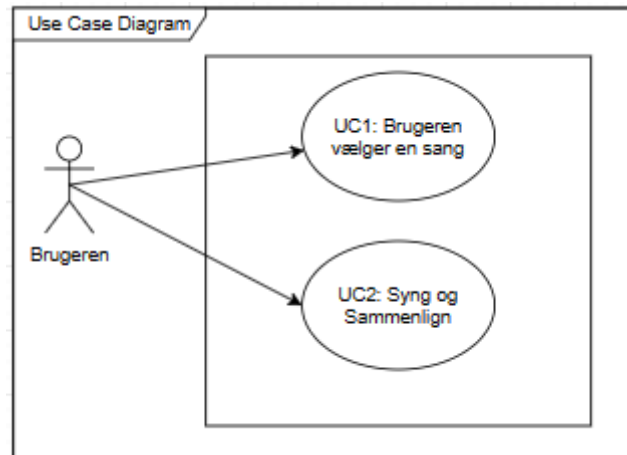
##### 3.1.2 Aktør beskrivelser

<i>Aktør Navn:</i>	Bruger
<i>Alternativ reference:</i>	Primær
<i>Type:</i>	Primær Aktør
<i>Beskrivelse:</i>	Brugeren vælger den ønskede sang, som systemet henter fra databasen og præsenterer brugerens ønskede sang, hvorefter systemet derefter starter med optagelsen og derefter scoring

Tabel 1 Aktør beskrivelser



### 3.1.3 Use Case diagram



Figur 2: Use Case Diagram

### 3.1.4 Use Case beskrivelser

- *Use Case 1: "Brugeren vælger en sang"*

Use casen "Brugeren vælger en sang" begynder med at brugeren vælger en sang igennem brugergrænsefladen, hvorefter RPi4'eren så vil hente den nødvendige sang data fra server og databasen og præsentere disse data for brugeren i form af en liste af sange. Herefter kan brugeren vælge den sang brugeren vil vælge at synge.

- *Use Case 2: "Syng og Sammenlign"*

Use casen "Syng og Sammenlign" beskriver hvordan vores sammenlignings proces foregår efter brugeren har valgt den sang de vil synge igennem brugergrænsefladen. Denne proces fortsætter indtil brugeren er færdig med at synge sin sang og har fået sin score fra systemet. Og kan i så fald starte forfra igen med udgangspunkt i Use Case 1.

### 3.1.5 Fully dressed Use Case 1: Brugeren vælger en sang

Navn:	UC1 - Brugeren vælger sang
Mål	Brugeren kan ved brug af brugergrænsefladen (GUI) vælge en ønsket sang at præstere
Initiering	Brugeren tænder systemet
Aktører	<b>Primær:</b> Brugeren
Antal samtidige forekomster	1
Prækondition	Systemet er opstartet og klar til at modtage brugerinput
Postkondition	Brugeren har valgt en sang fra valget af sanglisten
Hovedscenarie	<ol style="list-style-type: none"> <li>1. Brugeren starter programmet på <i>RPi4</i> gennem OS.</li> <li>2. <i>RPi4</i> henter sanglisten og scores fra databasen.</li> <li>3. <i>Brugergrænsefladen</i> præsenterer hovedmenuen for brugeren.</li> <li>4. Brugeren navigerer igennem <i>brugergrænsefladen</i> ved brug af piletasterne og/eller mus.</li> </ol> <p><b>[Ext 1: Brugeren klikker på "Best scores"]</b></p> <ol style="list-style-type: none"> <li>5. Brugeren klikker på elementet "Sange".</li> <li>6. Brugeren præsenteres med en liste af <i>sange</i>.</li> </ol> <p><b>[Ext 2: Brugeren klikker på "Back"]</b></p> <ol style="list-style-type: none"> <li>7. Brugeren vælger den ønskede <i>sang</i>.</li> <li>8. <i>RPi4</i> gør klar til indspilning via mikrofonen.</li> </ol>
Udvidelser/undtagelser	<p><b>[Ext 1: Brugeren klikker "Best scores"]</b></p> <ul style="list-style-type: none"> <li>• Der bliver præsenteret de 5 højeste scores der har været indspillet for alle sange.</li> <li>• Brugeren trykker "Tilbage til hovedmenuen".</li> <li>• UC1 Fortsætter fra punkt 3.</li> </ul> <p><b>[Ext 2: Brugeren klikker på "Back"]</b></p> <ul style="list-style-type: none"> <li>• Brugeren går tilbage til forrige vindue.</li> </ul>
Datavariationer	<p><b>Sange:</b> Sange dækker over den række af lydfiler, der er tilgængelige fra databasen og bliver præsenteret for brugeren ved brug af den grafiske brugergrænseflade.</p> <p><b>RPi4:</b> Den primære enhed for systemet, hvor på programmet køre.</p> <p><b>Brugergrænseflade:</b> Den del af vores system som står for de grafiske elementer.</p>

Tabel 2: Fully dressed UC1

### 3.1.6 Fully dressed use Case 2: Syng og Sammenlign

Navn:	UC2 - Syng og Sammenlign (SOS)
Mål	Indspille sang og få den sammenlignet med sangens originale version, og derved modtage en score der repræsentativt afspejler brugerens input
Initiering	Initiering forekommer efter brugeren har valgt en sang (Fortsættelse på UC1)
Aktører	<b>Primær:</b> Brugeren
Antal samtidige forekomster	1
Prækondition	Systemet er klar til at modtage brugerinput via mikrofonen (Afsluttet UC1)
Postkondition	Brugeren har indspillet den tilvalgte sangen, og fået fremvist sin score via brugergrænsefladen og kommet tilbage til hovedmenuen
Hovedscenarie	<ol style="list-style-type: none"> <li>1. Systemet initialiserer en nedtælling på X sekunder.</li> <li>2. <i>Brugergrænsefladen</i> viser lyrik i takt med instrumental.</li> </ol> <p><b>[Ext 1: Brugeren klikker på "Cancel"]</b></p> <ol style="list-style-type: none"> <li>3. Brugeren forsøger at ramme de rigtige toner.</li> <li>4. Når sangen er færdig, sammenlignes <i>lydfilerne</i>.</li> <li>5. <i>Brugergrænsefladen</i> viser en score i form af en numerisk værdi.</li> </ol> <p><b>[Ext 2: Brugeren klikker på "New song"]</b></p> <ol style="list-style-type: none"> <li>6. Brugeren klikker tilbage til hovedmenuen.</li> </ol>
Udvidelser/Undtagelser	<p><b>[Ext 1: Brugeren klikker på "Cancel"]</b></p> <ul style="list-style-type: none"> <li>• Brugeren afbryder sin indspilning og vender tilbage til sanglisten</li> </ul> <p><b>[Ext 2: Brugeren klikker på "New song"]</b></p> <ul style="list-style-type: none"> <li>• Brugeren bliver præsenteret for de valgmulige sange via grænsefladen.</li> </ul> <p>Brugeren vælger en ny sang og går tilbage til UC 1 punkt 6.</p>
Datavariationer	<p><b>X:</b> Et dynamisk antal sekunder fra melodien starter til det første brugerinput via mikrofonen, med henblik på at ramme timingen bedst.</p> <p><b>Lydfilerne:</b> De filer fra databasen, der skal sammenlignes for at udregne en repræsentativ score af brugerens input.</p> <p><b>Brugergrænseflade:</b> Den del af vores system som står for de grafiske elementer.</p>

Tabel 3: Fully dressed UC2

## 3.2 Funktionelle krav

### 3.2.1 FURPS

#### [F] Functionality (Funktionaltitet)

- 1.1. Systemet **skal** kunne konvertere det analoge signal fra mikrofonen til et digitalt signal.
- 1.2. Systemet **skal** sammenligne den behandlede signal-data med den prædefinerede data fra serveren.
- 1.3. Systemet **skal** kalkulere en score ud fra sammenligningen af de to signaler
- 1.4. Serveren **skal** være stabil og tilgængelig så den kan levere lydfiler, sammenligningsscores, sangnavne og generelt data vedrørende sange og logistik.
- 1.5. Systemets brugergrænseflade **skal** kunne navigere igennem systemets vinduer.
- 1.6. Brugeren **skal** kunne anvende brugergrænsefladen til at vise "Best Scores" vha. systemets vinduer.
- 1.7. Systemet **skal** have mindst 2 forskellige sange at vælge imellem.
- 1.8. Systemet **skal** igennem Brugergrænsefladen kunne vise sangteksten i takt med musikken.
- 1.9. Systemet **skal** kunne give en score som vises via brugergrænsefladen.
- 1.10. Systemet **skal** gemme scoren efter indspilning og sende den til databasen.
- 1.11. Systemet **burde** kunne transponere tonelejet, så oplevelsen er mere tilgængeligt.
- 1.12. "Guide Melody" - Systemet **kunne** spille den originale melodi i en entydig lyd, som brugeren kan følge.
- 1.13. "Live Score Feedback" - Systemet **kunne** have en visualisering til brugeren der viser sangerens udregnede score undervejs.
- 1.14. Der **kunne** være API integration med musiktjenester som Spotify eller Apple Music.

#### [U] Usability (Brugervenlighed)

- 1.1. Systemet **kunne** have mulighed for at tilføje flere mikrofoner for således at synge "duet".
- 1.2. Systemet **kunne** have en indbygget højttaler, således der kun var behov for en skærm.
- 1.3. Systemet **kunne** have en raffineret brugerdatabase for at gemme tidligere scorere og rekorder ift. brugere.

#### [R] Reliability (Pålidelighed)

- 1.1. Systemet **skal** være stabilt og pålideligt. Således at der ikke er nedbrud under brugerens indspilning.
- 1.2. Systemet **burde** fungere funktionelt uden afbrydelser eller nedbrud, i mindst 5 minutter.

*[P] Performance (Ydeevne)*

- 1.1. Systemet **burde** være hurtigt nok, så der ikke går mere end 30 sekunder efter indspilningen for at få sin score.

Indbygget højttaler. Systemet **kunne** have en indbygget højttaler, således der kun var behov for en skærm.

*[S] Supportability (Supportmuligheder)*

Der er ikke blevet implementeret nogen supportmuligheder for dette system.

### 3.2.2 MoSCoW

*[M] Must have*

For at systemet har de basale funktioner der skal til for at have en velfungerende prototype, må følgende være opfyldt:

- 1.1 Systemet **skal** kunne konvertere det analoge signal fra mikrofonen til et digitalt signal.
- 1.2 Systemet **skal** sammenligne den behandlede signal-data med den prædefinerede data fra serveren.
- 1.3 Systemet **skal** kalkulere en score ud fra sammenligningen af de to signaler
- 1.4 Systemet **skal** være stabil og pålideligt. Således at der ikke er nedbrud under brugerens indspilning.
- 1.5 Serveren **skal** være stabil og tilgængelig så den kan levere lydfiler, sammenligningsscores, sangnavne og generelt data vedrørende sange og logistik.
- 1.6 Systemets brugergrænseflade **skal** kunne navigere igennem systemets vinduer.
- 1.7 Brugeren **skal** kunne anvende brugergrænsefladen til at vise "Best Scores" vha. systemets vinduer.
- 1.8 Systemet **skal** have mindst 2 forskellige sange at vælge imellem.
- 1.9 Systemet **skal** igennem Brugergrænsefladen kunne vise sangteksten i takt med musikken.
- 1.10 Systemet **skal** kunne give en score som vises via brugergrænsefladen
- 1.11 Systemet **skal** gemme scoren efter indspilning og sende den til databasen.

*[S] Should have*

Systemet kan have stor gavn af at implementere nogle ekstra kvaliteter for at give en fuldere og bedre oplevelse af anvendelsen:

- 1.1 Systemet **burde** fungere funktionelt uden afbrydelser eller nedbrud, i mindst 5 minutter.
- 1.2 Systemet **burde** være hurtigt nok, så der ikke går mere end 30 sekunder efter indspilningen for at få sin score.
- 1.3 Systemet **burde** kunne transponere tonelejet, så oplevelsen er mere tilgængeligt.

### *[C] Could have*

Overvejelser omkring videreudvikling af systemet kan indebære følgende:

- 1.1 Systemet **kunne** have mulighed for at tilføje flere mikrofoner for således at synge "duet".
- 3.2 Systemet **kunne** have en indbygget højtaler, således der kun var behov for en skærm.
- 3.3 Systemet **kunne** have en raffineret brugerdatabase for at gemme tidligere scorere og rekorder ift. brugere.
- 3.4 "Guide Melody" - Systemet **kunne** spille den originale melodi i en entydig lyd, som brugeren kan følge.
- 3.5 "Live Score Feedback" - Systemet **kunne** have en visualisering til brugeren der viser sangerens udregnede score undervejs.

### *[W] Won't have*

Systemets størrelse og scope gør at det på nuværende tidspunkt ikke kunne tænke sig at have med

- 4.1 Der **kunne** være API integration med musiktjenester som Spotify eller Apple Music.

## 4 Indledende analyser

### 4.1 Overordnede teknologiske valg

Til udregning af scoren har det været et spørgsmål om hvilket program, der kunne være i stand til at importere og analysere de lydfile der ville skulle bruges i applikationen. Dette gjorde sig gældende når, der skulle udregnes en samlet score for brugerens indspilning. Denne skulle kunne sammenlignes med den valgte sang, og det tilhørende sample for den valgte sang. Af disse grunde udfoldede tanken om at bruge andre sprog end C++ til programmet, selvom det ellers er dette sprog der indtil nu har været mest undervisning og støtte i. Den bevægede sig over imod kodesproget Python, som minder om sproget fra Matlab, som der tidligere er blevet brugt i DSB faget, til netop analyse og import af lydfile. Udover dette har Python også adgang til en stor mængde af libraries, der gjorde det endnu simplere at skabe den ønskede funktionalitet som Matlab havde i Python.

Til systemets server og database er der hovedsageligt blevet brugt C++, med små praj af Python. Der er endvidere også blevet gjort brug af SQLite3 til databasen, da dette er en af de nemmeste databaser at lære og gøre brug af. Da SQLite3 allerede har alt den ønskede funktionalitet indbygget var det et indlysende valg at benytte.

For systemets GUI var der en række designovervejelser der skulle træffes. Målet med systemets GUI var at få lavet en intuitiv platform for brugeren, så de nemt ville kunne navigere, samt bruge systemet, uden at skulle have nogen forhåndsviden eller være særlig "tech-skarp". Et værktøj der

kunne opfylde disse krav og samtidig de behov, der var for funktionalitet for GUI'en var programmet Qt Creator. Fordelen ved at vælge Qt Creator var at dette program læner sig rigtig meget op ad C++. Qt Creator krævede dog stadig noget tilvænning og tid med programmet.

For systemets PSoC er der blevet gjort brug af SPI-kommunikation, for at kunne forbinde PSoC'en med resten af systemet. Til dette valg var der også alternativer som UART og I2C. men SPI blev valgt, da SPI er kendt for sin højere dataoverførselshastighed sammenlignet med UART og I2C. Dette var en vigtig faktor, for at systemet kunne opnå den ønskede funktionalitet. Til brug af SPI-kommunikationen er der også blevet gjort brug af Bcm2835 biblioteket som valg, over WiringPi til styring af SPI-kommunikationen på Raspberry Pi. Der var desuden et tilsvarende Python bibliotek, til styring af GPIO pins, men da dette blev opdaget relativt sent, kunne dette ikke afprøves.

## 4.2 Risikoanalyse for teknologiske valg

En omhyggelig risikoanalyse er blevet udarbejdet for de valgte teknologier, med fokus på nøgleovervejelser, sandsynlige scenarier og de mest alvorlige konsekvenser i tilfælde af uheld. Til bestemmelse af risiko-score er der blevet taget udgangspunkt i følgende model:

		Impact →				
		Negligible	Minor	Moderate	Significant	Severe
Likelihood ↑	Very Likely	Low Med	Medium	Med Hi	High	High
	Likely	Low	Low Med	Medium	Med Hi	High
	Possible	Low	Low Med	Medium	Med Hi	Med Hi
	Unlikely	Low	Low Med	Low Med	Medium	Med Hi
	Very Unlikely	Low	Low	Low Med	Medium	Medium

Figur 3 Risiko analyse

Anvendte software teknologier			
teknologi valg:	Sandsynlighed	Konsekvens	Risiko-score
GUI: C++	Der er sandsynligt at noget skulle gå galt med implementering af en brugergrænseflade, da dette er et nyt teknologiområde for gruppen.	Ikke at kunne få implementeret en brugergrænseflade er betydelige for projektet, da det er svært at synge karaoke uden tekst eller melodi.	Medium-High
SERVER: C++	Med afsæt i en implementering tidligere aflevering på semestret forudsætter vi at der ikke kommer til at være de store problemer med denne og at problemer er usandsynlige	Det er et mindre problem, hvis denne forbindelse ikke virker, da databasen kan flyttes og implementeres anderledes. Så serveren ikke er en nødvendighed	low
DATABASE: SQL-LITE3	Der er en moderat sandsynlighed for at noget skulle gå galt da dette er en ny teknologi for gruppen.	Konsekvenserne for at denne del ikke virker er moderate i det nuværende scope, men ekstreme i et større scope. Da det ville fylde for meget at gemme ting ellers.	Medium-high
Databehandling / udregning af score: Python	Det er usandsynligt at noget skulle gå galt her. Trods at Python er en ny teknologi er det supportet med en række biblioteker og et fællesskab der gør hjælp let at hente.	Det har kritiske konsekvenser for vores projekt hvis ikke dette modul fungerer. Da den står for at udregne scoren for sang sessionen.	Medium-high

Tabel 4: Anvendte Software Teknologier

Indledende analyser kan findes i bilag B, samt den udvidede risikoanalyse som kan findes i Bilag C.



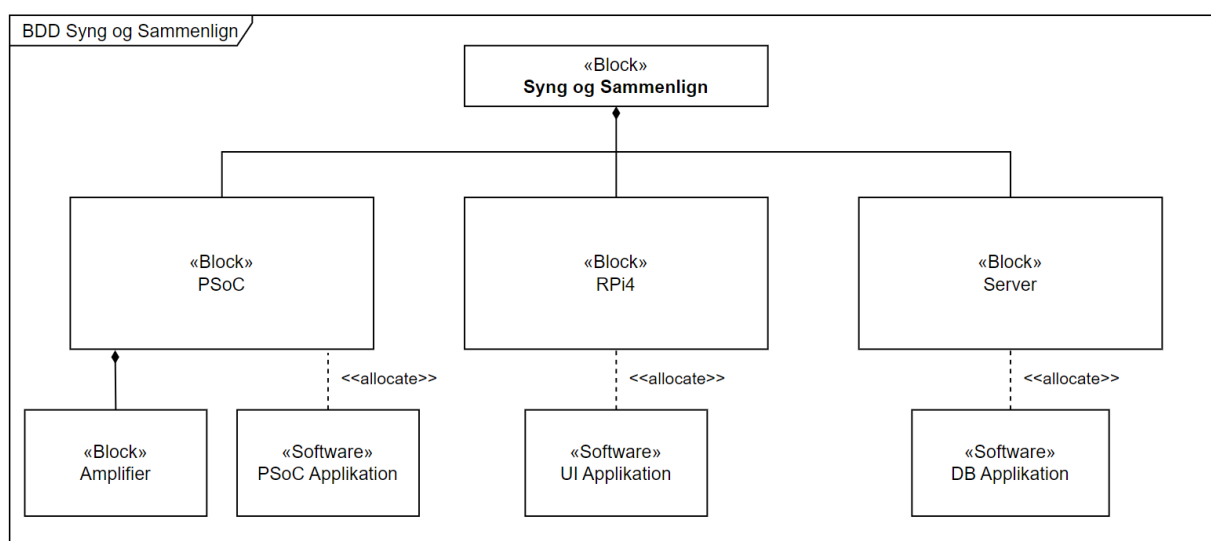
## 5 Systemarkitektur

### 5.1 Overordnet system

I dette afsnit vil arkitekturen for vores system blive beskrevet. SysML-arkitektur og design er blevet brugt som det grundlæggende fundament i oprettelsen af følgende diagrammer.

#### 5.1.1 BDD

På nedenstående figur kan et overblik over vores BDD ses. Denne figur er en overordnet repræsentation af vores system og inkluderer kun de mest essentielle dele:



Figur 4: BDD for overordnede system

#### 5.1.2 Blokbeskrivelse

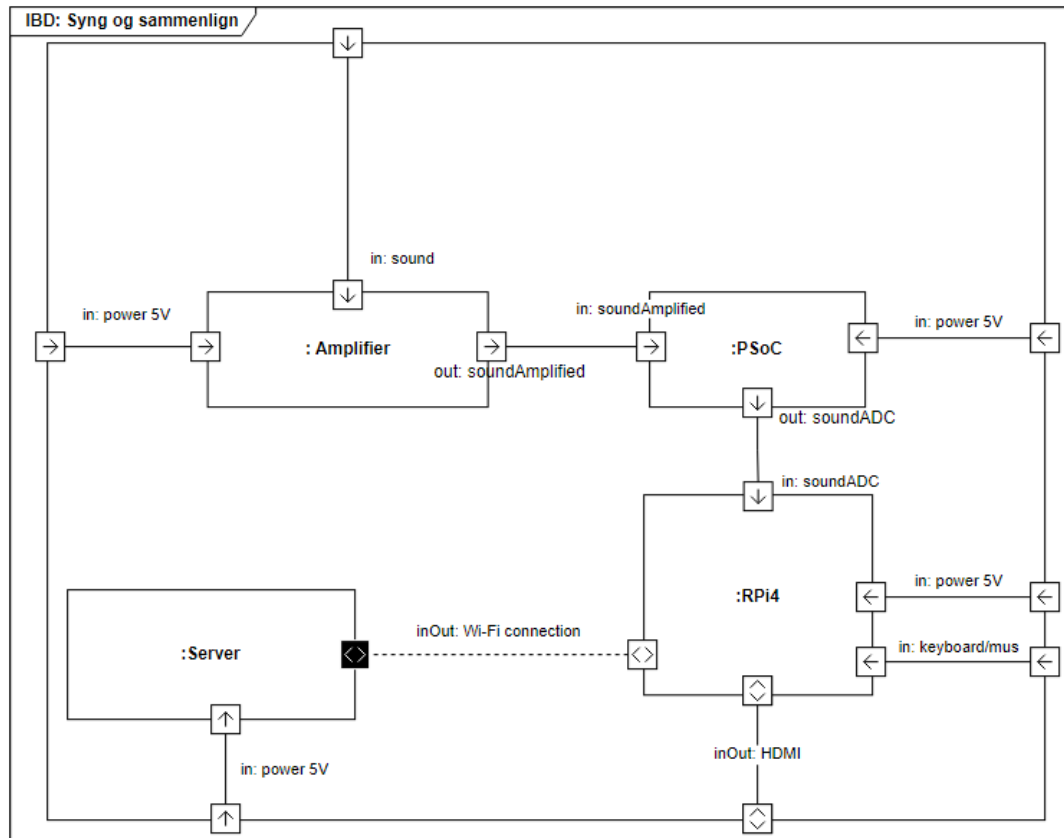
Der laves en tabel over de forskellige blokke i vores BDD, således en bedre opfattelse opnås:

Bloknavn	Beskrivelse
<b>PSoC</b>	PSoC'en er den sekundære hjerne af systemet, da den står for ADC-konvertering og SPI-transmitting af indkommende samples fra forstærkeren.
<b>RPi4</b>	RPi4'eren er hovedhjernen af systemet og står for at .wav formatere indkommende samples fra PSoC'en, hente relevant data fra serveren, samt sørge for logisk behandling af data.
<b>Server</b>	Serveren står for opbevaring af diverse data og fungerer som database for det overordnede system. Den kommunikerer også med RPi4 og sender / modtager data.
<b>Amplifier</b>	Amplifieren står for forstærkning af indkommende analogt signal fra mikrofonen og afsendelsen af dette til PSoC'en.

Tabel 5 Blokbeskrivelse

### 5.1.3 IBD

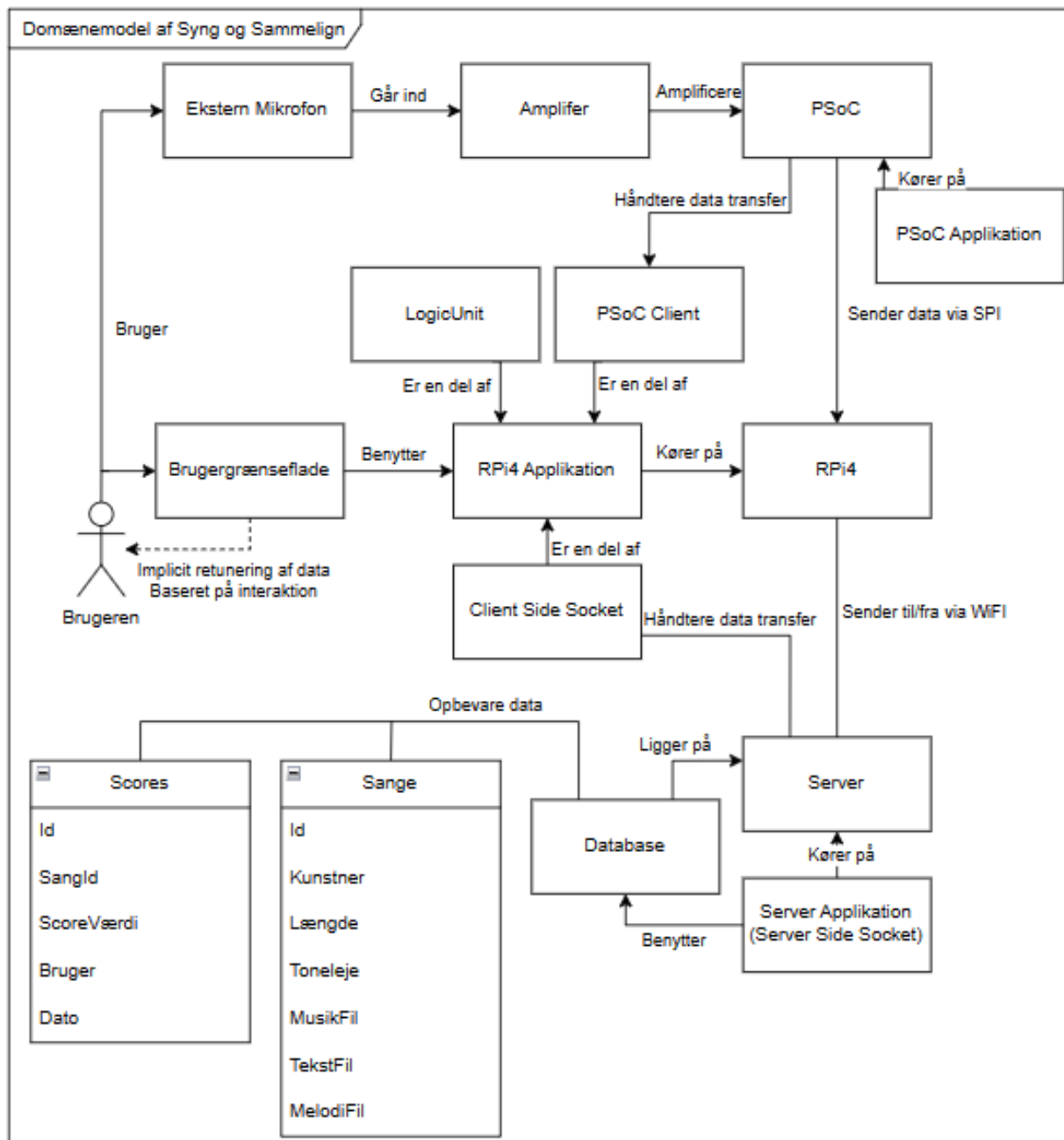
IBD'en fremviser hvordan de interne blokke spiller sammen og hvordan kommunikationen mellem disse fungerer på et dybere plan. Denne er lavet i sammenhæng med BDD'en og benytter sig af samme bloknavne:



Figur 5: IBD for overordnet system

### 5.1.4 Domænemodel

Domænemodelen er lavet på baggrund af projektets UC'er og den overordnede systemarkitektur. En udvidet domæneanalyse kan findes i bilag F, Multiplicitet er implicit sat som en - en. Da der kun er data i form af sange og scores som har anden relation, men da de ligger på databasen vil dette forklares yderligere i design fasen. Projektet bliver her også mere modulariseret så de enkelte problemer kan takles, så som ved introduktionen af 'LogicUnit'.

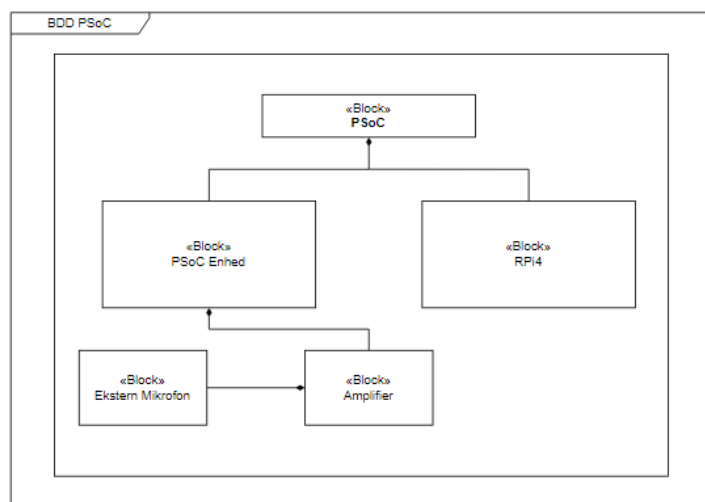


Figur 6: Domænemodel for det overordnede system

## 5.2 HW Arkitektur

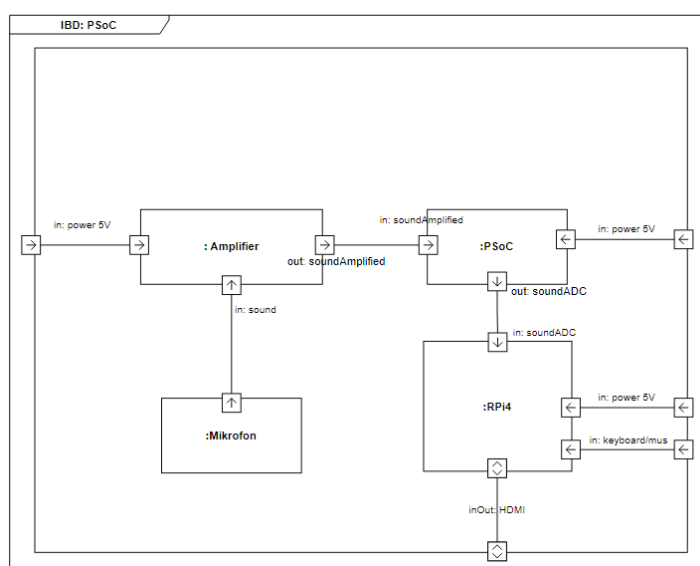
Da gruppen bestod af fuld SW-Studerende betyder det at projektet består af en minimal mængde HW. Derfor er den eneste reelle HW Komponent som videre bygges på med kabler i form af PSoC modulet som beskrives i følgende.

På nedenstående figur kan et overblik over vores BDD for PSoC modulet ses. Denne figur er en overordnet repræsentation af hvad modulet indebærer og inkluderer udelukkende det essentielle:



Figur 7: PSoC BDD

På nedenstående figur kan et overblik over vores IBD for PSoC modulet ses. Figuren fremviser hvordan de interne blokke spiller sammen og hvordan kommunikationen mellem enkelte blokke fungerer. Denne er lavet i sammenhæng med BDD'en og benytter sig af samme bloknavne for overensstemmelse:



Figur 8: PSoC IBD

### 5.2.1 Signalbeskrivelser

På nedenstående tabel er der signalbeskrivelser af IBD'en, for at danne overblik over, hvad hver signaltype gør:

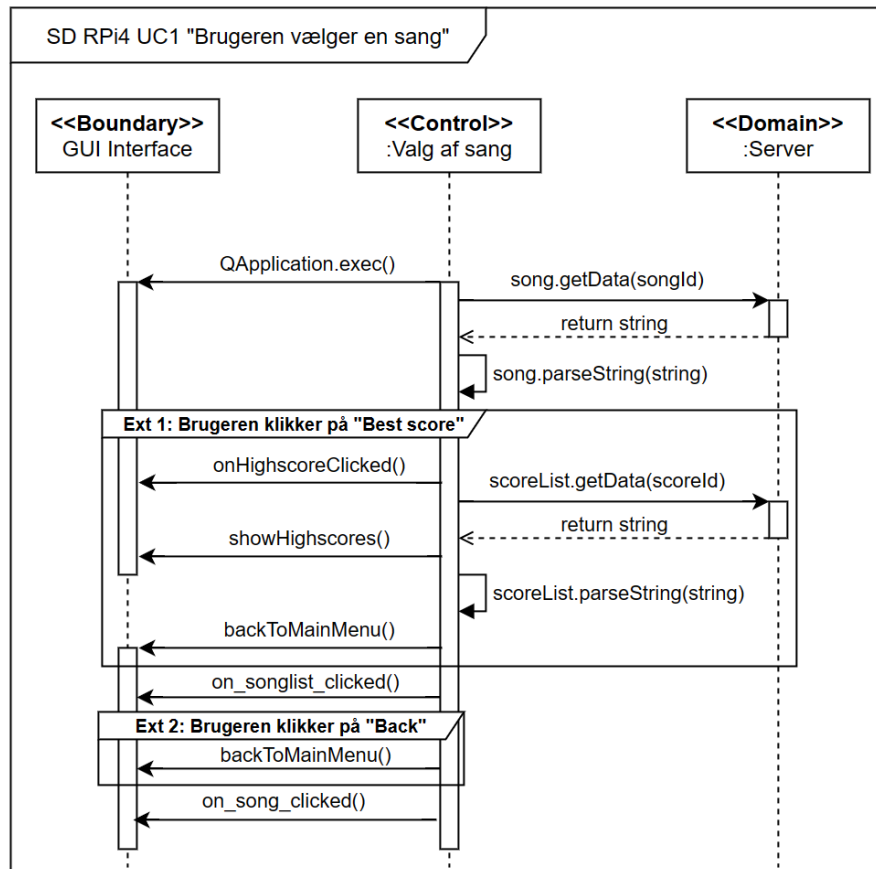
Signaltype	Funktionsbeskrivelse	Område	Port 1 (Source/Out)	Port 2 (Destination/In)
Sound	Lydsignal fra mikrofon	Mikrofon	Mikrofon	Amplifier
Power 5V	5V strømforsyning	Ekstern	Power Supply (intern i RPI4)	Amplifier, PSoC, & RPi4
SoundAmplified	Forstærket signal	Amplifier	Amplifier	PSoC
Keyboard/mus	Brugerinput fra tastaturet og/eller musen	Ekstern	Graphical User interface	RPI4
HMDI	Forbindelser for mus, skærm og tastatur til RPI4	Ekstern	Ekstern input enheder	RPI4
soundADC	Lydsignal med ADC-behandling	PSoC	PSoC	RPI4

*Tabel 6: IBD Signalbeskrivelser*

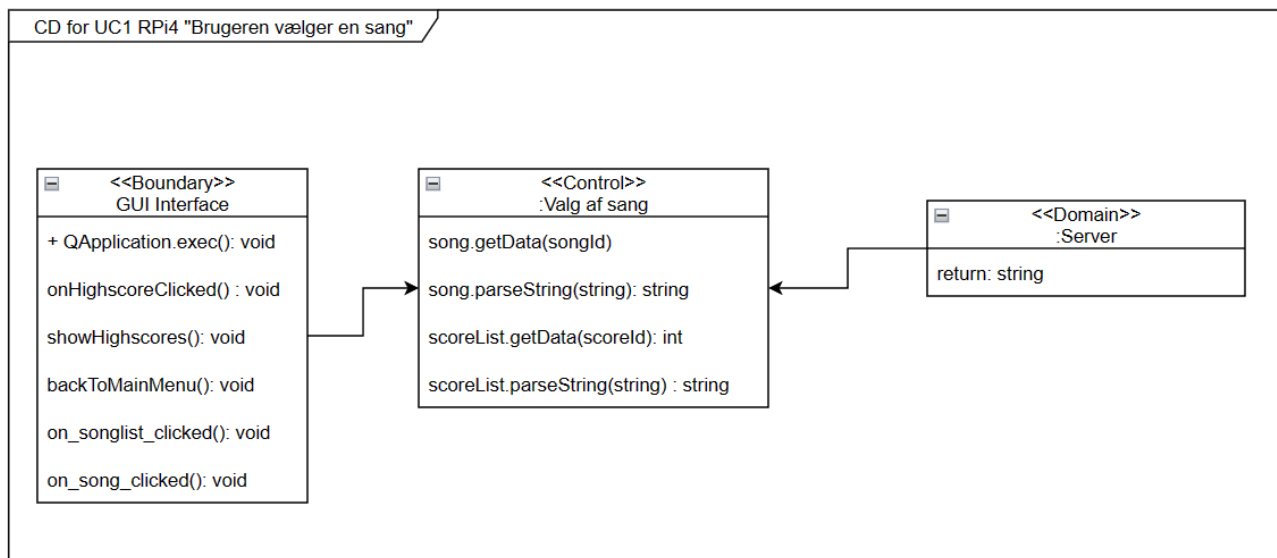
### 5.3 SW Arkitektur

For softwarearkitekturen fortages der en applikationsmodel med SD og CD for hver processor i projektets system, dvs. RPI4, Server & PSoC. Applikationsmodellen fortages på grundlag af den UC og den overordnede systemarkitektur, men ignorere modulerne fra domænemodel for at holde den overordnede ide om systemet klar og koncis.

### 5.3.1 RPI4 UC1

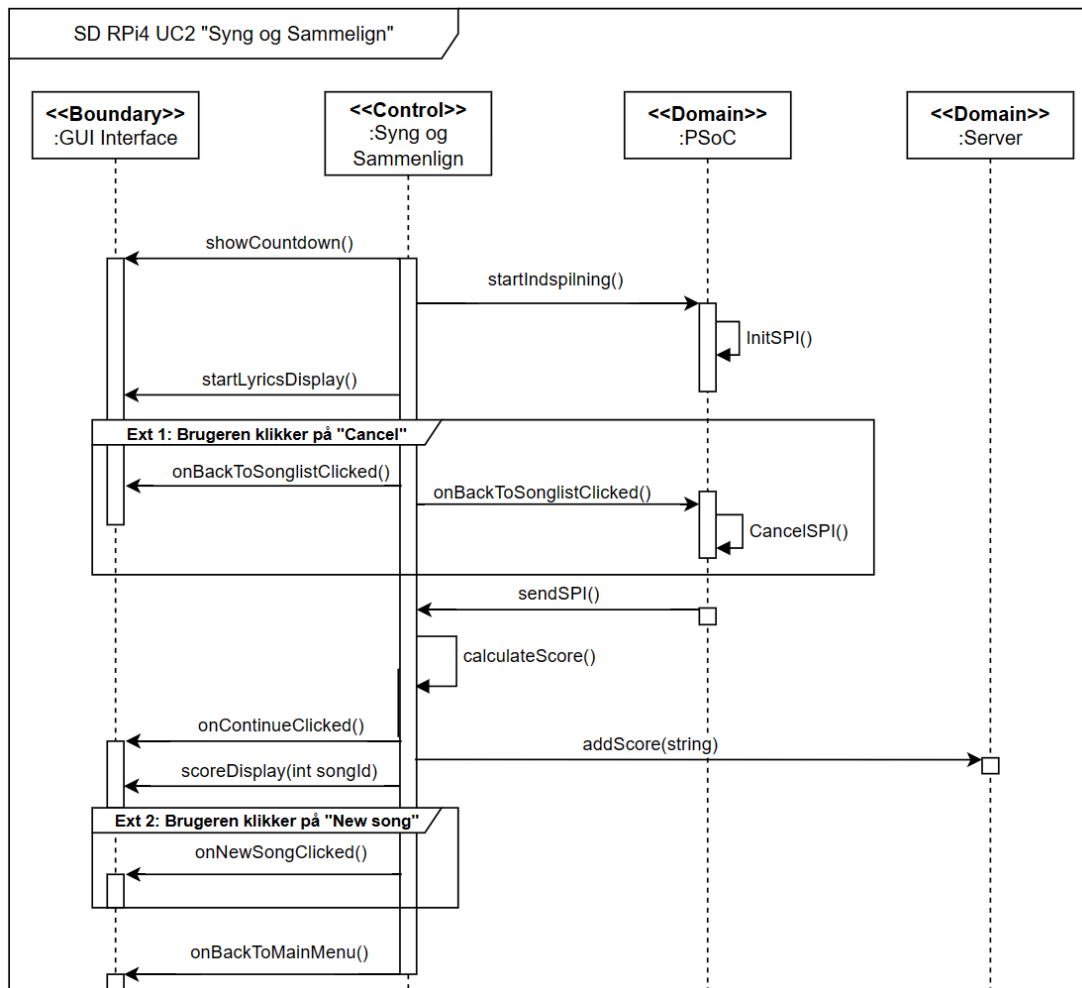


Figur 9: SD RPi4 UC1

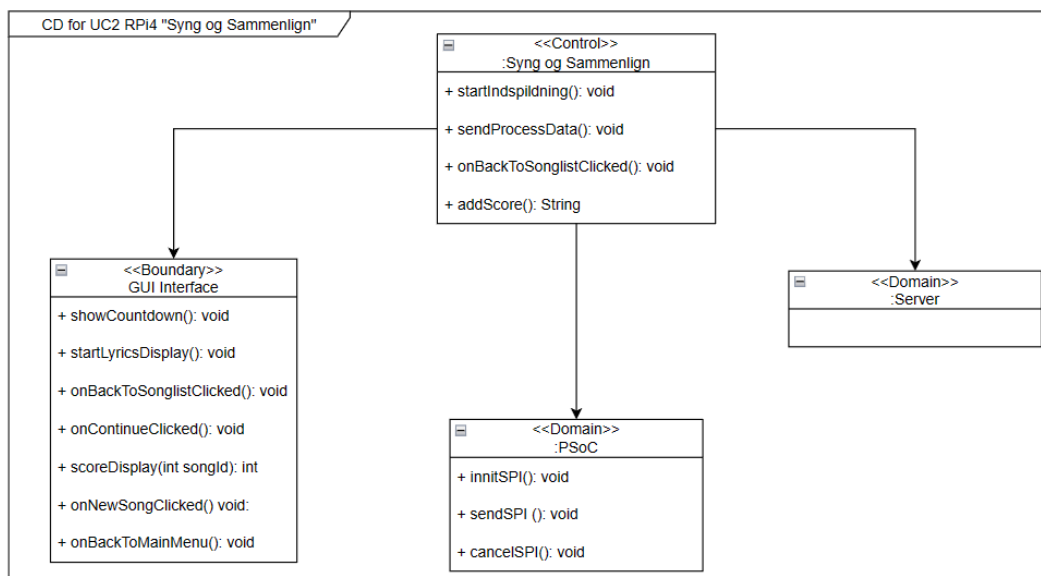


Figur 10: CD RPi4 UC1

## UC2

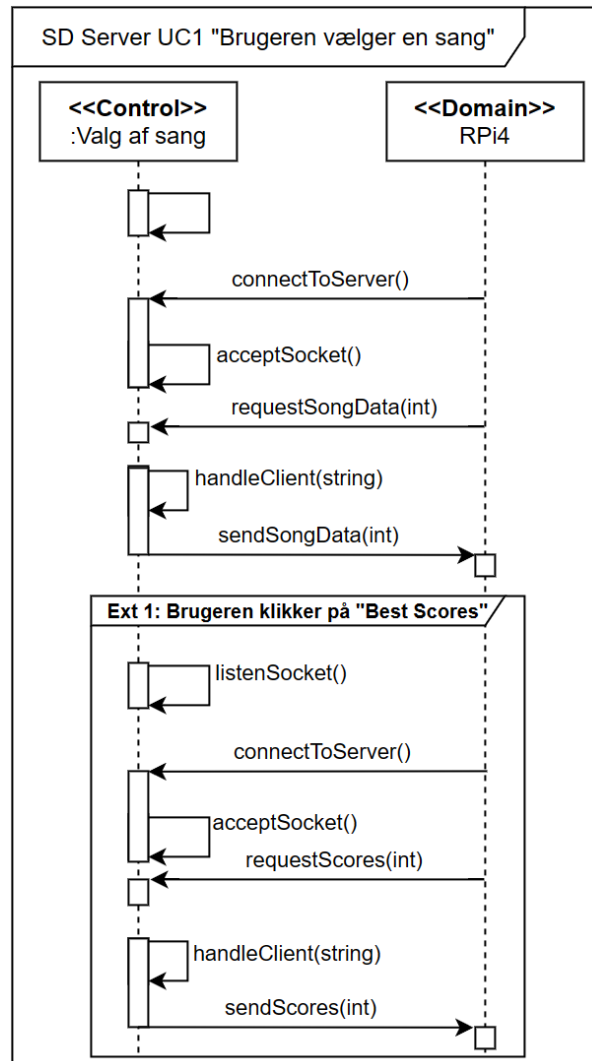


Figur 11: SD RPi4 UC2

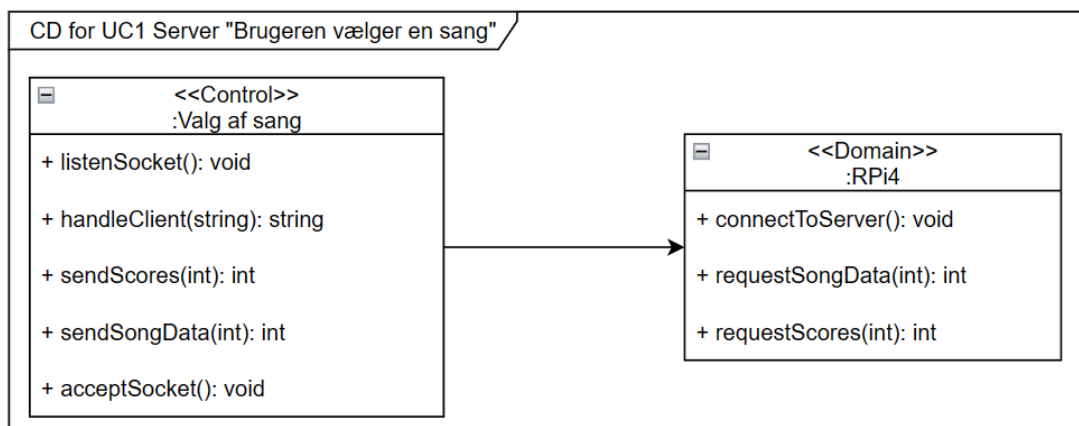


Figur 12: CD RPi4 UC2

### 5.3.2 Server UC1



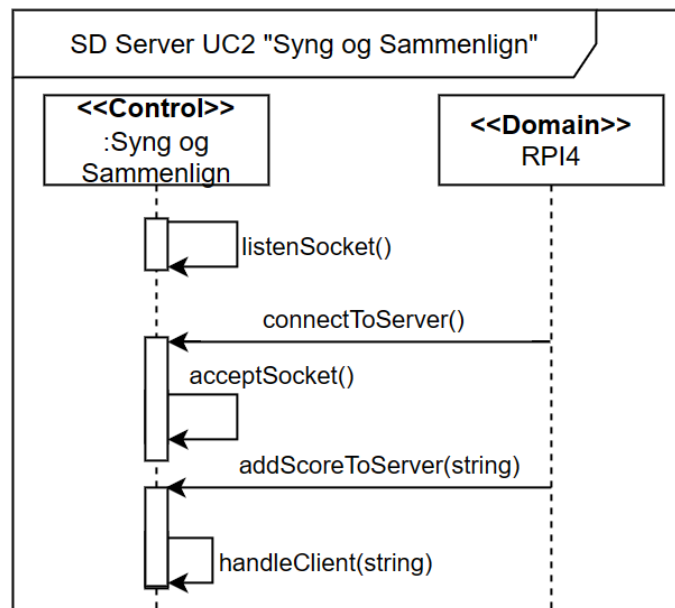
Figur 13: SD Server UC1



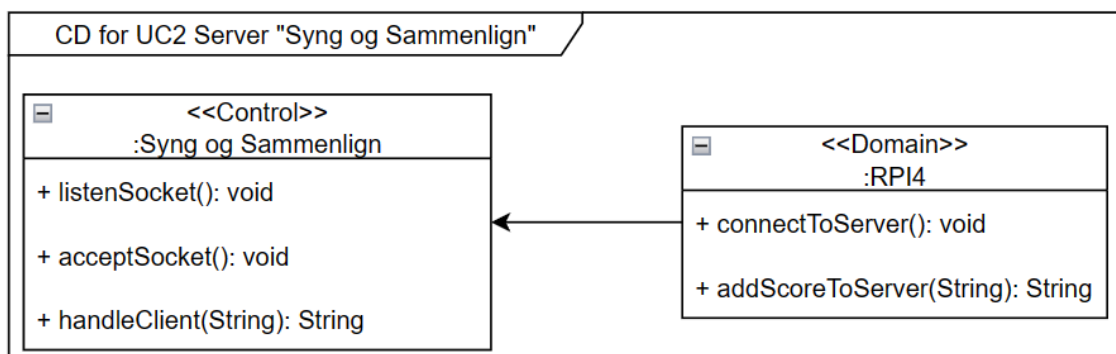
Figur 14: CD Server UC1



## UC2

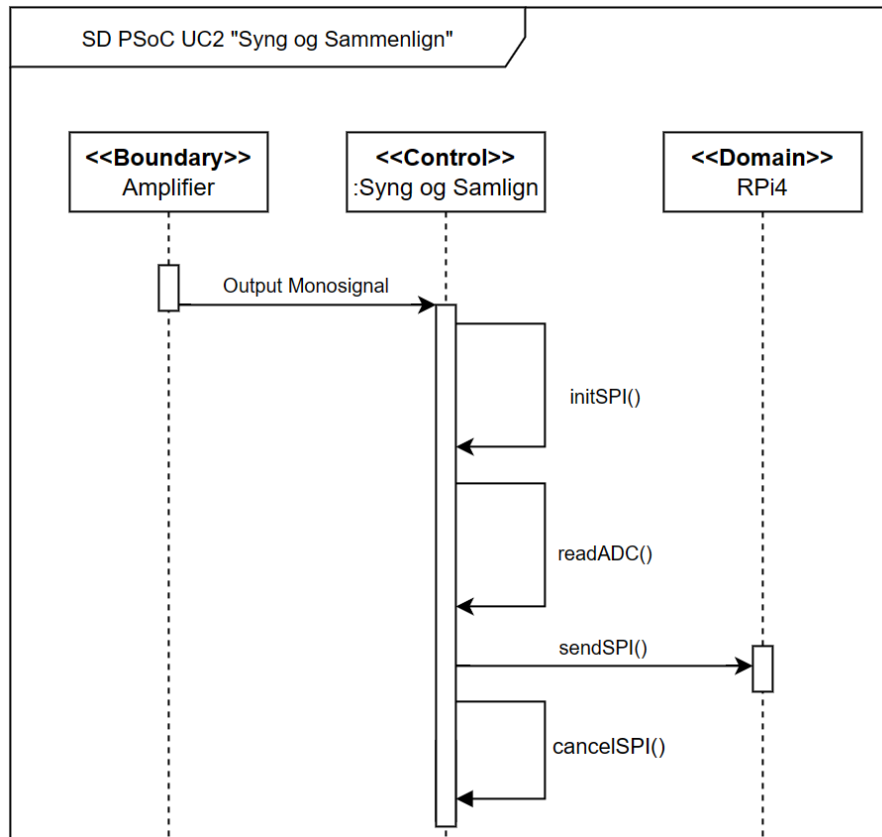


Figur 15: SD Server UC2

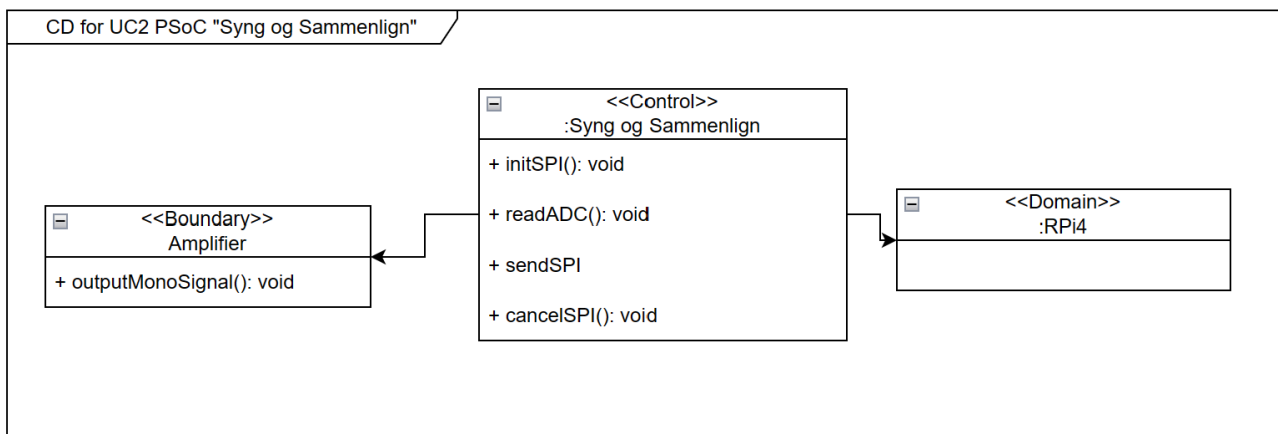


Figur 16: CD Server UC2

### 5.3.3 PSoC UC2



Figur 17: SD for PSoC UC2



Figur 18 CD for PSoC UC2

## 6 Modul- og kommunikationsdesign

Denne sektion af rapporten vil forgå som en kombineret refleksion af projektets system, igen da PSoC er den eneste reelle form for hardware i systemet. Denne tilgang vil formode en bedre oversigt over sammen spillet mellem HW & SW-design for PSoC modulet.

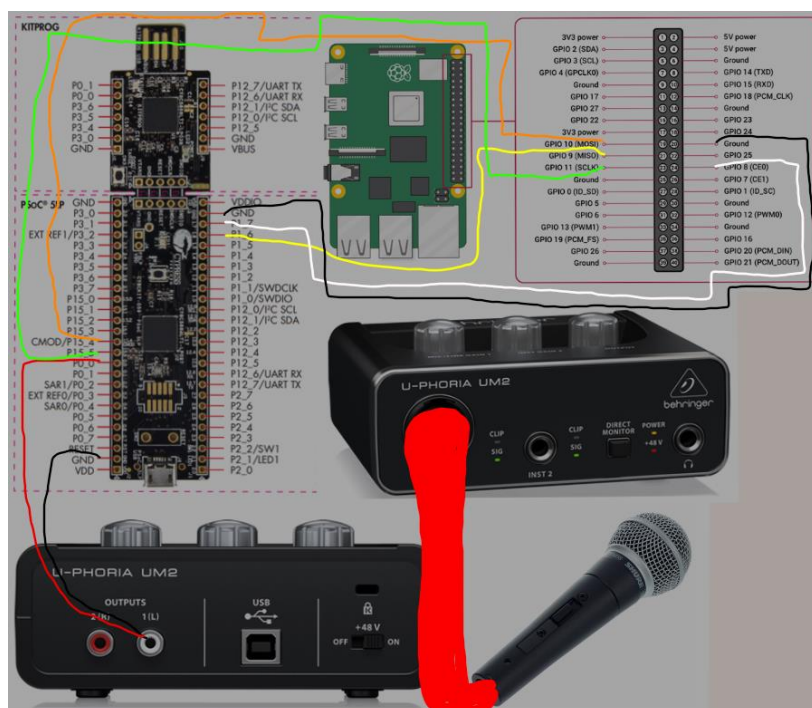
### 6.1 PSoC

I denne del-sektion vil der dykkes dybere ned i PSoC modulet, hvordan det fungerer ift. Hardwaren og Softwaren, samt hvad tankeprocessen var under designfasen.

#### 6.1.1 HW-design

I dette afsnit vil SOS' hardware design blive præsenteret.

I forbindelse med udarbejdelsen af hardwaren blev der designet et sketch (figur 19) over opstillingen for at visualisere forbindelserne med farvekodning, med formålet at danne bedre overblik over systemet som helhed og kommunikationen mellem komponenterne. Her blev RPIZ'en ikke inkluderet, da den er forbundet med RPI4'eren over Wi-Fi.

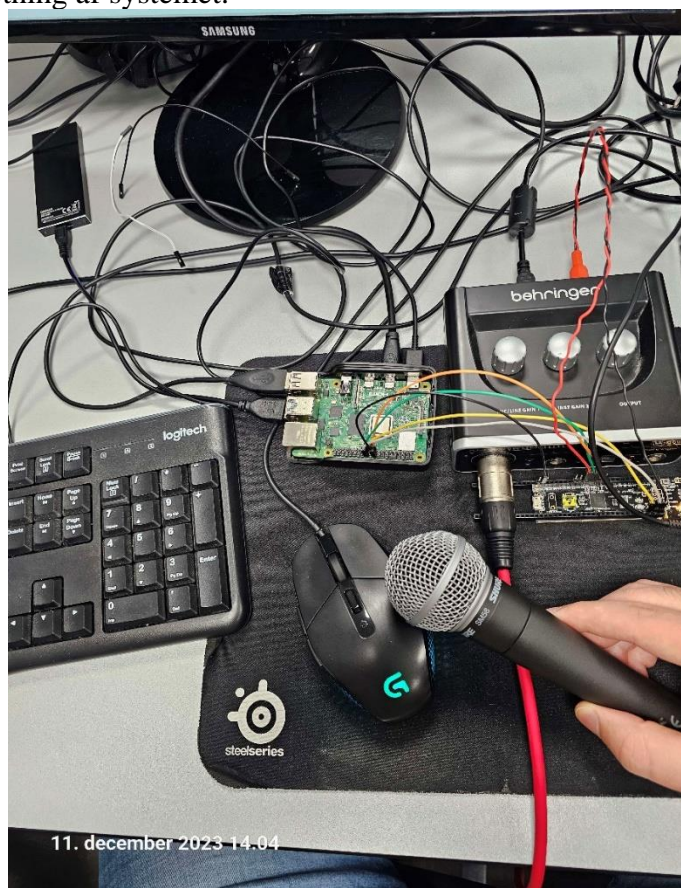


Figur 19: Hardware sketch

Angående HW-design består systemet af en Shure SM58S-mikrofon, U-Phoria UM2-forstærker, PSoC-5LP Prototyping Kit (CY8CKIT-059), og en Raspberry Pi 4 Model B.

Mikrofonen og forstærkeren er korrekt tilsluttet og konfigureret, sikret ved korrekt montering og tilslutning. PSoC er forbundet til mikrofonen via et produceret Phono-stik, og tilslutningen mellem PSoC og Raspberry Pi er oprettet ved hjælp af ledninger og Serial Peripheral Interface (SPI-protokollen).

Herunder ses den opsætning af systemet.



Figur 20: Opsætning af systemet (PSoC & RPi4).

Herunder ses farvekodning af forbindelserne mellem komponenterne:

<i>Ledning</i>	<i>Farve</i>	<i>Forbindelse mellem</i>	<i>Signal Type</i>
	Sort	Alle	GND
	Orange	PSoC og RPi4	MOSI
	Grøn	PSoC og RPi4	SLCK
	Gul	PSoC og RPi4	MISO
	Grå	PSoC og RPi4	SS/CS
	Rød (størrelse: lille)	Amplifier og PSoC	Forstærket Analog Signal
	Rød (størrelse: stor)	Mikrofon og Amplifier	Analog Signal

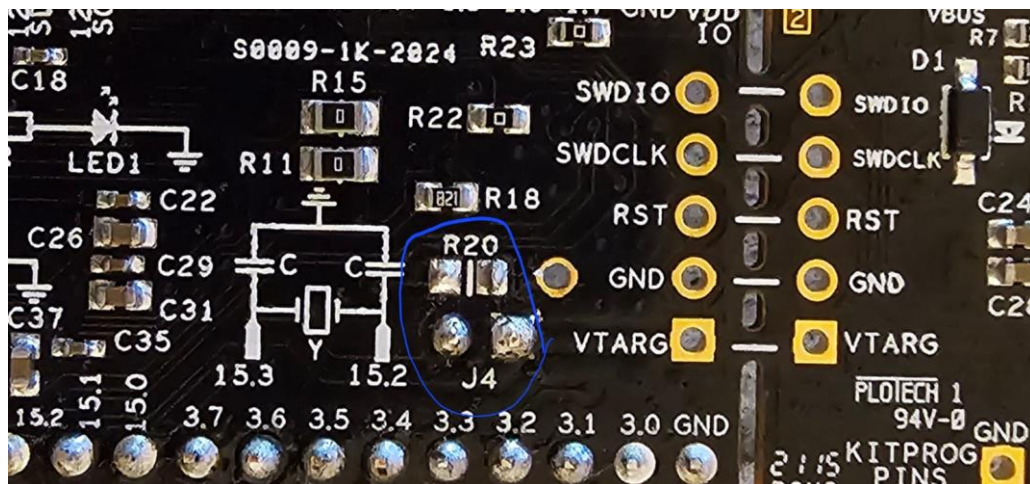
Tabel 7: Forbindelsesoversigt med farvekodning

### 6.1.2 Serieforbindelse mellem PSoC og RPi4

I udviklingsprocessen af hardwaredesignet er det blevet bestemt, at en SPI-forbindelse skal anvendes til at overføre dataene mellem PSoC og RPi4. Dette sikrer en effektiv og hurtig datakommunikation og er allerede bekendt fra pensum.

Ved brug af SPI-forbindelsen etableres en forbindelse, hvor MISO, MOSI, SCLK og SS/CS pins på PSoC og RPi4 er korrekt forbundne. Denne forbindelse muliggør tovejskommunikation, hvilket betyder, at RPi4 kan modtage data fra PSoC, i projektets tilfælde samples.

For at sikre en sikker og stabil dataoverførsel skulle PSoC'en modificeres til at være kompatibel med RPi4's 3.3v logikniveauer. Dette er tilfældet eftersom PSoC'en som standard kører på 5V med USB-forbindelse. Det realiseres ved at fjerne R20 modstanden og sætte en jumper på J4's plads<sup>1</sup>:



Figur 21: PSoC modificering

Var overstående ikke realiseret, kunne det resultere i overbelastning og sandsynligvis beskadige alle komponenterne, grundet spændingsforskelle.

### 6.1.3 ADC i PSoC'en

I udviklingsprocessen blev det bestemt af en ADC skulle anvendes til at konvertere analoge lydsignaler fra mikrofonen til digitalt signal, så RPi4'eren kan håndtere det korrekt. Konverteringen sker i PSoC'en og var rimelig ligetil at sætte op, derfor bliver der ikke gået i dybden med det.

<sup>1</sup> (Mikkelsen, 2021)



### 6.1.4 Amplifier og mikrofon

I udviklingsprocessen blev det klargjort at en hvilken som helst 'ready-to-go' mikrofon og amplifier skulle anvendes, da det ikke var betydningsfuldt for udkommet af projektet.

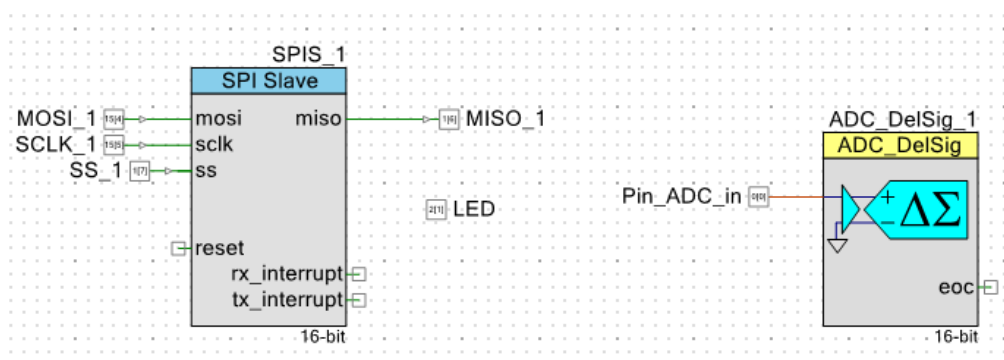
Da amplifiseren har RCA udgange og PSoC'en har pin-inputs, skulle et specielt kabel laves, så forbindelsen mellem amplifiseren og PSoC'en kunne realiseres. Her er løsningen at lodde to kabler fast til RCA-stikket henholdsvis (GND og Analog Output) og dermed sikre en fejlfri forbindelse:



Figur 22: RCA til hunstik kabel

### 6.1.5 PSoC projekt

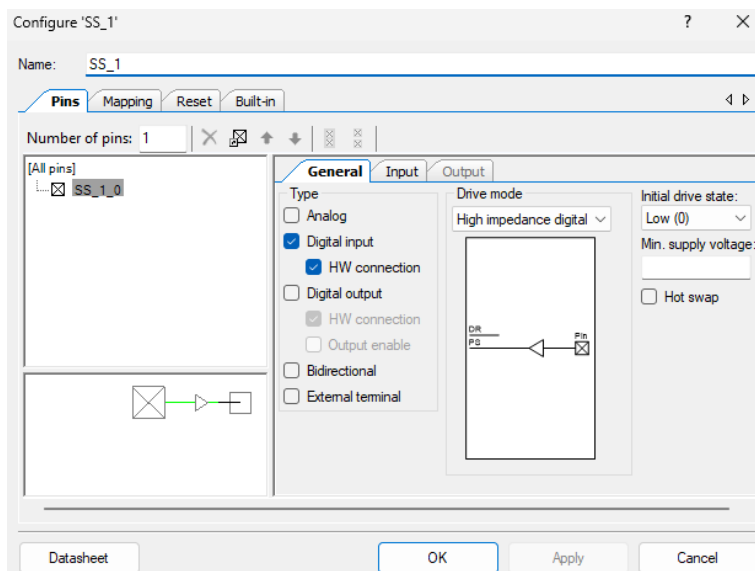
I udviklingsprocessen blev der truffet betydningsfulde valg ift. Valg af komponenter i topdesignet.



Figur 23: PSoC projekt topdesign

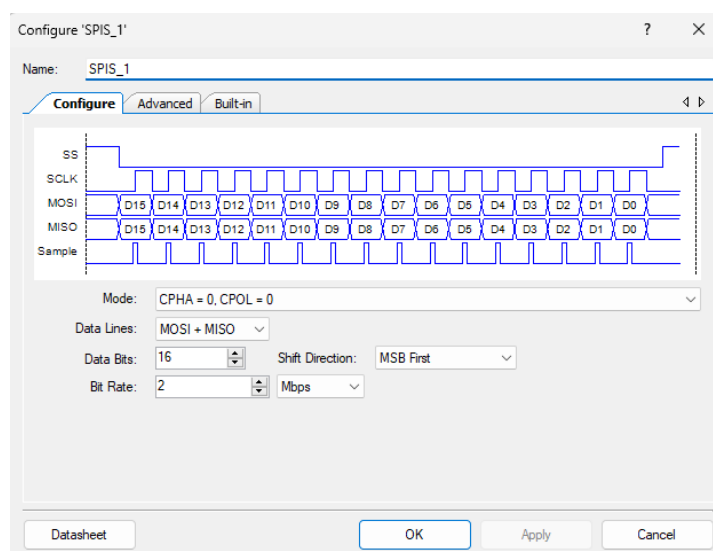
Det var oplagt at have en SPI slave og ADC-komponent - specifikt en Delta Sigma variant<sup>2</sup> Her blev SAR ADC-varianten<sup>3</sup> fravalgt, eftersom den udelukkende understøtter en maksimal resolution på 12-bit, hvilket ikke levede op til ønsket om en 16-bit resolution.

Hertil blev relevante pins forbundet til deres respektive porte på komponenter.



Figur 24: Konfiguration af SS pin i PSoC topdesign

Særligt for SS pin, blev den sat til digitalt input som HW connection, da CS fra RPi4'eren skulle forbindes med PSoC'en. Desuden blev SPI komponenten konfigureret.



Figur 25: Konfiguration af SPI-komponent i topdesign

<sup>2</sup> (CYPRESS, Delta Sigma Analog to Digital Converter (ADC\_DelSig), 2017)

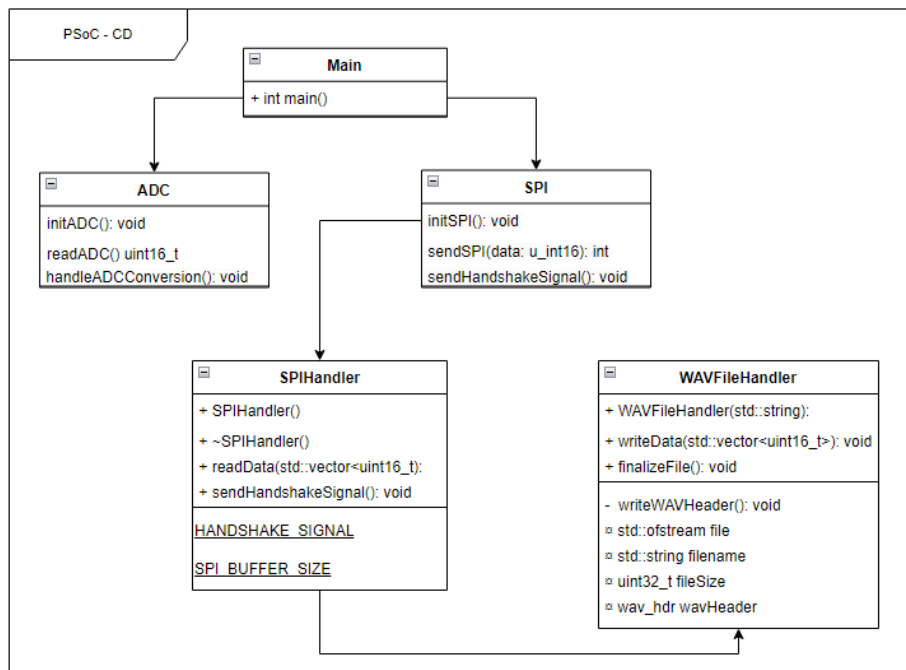
<sup>3</sup> (CYPRESS, ADC Successive Approximation Register (ADC\_SAR), 2017)

SPI-mode blev sat til SPI-mode = 0, hvor både CPOL, der indstiller polariteten af clocksignalet under inaktiv tilstand og CPHA sættes til 0. Betydningen er at datene samples i rising edge og shiftes ud i falling edge.

Desuden bruges overraskende nok datalinjen MOSI + MISO, med en bit-rate på 2Mbps, der sørger for en hurtig og stabil dataoverførsel, med så meget data som muligt.

### 6.1.2 SW-design

PSoC-modulet står for at lave ADC på indkommende analogt signal fra amplifieren, sende det digitale data til RPi4'eren gennem SPI-kommunikation og sørge for at .wav formatere det digitale data, så brugerinputtet fra mikrofonen er til afspilning. System fungerer desuden i forbindelse med Use Case 2 (Syng Og Sammenlign)



Figur 26: PSoC modul klassediagram

### 6.1.6 Software Designvalg.

**Moduler:** Implementeringen er organiseret i separate moduler for ADC, SPI og WAV-filhåndtering, hvilket sikrer en klar adskillelse af ansvarsområder og let vedligeholdelse, hvis der kræves debugging.

**Dataoverførsel:** SPI-kommunikation er valgt for dens pålidelighed og hastighed, som er velegnet til realtime behandling fra PSoC til RPi4, hvilket er godt for et system som karaoke maskine.



**Sampling rate:** En sampling rate på 44,1 kHz er valgt, hvilket er standarden for lydoptagelser og sikrer lydgengivelse af høj kvalitet.

**Buffers:** En bufferstørrelse på 512 bruges til at holde en balance mellem latenstid og memory effektivitet, hvilket tillader systemet at håndtere realtime behandling uden stor forsinkelse.

**Filformat:** WAV-formatet anvendes til lydfiler, grundet dets udbredte understøttelse og stabile format, hvilket sikrer høj effektivitet. Desuden er det bekendt fra 3. Semester faget DSB.

### 6.1.7 Kodeeksekvering

**PSoC:** Her er det tanken at lade en ADC-buffer fylde op før SPI-kommunikationen initieres. ADC-modulet vil kontinuerligt læse fra det analoge input fra amplifiseren og konvertere dette til digitalt format. Så snart bufferen er fuld, sendes et handshake signal til RPi4'eren, der svarer tilbage, efterfulgt af at ADC-bufferens indhold sendes afsted. Tanken her er at lade hele ovennævnte proces fortsætte uendelig langt indtil indspilning ønskes færdig.

**RPi4:** RPi4'eren sættes op således at den venter på handshake signalet før læsning af data. Når gyldige data modtages, oprettes modulet en .wav file med korrekt header og skriver daten i filen i realtid. WAV File Handler styrer fil-I/O, hvilket sikrer, at dataene er korrekt formateret og gemt.

### 6.1.8 Overvejelser

**Ydelse:** Ved brug af ikke-blokerende SPI-kommunikationsoperationer og interrupts på PSoc'en, minimeres CPU'ens inaktive tid og optimerer dermed systemets respons, som er essentielt for karaoke maskinen.

**Brugerinteraktion:** Systemet forbliver som nævnt aktivt, indtil et tastaturinput registreres på RPi4'eren, hvilket tillader brugerkontrolleret drift og lagring af lydfilen. Det er essentielt for at system som karaoke maskine, da den meget gerne skulle være brugervenlig.

**Skalérbarhed:** Det specifikke design af modulet tillader fremtidige udvidelser, såsom yderligere lydforarbejdningsfunktioner eller support til mere komplekse use cases, hvis ønskes.

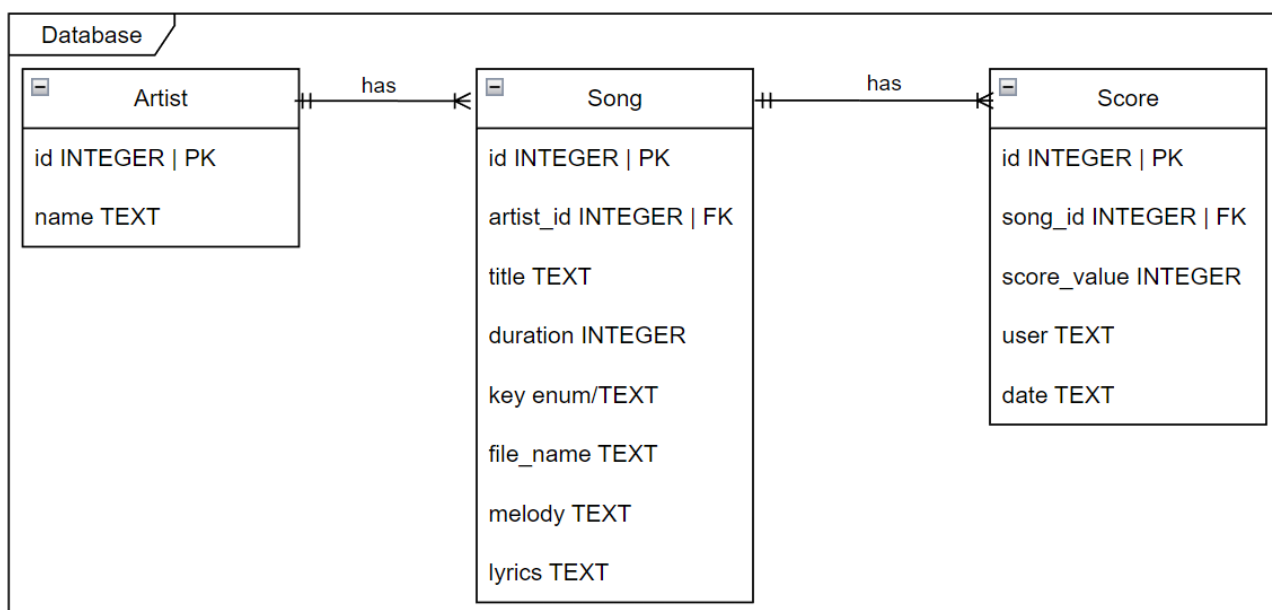
## 6.2 Server

Serveren står for at holde den dynamiske data for systemet i en database, dvs. Sange og Scores, som præsenteret i systemarkitektur afsnittet. Databasen blev skrevet med SQLite, da det var det oplagte valg for at køre en lille og effektiv database på relativt svagt hardware, såsom vores RPiZero, som

blev valgt for dette system. I teorien skal Server dog forstås som kørbart på hvilken som helst HW, så som i skyen. Derfor kørte simplicitet, som et hoveddesignprincip for serveren, da den skal være let at integrere med resten af systemet.

### 6.2.1 Database design

Som vist i systemarkitektur-afsnittet var SQL et valg som blev lavet tidligt i projektets proces, og inkorporeret i designet. I ledetråd med den tidligere viste struktur, vises nu et mere detaljeret SQL-Diagram for at vise hvordan den tænkes at implementeres.

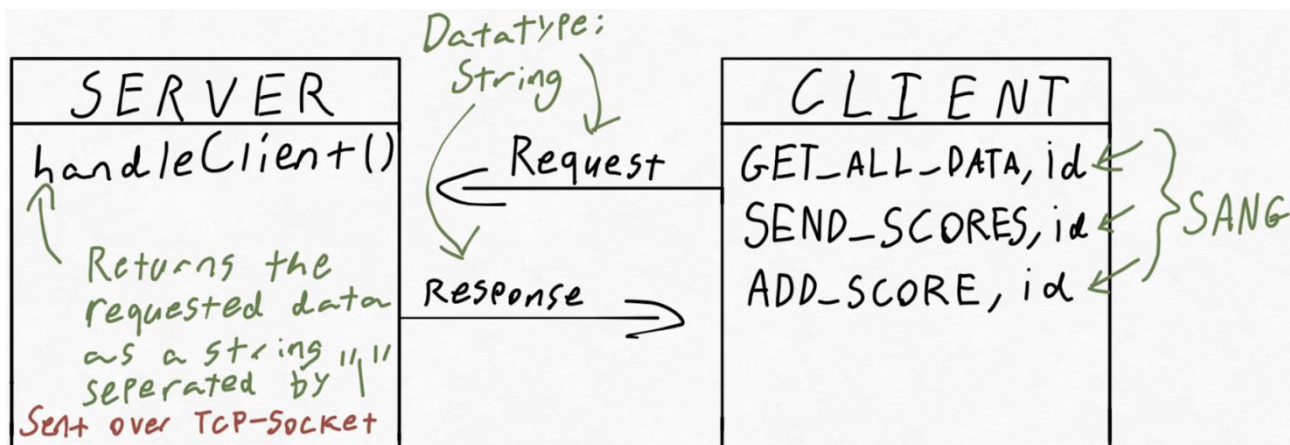


Figur 27: SQL Diagram of DB Design

### 6.2.2 Kommunikation mellem RPi4 & Server

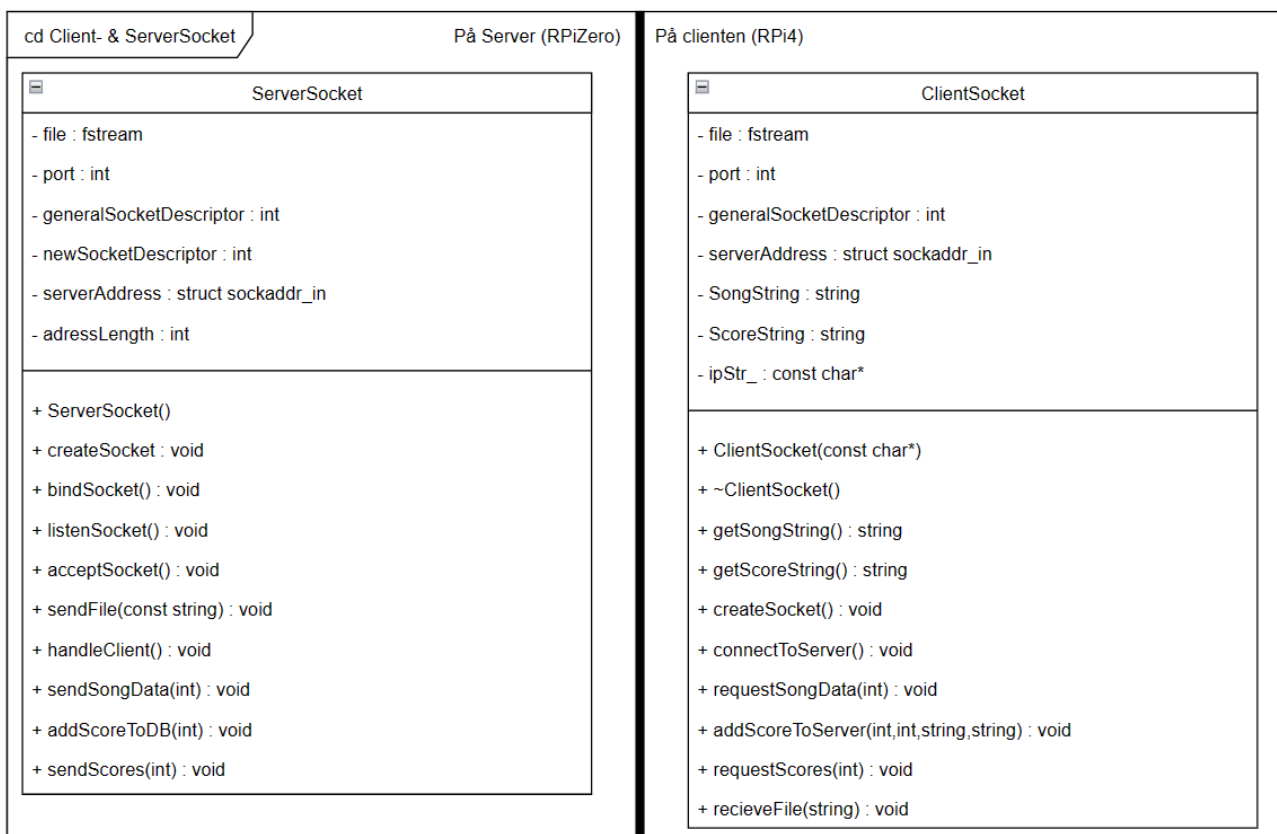
For at holde det simpelt blev der valgt at benytte en TCP-socket til kommunikation mellem de to processorer over Wi-Fi. Udgangspunktet blev sat på koden fra NGK-undervisningen<sup>4</sup>, hvor en sådan integration blev lavet med sending af en fil. Som vist i SQL-Diagrammet vil der forekomme mange forskellige datatyper som skal sendes over denne TCP-socket. Den bedste fremgangsmåde var at sende denne data i form af en streng, som vist i systemarkitekturen. Derved formodes en entydig kommunikation i form af en streng for både sendingen- og forespørgsel af data. For bedre at visualisere denne kommunikation er følgende tegning inkluderet.

<sup>4</sup> (SjioGG, 2023)



Figur 28: Tegning af kommunikation-koncept over TCP-sockets

Kodemæssigt betød dette at der skulle implementeres afsendelse og modtagelse af tre filer, SQL-integration og en længere række af string-parsing funktionalitet på begge sider af forbindelsen. I den iterative proces endte den endelige struktur således for systemet.

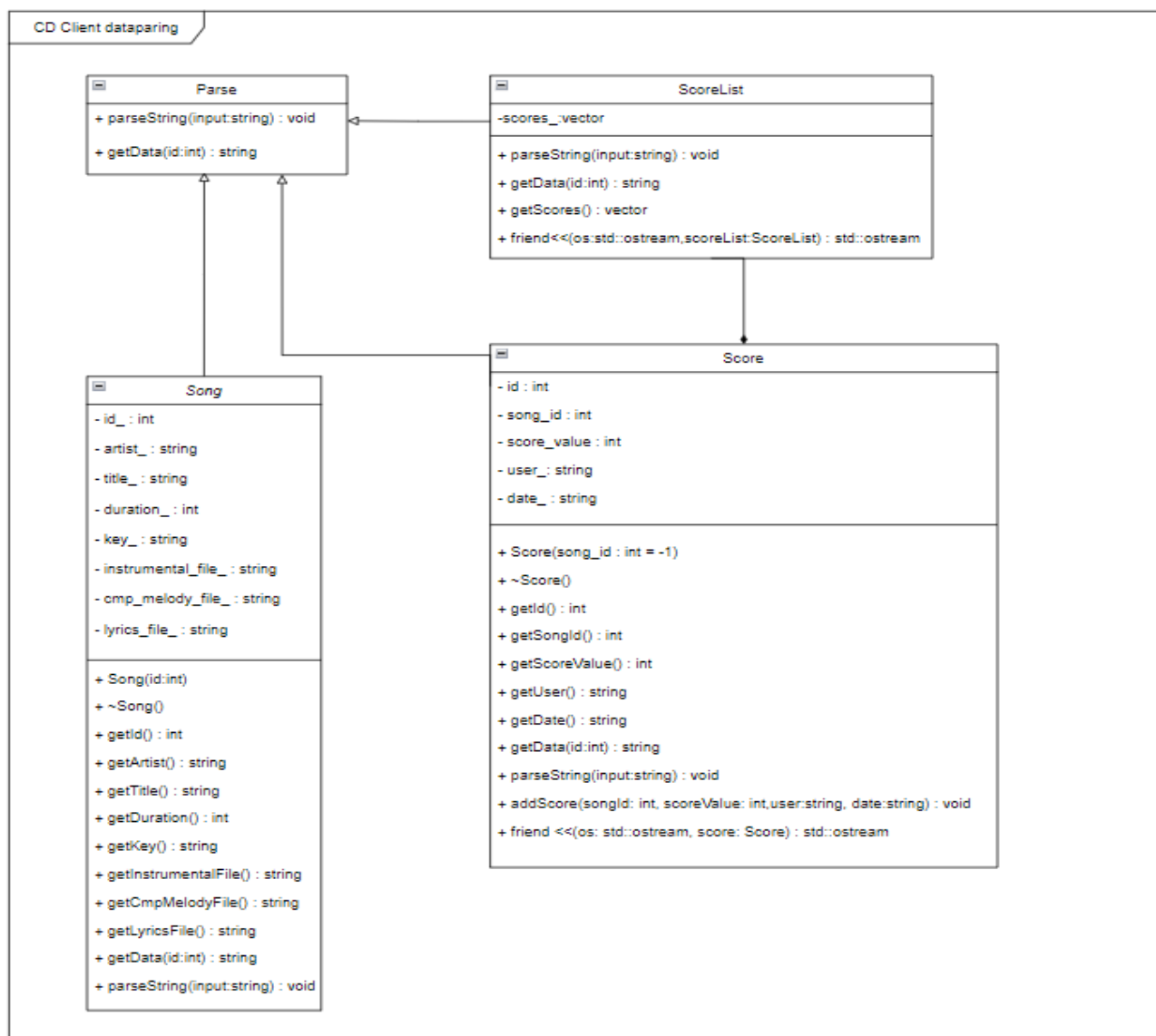


Figur 29: CD endelige implementation af Client- og ServerSockets

### 6.2.3 Kommunikation mellem moduler på RPi4

Kommunikationen mellem de forskellige moduler på RPi4, sker igennem klasser. Overordnet set har det været nødvendigt at udtænke en designmåde, hvorpå alt data kunne blive behandlet korrekt.

Med udgangspunkt i dette er der blevet opsat en klient, der modtager data fra server. Alt efter hvilken form for streng, der modtages, vil en af disse klasser gå ind og behandle dataene. Alt kommunikation og databehandling der foregår på RPi4 sker via klasser. På nedenstående figur er en visuel repræsentation af disse klasser:



Figur 30: CD for Client dataparsing

Det kan ses at de forskellige klasser alle er forbundet og indeholder specifikke data, alt efter hvilken slags streng klienten modtager fra databaseserveren. Slutresultatet af dette vil være en opdelt streng, der har fået sine interne elementer opdelt i den korrekte datatype. F.eks. kunne klienten modtage en "Song" streng der lyder på "(id, artist, title, duration, key, instrumental\_file, cmp\_melody\_file, lyrics\_file) herefter vil den blive parset igennem og få uddelt en datatype. Her vil "id" blive til en int, hvorimod "artist" vil forblive en streng.

Grundet C++ og Python benyttes på RPi4 til forskellige moduler og den samme data skal kunne benyttes af begge moduler, er der blevet lavet en Python wrapper, som gør det muligt at få dataene igennem forskellige get-metoder. Denne tager kort sagt de tidligere nævnte C++ klasser og gør dem tilgængelige i Python-segmentet. På denne måde kan systemet stadig fungerer og kommunikere med andre moduler, som den skal uden at støde på problemer.

I designet blev der også taget højde for, at det skulle kunne tilpasse sig til de andre modulers kalde hierarkier, hvilket vil sige at der på klient-siden skulle kunne åbnes en TCP-socket og lukke i selv samme kald, hvor data forespørges. Gennem designet af dette modul burde det være let at integrere med de andre moduler grundet det simple design, og fokus omkring C++ klasser som alle projektmedlemmer har et godt bekendtskab med.

## **6.3 LogicUnit**

### **6.3.1 Logik behandling**

Den logiske behandling af vores data foregår i dette modul, hvor vores mål er at generere en repræsentativ score baseret på brugerens sangpræstation via mikrofonen. For at opnå en god repræsentativ score er det en forudsætning, at brugeren har forsøgt at imitere den valgte sang på en måde, der efterligner originalen. Dataene behandles ved brug af dynamisk tidsforvrængning fra stokastisk signalbehandling. Scoren beregnes ud fra to faktorer: timing og energi. Det skal bemærkes, at på grund af de forudsatte afgrænsninger tages der ikke højde for tonen i sangen.

### **6.3.2 Design af Matlab skelet og implementeringsstrategi**

Før vi går i gang med kodeimplementering, udarbejder vi først et skelet for den logiske behandling. Dette skelet bliver skabt i MATLAB, da vi ikke er så erfarne med Python-programmeringssproget og foretrækker at teste vores implementering i det velkendte MATLAB-miljø, inden vi begynder selve kodningsprocessen. Denne tilgang giver os mulighed for at opbygge en solid forståelse i et miljø, vi er komfortable med, før vi oversætter det til Python. Fortsat ses vores design logik i Matlab.

Først så skal vi have importeret vores data fra Lydfilerne. I Matlab er en af mulighederne for dette ved brug af audioread funktionen. Som importerer Dataen fra vores lydfile, vi importerer dog ikke

vores Sample rate her. Da det ikke er noget vi har haft brug for til vores analyse, som også kan ses i de nedenstående code snippets.

```
% Antag, at recordingA og recordingB er dine lydsignaler
recordingA = audioread('original.wav'); % Indsæt stien til din lydfil
recordingB = audioread('cover.wav'); % Indsæt stien til din anden lydfil
```

*Figur 31 Import af data*

Vi Definere her størrelsen på de segmenter vi vil arbejde med, Det at vi har en længde på 10000 samples gør dog også at vi ender op på 676.48 segmenter, hvilket gør at vores program tager en del tid om at køre når den laver sine udregninger

```
% Definér længden af de segmenter, du vil behandle (f.eks. 10000 samples)
segmentLength = 10000;
% For at teste kan numSegments resultatet kommenteres ud og der kan sættes et antal fx 10.
numSegments = min(length(recordingA), length(recordingB)) / segmentLength;

% Initialiser variabler til at holde den samlede DTW-afstand og score
totalDtwDistance = 0;
totalMaxDtwDistance = 0;
```

*Figur 32 Definition for segmentlængde*

Vi bruger her et Loop, til at iterere igennem alle vores 676.48 segmenter, så vi får lavet udregningerne på segmenterne hver for sig.

```
% Loop igennem segmenter og beregn DTW for hver
for i = 1:numSegments
    % Hent det aktuelle segment fra hver optagelse
    segmentA = recordingA((i-1)*segmentLength + 1:i*segmentLength);
    segmentB = recordingB((i-1)*segmentLength + 1:i*segmentLength);

    % Beregn DTW afstanden for det aktuelle segment
    dtwDistance = dtw(segmentA, segmentB);

    % Opdater den samlede DTW-afstand og maksimale DTW-afstand
    totalDtwDistance = totalDtwDistance + dtwDistance;
    totalMaxDtwDistance = totalMaxDtwDistance + max(length(segmentA), length(segmentB));
end
```

*Figur 33 Loop funktion*

Når vi har fået lavet vores udregninger af DTW på alle segmenterne hver for sig, så udregner vi en samlet værdi for alle segmenterne og finder gennemsnittet af disse. Så vi i sidste ende får en total værdi for den udregnede DTW-værdi.

```
% Beregn den gennemsnitlige DTW-afstand og score over alle segmenter-
averageDtwDistance = totalDtwDistance / numSegments;
averageMaxDtwDistance = totalMaxDtwDistance / numSegments;
finalScoreDTW = 100 * (1 - (averageDtwDistance / averageMaxDtwDistance));
```

Figur 34 DTW funktions

Energi:

Som skrevet i code snippet under, så oversætter vi vores lydfile så de kommer på Vektor form. Da vi ellers ikke har mulighed for at arbejde med dem i forbindelse med energy-funktionerne vi tager brug af i matlab.

```
% Sørg for, at både recordingA og recordingB er vektorer
recordingA = recordingA(:);
recordingB = recordingB(:);
```

Figur 35 Konversion til Vektorer

Når vi har fået oversat vores lydfile om til vektore, begynder vi så at finde energien for hver lydfile. Den matematiske formel for denne udregning ser sådan ud:

$$E = \sum_{n=1}^N x[n]^2$$

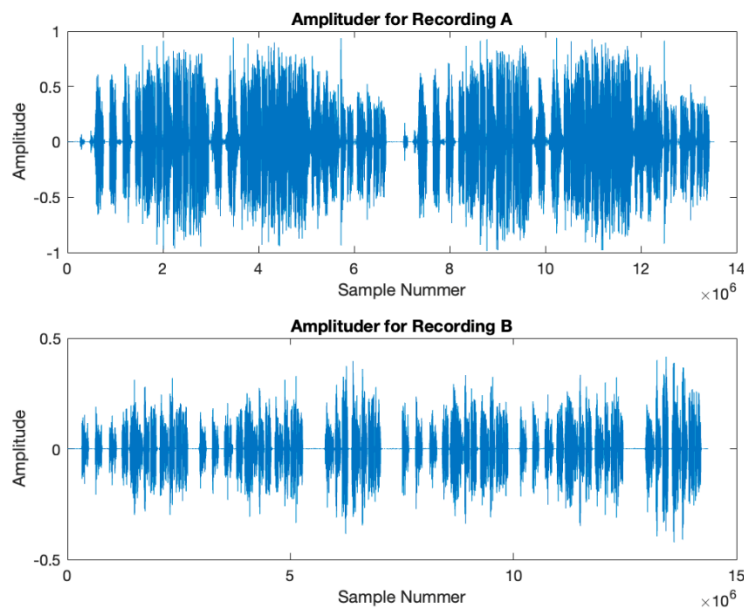
Hvor at  $x[n]$  representere det individuelle antal af samples og  $N$  er det totalte nummer af samples. Skrevet i matlab kode, ser dette ud som nedenstående code snippet viser. Hvor vi så i vores finalScoreEnergy funktion beregner en score ud fra at sammenligne vores energi værdier

```
% Beregn energien i hvert signal
energyA = sum(recordingA.^2);
energyB = sum(recordingB.^2);

% Beregn en score baseret på, hvor godt recordingB matcher energien i recordingA
finalScoreEnergy = (min(energyA, energyB) / max(energyA, energyB)) * 100;
```

Figur :36 Den samlede energy beregning

For at opnå en bedre forståelse af, hvorfor energiscoren er som den er, kan vi undersøge amplituderne i recordingA (originalen) og recordingB (coverversionen). Det er tydeligt, når vi analyserer amplituderne, at recordingA har amplituder, der når helt op til 1, mens amplituderne i recordingB næsten kun når 0,5. Dette resulterer i en betydelig lavere amplitudescore for recordingB.



Figur 37 Amplituder for recordingA (originalen) og recordingB (coverversionen)

Den endelige score beregnes som gennemsnittet af to sammenligningsfaktorer: timing og energi. Vores design afspejler tydeligt, at det er lettere at opnå en højere timing-score end en højere amplitude-score. Dette afspejler karaokefilosofien, hvor det primære fokus er at skabe en sjov og positiv oplevelse. Det vigtigste er at være til stede, have det sjovt og gøre sit bedste, snarere end at stræbe efter perfekt timing. I karaoke handler det om den sociale og underholdende oplevelse, hvor teknisk præcision ofte prioriteres lavere.

```
finalScore = finalScoreEnergy+finalScoreDTW/2
```

Figur 38 Funktion for finalScore

Designskelettet er struktureret med en klar prioritering af timing og energi. Dette skelet fungerer som fundamentet for vores kommende Python-implementering og giver os mulighed for at beregne en objektiv score. Dette skridt er afgørende for at opnå en præcis evaluering af sangpræstationen med fokus på timing og energi.

Der er desværre et stort problem, der skal adresseres, og det vedrører køretiden for den dynamiske tidsforvrængning. Den tager meget lang tid på hardware, der har betydeligt mindre hukommelse end vores planlagte single-board computer, Raspberry Pi 4. Derfor besluttede vi i stedet at anvende krydskorrelation fra stokastisk signalbehandling. Denne funktion analyserer signalernes lighed og giver en krydskorrelationsværdi, der repræsenterer den estimerede korrelation eller lighed mellem de to signaler.



Vi erstatter vores dynamiske tidsforvrængning, med krydskorrelation i stedet. Her definerer vi længden af vores segmenter. Denne er sat relativt lavt, da vi ønsker at se ligheden på timing og der derfor tages mange små samples der sammenlignes og scores.

```
% Definer længden af segmenterne
segmentLength = 1024; % For eksempel 1024 samples per segment
% Beregn antallet af segmenter (for det korteste signal)
numSegments = floor(min(length(recordingA), length(recordingB)) / segmentLength);

% Initialiser samlet score
totalScore = 0;
```

Figur 39 Definitionerne for nogle af vores variable

Der fortsættes testning med recordingA (originalen) og recordingB (coverversionen). Ved at bruge segmenter med en længde på 1024 samples, opnås i alt 6606 segmenter, som der itereres igennem. For hvert af disse segmenter udføres en krydskorrelation, og værdien akkumuleres i en samlet score.

```
for i = 1:numSegments
    % Ekstraher segmenter
    segmentA = recordingA((i-1)*segmentLength+1 : i*segmentLength);
    segmentB = recordingB((i-1)*segmentLength+1 : i*segmentLength);

    % Krydskorrelation
    [corr, lags] = xcorr(segmentA, segmentB, 'coeff');

    % Find maksimal korrelation
    maxCorr = max(abs(corr));

    % Konverter maksimal korrelation til score (0-100)
    score = maxCorr * 100;

    % Tilføj scoren til den samlede score
    totalScore = totalScore + score;
end
```

Figur 40 Loop funktion

Vi benytter os af en gennemsnitlig score for alle segmenterne, da det giver en generel idé om, hvor godt recordingB efterligner recordingA. Dette resulterer i en krydskorrelationsscore på 18.01.

```
% Gennemsnitlig score for alle segmenter
finalScoreXcorr = totalScore / numSegments;
```

Figur 41 Final score definition

Denne implementering er ikke lige så præcis som den dynamiske tidsforvrængning, da krydskorrelation ikke kun fokuserer på timing, men også på det generelle billede af hvor meget et signal ligner et andet. Dette indebærer også amplituden, hvilket betyder, at den endelige score tager højde

for amplituden to gange. Selvom dette ikke er ideelt, er det et nødvendigt kompromis for at sikre, at systemet kan køre på en single board computer med begrænset hukommelse. Derfor vil vores realisering af systemet laves med kryds-korrelation fremfor dynamisk tidsforvrængning.

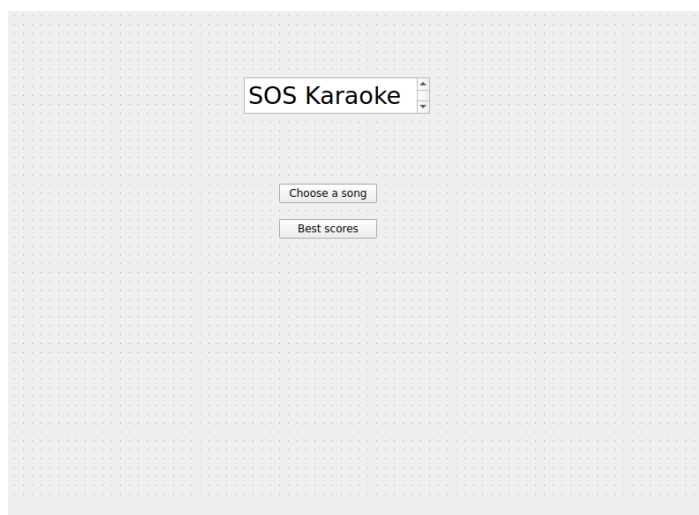
## 6.4 Brugergrænseflade (GUI)

Brugergrænsefladen i karaoke-systemet har ansvaret for al interaktion med brugeren. I praksis fungerer brugergrænsefladen som ”front-end”, mens modtagelse af data behandles via klassen ’socket’ i servermodulet. I teorien betyder dette, at grænsefladen styrer, hvordan brugeren interagerer med system, herunder navigation, indtastning af valg samt visning af information.

### 6.4.1 Design

Som en start skulle der først og fremmest laves nogle mockups, for at have en idé om hvordan en ideel brugergrænseflade kunne se ud ved at gøre brug af QtCreator, da der ikke var særlig meget kendskab til programmet. Ved begyndelsen benyttes der udelukkende Qt’s form-filer/ui-filer, hvor man blot dropper og dragger de ønskede widgets, dette var med til at skabe et fundament af hvordan brugergrænsefladen kunne se ud.<sup>5</sup>

Dette er forsidens første design:

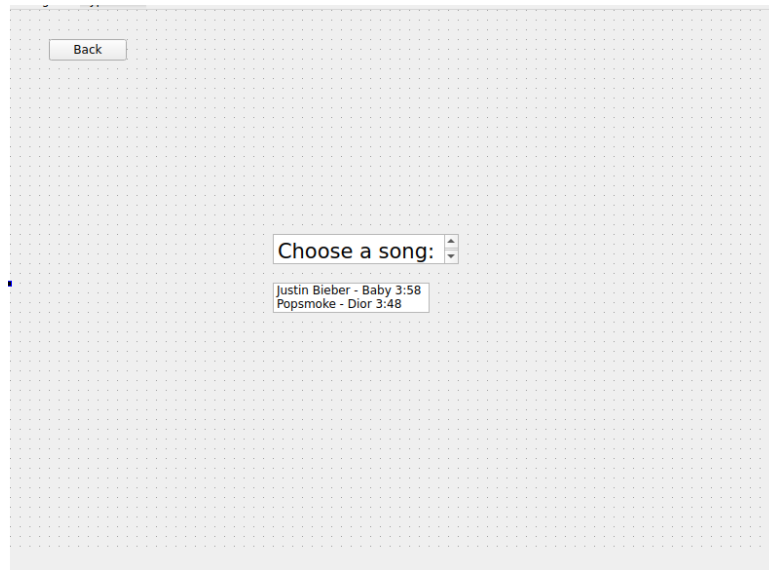


Figur 42: Mockup Menu

Idéen var at have et overfladisk design og derefter skærpe det med bedre layout og pænere farver. Herefter laves der en mockup af songlist:

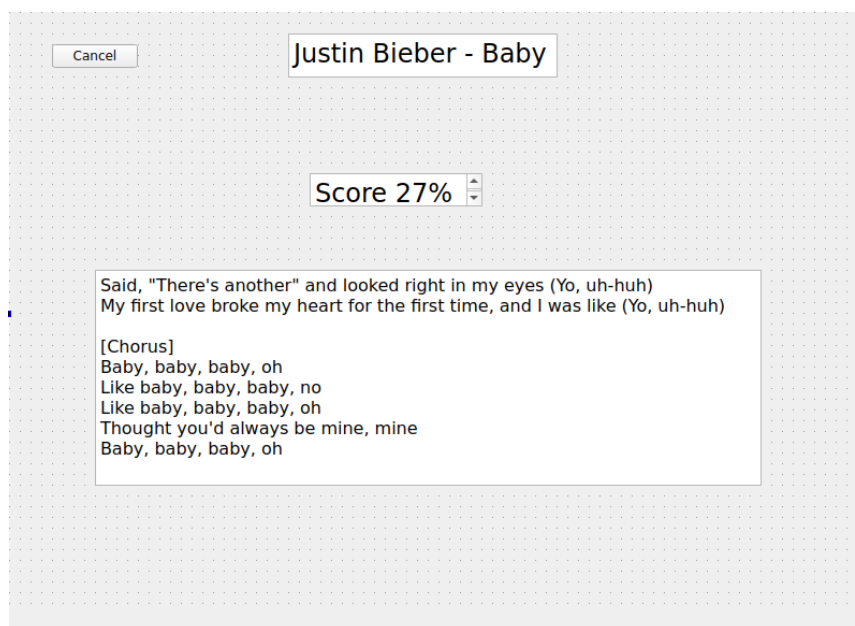
---

<sup>5</sup> (ProgrammingKnowledge, 2016)



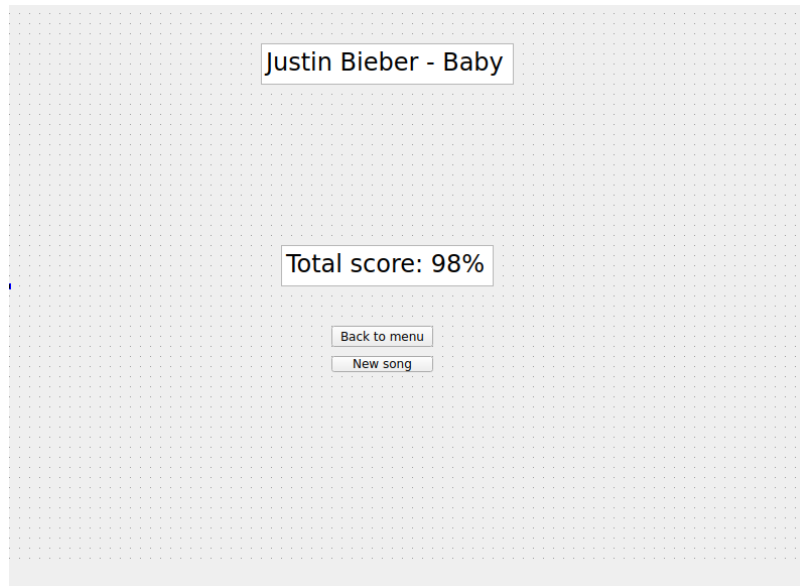
Figur 43: Mockup Songlist

Dertil laves der endnu et vindue for når en sang er valgt, hvor der bl.a. vises sangtekst på skærmen:



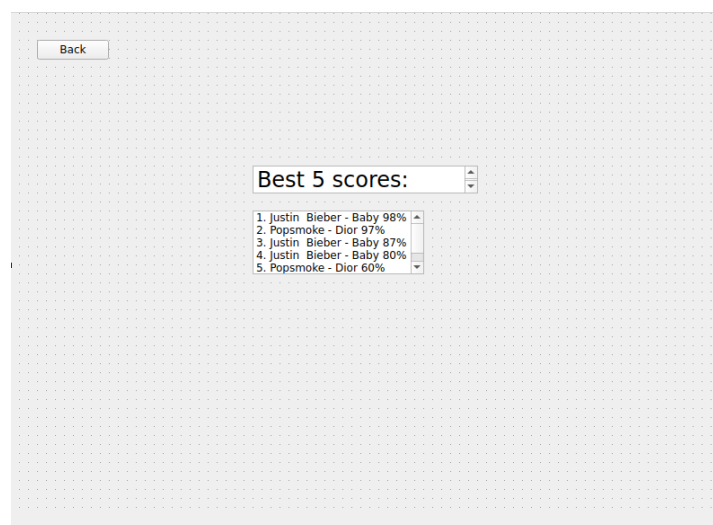
Figur 44: Mockup Song

Derefter laves der et resulterende vindue med brugerens præstation på skærmen der viser hvor godt brugeren har klaret sig:



Figur 45: Mockup Results

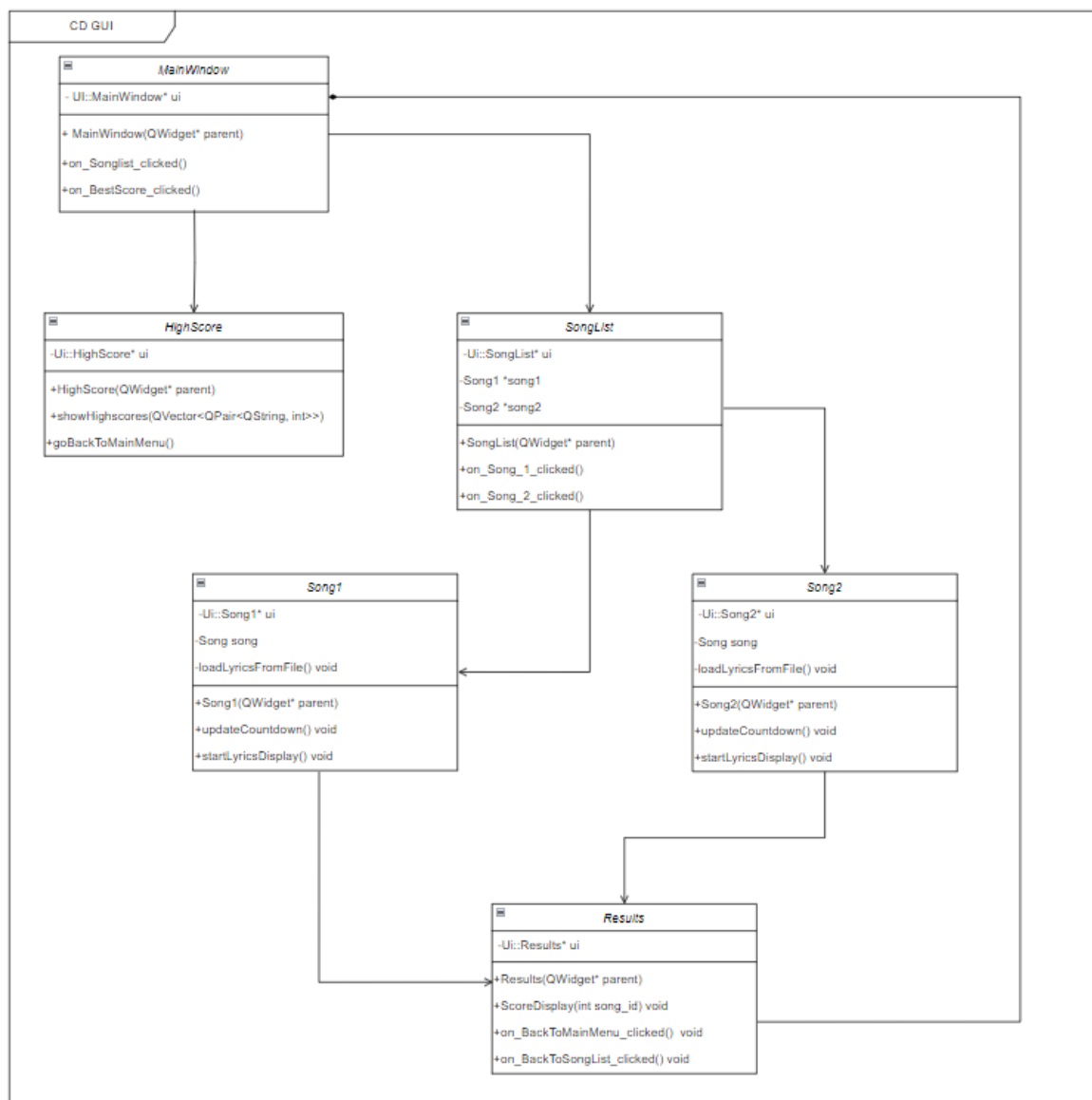
Og sidst men ikke mindst skal der være mulighed for at brugeren kan tjekke de 5 bedste scores for systemet:



Figur 46: Mockup Best score

Overordnet set blev der lagt et fundament af hvordan brugergrænsefladen skulle fremvises.

Dette var med til at skabe et indblik af hvordan de forskellige moduler hænger sammen og hertil udviklet et klassediagram:



Figur 47: CD for GUI

I overstående figur er, der blevet udarbejdet et klassediagram, som skal repræsentere klassernes struktur i forhold til systemets brugergrænseflade ved at vise deres attributer og de relationer som klasserne har med hinanden. I klassediagrammet for brugergrænsefladen er der identificeret følgende klasser: “MainWindow”, “Song1”, “Song2”, “HighScore” og “Results”. Følgende klasser udgør de centrale komponenter i følgende grænseflade.

## 7 Realisering af systemet

I denne sektion vil SW- & HW Design realiseres i form af forbindelser

### 7.1 PSoC

I dette modul er der skrevet i sprogene c/c++, hvor c-koden kører på PSoC'en, mens c++ koden kører på RPi4'eren.

For at implementere ADC'en blev følgende funktioner oprettet:

```
// Initialiserer ADC
void InitADC(void) {
    ADC_Delsig_1_Start();          // Starter ADC
    ADC_Delsig_1_StartConvert();   // Starter konvertering
}

// Læser data fra ADC
uint16_t ReadADC(void) {
    ADC_Delsig_1_IsEndConversion(ADC_Delsig_1_WAIT_FOR_RESULT); // Venter på konvertering
    return ADC_Delsig_1_GetResult16();                          // Returnerer resultat
}

// Håndterer ADC konvertering
void HandleADCCConversion(void) {
    uint32 status = ADC_Delsig_1_IsEndConversion(ADC_Delsig_1_RETURN_STATUS); // Tjekker status på konvertering
    if (status == ADC_Delsig_1_RETURN_STATUS) {                                // Hvis succesfuld
        LED_Write(1);                                                           // Tænder LED ved succesfuld konvertering
    } else {
        LED_Write(0);                                                           // Slukker LED hvis fejl
    }
}
```

*Figur 48: ADC-implementering på PSoC*

Implementering af disse funktioner er essentiel og sørger for at konverteringen er muligt, ved at initialisere ADC'en (initADC) og samtidig læse fra den, med readADC(), hvor det digitale data gemmes som 16-bit Unsigned Integer. handleADCCConversion() sørger derudover for at tjekke status på ADC-konverteringen, hvor en LED tænder eller slukker alt efter om konverteringen er afsluttet.

For at implementere SPI'en blev følgende funktioner oprettet:

```
#include "spi.h"

// Initialiserer SPI
void InitSPI(void) {
    SPIS_1_Start(); // Starter SPI
}

// Sender data over SPI
int SendSPI(uint16_t data) {
    while (!(SPIS_1_ReadTxStatus() & SPIS_1_STS_TX_FIFO_NOT_FULL)); // Venter på at TX FIFO ikke er fuld
    SPIS_1_WriteTxData(data); // Skriver data til TX FIFO
    while (!(SPIS_1_ReadTxStatus() & SPIS_1_STS_SPI_DONE)); // Venter på at SPI er færdig
    return (SPIS_1_ReadTxStatus() & SPIS_1_STS_SPI_DONE) ? 1 : 0; // Returnerer 1 hvis succesfuld, 0 hvis fejl
}

// Sender handshake signal over SPI
void SendHandshakeSignal(void) {
    SendSPI(HANDSHAKE_SIGNAL); // Sender handshake signal
    CyDelay(5); // Lille forsinkelse for at sikre RPI er klar
}
```

Figur 49: SPI-implementering på PSoC

Implementering af disse funktioner er essentiel og sørger for at SPI-kommunikationen er mulig, ved at initialisere SPI-slaven (initSPI), så den er klar til at sende og modtage. Samtidig sørges der gennem funktionen (sendSPI) at modtage en 16-bit unsigned integer.

Den venter først på at bufferen ikke er fuld ved at tjekke status på (SPIS\_1\_WriteTxData()) og flaget FIFO\_NOT\_FULL. Derefter skriver den dataene til transmitter FIFO vha.

(SPIS\_1\_WriteTxData(data)). Efterfølgende venter den på, at dataoverførsel er fuldført ved at returnere 1 for succesfuld og 0 hvis der opstod fejl.

(sendHandShakeSignal()) funktionen bruges til at sende et handshake-signal defineret til '0xFFFF', til RPi4'eren, efterfulgt af et 5 sekunders delay, så der sikres en stabil forbindelse.

For at implementere main funktion blev følgende gjort:

```
#define SAMPLE_RATE 44100      // Samplerate sat til 44.1 kHz
#define BUFFER_SIZE 512        // Buffer størrelse sat til 512

volatile uint16_t adcBuffer[BUFFER_SIZE]; // ADC buffer
volatile int bufferIndex = 0;             // Buffer index

int main(void) {
    CyGlobalIntEnable; // Aktiverer globale interrupts
    InitADC();          // Initialiserer ADC
    InitSPI();          // Initialiserer SPI

    // Uendelig løkke
    for (;;) {
        if (bufferIndex < BUFFER_SIZE) {
            // Hvis buffer ikke er fyldt
            adcBuffer[bufferIndex++] = ReadADC(); // Læser data fra ADC og gemmer i buffer
            HandleADCCConversion();              // Håndterer ADC konvertering
        } else {
            SendHandshakeSignal();                // Sender handshake signal
            for (int i = 0; i < BUFFER_SIZE; i++) { // Sender buffer over SPI
                if (!SendSPI(adcBuffer[i])) {      // Hvis fejl
                    LED_Write(0);                 // Slukker LED
                    break;                         // Stopper for loop
                }
            }
            bufferIndex = 0;                       // Nulstiller buffer index
        }
    }
}
```

Figur 50: PSoC main()

Allerførst aktiveres Globale interrupts, hvilket tillader systemet at reagere på eksterne og interne hændelser Herefter initialiseres ADC og SPI vha. `InitADC()` og `InitSPI()` funktionerne. Programmet går derefter ind i en uendelig løkke `for(;;)`, der kører kontinuerligt og udfører to hovedopgaver:

Hvis `bufferIndex` er mindre end `BUFFER\_SIZE` (sat til 512), læser den et data punkt fra ADC'en vha. `ReadADC()` funktionen og gemmer dataen i `adcBuffer` ved den nuværende `bufferIndex`. Derefter inkrementeres `bufferIndex` og bufferen fyldes op, hvor `HandleADCCConversion()` funktionen også kaldes, der håndterer ADC-konverteringen.

Hvis bufferen er fuld, sender programmet et handshake-signal til RPi4'eren vha. `SendHandshakeSignal()` og går derefter ind i en `for`-løkke, der sender hele indholdet af `adcBuffer` over SPI vha. `SendSPI()` funktionen.

Efter at have tømt bufferen og forsøgt at sende alle data over SPI'en, nulstilles `bufferIndex`, så bufferen straks kan begynde at blive fyldt igen, hvor samme loop gennemløbes.



Her er implementeringen for SPIHandler, der er står for at initialisere og håndtere SPI-kommunikationen.

```
#include "SPIHandler.hpp"
#include <bcm2835.h>
#include <iostream> // std::cerr

// Constructor
SPIHandler::SPIHandler() {
    initSPI(); // Initialiserer SPI.
}

// Destructor
SPIHandler::~SPIHandler() {
    bcm2835_spi_end(); // Afslutter SPI.
    bcm2835_close(); // Afslutter BCM2835 biblioteket.
}

// Initialiserer SPI.
void SPIHandler::initSPI() {
    // Initialiserer BCM2835 biblioteket.
    if (!bcm2835_init()) {
        std::cerr << "BCM2835 init ERROR." << std::endl; // Kaster en runtime error, hvis BCM2835 biblioteket ikke kan initialiseres.
    }
    // Initialiserer SPI.
    if (!bcm2835_spi_begin()) {
        bcm2835_close(); // Lukker BCM2835 biblioteket.
        std::cerr << "SPI init ERROR." << std::endl; // Kaster en runtime error, hvis SPI ikke kan initialiseres.
    }

    // Konfigurerer SPI indstillinger
    bcm2835_spi_setBitOrder(BCM2835_SPI_BIT_ORDER_MSBFIRST); // MSB først for dataoverførsel.
    bcm2835_spi_setDataMode(BCM2835_SPI_MODE0); // SPI mode 0.
    bcm2835_spi_setClockDivider(BCM2835_SPI_CLOCK_DIVIDER_256); // Sætter SPI clock divider.
    bcm2835_spi_chipSelect(BCM2835_SPI_CS0); // Vælger SPI chip (CS0).
    bcm2835_spi_setChipSelectPolarity(BCM2835_SPI_CS0, LOW); // CS0 er aktiv lav.
}

// Læser data fra SPI bufferen.
std::vector<uint16_t> SPIHandler::readData() {
    std::vector<uint16_t> data; // Data vektor.
    uint8_t spiBuffer[SPI_BUFFER_SIZE]; // SPI buffer.

    uint8_t handshakeBuffer[2]; // Handshake buffer.
    bcm2835_spi_transfer(reinterpret_cast<char*>(handshakeBuffer), 2); // Overfører data fra SPI bufferen til handshake bufferen.
    uint16_t handshakeSignal = static_cast<uint16_t>(handshakeBuffer[0]) << 8 | handshakeBuffer[1]; // Konverterer SPI data til 16-bit data.

    // Hvis handshake signal modtages, modtag data.
    if (handshakeSignal == HANDSHAKE_SIGNAL) {
        bcm2835_spi_transfer(reinterpret_cast<char*>(spiBuffer), SPI_BUFFER_SIZE); // Definér HANDSHAKE_SIGNAL passende
        // Overfører data fra SPI bufferen til SPI data bufferen.
        for (int i = 0; i < SPI_BUFFER_SIZE; i += 2) { // Læser data fra SPI bufferen.
            uint16_t spiData = static_cast<uint16_t>(spiBuffer[i]) << 8 | spiBuffer[i + 1]; // Konverterer SPI data til 16-bit data.
            data.push_back(spiData); // Tilføjer data til data vektoren.
        }
    }
    return data; // Returnerer data vektoren.
}
```

Figur 51: RPI SPIHandler implementering

## Constructor `SPIHandler::SPIHandler():

- Kalder `initSPI()` for at initialisere SPI-kommunikationen.

## Destructor `SPIHandler::~SPIHandler():

- Afslutter SPI-kommunikation ved at kalde `bcm2835\_spi\_end()`.
- Lukker BCM2835-biblioteket ved at kalde `bcm2835\_close()`.

**Funktionen `void SPIHandler::initSPI()`:**

Denne funktion sørger for at initialisere BCM2835-biblioteket ved at kalde på `bcm2835_init()` og derudover også `bcm2835_spi_begin()`. Desuden konfigureres nødvendige SPI-indstillinger så de matcher med PSoC'en:

- Sætter bitorder til MSB.
- Sætter SPI-mode til mode 0.
- Sætter SPI clock divider til 256.
- Sætter chip select til at bruge GPIO pin som chip select (CS) og indstilles til low, hvilket vil sige chippen er aktiv når CS er low. (McCauley, ukendt)

**Funktionen `std::vector<uint16_t> SPIHandler::readData()`:**

Denne funktion sørger for at initialisere en vektor til at holde på datene, der skal læses fra SPI.

Opretter en buffer til at holde på handshake-signalet og udfører en SPI-overførsel for at læse handshake-signalet fra PSoC'en. Her konverteres den indkommende Data fra en rå databuffer til en `'uint16_t'` type. Her begynder den med at læse fra SPI-bufferen og gemme i vektoren, hvis den modtagende `'HANDSHAKE_SIGNAL'` matcher med det forventede `'0XFFF'`. Til slut returnerer den vektoren med indlæst data.

Her er implementeringen for WAVFileHandler, der står for at åbne, skrive og afslutte en .wav fil korrekt:

```
#include "WAVFileHandler.hpp" // WAVFileHandler
#include <cassert>              // assert

// Constructor
WAVFileHandler::WAVFileHandler(const std::string& filename) : filename(filename), fileSize(0) {
    file.open(filename, std::ios::binary | std::ios::out | std::ios::trunc); // Åbner output filen for at skrive data.
    if (!file.is_open()) {                                                    // Hvis filen ikke kan åbnes, kastes en exception.
        throw std::runtime_error("cannot open file.");                        // Kaster en runtime error.
    }
}

// Skriver data til WAV filen.
void WAVFileHandler::writeData(const std::vector<uint16_t>& data) {
    for (auto& d : data) {
        file.write(reinterpret_cast<const char*>(&d), sizeof(d));
        fileSize += sizeof(d); // Filstørrelsen opdateres.
    }
}

// Afslutter WAV filen.
void WAVFileHandler::finalizeFile() {
    writeWAVHeader(); // Skriver WAV headeren til filen.
    file.close();     // Lukker filen.
}

// Skriver WAV headeren til filen.
void WAVFileHandler::writeWAVHeader() {
    static_assert(sizeof(wav_hdr) == 44, ""); // Write WAV header.
    wavHeader.wavSize = fileSize + sizeof(wav_hdr) - 8; // Assert at størrelsen af wav_hdr er 44 bytes.
    wavHeader.dataChunkSize = fileSize;                // Filstørrelsen opdateres.
    file.seekp(0, std::ios::beg);                       // Størrelsen på data sub-chunk opdateres.
    file.write(reinterpret_cast<const char*>(&wavHeader), sizeof(wav_hdr)); // Sætter filpointeren til starten af filen.
} // Skriver headeren til filen.
```

Figur 52: RPI WAVFileHandler implementering

### Constructor `WAVFileHandler::WAVFileHandler(const std::string& filename)`:

Den tager en filsti som parameter og åbner en file stream (`std::ofstream file`) i binær og output-tilstand. Den sørger desuden for at eksisterende filindhold bliver slettet med (`std::ios::trunc`).

### Metoden `void WAVFileHandler::writeData(const std::vector<uint16\_t>& data)`:

Den tager en vektor af 16-bit unsigned integers og itererer over den, hvor der for hvert element i vektoren skrives data til filen ved at konvertere `uint16\_t` værdien til en raw bytestream vha. `reinterpret\_cast`

Den opdaterer desuden `fileSize` variablen for at holde styr på den samlede størrelse af de skrevne data.

### Metoden `void WAVFileHandler::finalizeFile()`:

Den laver et kald til `writeWAVHeader()` for at skrive header-informationen for WAV-filen, i filen, hvilket er nødvendigt for at filen bliver genkendt som en valid WAV-fil.

Den lukker derefter file streamen, hvilket afslutter write processen.

### Metoden `void WAVFileHandler::writeWAVHeader()`:

Den bruger `static_assert` for at sikre, at størrelsen af WAV-header-strukturen (`wav_hdr`) er 44 bytes, hvilket er standarden for WAV-filer. (ukendt, ukendt)

Den Opdaterer `wavSize` og `dataChunkSize` i `wav_hdr` strukturen med korrekt filstørrelse, minus 8 bytes for 'RIFF' og størrelsen på `wavSize`, og sætter filens skriveposition til starten (ved hjælp af `file.seekp()`), så headeren kan blive skrevet først i filen, fremfor et hvilket som helst sted i filen.

Slutvis skriver den opdaterede header til filen ved hjælp af `file.write()`, der også bruger `reinterpret_cast` til at konvertere `wav_hdr` strukturen til en raw bytestream.

## 7.2 Server

I løbet af udviklingsprocessen blev server og klient testet gennem 'VMWare' med VM's. Kodemæssigt for dette modul blev der som tidligere nævnt i designafsnittet brugt C++ til at implementere den ønskede funktionalitet. I løbet af realiseringen blev det valgt at benytte PyBind11<sup>6</sup> til at wrappe koden og gøre den tilgængelig for Python moduleerne. I implementering af PyBind11 blev der også benyttet CMake og lavet en trampolin klasse, da der blev benyttet 'inheritance' i implementeringen af sange og scores. Der var mange små skridt, men grundet deres gode dokumentation var det dog en rimelig smerteløs proces. For at vise et eksempel på hvordan C++ metoderne bindes til Python metoder inkluderes følgende kode-udsnit, hvor en abstrakt klasse bindes. For denne klasse er der også lavet en nødvendig trampolin klasse, men syntaktisk er processen den samme for de andre overliggende metoder.

```
PYBIND11_MODULE(py_dbserver, m) {
    py::class_<Parse, Parse>(m, "Parse")
        .def(py::init<>())
        .def("parse_string", &Parse::parseString)
        .def("get_data", &Parse::getData);
}
```

Figur 53: PyBind11 module initiering

For implementering af sockets var det hovedsagelige problem med data allokering imellem afsendelse af data. Dette forekom da den originale kode vi byggede på, var meget upræcis i sin allokering af data-størrelser. Denne fejl blev så derefter gentaget flere gange, hvilket gjorde

---

<sup>6</sup> (Jakob, 2017)

problemet værre og ødelagde kommunikationen mellem sockets. Da problem blev identificeret, gik en større refaktoreringsproces i gang for at få data-størrelserne til at gå op med hinanden, så begge sider havde en klar besked om hvornår den anden var færdig med at sende/modtage data. Følgende kode udklip demonstrerer den rette måde at opsætte dette.

```
string request = "GET_ALL_DATA," + to_string(data);  
char buffer[STRING_SIZE];  
strncpy(buffer, request.c_str(), STRING_SIZE - 1);  
buffer[STRING_SIZE - 1] = '\\0';  
send(generalSocketDescriptor, buffer, STRING_SIZE, 0);
```

*Figur 54: Korrekt afsendelse af data, med data allokering*

```
char requestBuffer[STRING_SIZE];  
recv(newSocketDescriptor, requestBuffer, STRING_SIZE, 0);
```

*Figur 55: Korrekt modtagelse af data, med data allokering*

Dette eksempel er for at illustrere den generelle proces hvor en statisk streng sendes, men flere edge cases sker, så som ved sendelse af fil, hvor størrelsen er dynamisk og endians også spiller en rolle på tværs af styresystemer. Så for at sikre sig en ren overførsel af dynamisk data benyttes 'htonl' og 'ntohl', som sikre en ren overgang af data størrelse på tværs af systemarkitekturer<sup>7</sup>.

I implementeringen af den overliggende klasse struktur var der til starts, glemt at tage højde for scores mange til en – forhold med sang, hvilket betød at 'ScoreList', blev implementeret i en hast. Dette er grunden til en ubrugt 'getData()' funktion i kaldehierarkiet hos klassen 'Score'.

Afsluttende inkluderes også Main for Serveren, som kører i et while-loop for kontinuer kørsel, så klienten kan anmode om data når den har brug for det.

---

<sup>7</sup> (POSIX, 2001)

```

int main(int argc, char *argv[])
{
    printf("Starting server...\n");
    ServerSocket server;
    server.createSocket();
    server.bindSocket();
    while (1)
    {
        server.listenSocket();
        server.acceptSocket();
        server.handleClient();
        //server.sendFile();    // -> server.handleClient();
                                // will either receive or send data,
                                // depending on the client's request.
                                // Probably either "add a row" or "send all data"

    }
    return 0;
}

```

old pseudo

←

SRVM...

Figur 56: Main for the Server-processor

## 7.3 LogicUnit

### 7.3.1 Python implementering af præstations vurdering

Struktur for afsnit:

For selve vores implementering af vores Logic unit, brugte vi som tidligere nævnt Python. Grunden til vi tog dette valg var så vi kunne få adgang til disse funktioner:

**Numpy** - (NumPy, n.d.)<sup>8</sup>

**scipy.signal** - (SciPy v1.11.4 Manual, n.d.)<sup>9</sup>

**Scipy.io.wavfile** - (SciPy v1.11.4 Manual, n.d., n.d.)<sup>10</sup>

Disse funktioner og biblioteker har været nødvendige for at vi kunne lave det system vi gerne ville i måls med. Og vi beskriver senere i dette afsnit hvordan vi har brugt dem til få vores ønskede implementation af den tidligere viste Matlab funktionalitet.

<sup>8</sup> [NumPy documentation — NumPy v1.26 Manual](#)

<sup>9</sup> [Signal processing \(scipy.signal\) — SciPy v1.11.4 Manual](#)

<sup>10</sup> [scipy.io.wavfile.read — SciPy v1.11.4 Manual](#)

Vi indlæser vores optagelser og gemmer dem ligesom i Matlab skelettet som recordingA og recordingB

```
# Indlæs lydfileerne
sample_rate_A, recordingA = scipy.io.wavfile.read('original.wav')
sample_rate_B, recordingB = scipy.io.wavfile.read('cover.wav')
```

*Figur 57 Import af Lydfiler i Python*

Der defineres længden af de segmenter vi ønsker at bruge til kryds-korrelation, og der beregnes et antal segmenter ud fra dette. Yderligere initialiseres totalScoreXcorr også, som er den variabel vi gemmer vores akkumulerede score i.

```
# Definer længden af segmenterne
segmentLength = 1024

# Beregn antallet af segmenter (for det korteste signal)
numSegments = min(len(recordingA), len(recordingB)) // segmentLength

# Initialiser samlet score for krydskorrelation
totalScoreXcorr = 0
```

*Figur 58: definition for variable in Python*

Ligesom i Matlab skelettet gør vi brug af et for loop, der litterær gennem alle segmenterne og gemmer deres score akkumulativt og finder en gennemsnitlig score.

```
for i in range(numSegments):
    segmentA = recordingA[i * segmentLength:(i + 1) * segmentLength]
    segmentB = recordingB[i * segmentLength:(i + 1) * segmentLength]

    corr = scipy.signal.correlate(segmentA, segmentB, mode='full', method='auto')
    maxCorr = np.max(np.abs(corr)) / segmentLength
    score = maxCorr * 100
    totalScoreXcorr += score

# Gennemsnitlig score for alle segmenter
finalScoreXcorr = totalScoreXcorr / numSegments
print ('finalScoreXcorr: ' + str(finalScoreXcorr))
```

Figur 59 Loop i Python

Der skal nu måles for energisammenligningen i recordingA og recordingB, her benyttes der er flatten funktionen der konverterer vores 2d array til et 1d array. Herefter beregnes energien i hvert signal og en finalscore for energi udregnes.

```
# Sørg for, at både recordingA og recordingB er vektorer
recordingA = recordingA.flatten()
recordingB = recordingB.flatten()

# Beregn energien i hvert signal
energyA = np.sum(recordingA ** 2)
energyB = np.sum(recordingB ** 2)

# Beregn en score baseret på, hvor godt recordingB matcher energien i recordingA
finalScoreEnergy = (min(energyA, energyB) / max(energyA, energyB)) * 100
print ('finalScoreEnergy: ' + str(finalScoreEnergy))
```

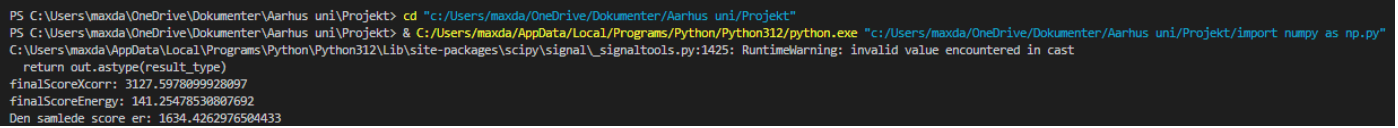
Figur 60 Energy formular i Python



Til slut tages de to resultater og en gennemsnitsscore findes.

```
# Samlet endelig score (gennemsnit af de to scores)
finalScore = (finalScoreEnergy + finalScoreXcorr) / 2
print('Den samlede score er:', finalScore)
```

*Figur 61 Final score og print funktioen i Python*



```
PS C:\Users\maxda\OneDrive\Dokumenter\Aarhus uni\Projekt> cd "c:/Users/maxda/OneDrive/Dokumenter/Aarhus uni/Projekt"
PS C:\Users\maxda\OneDrive\Dokumenter\Aarhus uni\Projekt> & C:/Users/maxda/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/maxda/OneDrive/Dokumenter/Aarhus uni/Projekt/import numpy as np.py"
C:\Users\maxda\AppData\Local\Programs\Python\Python312\Lib\site-packages\scipy\signal\_signaltools.py:1425: RuntimeWarning: invalid value encountered in cast
  return out.astype(result_type)
finalScoreXcorr: 3127.5978099928097
finalScoreEnergy: 141.25478538807692
Den samlede score er: 1634.4262976584433
```

*Figur 62: Print fra terminalen ved kørsel af kode*

Som det fremgår af billedet, er der en markant forskel på de scoringer der beregnes i Python-miljøet sammenlignet med Matlab-miljøet, og det er tydeligt, at proportionerne er betydeligt anderledes. Dette indikerer, at der i implementering af score-beregningen i Python ikke opfylder forventningerne. Selvom Python-implementering følger strukturen fra Matlab, indikerer resultaterne fra udskrifterne, at beregningsmetoden for krydskorrelation og energi afviger fra den oprindelige skabelon. Dette har medført nogle udfordringer, og for at adressere disse har vi udarbejdet en tabel, der illustrerer mulige problemområder. De udskrevne scores er som følger: finalScoreXcorr er 3127.59, finalScoreEnergy er 141.25, og den samlede score er 1634.42. Alle scores afviger fuldstændigt fra afgrænsningerne der er lavet i Matlab der gerne skulle holde scoren på en repræsentativ 0 til 100.

<b>FUNKTION</b>	<b>PROBLEM</b>	<b>LØSNING</b>
<i>scipy.io.wavfile.read()</i>	Virkede som forventet	Brugt denne i forbindelse med indlæsning af lydfile
<i>scipy.signal.correlate()</i> :	Virkede som forventet, men giver nogle anderledes data. Og højst sandsynligt her at koden går i stykker	Afgrænsninger, med henblik på at sætte en værdi der agerer 100% og regne ud fra afvigelser af denne.
<i>np.sum()</i> :	Virkede som forventet	Udregnede summen af elementerne i de pågældende arrays
<i>np.abs()</i> :	Virkede som forventet	Returnerede den absolutte værdi for vært element i den pågældende array
<i>np.flatten()</i> :	Virkede som forventet, men må være anderledes end metoden anvendt i Matlab skelletet grundet forskellige data	Afgrænsninger, med henblik på at sætte en værdi der agerer 100% og regne ud fra afvigelser af denne.
<i>min() and max()</i> :	Virkede forventet	Fandt de min og max-values af vores Energy variable
<i>print()</i> :	Virkede som forventet	Brugt i forbindelse med visualisering af data.

Tabel 8: Diagram over Funktionerne for logic\_Unit

## 7.4 Brugergrænseflade

I denne afsnit af projektet beskriver vi udviklingen af brugergrænsefladen. Den er skrevet i C++ og benytter funktionaliteten fra Qt til at håndtere logik og brugerinteraktion, der er oprettet forskellige klasser, der er fremhævet i brugergrænsefladens klassesdiagram.

I den følgende funktion initialiseres hovedvinduet for projektets program. Dette vindue indeholder en brugergrænseflade med en titel, to knapper og links til både en "highScoreDialog" og "song".

Disse links muliggør skift mellem forskellige vinduer. Herunder så har funktionen følgende parameter "parent" (QWidget pointer), som er en pointer til det overordnede vindue, som det nye hovedvindue er tilknyttet.

```
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent), ui(new Ui::MainWindow),
    songs(nullptr),
    highScoreDialog(nullptr)
{
    // Initialiserer brugergrænsefladen ved at kalde Ui_mainwindow's setupUI-funktion
    ui->setupUi(this);

    // Viser MainWindow i fuldskærmstilstand
    this->showFullScreen();

    // Opretter en QLabel med en velkomstbesked og konfigurerer dens udseende
    QLabel *titleLabel = new QLabel("Welcome to SOS Karaoke", this);
    QFont titleFont = titleLabel->font();
    titleFont.setPointSize(24);
    titleLabel->setFont(titleFont);
    titleLabel->setAlignment(Qt::AlignCenter);

    // Opretter et QVBoxLayout til layoutet af MainWindow
    QVBoxLayout *mainLayout = new QVBoxLayout();
    mainLayout->addWidget(titleLabel); // Tilføjer overskriftsetiketten til layoutet
    mainLayout->addWidget(ui->pushButton); // Tilføjer knappen med teksten "Start" til layoutet
    mainLayout->addWidget(ui->pushButton_2); // Tilføjer knappen med teksten "High Scores" til layoutet

    // Opretter et centralt widget og tildeles det oprettede layout
    QWidget *centralWidget = new QWidget(this);
    centralWidget->setLayout(mainLayout);
    setCentralWidget(centralWidget); // Sætter det centrale widget som centrum for MainWindow

    // Konfigurerer stilen for knapperne ved hjælp af CSS
    QString buttonStyle = "QPushButton {"
        "background-color: #445566;"
        "color: white;"
        "border-radius: 5px;"
        "padding: 10px;"
        "font-size: 16px;"
        "font-weight: bold;"
        "}"
        "QPushButton:hover {"
        "background-color: #667788;"
        "}"
    ;
    setStyleSheet(buttonStyle);

    // Opretter et HighScore-dialogvindue og tildeler det MainWindow som overordnet widget
    highScoreDialog = new HighScore(this);
}
```

Figur 63: Kodeudsnit af MainWindow

I den nedenstående figur bestående af kodeafsnit af Song1, gennemgås der en grundlæggende beskrivelse af den overordnede funktionalitet.

```
// Hent sangtekster fra fil ved start
std::string songString = song.getData(1);
song.parseString(songString);
std::string fp = "../../shared_cache/" + song.getLyricsFile();
QString qstring_fp = QString::fromStdString(fp);
loadLyricsFromFile(qstring_fp);
```

Figur 64: Indlæsning af sangteksten fra serveren

I overstående figur hentes sangteksten fra starten af programmet. Dette håndteres i constructoren, hvor sangteksten modtages som en almindelig c-streng. Da Qt bruger QString, er der behov for at implementere en funktion til konvertering fra c-streng til QString.

Når der så hentes sange serveren er der implementeret en funktion "loadLyricsFromFile", som er ansvarlig for at indlæse sangtekster, overordnet set så sikre funktionen at brugeren har tekstindholdet tilgængeligt for den pågældende sang, som de ønsker at synge koden nedenfor, viser dette:

```
void Song1::loadLyricsFromFile(const QString &filePath) {
    QFile file(filePath);
    // Forsøg på at åbne filen med den angivne sti i læsemodus.
    if (!file.open(QIODevice::ReadOnly | QIODevice::Text)) {
        // Hvis filen ikke kan åbnes, logges en fejlmeddelelse.
        qDebug() << "Cannot open the file for reading:" << filePath;
        return;
    }

    // Opretter en QTextStream for at læse fra filen.
    QTextStream in(&file);
    // Læser filen linje for linje indtil slutningen af filen.
    while (!in.atEnd()) {
        // Tilføjer hver linje til listen over sangtekster.
        ghostLyrics.append(in.readLine());
    }
    // Lukker filen efter fuldført læsning.
    file.close();

    // Kontrollerer om der blev indlæst nogen sangtekster.
    if (!ghostLyrics.isEmpty()) {
        // Bekræfter at teksterne er indlæst og starter visningsprocessen.
        qDebug() << "Lyrics loaded successfully!";
        startLyricsDisplay();
    }
    else {
        // Hvis ingen tekster blev indlæst, logges en fejlmeddelelse.
        qDebug() << "No lyrics loaded!";
    }
}
```

Figur 65: Indlæsning af sangtekster i Qt applikation

Dens parameter, "const QString &filePath" repræsenterer stien til tekstfilen, der indeholder sangteksterne. Desuden returnerer funktionen ikke en værdi, da den er deklareret som "void." Dog meddeler den via logmeddelelser, om teksterne er "loaded" eller "not loaded," når visningen af teksterne starter.

I funktionen "startLyricsDisplay" initialiserer og starter visningen af sangtekster og tilhørende instrumentale lydfil i karaoke-applikation. Der opsættes en timer, som forudbestemt forsinkelser og aktiver opdateringen af sangtekster på skærmen

```
void Song1::startLyricsDisplay() {
    // Forbinder lyricsTimer's timeout signal til denne klasse's updateLyrics slot
    // Dette betyder, at når lyricsTimer udløber (timeout), kaldes updateLyrics funktionen
    connect(lyricsTimer, &QTimer::timeout, this, &Song1::updateLyrics);

    // Sætter lineCounter til 0, dette vil spore, hvilken linje i sangteksten vi er på
    lineCounter = 0;

    // Sætter en forsinkelsestid før visning af tekster starter, her er det 14 sekunder
    int delayTime = 14000; // Initial delay time for the first line (14 seconds)

    // Starter timeren med den definerede forsinkelsestid
    lyricsTimer->QTimer::start(delayTime);

    // Tjekker om soundEffect objektet ikke allerede er initialiseret
    if (!soundEffect) {
        // Bygger filstien til den instrumentale lydfil ud fra sangobjektet
        std::string fp = "../../shared_cache/" + song.getInstrumentalFile();
        // Konverterer std::string til QString, som bruges af Qt
        QString qstring_fp = QString::fromStdString(fp);

        // Logger den fulde sti til konsollen for debug formål
        qDebug() << "vellykket: "<<qstring_fp; // Output til terminal

        // Initialiserer soundEffect med stien til den instrumentale fil og denne klasse som parent
        soundEffect = new QSound(qstring_fp, this);

        // Starter afspilningen af den instrumentale lydfil
        soundEffect->play();
    }
}
```

Figur 66: initialisere af sangvisning og lydafspilning

Funktionen er der ingen eksterne parametre. Funktionen opererer med klassens interne medlemsvariabler, herunder "lyricsTimer", "lineCounter" og "soundEffect". Funktionen returnerer heller ingen værdi, da formålet er at initialisere visning og lydafspilning. For dette skal opnås benyttes den en "getInstrumentalFile"-metode til at hente stien til sangene.

```
// Konstruktor for HighScore-klassen
HighScore::HighScore(QWidget *parent) :
    QDialog(parent), // Kaldet konstruktøren for basisklassen QDialog
    ui(new Ui::HighScore), // Initialiserer brugergrænsefladen for HighScore
    menu(nullptr) // Sætter menu-pegeren til null
{
    ui->setupUi(this); // Konfigurerer UI-elementerne fra Ui::HighScore

    ScoreList scoreList; // Opretter et ScoreList objekt
    scoreList.getData(1); // Anmoder om data for spiller med ID 1

    QVector<QPair<QString, int>> testHighscores; // Opretter en liste til at opbevare highscores
    vector<Score> Top10 = scoreList.getScores(); // Henter top 10 scores fra scoreList

    // Loop igennem og tilføj kun de første 5 scores til testHighscores
    for(int i = 0; i < 5; i++) {
        std::string fp = Top10[i].getUser(); // Henter brugernavnet fra score objektet
        QString qstring_fp = QString::fromStdString(fp); // Konverterer std::string til QString
        testHighscores.append(QPair<QString, int>(qstring_fp, Top10[i].getScoreValue())); // Tilføjer til listen
    }

    // Vis highscores på skærmen
    showHighscores(testHighscores);

    // Opretter en knap, der giver brugeren mulighed for at vende tilbage til hovedmenuen
    QPushButton *on_pushButton_clicked = new QPushButton("Tilbage til Hovedmenu", this);
    // Forbinder knappens klik-hændelse til den relevante slot i denne klasse
    connect(on_pushButton_clicked, &QPushButton::clicked, this, &HighScore::on_pushButton_clicked);

    // Opsætter layoutet med highscore label og tilbage-knappen
    QVBoxLayout *layout = new QVBoxLayout;
    layout->addWidget(ui->highscoreLabel);
    layout->addWidget(on_pushButton_clicked);
    setLayout(layout); // Anvender layoutet til denne dialog

    // Konfigurerer farveskemaet for dialogboksen
    QPalette pal = palette(); // Henter dialogboksens nuværende palette
    pal.setColor(QWidget::Window, QColor(173, 216, 230)); // Sætter vinduets baggrundsfarve til lyseblå
    setAutoFillBackground(true); // Søger for at baggrunden tegnes automatisk
    setPalette(pal); // Anvender den nye palette til dialogboksen
}
```

Figur 67: kodeudsnit funktionen "HighScore"

Funktions formål er at fremvise top "highcores" i brugergrænsefladen, og lade brugeren tillade at navigere tilbage til hovedmenuen. Der hentes spillerdata fra serveren ved brug af 'getData', derudover så anvendes funktionen 'getScores' som returner en liste af "Score" objekter, der indeholder brugernavne og vedkommendes scoreværdi.

```
void HighScore::showHighscores(const QVector<QPair<QString, int>>& newHighscores)
{
    // Opdaterer den interne liste med de nye highscores
    highscoreData = combinedHighscores;

    // Starter en besked til at vise highscores med en stor overskrift
    QString message = "<span style='font-size: 72px; color: red;'><b>Highscores:</b></span><br><br>";

    // Tilføjer hver highscore til beskeden
    for (const auto& highscore : highscoreData) {
        message += "<span style='font-size: 36px;'><b>" + highscore.first + " :</b> " + QString::number(highscore.second) + "</span><br>";
    }

    // Opdaterer highscoreLabel (et UI-element) med den sammensatte besked
    ui->highscoreLabel->setText(message);

    // Indstiller dialogvinduet til at vise i fuld skærm
    setWindowState(Qt::WindowFullScreen);
}
```

Figur 68: Kodeudsnit af funktionen showHighscores

Funktionen 'showHighscores' er en QVector af QPair, hvor hvert 'pair' indeholder et QString-objekt med et brugernavn og en score. Den opdaterer og præsenterer highscores i karaoke-applikationen og viser, hvordan brugeren præsterer i forhold til andre spillere.

```
// Definerer en funktion, ScoreDisplay, som tager en sangs ID og bruger det til at vise scoren for den sang
void Results::ScoreDisplay(int song_id) {
// Antager at der findes en funktion, getScoreForSong, som kan hente en score baseret på en sangs ID
// Denne funktion er ikke vist her, men den vil hente scoren fra et sted i systemet
    Score score = getScoreForSong(song_id);

// Henter score værdien fra score objektet ved hjælp af en getter metode, getScoreValue
    int scoreValue = score.getScoreValue();

// Henter brugerens navn fra score objektet ved hjælp af en getter metode, getUser
    QString user = score.getUser();

// Henter datoen for scoren fra score objektet ved hjælp af en getter metode, getDate
    QString date = score.getDate();

// Forbereder en tekststreng, der indeholder brugerens score, navn og datoen for scoren
// %1, %2, og %3 er pladsholdere, der vil blive erstattet med score værdien, brugernavn og dato

    QString scoreText = QString("Score: %1\nUser: %2\nDate: %3")
        .arg(scoreValue) // Erstatte %1 med score værdien
        .arg(user) // Erstatte %2 med brugernavn
        .arg(date); // Erstatte %3 med dato
    ui->label->setText(scoreText);
// Opdaterer teksten i QLabel (et GUI element) med navnet 'label' til at vise den forberedte tekststreng
}
}
```

Figur 69: kodeudsnit af ScoreDisplay

ScoreDisplay-funktionen modtager en sang-ID som input og det bliver brugt til at hente og præsentere scoren for den pågældende sang. Ved at kalde ”getScoreForSong” antages det, at et Score-objekt returneres med oplysninger om sangens score. Derefter bruges objektets get-metoderne til at hente scoreværdien, brugesdrens navn og datoen for scoren. Disse oplysninger kombineres i en QString, som fremvises på brugergrænsefladens skærm. For en ydligere forståelse for anvendte biblioteker, henvises til bilags mappen for brugergrænseflade (bilag B).<sup>11</sup>

---

<sup>11</sup> (Qt, 2023)

## 8 Test og resultater

### 8.1 Modul- og integrationstest

#### 8.1.1 PSoC

Modultesten for PSoC fokuserer på at validere funktionaliteten af hver individuelle komponent ved at sikre deres korrekte integration. Testen verificerer at hvert modul opfylder dets specificerede krav, hvoraf integrationen med de resterende moduler er operationsdygtigt. Modultestens komponenter består Raspberry Pi 4 Model B (RPI4) og PSoC-5LP Prototyping Kit CY8CKIT-059 (PSoC), og testen for disse komponenter indeholder følgende.

#### Testudførelse

I testudførelsen laves en evaluering af systemets funktionalitet og effektivitet i overensstemmelse med de ønskede og forventede resultater. Da denne fase af projektet er væsentligt orienteret mod hardware, blev det besluttet at gennemføre individuelle tests for hver komponent for at sikre korrekt funktionalitet, og dertil er der blevet udformet en video for testen af PSoC'en, hvilket findes i Bilag.

#### Test 1: SPI-kommunikation mellem PSoC & RPI4

I test 1 blev der undersøgt, om PSoC'en og RPI'en kunne kommunikere igennem SPI-kommunikationen. Dertil blev en programmeringskode udviklet i PSoC Creator med en tilhørende video for at dokumentere kompileringen og en succesfuld kommunikation. Hertil opstod der dog problemer, idet RPI'en tilsyneladende ikke modtog data fra PSoC'en, på trods af at koden kørte og fyldte bufferen med data. Af disse grunde mistænkes der, at problemet skyldes hardware, og hertil hardwareforbindelserne mellem enhederne, hvilket har medført problemerne med SPI-kommunikationen.

#### Test 2: Analog signaltest

I test 2 blev PSoC'ens evne til at konvertere det monofoniske inputsignal evalueret for at sikre, at denne konvertering blev udført korrekt uden tab af lydkvalitet eller risiko for korruption af lydfilen. Denne test var betydelig for at verificere, at PSoC'en kunne håndtere den analoge indgang fra mikrofonen og forvandle den til et digitalt format på en pålidelig måde. Hertil opstod der en fejl ved korruption af lydfilen. Den korrupte tilstand af lydfilen forhindrede os i at opnå en brugbar lydfil.

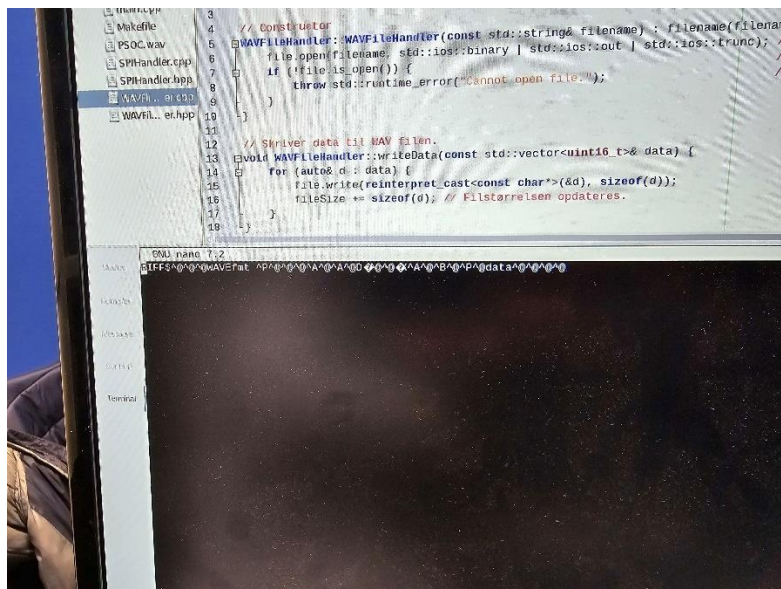


Under testprocessen observeres der yderligere potentielle fejl, som desværre dog ikke blev dokumentet grundet i mødegangen til afleveringsfristen. Rapportskrivningen blev derfor prioriteret fremfor fejlhåndteringen af systemfejl, hvoraf en rettelse af systemfejlen kunne løses senere hen.

### Test 3: WAV-formatering I realtid

I test 3 evalueres RPI'ens evne til i realtid at konvertere modtaget data fra PSoC'en til en WAV-fil i overensstemmelse med WAV-standard. Filens indhold analyseres omhyggeligt for at sikre overholdelse af det forventede lydsignal, herunder vurdering af header-information og andre relevante parametre i henhold til WAV-standard.

Resultaterne af testen indikerede, at RPI'en succesfuldt kunne generere WAV-filen med korrekte header-informationer og parametre, som også blev dokumenteret i den tidligere beskrevne testvideo. Da vi ankom til konklusionen af, at der var fejl i SPI-kommunikationen, indikerede dette også, at WAV-filen ikke kunne hente de nødvendige data fra RPI'en. Dette resulterede i en "tom" WAV-fil, der kun indeholdt header-informationer uden selve lyddataene. Den ovenstående beskrivelse af testen kan også ses på nedenstående billede, hvilket viser indholdet af PSoC.WAV-filen efter kompileringsprocessen.



Figur 70: Test af WAV-formatering

Testresultaterne afslørede udfordringer i systemets funktionalitet, primært relateret til hardwarekomponenterne og deres integration. Mikrofonen og forstærkeren udførte korrekte

lydoptagelser, men uventede dataoverførselsfejl og korrupte lydfiler antydede problemer med PSoC'en og RPi4'en under dataoverførsel og behandling af det analoge signal.

RPi4'en kunne oprette en .wav-fil med korrekt header, men "korrupt data" indikerer problemer i overførslen af lydsignalet fra PSoC til RPi4. Den upålidelige SPI-kommunikation mellem PSoC og Raspberry Pi var en betydelig faktor i systemets utilstrækkelige ydeevne. Tidsmangel på grund af eksterne opgaver påvirkede også rapportudførelsen og andre administrative opgaver, hvilket begrænsede muligheden for finjustering og fejlfinding.

Samlet set indikerer disse udfordringer behovet for nærmere undersøgelser og fejlfinding på både hardware- og softwareniveau. Forbedringer i komponentforbindelser og optimering af dataoverførselsprotokoller vil sandsynligvis være afgørende for at rette op på systemets manglende funktionalitet og opnå pålidelig og effektiv ydeevne i fremtidige arbejder.

## 8.2. Server

For at teste server samt moduler er der blevet oprettet en test fil, der tester alle individuelle klasser. På denne måde kan funktionaliteten af vores funktioner ses. Forneden kan et kode-udsnit ses, der viser hvordan "Song" klassen testes:

```
int main()
{
    // SONG CLASS TEST
    Song song(1);
    string songString = song.getData(1);
    cout << "OG STRING:    " << songString << endl;
    song.parseString(songString);
    cout << "ARTIST:      " << song.getArtist() << endl;
    cout << "TITLE:       " << song.getTitle() << endl;
    cout << "LENGTH:     " << song.getDuration() << endl;
    cout << "KEY:        " << song.getKey() << endl;
    cout << "INSTRUMENTAL: " << song.getInstrumentalFile() << endl;
    cout << "MELODY:     " << song.getCmpMelodyFile() << endl;
    cout << "LYRICS:     " << song.getLyricsFile() << endl;
```

Figur 71: 'Song' test C++

Der oprettes en main, hvori song kaldes. Som tidligere nævnt i vores design afsnit vil denne printe de korrekte data for en song string, alt efter hvilken funktion der bliver kaldt. F.eks. når "getArtist" funktionen bliver kaldt, vil den blive slottet ind i "ARTIST:" Samme funktionalitet er gældende for de resterende funktioner, på nær for "addScore":

```
// TEST ADD FUNCTION
Score score;
score.addScore(1, 9000, "Boobman69", "2011-11-11");
```

Figur 72: 'addScore' test C++

Her tilføjes der en vilkårlig score, med attributterne for 'ScoreString'. Denne dukker op i vores terminal når testprogrammet køres. Den resterende kode for test klassen kan findes under bilag.

```
Received 27059402 bytes (100.00%)
Melody: ghost-vocals.wav
File opened
File size: 1058
FSN: 570687488
Received 1000 bytes (94.52%)
Received 1058 bytes (100.00%)
Lyrics: ghost-lyrics.txt
OG STRING: 1|Justin Bieber|Ghost|153|D|ghost-music.wav|ghost-vocals.wav|ghost-lyrics.txt
ARTIST: Justin Bieber
TITLE: Ghost
LENGTH: 153
KEY: D
INSTRUMENTAL: ghost-music.wav
MELODY: ghost-vocals.wav
LYRICS: ghost-lyrics.txt
Socket created
Connected to server!
result: 1|1|90|user1|2023-01-01,3|1|9000|Boobman69|2011-11-11,4|1|9000|Boobman69|2011-11-11,5|1|9000|Boobman69|2011-11-11,6|1|9000|Boobman69|2011-11-11,7|1|9000|Boobman69|2011-11-11,8|1|9000|Boobman69|2011-11-11,
OG STRING: 1|1|90|user1|2023-01-01,3|1|9000|Boobman69|2011-11-11,4|1|9000|Boobman69|2011-11-11,5|1|9000|Boobman69|2011-11-11,6|1|9000|Boobman69|2011-11-11,7|1|9000|Boobman69|2011-11-11,8|1|9000|Boobman69|2011-11-11,
ScoreList:
ID: 1 Song ID: 1 Score: 90 User: user1 Date: 2023-01-01
ID: 3 Song ID: 1 Score: 9000 User: Boobman69 Date: 2011-11-11
ID: 4 Song ID: 1 Score: 9000 User: Boobman69 Date: 2011-11-11
ID: 5 Song ID: 1 Score: 9000 User: Boobman69 Date: 2011-11-11
ID: 6 Song ID: 1 Score: 9000 User: Boobman69 Date: 2011-11-11
ID: 7 Song ID: 1 Score: 9000 User: Boobman69 Date: 2011-11-11
ID: 8 Song ID: 1 Score: 9000 User: Boobman69 Date: 2011-11-11
9000
Socket created
Connected to server!
9000|Boobman69|2011-11-11|C-string: 9000|Boobman69|2011-11-11|size: 25
ase@ubuntu:~/Syng og Sammenlign/db server/clients$
```

Figur 73: Terminal vindue af C++ Test

Ovenstående billede viser resultatet i terminalen når testen køres. Her kan det ses at alt funktionalitet fungerer som forventet. Det samme er gældende for vores Python test klasse. Dette kan også ses på nedenstående kode-udsnit:

```
def test_song():
    # SONG CLASS TEST
    song = py_dbserver.Song(1)
    song_string = song.get_data(1)
    print("OG STRING: ", song_string)
    song.parse_string(song_string)
    print("ARTIST: ", song.get_artist())
    print("TITLE: ", song.get_title())
    print("LENGTH: ", song.get_duration())
    print("KEY: ", song.get_key())
    print("INSTRUMENTAL: ", song.get_instrumental_file())
    print("MELODY: ", song.get_cmp_melody_file())
    print("LYRICS: ", song.get_lyrics_file())
```

Figur 74: 'Song' test Python

Som det kan ses, fungerer Python test af ”Song” klassen på nogenlunde samme måde som vores C++ test. Det samme gør sig gældende for de resterende test, inklusiv ”addScore”.

```
Received 27059402 bytes (100.00%)
Melody: ghost-vocals.wav
File opened
File size: 1058
FSN: 570687488
Received 1000 bytes (94.52%)
Received 1058 bytes (100.00%)
Lyrics: ghost-lyrics.txt
OG STRING: 1|Justin Bieber|Ghost|153|D|ghost-music.wav|ghost-vocals.wav|ghost-lyrics.txt
ARTIST: Justin Bieber
TITLE: Ghost
LENGTH: 153
KEY: D
INSTRUMENTAL: ghost-music.wav
MELODY: ghost-vocals.wav
LYRICS: ghost-lyrics.txt
Socket created
Connected to server!
result: 1|1|90|user1|2023-01-01,3|1|9000|Boobman69|2011-11-11,4|1|9000|Boobman69|2011-11-11,5|1|9000|Boobman69|2011-11-11,6|1|9000|Boobman69|2011-11-11,7|1|9000|Boobman69|2011-11-11,8|1|9000|Boobman69|2011-11-11,9|1|9000|Boobman69|2011-11-11,
OG STRING: 1|1|90|user1|2023-01-01,3|1|9000|Boobman69|2011-11-11,4|1|9000|Boobman69|2011-11-11,5|1|9000|Boobman69|2011-11-11,6|1|9000|Boobman69|2011-11-11,7|1|9000|Boobman69|2011-11-11,8|1|9000|Boobman69|2011-11-11,9|1|9000|Boobman69|2011-11-11,
ScoreList:
9000
get_id: 1
get_song_id: 1
get_score_value: 90
get_user: user1
get_date: 2023-01-01
get_id: 3
get_song_id: 1 . . .
```

Figur 75: Terminal billede af Python Test

Grundet den manglende overload af '<<' i vores wrapper, kommer printet selvfølgelig til at se anderledes ud. Den fortsætter for hvert element i vektoren og ender med 'addScore', som også fungerer som vist i C++ testen.

### 8.1.2 LogicUnit

Når der køres modultest af Logic enheden er der blevet sammenlignet med en på forhånd optaget lydfil til at kunne simulere vores endelige system, da vi desværre ikke var i stand til at optage direkte fra vores mikrofon, og få på den lydfil optaget igennem denne. Derefter tager Logic enheden og sammenligner den optagede lydfil, med den valgte Sample sang. I dette tilfælde er der blevet sammenlignet med Sang 1 som er Justin Bieber's sang "Ghost". Når denne sammenligning sker, kommer der dette print ud i terminalen, som fortæller brugerens samlede score for sin indspilning som kan ses på følgende Billede:

Den samlede score er: 1634.4262976504433

Figur 76 Modultest for logic\_Unit

Modultesten gik ud på at se om systemet var i stand til korrekt at sammenligne de 2 lydfiler og give en samlet score fra 1-100. Dette er dog som det kan ses ikke lykkedes i forhold til den ønskede implementation og effekt.

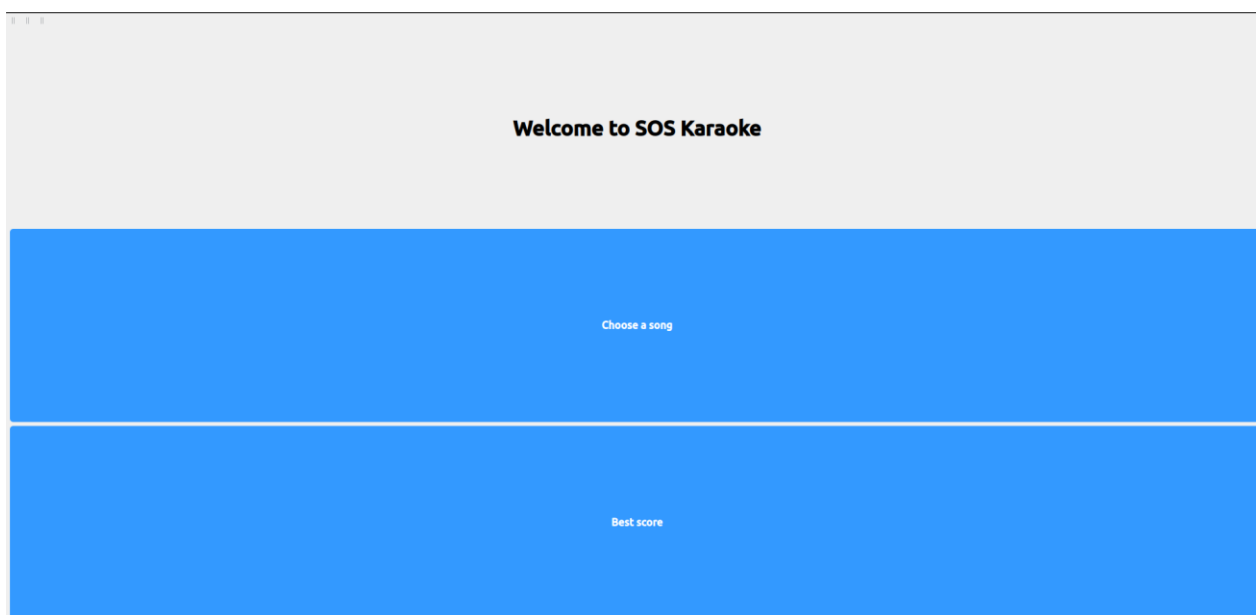
Det har desværre vist sig at være en udfordrende opgave at fuldføre en velfungerende Python-implementering baseret på det tidligere udviklede MATLAB-skelet. På trods af vores ihærdige bestræbelser og tidsmæssige forpligtelser var det ikke muligt for os at levere en fuldt fungerende Python-implementering inden for den fastsatte afleveringsfrist. Dette har også betydet at integration med de andre moduler ikke har været en mulighed i det logikken først og fremmest ikke har virket som ønsket.

Denne situation har desværre medført kritiske konsekvenser i det samlede system, idet vores scorings-funktionalitet ikke kan integreres effektivt med de resterende moduler. Dette udgør en betydelig udfordring, da det underminerer vores evne til at opnå en præcis evaluering af brugerens sangpræstation og opfylde vores projektmål.

### 8.1.3 Brugergrænseflade

Da der blev foretaget modultest af brugergrænsefladen for SOS karaoke-systemet, blev det sikret, at hver selvstændig del af systemet fungerede korrekt inden integrationen med andre moduler.

Startmenuen er her hvor brugeren præsenteres for startmenuen, hvor de har mulighed for at vælge mellem at se "Best score" eller "Choose a song"



Figur 77: Visning af menu

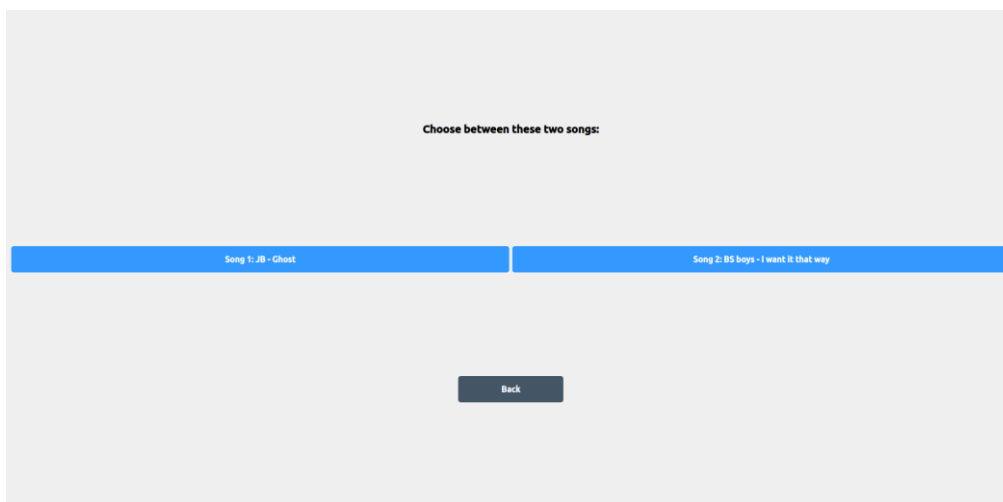
Ved valget af "Best score" kan brugeren navigere til highscore-skærmen. Her tester vi brugergrænsefladen for at vise scores fra en hardcoded liste. Derudover ses også tilbageknappen, som giver brugeren mulighed for at vende tilbage til startmenuen.



Figur 78: Visning af highscore

Ved at trykke tilbage vender man blot tilbage til startvinduet/menuen.

Når der vælges "Choose a song", føres der videre til sangliste-skærmen, hvor der kan vælges mellem de tilgængelige sange, her tester vi overgangene mellem skærme samt selve sangvalget.



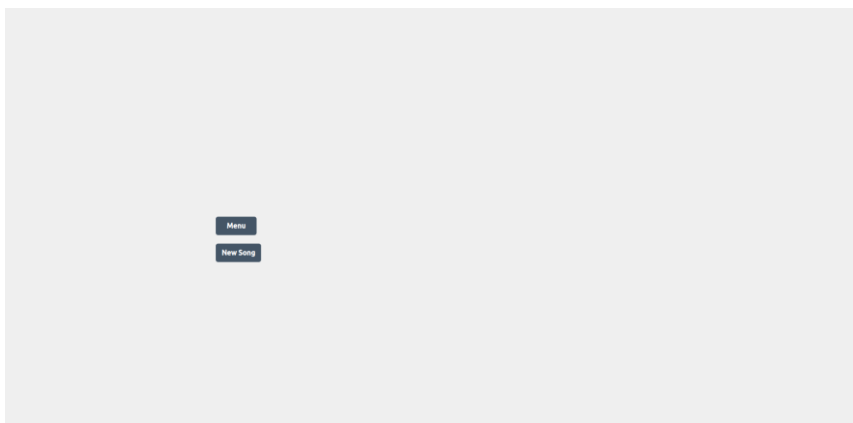
Figur 79: Visning af songlist

Herfra kan man vælge at vende tilbage til menuen eller vælge ønskede sang hvorefter nedtællingen og visning af sangteksterne påbegynder. Der testes brugergrænsefladen til at håndtere synkronisering mellem tekstvisning og nedtælling. Her kan brugeren enten vælge at "Cancel" eller fortsætte efter sangen ved at trykke på "Continue", hvor scoren fremvises.



*Figur 80: Visning af song1*

Da manglede implementering af LogicUnit så er resultat-skærmen tom på nuværende tidspunkt. Dog tester der stadigvæk brugernes mulighed for at navigere tilbage til sanglisten eller tilbage til startmenuen.



*Figur 81: Visning af results*

Ved anvendelsen af qDebug bliver programmet handlinger under kørsel logget i terminalen, hvilket giver feedback til programmets tilstand og hjælper med at identificere problemer i realtid.

```

Lyrics loaded successfully!
Lyrics loaded successfully!
Updating lyrics, lineCounter: 0
Updating lyrics, lineCounter: 0
Updating lyrics, lineCounter: 4
Updating lyrics, lineCounter: 4
Updating lyrics, lineCounter: 8
Updating lyrics, lineCounter: 8
  
```

*Figur 82: Visning af terminalen*



#### 8.1.4 Server & UI

Integration mellem Server & UI var en lettere proces. Da projektet endte med at have en utraditionel kombination, er en nærmest 'Coupled UI', hvor 'backend' og 'frontend' er i det samme modul, såvel som IDE i form af 'QTcreator'. Den eneste udfordring som ikke var forventet var 'QT's' meget integrerede system hvorpå de har deres egne Q- versioner af de normale datatyper man kender. Dette gjorde at det blev nødvendigt at lave en hurtig konversion, som heldigvis ikke var et problem da 'QT' har en lang række hjælpefunktioner til konvertering fra populære datatyper. Dette var så også første gang at serveren blev implementeret på den reelle HW. Dette blev gjort ved brug af mobilhotspot som blev forbundet til RPi4 og RPiZero, som var en relativ let proces.

## 8.2 Accepttest

Grundet flere problematikker i implementeringsfasen endte det med at det ikke lykkedes at gennemføre accepttesten.

Use Case 1: "Brugeren vælger en sang"	Test	Forventet resultat	Resultat	Godkendt/ kommentar
1	Brugeren starter programmet ved brug af en ekstern mus.	Den grafiske brugergrænseflade præsenteres for brugeren på hovedmenuen.	O.K.	Godkendt
2	Systemet henter sanglisten fra databasen	Serveren sender dataene angående sanglisten til systemet, og kan ses som et print statement i terminalen under test.	Sender data på det forkerte tidspunkt	Delvist Godkendt
3	Brugeren navigerer gennem systemet med musen.	de forskellige vinduer kan tilgås ved brug af brugergrænsefladen og musen.	O.K.	Godkendt
4	[EXT1] Brugeren tilgår "Best score", fra hovedmenuen med musen	Brugeren præsenteres for "Best Score" vinduet.	O.K.	Godkendt
5	Brugeren trykker på menuen "sange"	Brugeren bliver præsenteret med vinduet hvorpå de mulige sange er tilgængelige.	O.K.	Godkendt
6	[EXT 2] brugeren klikker på "tilbage" knappen	Brugeren bliver taget tilbage til det forrige vindue.	O.K.	Godkendt
7	Brugeren vælger en sang fra vinduet hvorpå sangene er præsenteret	Brugeren vælger en sang og en nedtælling starter.	O.K.	Godkendt

Tabel 9 Accepttest UC1

Use Case 2: "Syng og sammenlign"	Test	Forventet resultat	Resultat	Godkendt/ kommentar
1	Brugeren har trykket på den ønskede sang	En nedtælling begynder, hvorefter hele sangens lyrik præsenteres i sekventielt skiftende strofer.	O.K.	Godkendt
2	[EXT 1] Brugeren trykker på "cancel/back" under indspilning af sangen	Brugeren præsenteres for vinduet med sanglisten igen.	O.K.	Godkendt
3	Systemet sammenligner lydclip og kommer med en score.	En repræsentativ score for brugerens input, præsenteres på brugergrænsefladen.	Ikke implemente ret	Ikke godkendt
4	Brugeren benytter sig af "hovedmenu" knappen efter indspilning.	Efter en indspilning benytter brugeren sig af "hovedmenu" knappen og præsenteres herefter for hovedmenuen.	Ikke Implemente ret	Ikke godkendt
5	[EXT 2] brugeren klikker på "new song", efter en indspilning	Efter en indspilning benytter brugeren sig af "new song" knappen og præsenteres herefter for sanglisten.	Ikke Implemente ret	Ikke godkendt

Tabel 10 Accepttest UC2

## 9 Diskussion af resultater

Projektet har, generelt set, desværre ikke levet op til de oprindelige forventninger. Fra starten var ambitionen at skabe et produkt, der opfyldte de indledende specifikationer, men dette mål blev aldrig realiseret. Flere faktorer bidrog til denne udfordring, herunder især kommunikationsproblemer og usikkerhed omkring ansvarsområder. En primær årsag til projektets vanskeligheder var den manglende kommunikation mellem de forskellige moduler. Dette var særligt tydeligt i samspillet mellem UI og LogicUnit, hvor der var en vedvarende uenighed og forvirring omkring, hvor grænserne for hvert moduls ansvar lå. Efter en periode med usikkerhed blev det dog besluttet, at LogicUnit skulle fokusere på databehandling, mens UI overtog en større del af funktionaliteten – en rollefordeling, der i bagspejlet burde have været klarlagt som en del af 'backend'- funktionalitet.

En anden væsentlig faktor, der bidrog til projektets udfordringer, var en mangel på teknisk viden inden for specifikke domæner. For eksempel var det oprindelige koncept med live-sampling ved hjælp af PSoC måske for ambitiøst, givet projektgruppens tekniske baggrund. Dette, kombineret med en tung semesterbelastning, resulterede i, at flere nødvendige opgaver ikke blev fuldført inden for den fastsatte tidsramme.

På trods af disse udfordringer havde projektet også positive aspekter. Nogle moduler fungerede delvist, som det var forventet, og opfyldte kravene i accepttesten til en vis grad. Dette indikerer, at der trods alt var elementer i projektet, der blev succesfuldt implementeret.

Derudover skal det også nævnes, at alle projektdeltagere blev udfordret både teknologisk og i deres evne til at administrere et større projekt. Denne erfaring, selvom den var udfordrende, har været lærerig og vil utvivlsomt bidrage til deltagernes fremtidige professionelle udvikling.

Afslutningsvis, selvom det endelige produkt ikke levede op til de oprindelige forventninger, har projektet givet vigtige lektioner i projektledelse, teamkommunikation, og den tekniske udviklingsproces.

## 10 Konklusion & Perspektivering

Projektet Syng og Sammenlign, har været en lærerig rejse, med projektets centrale mål at implementere en karaoke-maskine, der udnyttede de tekniske og faglige komponenter fra semestret som helhed.

Projektet har med succes udforsket og anvendt digital signalbehandling til logikhåndteringen af dataene og netværksprogrammering til at etablere en server, hvilket har resulteret i en dyb forståelse af disse teknologier. Desuden har projektet introduceret os for nye områder som databaser og brugergrænsefladedesign, hvilket har styrket vores kompetencer og viden inden for disse områder, samt gjort brug af tidligere erfarede teknologier som SPI-kommunikation og analog til digital konvertering.

### **Vurdering af teamwork og projektstyring:**

Projektet startede med en udfordrende periode, hvor gruppens medlemmer havde travlt med at tilpasse sig de nye fag på semesteret. Dette medførte ujævn deltagelse i gruppearbejdet i starten, men gruppen formåede stadig at holde faste møder hver torsdag kl. 11 under vejledning af Lars Mortensen og fik kort efter en bedre struktur på hvordan gruppen målrettet arbejdede sammen, som følger af SCRUM-kurset ved Systematic. Med valgt SCRUM-master og produktejer, havde vi nogle klare rammer og beslutningstagere der var gjorde ledelse af projektet stærkere og bedre.

### **Perspektiver for det udviklede system:**

Selvom projektet ikke blev fuldt realiseret inden for tidsrammen for afleveringsfristen, har det stadig resulteret i værdifulde tekniske erfaringer og resultater. Selvom accepttesten ikke blev fuldt gennemført, har vi eksperimenteret, fejlet og afprøvet en overflod af metoder og tilgange for at opnå det bedst mulige resultat.

Det udviklede system var ambitiøst og strakte sig ud over projektets oprindelige omfang. Et fuldt fungerende karaokesystem er en betydelig opgave, og mange af de implementeringer og metoder, vi har brugt, er kun en brøkdel af, hvad et sådant system kræver. Desværre har vi ikke implementeret alle de ønskede kerne-funktionaliteter, hvilket er ærgerligt, da det ville have været spændende at teste systemet i sin helhed.

Samlet set har projektet udfordret os teknisk og fagligt og givet os mulighed for at anvende vores tillærte viden i praksis. Det har også styrket vores evne til at arbejde i teams og håndtere de udfordringer projektarbejder kommer med. Selvom det udviklede system ikke er fuldendt, har det skabt en solid platform for fremtidig udvikling og læring.

## Referenceliste

- alldatasheet. (n.d.). *alldatasheet*. Retrieved from BCM2835 Datasheet (PDF) - Broadcom Corporation.: <https://www.alldatasheet.com/datasheet-pdf/pdf/502533/BOARDCOM/BCM2835.html>
- BilstereoKlub. (2020, maj 26). *Youtube*. Retrieved from Phonokabel- RCA stik gør det selv guide: [https://www.youtube.com/watch?v=\\_DnPgpdONfM](https://www.youtube.com/watch?v=_DnPgpdONfM)
- CYPRESS. (2017, 7 26). *ADC Successive Approximation Register (ADC\_SAR)*. Retrieved from file:///C:/Program%20Files%20(x86)/Cypress/PSoC%20Creator/4.4/PSoC%20Creator/psoc/content/CyComponentLibrary/CyComponentLibrary.cylib/ADC\_SAR\_v3\_10/ADC\_SAR\_v3\_10.pdf
- CYPRESS. (2017, 12 12). *Delta Sigma Analog to Digital Converter (ADC\_DelSig)*. Retrieved from file:///C:/Program%20Files%20(x86)/Cypress/PSoC%20Creator/4.4/PSoC%20Creator/psoc/content/CyComponentLibrary/CyComponentLibrary.cylib/ADC\_DelSig\_v3\_30/ADC\_DelSig\_v3\_30.pdf
- Fahad, E. (2020, November 12). *electronicclinic*. Retrieved from SPI Serial Peripheral Interface in Raspberry Pi: <https://www.electronicclinic.com/spi-serial-peripheral-interface-in-raspberry-pi/>
- fileformat. (n.d.). *fileformat*. Retrieved from WAV fileformat: <https://docs.fileformat.com/audio/wav/>
- geeksforgeeks. (2023, November 24). *geeksforgeeks*. Retrieved from .wav Audio Format: <https://www.geeksforgeeks.org/wav-audio-format/>
- Jakob, W. (2017). *pybind11 documentation*. Retrieved from pybind11 readthedocs.io: <https://pybind11.readthedocs.io/en/stable/basics.html>
- lotguider. (2019, Februar 3). *iot-guider*. Retrieved from Using Serial Peripheral Interface (SPI) in Raspberry Pi: <https://iot-guider.com/raspberrypi/using-spi-in-raspberry-pi/#:~:text=First%20of%20all%2C%20we%20will%20enable%20SPI%20in,Finishbutton%20S elect%20Yeswhen%20it%20asks%20for%20a%20reboot>
- McCauley, M. (ukendt, ukendt ukendt). *bcm2835*. Retrieved from [https://www.airspayce.com/mikem/bcm2835/group\\_\\_spi.html](https://www.airspayce.com/mikem/bcm2835/group__spi.html)
- Mikkelsen, P. H. (2021). *Connecting Rpi/ASE fHat to PSoC*. Retrieved from [https://redweb.ece.au.dk/devs/projects/raspberry-zero/wiki/Connecting\\_RpiASE\\_fHat\\_to\\_PSoC](https://redweb.ece.au.dk/devs/projects/raspberry-zero/wiki/Connecting_RpiASE_fHat_to_PSoC)
- NumPy. (n.d.). Retrieved from <https://numpy.org/>
- POSIX. (2001). *linux.die.net*. Retrieved from Linux Man Page: <https://linux.die.net/man/3/ntohl>
- ProgrammingKnowledge. (2016, april). *youtube*. Retrieved from QT C++ GUI Tutorial For Beginners: <https://www.youtube.com/playlist?list=PLS1QuIWo1RIZiBcTr5urECberTITj7gjA>
- Qt. (2023). *Qt*. Retrieved from All Qt C++ classes: <https://doc.qt.io/qt-6/classes.html>
- raspberrypi. (n.d.). *raspberrypi*. Retrieved from Raspberry Pi Documentation: <https://www.raspberrypi.com/documentation/computers/processors.html>
- SciPy v1.11.4 Manual. (n.d.). Retrieved from <https://docs.scipy.org/doc/scipy/reference/signal.html>
- SciPy v1.11.4 Manual, n.d. (n.d.). Retrieved from <https://docs.scipy.org/doc/scipy/reference/generated/scipy.io.wavfile.read.html>
- SjioGG. (2023, Oktober 12). *Github.com*. Retrieved 12 4, 2023, from NGK/TCP\_Basic\_File\_server: [https://github.com/SjioGG/NGK/tree/main/TCP\\_Basic\\_File\\_server](https://github.com/SjioGG/NGK/tree/main/TCP_Basic_File_server)

ukendt. (ukendt, ukendt ukendt). WAV. Retrieved from FILEFORMAT:  
<https://docs.fileformat.com/audio/wav/>



## Bilagsoversigt

BILAG:	NAVN:	FORMAT:
A:	Bilag A - Udvidet Procesbeskrivelse	PDF
B:	Bilag B - Brugergrænsefladen (GUI)	PDF
C:	Bilag C - Udvidede Risikoanalyse	PDF
D:	Bilag D - Tidsplan	PDF
E:	Bilag E - Funktionsbeskrivelser	PDF
F:	Bilag F – Udvidet domæneanalyse	PDF
G:	Bilag G - Datablad	PDF
H:	Bilag H - ProjektFormulering	PDF
I:	Bilag I - Vejleder-Reviewmøder	PDF
J:	Bilag J – Samarbejdskontrakt	PDF
K:	Bilag K – Logbog	PDF
Y:	Bilag Y – Source Code	ZIP
X:	Bilag X - Accept-Modul-Test-Video	MP4

Source code also available on GitHub: [SjioGG/Syng\\_og\\_Sammenlign \(github.com\)](https://github.com/SjioGG/Syng_og_Sammenlign)