Design Specification

Editor: Klas Gudmundsson

Version 0.1

Status

Reviewed		
Approved		

PROJECT IDENTITY

Group members

Name	Responsibility	Phone	Email
Linnea Faxén	Project Manager (PM)	073-829 65 09	linfa440@student.liu.se
Klas Gudmundsson	Documentation Manager (DM)	072-714 06 66	klagu863@student.liu.se
Eskil Jörgensen	Chief of Design (CoD)	076-222 53 72	eskjo325@student.liu.se
Stefan Lundström	Hardware Specialist	070-379 80 79	stelu332@student.liu.se
Ema Becirovic	Test Manager (TM)	073 564 15 73	emabe844@student.liu.se
Javier Preciado	Software Specialist	+34 647 92 61 38	javpr698@student.liu.se

Customer: ISY, Linköpings universitet, 581 83 Linköping
Customer contact: Mikael Olofsson, mikael.olofsson@liu.se
Course leader: Danyo Danev, danyo.danev@liu.se
Tutor: Christopher Mollén, christopher.mollen@liu.se

Contents

Document History

Version	**Date**	**Changes**	**Sign**	**Reviewed**
0.1	2016-09-30	First draft.	KG	EJ,EB,KG

1 Introduction

In the course TSKS05, a group of students are tasked with a project. This year, the group specified under Project Identity have been given the task to construct a Massive Audio Beamformer (MAB) that can be used as a demonstrator for Massive Multiple Input, Multiple Output (MIMO). This document will provide a detailed description of the planned system so that the reader can get to know how it will work.

The MAB will be able to focus, or beamform, sound to terminals. It will have an array that transmits signals. By beamforming it will direct these signals to specific terminals. Terminals will receive the signal intended for them along with noise but should not receive the signals of the other terminals. The MAB is illustrated in Figure 1. The group has decided to name this Massive Audio Beamformer "BAM!" which is MAB backwards and then an exclamation mark at the end to make it sound more powerful.

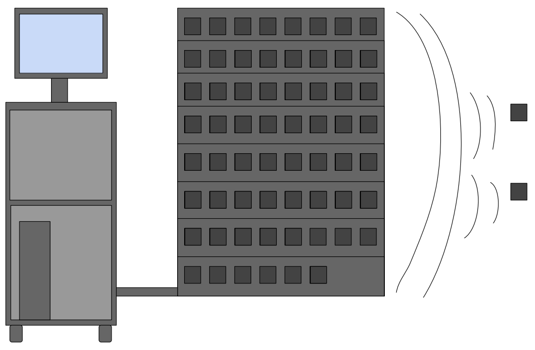

Figure 1: Example of finished product

A user will interact with the MAB via a user interface on a computer. Using the MAB the user can control which antennas should be in the array or in the terminals and what data to transmit. The antennas in the MAB will be 64 combined loudspeaker and microphone units (L/M units). Since the MAB will beamform audio, transmitting a signal corresponds to playing it in a loudspeaker. Receiving a signal corresponds to recording the sound, using a microphone.

To demonstrate the powers of massive MIMO in a pedagogic manner, two different sounds will be played at two different terminals in a room. The sounds are played from the array but through beamforming only one of the sounds is heard at one terminal and the other sound at the other terminal. This will demonstrate the abilities of massive MIMO to direct signals to different users.

1.1 Purpose of BAM!

The purpose of BAM! is to be an easy to use, easy to upgrade, demonstrator for massive MIMO.

1.2 Definitions

Throughout this document a number of abbreviations are used. All of them are defined in Table 1.

Table 1: Abbreviations used in the document

Abbreviation	**Definition**
A/D	Analog to digital
API	Application programming interface
APP	MAB Application
D/A	Digital to analog
DAL	Data Acquisition Library
GUI	Graphical user interface
I2C	Inter-Integrated Circuit
L/M unit	Loudspeaker/microphone unit
LED	Light-emitting diode
MAB	Massive Audio Beamformer
MIMO	Multiple input multiple output
PCI	Peripheral Component Interconnect
SNR	Signal to Noise Ratio
USB	Universal Serial Bus

2 Overview of the System

The system consists of two subsystems, hardware and software. Software is further divided into an application programming interface (API) and an Application. The user interacts with the Application which uses functions from the API to control the hardware. The hardware contains all L/M units and the necessary electronics to control them. Figure 2 shows a basic overview of the system. There are three large areas of the project: assembling the hardware, making the API work and creating an Application. In the coming sections, the hardware, API and Application will be described.

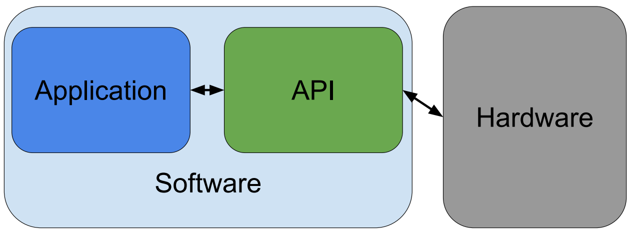

Figure 2: System overview

The hardware is explained in detail in Section 3.

3 Hardware

The design of the whole system hardware has been created by the customer. What remains is to construct the hardware according to the design and control it via a computer. To construct the hardware all L/M units must be soldered, assembled and tested. Some other parts of the hardware described in the coming sections such as signal splitters and cross coupling board must also be assembled and tested. In order to control the hardware, the data to transmit must be converted from digital to analog, the data to receive must be converted from analog to digital and the mode of each L/M unit must be set. Figure 3 shows an overview of the hardware and its subsystems.

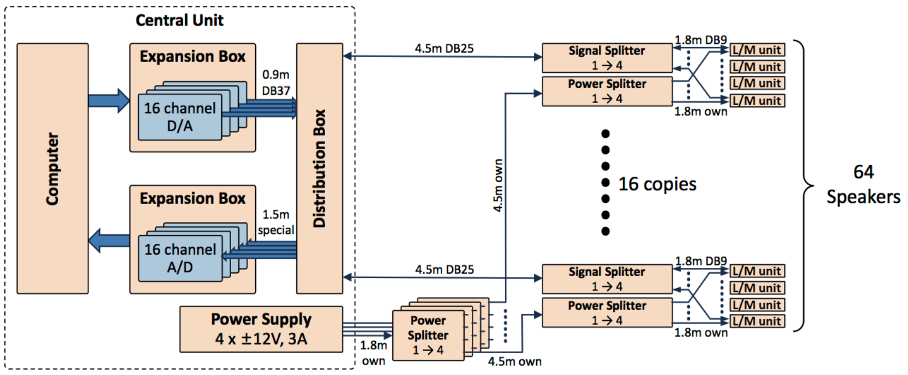

Figure 3: Hardware overview

As can be seen in Figure 3, the hardware will have a central unit that directs the data to send and control signals from the computer to the L/M units with the help of two expansion boxes and a distribution box. It will also direct the received data from the L/M units to the computer. The cables carrying the signals to and from the 64 speakers are connected to the central unit. There are 16 cables, going out from the distribution box which are split into 4 new cables each at the signal splitters. A power supply will also be located in the central unit. The power is then split with power splitters and distributed to all L/M units. The rest of this chapter will describe each part of the hardware.

3.1 Central Unit

The purpose of the central unit is to direct all signals. This consists of directing the data to transmit to the D/A-card, the converted signals to the correct L/M-units, directing the received data from the L/M-units to the A/D-converter, directing the converted signals back to the computer and directing control signals from the computer. It will consist of a computer, two expansion boxes, a distribution box and a power supply, as can be seen in Figure 3.

3.1.1 Computer

The computer is a Windows PC. Each of the two expansion boxes will be connected via a DVI cable connected to a PCI-Express card on the motherboard of the computer. The computer will perform all signal processing needed to perform beamforming. It will also control the A/D- and D/A-cards.

3.1.2 Expansion Box

The first expansion box will contain four 16-channel A/D-converters and the second will contain four 16-channel D/A-converters. The D/A-converters are used to convert the digital signals from the computer to analog signals to be played at the L/M units. Similarly, the A/D-converters are used to convert the recorded analog signals at the L/M units into digital signals to be stored in the computer.

The A/D-boards have a sample rate of 100 ksamples/s in total. The converter converts one channel at a time. Therefore the sample rate per channel is 6250 samples/s. The D/A-boards have a sample rate of 100 ksamples/s per channel since it contains 16 D/A-converters that are run simultaneously. There are four 16-channel boards giving a total of 64 channels. The A/D-board is of type Contec AD12-16 (PCI) and the D/A-board is of type Contec DA12-16 (PCI).

To avoid folding or aliasing in the sampled signals, the bandwidth of the sampled signal must be less than half the sampling frequency, also known as the Nyquist frequency;

$$B < \frac{f_s}{2}$$

.

In our system, $f_s = 6250$ Hz as stated above, which gives us that $B < 3125$ Hz. This limits the possible signals to transmit. The human hearing range covers 20-20 000 Hz so there is a significant part of it we cannot transmit in. However, in telephony the voice frequency covers approximately 300-3000 Hz. Therefore at least speech should be possible to transmit.

3.1.3 Distribution Box

The distribution box contains electronics to distribute power and signals between the A/D- and D/A-converters and the L/M units. It also contains electronics to control the transmit and receive relays in the L/M units. In Figure 4, an overview of the distribution box is presented.

The distribution box has three major groups of components for three different tasks. The first group is connection boards which connect data cables from the D/A-cards to the L/M units. The second group is analog boards that connect the data cables from the A/D-cards to the L/M units. The third task is to direct control signals which is done in a cross coupling board. The control signals from this board is then connected to digital boards that control the relays in the L/M-units. The digital boards contain Arduinos to distribute the relay signal to 16 L/M units each.

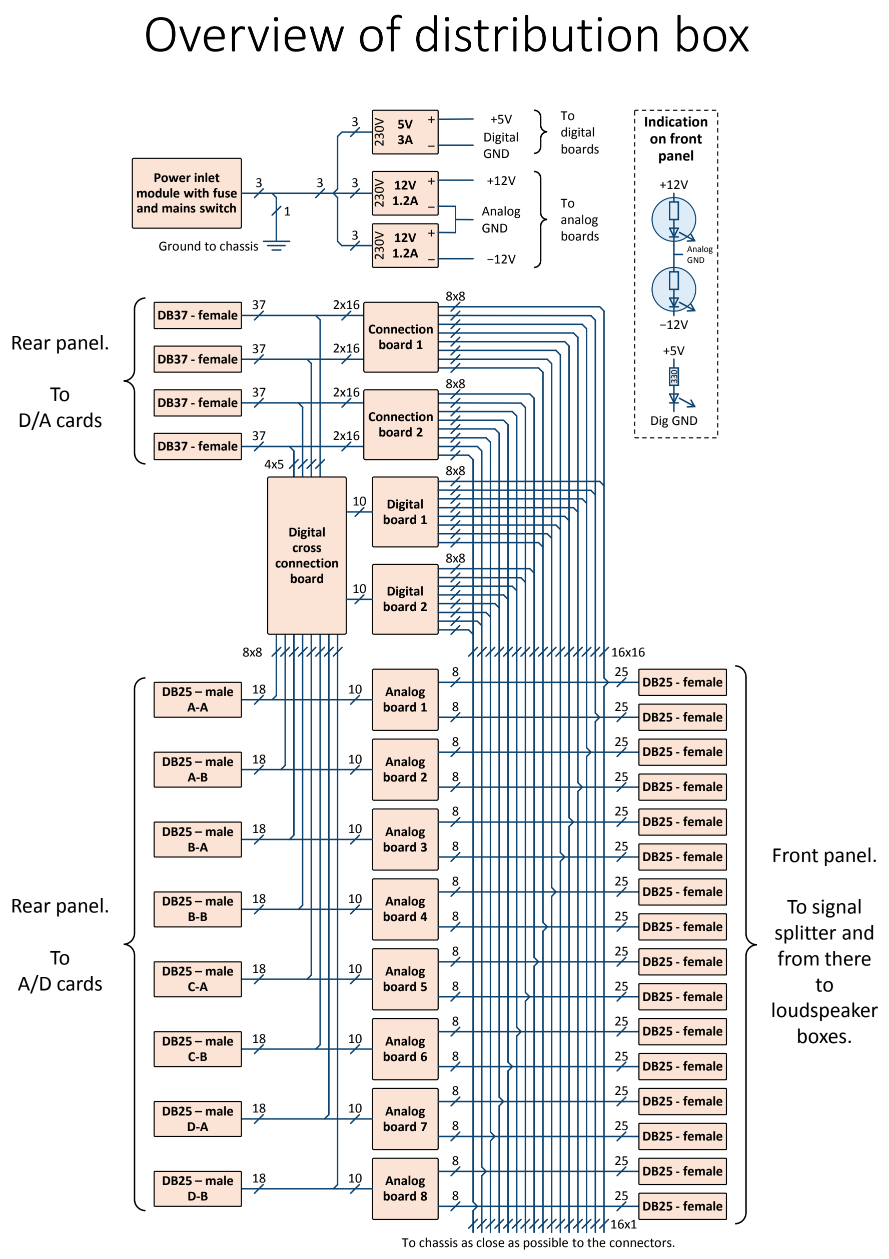

Figure 4: Distribution Box

3.1.4 Cross-coupling board

The cross-coupling board is a coupling board in the distribution box. Control signals from the A/D- and D/A-cards can here be directed to ports on the digital boards. The cross-coupling board can be seen in Figure 5. The configuration of connections is flexible. Since it is a coupling-board, the cables are not soldered but can be easily moved. There are two tasks that the directed control signals must accomplish. The first is t synchronize the A/D- and D/A-cards. The second is to direct a signal to the L/M-units relays.

To synchronize the A/D- and D/A-cards they will all use the same external clock and external trigger. It is important that all A/D-cards are synchronized with each other and all D/A-cards are synchronised with each other. Only then will the samples from all channels be transmitted or received simultaneously. This is necessary to obtain a reliable channel estimation and to avoid transmitting delayed signals. One Arduino on one digital board will generate a clock pulse and output it on a pin. This is used as an external clock for all A/D- and D/A-cards. One A/D-card (A/D master) will use one of its digital output pins to generate an external trigger signal for the other cards.

The A/D master is also used to generate the 'Select' signal in the Digital Boards. This is implemented in the hardware by connecting a 'digital output' pin on this A/D-card to the 'Select' signals of each Digital Board.

All of the Arduino boards communicate using an I2C bus with dedicated pins. These pins are also connected using the cross-coupling board.

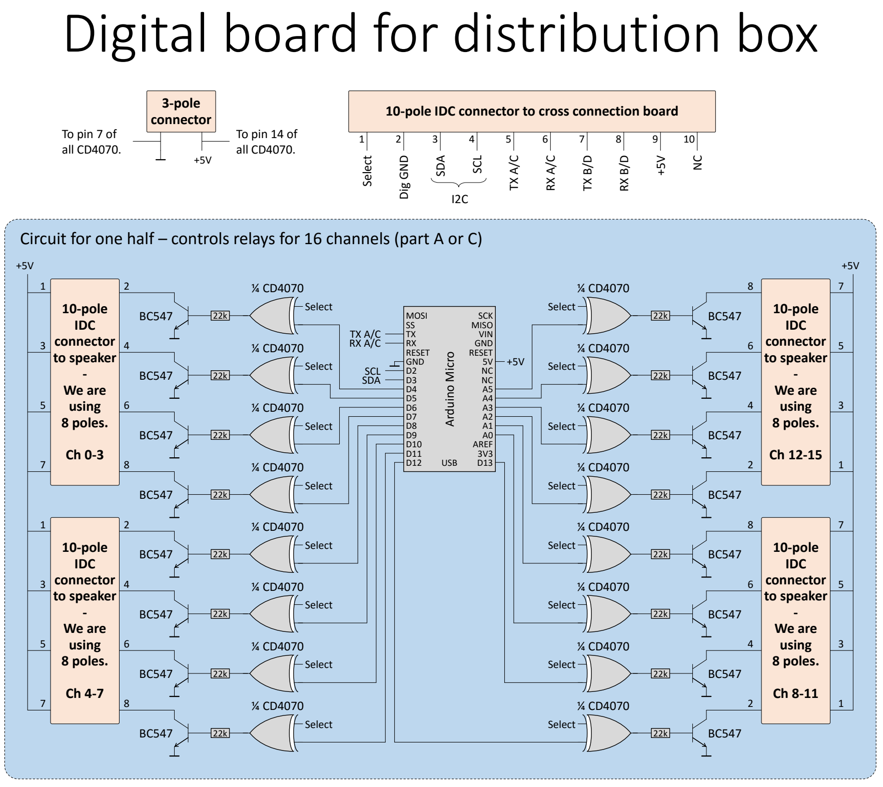
Digital board for distribution box
3-pole connector
To pin 7 of all CD4070.
To pin 14 of all CD4070.
+5V
10-pole IDC connector to cross connection board
Select
Dig GND
SDA
SCL
I2C
TX A/C
RX A/C
TX B/D
RX B/D
+5V
NC
Circuit for one half – controls relays for 16 channels (part A or C)
10-pole IDC connector to speaker - We are using 8 poles. Ch 0-3
10-pole IDC connector to speaker - We are using 8 poles. Ch 4-7
10-pole IDC connector to speaker - We are using 8 poles. Ch 12-15
10-pole IDC connector to speaker - We are using 8 poles. Ch 8-11
BC547
22k
¼ CD4070
Select
Arduino Micro
TX A/C
RX A/C
SCL
SDA
USB

Figure 5: Digital Board

3.1.5 Digital Boards

A schematic of half a digital board is illustrated in Figure 5. In total, there are two digital boards. Each digital board contains two Arduinos that control the relay of 16 channels each. One board thus controls 32 channels. Channel 1-16 is called A, channel 17-32 is called B, channel 33-48 is called C and channel 49-64 is called D. The first digital board contains A and B while the second digital board contains C and D. Figure 5 only shows part A (or C) but the other half including part B (or D) is almost an identical copy. The purpose of these boards is to output digital signals that are connected to the relays of the L/M units and switch the behaviour of the L/M unit between loudspeaker and microphone. The relay signal to the L/M unit is produced by an XOR between the 'Select' signal and a signal describing which group the L/M unit belongs to, in this section referred to as 'group' signal.

The group signal is unique for each L/M unit and divides the units into two groups: array units and terminals respectively. The group signals come from the Arduino boards, which in turn get this information from the user application. The user application in the computer sends the layout information via a USB-cable to the the master Arduino board, which distributes it to the slave Arduinos using I2C.

3.2 Signal Splitters

To lessen the number of cables, boxes called signal splitters are used between the distribution box and the L/M units. Each signal splitter distributes audio signals to and from four L/M units. An overview of the signal splitter can be seen in Figure 6, where DB25 and DB9 are specific types of the D-subminiature which is a common type of connector.

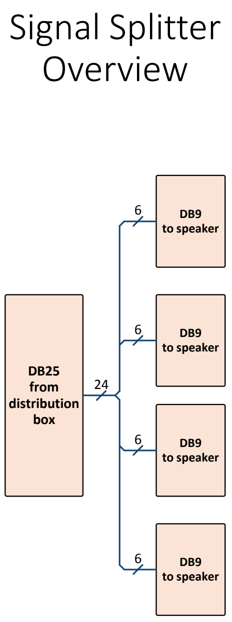

Figure 6: Signal splitter

3.3 L/M Units

Each L/M unit has an electronics board that handles power and microphone amplification of signals, as well as switching between transmit and receive mode. They have two external LED's, one that indicates if there is power and one that indicates if the unit is used as a transmitter or receiver. Table 2 describes the in and output signals from the L/M unit and the unit itself can be seen in Figure 7. The text "OBS!" in Figure 7 refers to the adjustments done to the printed board, illustrated with black lines.

Table 2: Input and Output pins on the L/M unit

Pin	**Description**
+12 V, 0V and -12 V	Power supply
Rel-, Rel+	Chooses whether the L/M unit transmits or receives sound
In- and In+	Input to the L/M unit when used as a loudspeaker
Out- and Out+	Output from the L/M unit when used as a microphone

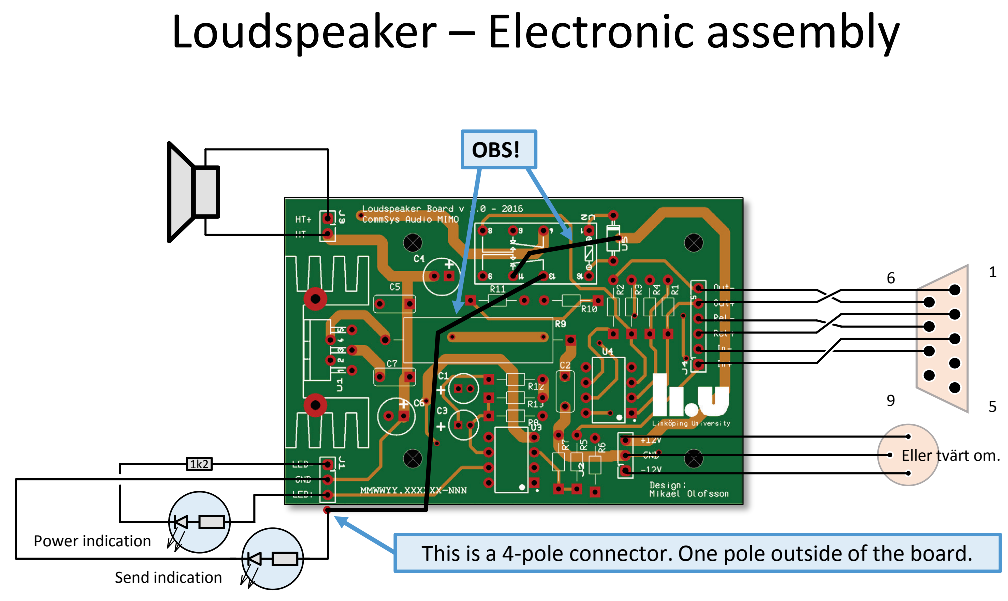
Loudspeaker – Electronic assembly
OBS!
1k2
Power indication
Send indication
This is a 4-pole connector. One pole outside of the board.
6
1
9
5
Eller tvärt om.

Figure 7: Loudspeaker

3.4 Power Supply

All L/M units are powered by the same power supply unit, which is assembled by the sponsor. The power supply can be seen in Figure 3 as a part of the central unit. The power supply has four sockets, each consisting of three banana connectors: -12 V, 0 V and 12 V. These potentials are then distributed to the L/M units by a series of power splitters.

4 Software

The software for the system will be used to control all speakers. It consists of two parts: API and Application. The API is the interface to the hardware and provides functions to use in the Application. The Application is the user interface and the signal processing needed to perform beamforming. The API is intended to be easy to use so that different applications (for example from other projects) can use the same API but perform different signal processing. This will make it easy for later projects to implement different signal processing projects on the MAB. An overview of the software, and what interfaces it has to the hardware is presented in Figure 8.

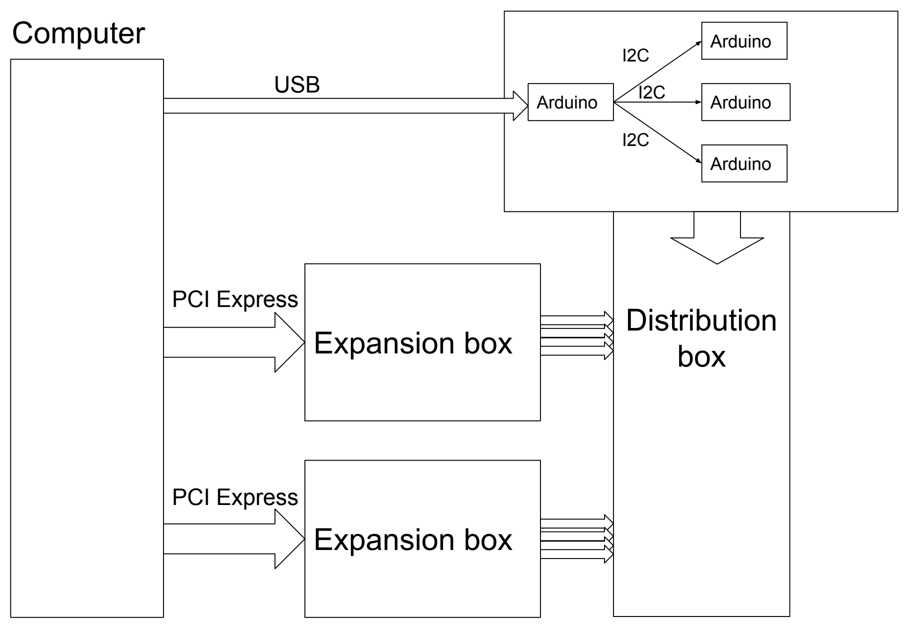

Figure 8: An overview of the tasks of the software

The following chapters will discuss the planned design for the Application and the API.

5 API

The purpose of the API is to provide three simple functions to the Application: initialization, transmission of a signal and reception of a signal. The initialization will select L/M units to use in array and terminal as well as setup the A/D- and D/A-converters. This setup consists of setting sampling frequency, sampling clock and triggers. In addition to this, the Application should be able to specify a signal to transmit at an array or a terminal. The recorded signals should then be returned to the Application. These functions are summarized in Table 3.

Table 3: API Functions

Function	Description	Input	Output
`init_boards`	Initialize the A/D- and D/A-boards	-	-
`set_array`	Sets the specified L/M units to group array	L/M units used for array	-
`set_terminal`	Sets the specified L/M units to group terminal	L/M units used for terminal	-
`transmit_array`	Transmits the specified signal on configured array	signal to transmit	received signals from terminal
`transmit_terminal`	Transmits the specified signal on configured terminal	signal to transmit	received signals from array
`record_array`	Record signals at array for a given duration	duration of recording	received signals
`record_terminal`	Record signals at terminal for a given duration	duration of recording	received signals

5.1 Implementation

To perform initialization, transmission of a signal and reception of a signal, the API must communicate with the hardware. This will be done using three different interfaces which can be seen in Figure 8. The Universal Serial Bus (USB) interface is used to instruct the master Arduino of which L/M units that will be used as terminals. The indexes of the L/M-units are then directed to the affected Slave Arduinos via Inter-Integrated Circuit (I2C). Peripheral Component Interconnect (PCI) Express is the interface to the expansion boxes that contain the A/D- and D/A-boards. The following subsections will describe how each interface will be implemented: MATLAB to Master Arduino, Master Arduino to Slave Arduinos and MATLAB to L/M units.

5.1.1 Assignment of Arrays and Terminals

The Arduino can assign an L/M unit to either terminal or array. However if the user does not wish to use all available L/M units there must be a way to handle unused L/M-units. The Application will keep track of if an L/M unit is a terminal, array or not used. If it is not used no signals will be sent from the Application to it. Therefore the hardware only keeps track of which L/M units that are terminals — the others are assumed to be array.

5.1.2 From MATLAB via USB to Master Arduino

As noted in the previous section, the Arduinos only need to know which L/M units to use as terminals. This information is transmitted through serial communication via USB to the master Arduino [2]. Table 4 shows what the code in MATLAB and the code run on the master Arduino will do.

Table 4: Description of code in MATLAB and on Master Arduino

Function	**MATLAB**	**Master Arduino**
`set_terminal`	Transmit number of terminals to use and then transmit the index of the chosen terminals	Receive indexes and send commands to affected slaves

5.1.3 From Master Arduino via I2C to Arduino Slaves

The master Arduino is connected via I2C to the slaves. Indexes for chosen terminals are here transmitted from the master to the slaves. Table 5 shows what the code on the master Arduino and the code run on the slave Arduinos will do. Arduino supports serial communication and has a specific library for I2C communication which will be used for this interface [3].

Table 5: Description of code on Master Arduino and on Slave Arduinos

Function	**Master Arduino**	**Slave Arduino**
`set_terminal`	Transmit index to affected slave	Receive indexes and set corresponding output-port

5.1.4 From MATLAB to L/M units

This interface must perform three tasks: set parameters for the A/D- and D/A-boards, transmit and receive data, and control the select signal that decides if the array or terminal should transmit. To perform these tasks, the API must control the A/D- and D/A-boards. These boards are provided with an API, called API-AIO(WDM) that can be used to control them. The distributor of the boards, Contec, also provides a Data Acquisition Library (DAL) called ML-DAQ. By installing this, one can directly use MATLAB:s Data Acquisition Toolbox to communicate with the boards. However, ML-DAQ only supports older MATLAB versions and Contec has no plans on updating it. Therefore, a new, simpler DAL will be written using the provided API-AIO(WDM). This will be used instead of the ML-DAQ. The chain of communication is illustrated in Figure 9. To the left, the figure shows what the communication looks like using ML-DAQ and the Data Acquisition Toolbox. To the right, the figure shows how the implementation in this project will work.

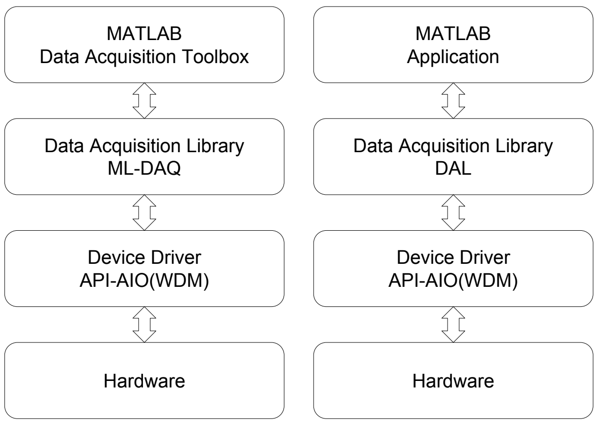

Figure 9: Communication chain between boards and MATLAB

5.2 Data Acquisition Library

The DAL will be written in C++ in Visual Studio. It will be a set of functions to be run from MATLAB. The interaction between the Application and the DAL is illustrated in Table 6 and 7. Table 7 also describes what functions the DAL will provide. Note that the input bit for selection (0 or 1) is reversed for the record functions. This is since they will record the signal, not transmit it at the specified group (array or terminal).

Table 6: API calls to DAL

Function	**DAL function**	**Input**
`init_boards`	init	-
`transmit_array`	`transmit`	`array`, 1
`transmit_terminal`	`transmit`	`array`, 0
`record_array`	`record`	0
`record_terminal`	`record`	1

Table 7: DAL Functions

Function	**Description**	**Input**	**Output**
`init`	Initializes the boards	`int`, `char`	-
`transmit`	Start D/A conversion on the input array and A/D conversion on received signals, set select to the input int	`array`, `int`	`array`
`record`	Start A/D conversion on received signals, set select to the input int	`int`	`array`

5.2.1 DAL Function "`init`"

The purpose of the initialization function is to initialize the settings of all boards. The settings are: sampling frequency, memory type, clock and triggers. All A/D- and D/A-boards will use an external clock and external trigger. One of the digital outputs on one of the A/D-boards will be used as the external trigger. The sampling frequency is set to the int given as an input.

5.2.2 DAL Function "`transmit`"

The transmit function starts D/A- and A/D-conversion. It sets Select by setting one of the digital outputs on the A/D-board to the input int. This function both transmits and records data.

5.2.3 DAL Function "`record`"

The record function starts A/D conversion. It sets Select by setting one of the digital outputs on the A/D-board to the input int. Note that this function does not start D/A conversion since it only records signals.

5.2.4 Running DAL from MATLAB

To be able to run the functions implemented in C++, MEX-files will be created. This allows the Application to use the C++ functions from MATLAB as if they were built-in functions. Practically, the C++ program is modified to include a wrapper function that handles the input and output to MATLAB. Thereafter, a binary MEX-file is created. The function can then be run from MATLAB, which invokes the MEX-file. [4]

5.2.5 Risks

The original idea was to use ML-DAQ and the Data Acquisition Toolbox. However, forcing the MAB to use an older version of MATLAB would put limitations on later projects. The MAB would also be an inflexible system. Therefore the idea is to write a new DAL which will take more time than learning to use the existing ML-DAQ. A simple DAL will be implemented to start with to see if it is feasible to do. Thereafter, the amount of time put into implementing it can be reviewed. If the implementation takes more than 20 hours, the ML-DAQ will be used instead, together with the Data Acquisition Toolbox, as depicted in the left part of Figure 9.

6 Application

The Application (APP) is the software system built on top of the API which consists of two parts, the first part performs some desired audio beamforming function and the second part provides a convenient user interface. The MAB system has a great potential for a wide variety of beamforming APPs. Most of these possible APPs will consist of some basic signal processing and beamforming subroutines that are similar. For this reason, some basic MIMO theory will be introduced in the following section. The theory section is followed by a section describing the actual APP of this project: directive sound transmission.

6.1 Background theory

In a MIMO system, there is one set of input elements and one set of output elements with a channel describing how the output signals depend on the input signals. In a Massive MIMO system, at least one of the sets contains a 'massive' amount of elements, which often means at least a few tens of elements. In proposed Massive MIMO mobile communication systems, the set containing the most elements is a base station and the other set consists of mobile units. In the APP of this project, these sets corresponds to the (base station) array and the terminals respectively. Figure 10 illustrates how the system could look with 62 array units and 2 terminal units. The array geometry will affect the system performance but that is outside the scope of this project. For simplicity, a rectangular array will be assumed, with dimensions adjusted depending on where the system will be set up. Other array configuration might be considered in future applications.

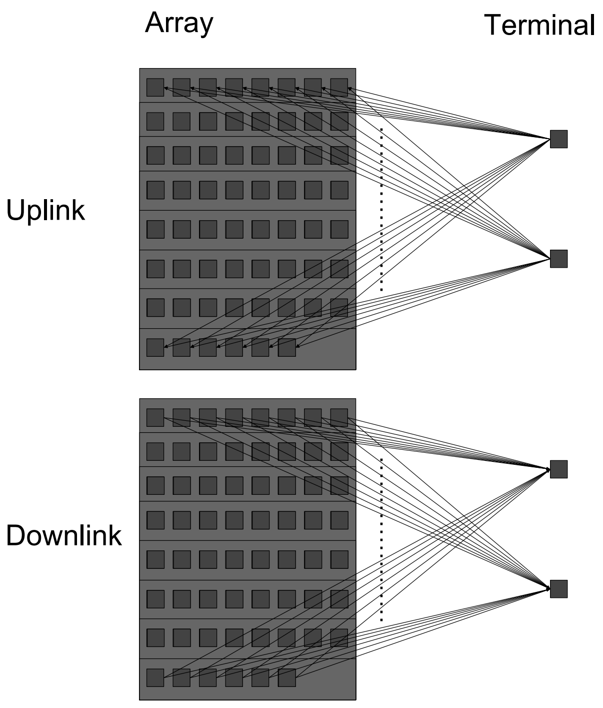

Figure 10: An overview of a system performing beamforming. Array geometry may vary.

6.1.1 Mathematical notation

Here follows a definition of the mathematical notation used in this document when describing MIMO theory. Let M be the total number of antennas in the array and K be the total number of terminals.

Let

$$\mathbf{G} = \begin{pmatrix} g_1^1 & \dots & g_1^K \\ \vdots & \ddots & \vdots \\ g_M^1 & \dots & g_M^K \end{pmatrix} \tag{1}$$

be the channel matrix where g_k^m is the channel impulse response between terminal k and antenna m. If denoted with a hat, $\hat{\mathbf{G}}$, it represents the estimated channel matrix.

Let

$$\phi_i = \begin{pmatrix} \phi_i^1 & \dots & \phi_i^N \end{pmatrix} \tag{2}$$

be the pilot signal for terminal i. N is the length of the pilot signal which is used to estimate the channel.

Let

$$\mathbf{\Phi} = \begin{pmatrix} \phi_1 \\ \vdots \\ \phi_K \end{pmatrix} \tag{3}$$

be the pilot matrix.

Let the received matrix at the array in the training phase be:

$$\mathbf{Y} = (\mathbf{G\Phi})^{\mathbf{T}} + \mathbf{N} \tag{4}$$

where $\mathbf{N}$ is a noise matrix.

For the downlink, let

$$\mathbf{q} = \begin{pmatrix} q_1 & \dots & q_K \end{pmatrix} \tag{5}$$

be one sample of each desired signal at the K terminals.

Let

$$\mathbf{W} = \begin{pmatrix} w_1^1 & \dots & w_1^M \\ \vdots & \ddots & \vdots \\ w_K^1 & \dots & w_K^M \end{pmatrix} \tag{6}$$

be the precoder matrix.

This gives one sample of the signals to transmit as:

$$\mathbf{x} = \mathbf{qW} \tag{7}$$

$\mathbf{x}$ has dimension 1xM, each value x_i is a sample to be transmitted at antenna i.

6.1.2 Linear beamforming precoders

Massive MIMO systems work by using beamforming patterns that are different from phased arrays in that the phaseshift difference between two adjacent elements does not have to be constant throughout the array. The array units in turn do not have to be placed in a regular pattern. Instead the beamforming patterns take all of the channel matrix into account to produce more complex patterns that increase spatial multiplexing gains.

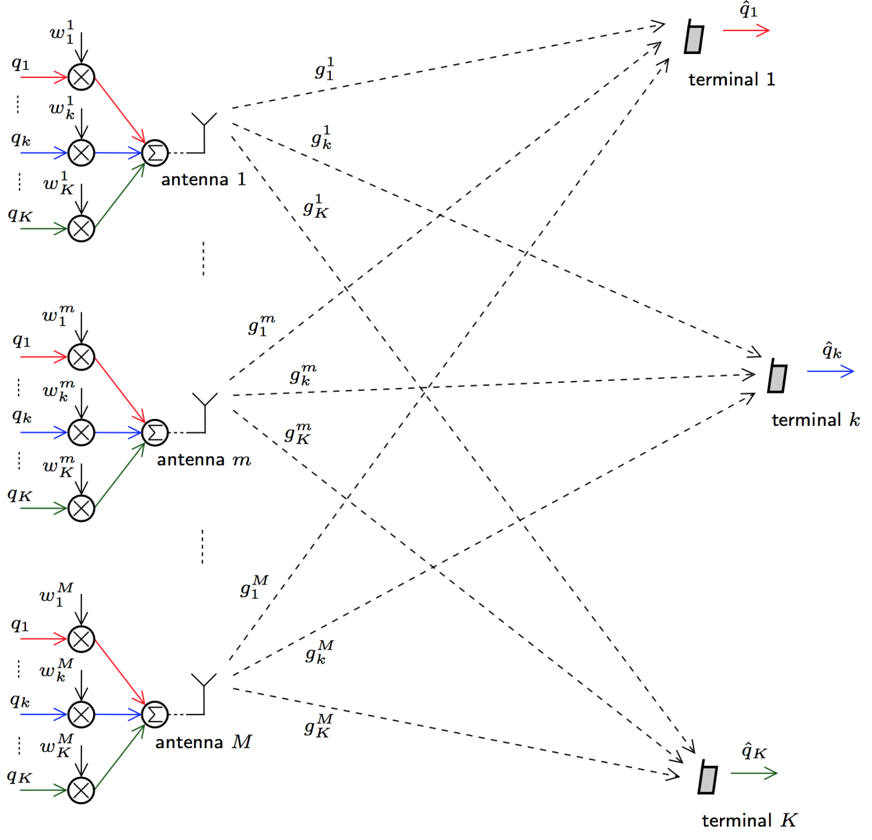

Figure 11: Linear beamforming, [5]

Linear beamforming is done over flat-fading sub-bands, which leads to the possibility of describing the channel matrix with complex gains. An example is illustrated with complex channel gains g_k^m as in equation (1). Every vector $\mathbf{q}$, transmitted to the receiver is distributed to the transmitter units by multiplication with a precoding matrix $\mathbf{W}$, so that $\mathbf{x} = \mathbf{qW}$ is transmitted. The received vector $\hat{\mathbf{q}}$ will be

$$\hat{\mathbf{q}} = \mathbf{xG} = \mathbf{qWG} \tag{8}$$

as shown in Figure 11. There are different ways of choosing the precoder matrix. The most common are summarized in table 8.

Table 8: Precoders

Precoder	**W**	**Description**
Maximum ratio (MR)	$\mathbf{G^H}$	MR of signal energy to given terminal
Zero-forcing (ZF)	$\mathbf{G^H}(\mathbf{GG^H})^{-1}$	Completely inverting channel
Regularized ZF	$\mathbf{G^H}(\mathbf{GG^H} + \lambda\mathbf{I})^{-1}$	Compromise of the two above

6.1.3 Channel Estimation

The previous description of precoding requires complete knowledge of the channel matrix $\mathbf{H}$. Knowledge about the channel is acquired by measuring the channel, hence $\mathbf{H}$ can in principle never be known exactly. However, good techniques of channel estimation can produce accurate approximations of the channel matrix.

The channel is assumed to be a linear system. Hence the channel response can be found by transmitting predefined pilot sequences over the channel. The pilots should have their energy spread over all of the spectrum of interest in such a way that the SNR is relatively constant. The pilots should also be orthogonal. This provides for accurate channel estimation. If the pilots, $\mathbf{\Phi}$, are orthonormal then the channel can be estimated as

$$\hat{\mathbf{G}} = \mathbf{Y}^{\mathbf{T}}\mathbf{\Phi}^{\mathbf{H}} \approx \mathbf{G}\mathbf{\Phi}\mathbf{\Phi}^{\mathbf{H}} = \mathbf{G} \tag{9}$$

where $\mathbf{Y}$ is the received matrix. Pilots should be sent from the terminals to the arrays. This will make orthogonality easier to achieve since the terminals are fewer and thus require fewer pilots.

After estimation, the channel frequency response is quantized and subdivided into flat-fading sub-bands, each having a complex channel matrix and thus a precoding matrix.

6.1.4 Equalization

Equalization is methods of dealing with frequency-selective fading or frequency-selective transmission. This is automatically dealt with if unnormalized ZF precoding (see Table 8) is used. Since different APPs have different requirements, they might use different equalization schemes. In the APP of this project, the equalization has to be done at transmitter side since directed sound transmission should work without a terminal units listening.

6.2 Directive Sound Transmission

The aim of the Application is to play several different sounds to several different terminals, or people, preferably without mutual overhearing. These sounds could be anything from narrowband sounds to music. One way to do this would be, to use knowledge of the geometry of the array. However, to demonstrate massive MIMO, the Application will estimate the channel using a pilot and then use this estimate to beamform the signal. The subsystems required to accomplish this is shown in Figure 12. The sound passes through the precoder and then gets played at the array L/M units. The precoder is determined by estimating the channel based on recorded pilot signals. The subsystems are further described in the following sections.

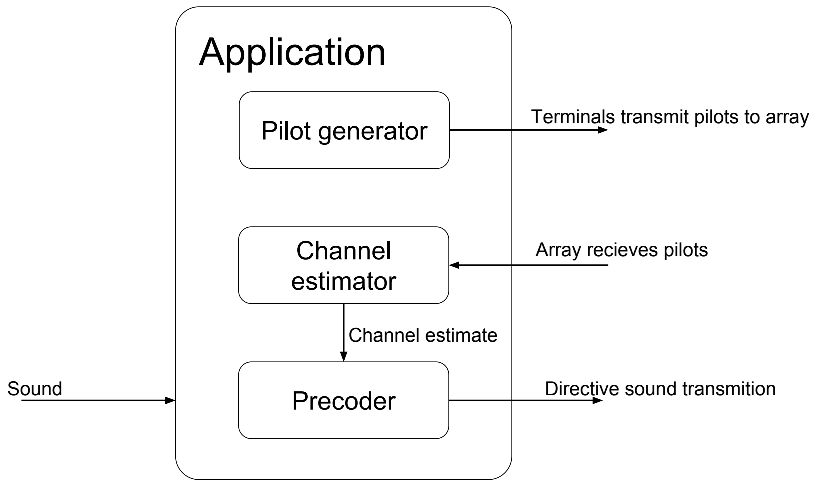

Figure 12: An overview of the Application

6.2.1 Pilot generator

The choice of pilots depend on the channel characteristics. Most important is that they are orthogonal between terminals and that they provide enough energy in the transmission band. One simple choice would be to sum a subset of the Fourier series components corresponding to the pilot duration T:

$$\phi_i(n) = \sum_{i \in A_i} \sin(2\pi(n/N)(i/T)) \tag{10}$$

The subsets $A_i \subset \mathbb{N}$ are disjoint so that the pilots ϕ_i are orthogonal.

For fading channels, the best way is to alternate consecutive frequencies between terminals, in order to properly cover all of the band for all terminals. This is illustrated in Figure 13. Using coherence bandwidth B_c, a $p \in \mathbb{N}$ should be chosen small enough so that the frequency spacing Δf satisfies

$$\Delta f := p/T < B_c. \tag{11}$$

Notice that p is the rate of which pilot frequencies are sampled from the complete set of orthogonal frequencies. Otherwise, there will be substantial fading that is not taken into account.

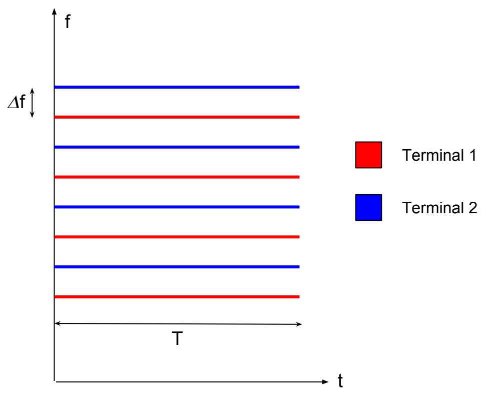

Figure 13: Frequency distribution of pilots

Pilots have to be transmitted every time the channel is substantially different. For the APP of this project, the hope is that the channel will be approximately static. The idea is that the movement of people has a small impact on propagation, and also that the movement of people will be as small as possible during operation. If the channel is static, only one pilot sequence is needed before the sound transmission. If experiments show that the channel unavoidably will change in time, then a new pilot sequence has to be transmitted periodically. See Figure 14.

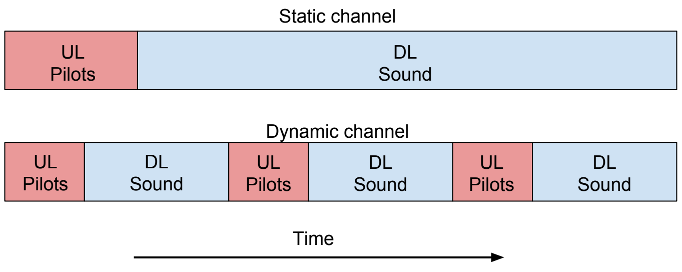

Figure 14: Time plot of pilots and sound transmissions

The pilot generator in the Application will have one function that is described in Table 9.

Table 9: Pilot functions

Function	**Description**	**Input**	**Output**
`generate_pilot`	Generates pilots	M	$\mathbf{\Phi}$

6.2.2 Channel estimator

As described in Equation (9), the channel estimation will be straightforward if pilots are orthogonal. The channel estimator part of the Application will have one function that is described in Table 10.

Table 10: Channel estimator functions

Function	Description	Input	Output
`estimate_channel`	Estimates the channel	$\mathbf{Y}, \mathbf{\Phi}$	$\hat{\mathbf{G}}$

6.2.3 Precoder

To avoid overhearing, the ZF-precoder should be used. However, complete zero-forcing is only possible with perfect channel state information. If highly accurate channel estimation is hard to obtain or if the time coherence due to moving people is far too small, then ZF-precoder might not be the best choice for our APP. Therefore, ZF-precoding will be implemented and tested. Should the scenario prove to be too demanding for ZF-precoding, a new solution will be implemented using MR-precoding. In this case, the precoder of each subband needs to be scaled to account for frequency-selective fading. The precoder part of the Application will have two functions described in Table 11.

Table 11: Precoder functions

Function	Description	Input	Output
`generate_precoder`	Generates precoder, see Table 8	$\hat{\mathbf{G}}$, mode	$\mathbf{W}$
`generate_signal`	Generates signal to transmit, see (7)	$\mathbf{q}, \mathbf{W}$	$\mathbf{x}$

6.2.4 Main program

Having described the Application and the API, the defined functions can be used to describe the main program.

```
% init
    M = 2;
    init_boards();
    set_array([3:64]);
    set_terminal([1, 2]);
% main
    pilot = generate_pilot(M);
    response = transmit_terminal(pilot);
    estimate = estimate_channel(response, pilot);
    precoder = generate_precoder(estimate, 'ZF');
    transmit_array(generate_signal(signal, precoder));
```

6.3 User Interface

The User Interface (UI) is a graphical user interface to make the system more user friendly. In the UI the user can choose what L/M units that will be used for transmitting or receiving sound. There is also a possibility to choose what sounds or music to be transmitted to each of the terminals. The UI also has an option if the user wants to save the recorded sound in the terminals. The graphical user interface is illustrated in Figure 15.

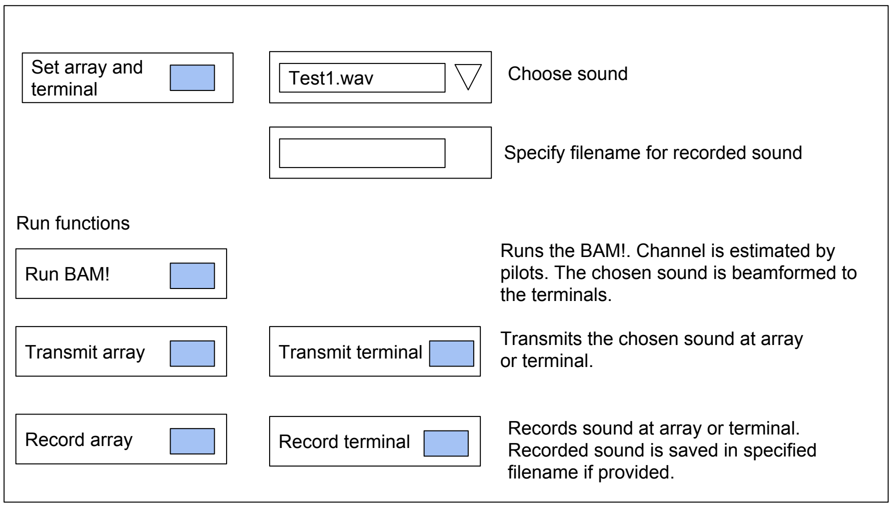
Set array and terminal
Test1.wav
Choose sound
Specify filename for recorded sound
Run functions
Run BAM!
Runs the BAM!. Channel is estimated by pilots. The chosen sound is beamformed to the terminals.
Transmit array
Transmit terminal
Transmits the chosen sound at array or terminal.
Record array
Record terminal
Records sound at array or terminal. Recorded sound is saved in specified filename if provided.

Figure 15: Overview of the graphical user interface

References

[1] T.Svensson and C.Krysander, in *LIPS*, Version 1.0, Compendium, LiTH, 2002.

[2] Mathworks, in *fscanf (serial)* [Online]. Available: https://se.mathworks.com/help/matlab/ref/serial.fscanf.html

[3] Arduino, in *Wire Library* [Online]. Available: https://www.arduino.cc/en/Reference/Wire

[4] Mathworks, in *Create C Source MEX File* [Online]. Available: https://se.mathworks.com/help/matlab/matlab_external/standalone-example.html

[5] Thomas L. Marzetta, Erik G. Larsson, Hong Yang, Hien Quoc Ngo, in *Fundamentals of Massive MIMO*, 1st ed. Cambridge University Press, 2016