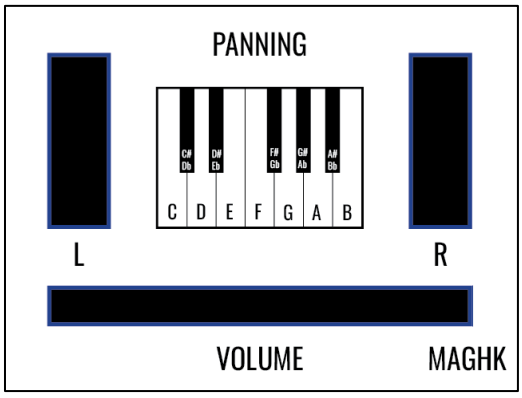
1



MIDI-Player

PROJECT REPORT

System Design, TSIU03

Version 1.1

[1]

MIDI-Player

Group 02, HT-23

Faculty of Science and Engineering at Linköping University

Name	Role	Email
Aidin Jamshidi	Developer	Aidja644@student.liu.se
Ghady Al Hadad	Developer	ghaal763@student.liu.se
Hannes Fröberg	Developer	hanfr262@student.liu.se
Martin Castro Bildhjerd	Developer	Marca851@student.liu.se
Kebba Jeng	Developer, Project manager	Kebje541@student.liu.se

Website: https://gitlab.liu.se/maghk/tsiu03

Client: Kent Palmkvist, 581 00, Linköping, kent.palmkvist@liu.se

Course responsible: Kent Palmkvist, 581 00, Linköping, kent.palmkvist@liu.se

Supervisor: Mikael Henriksson

Contents

1. Introduction

As part of the course System Design, TSIU03 at Linköping University, a MIDI-player was designed and built. The project aimed to create a complete functional system on the Altera DE2-115 FPGA board. The task consisted of using the DE2-115 board to modify/analyze a stereo sound signal. The input sound signal was to be converted into digital form and modified on the FPGA. The output sound was then to be sent to the loudspeakers. Relevant information was to be displayed graphically on a screen through VGA attached to the board.

A MIDI-player is a device used to send MIDI signals, a standardized communication protocol for communication between electronic musical devices. In this project the MIDI-player can produce notes within one chromatic scale based on keypresses on a PS/2 keyboard connected. The board also allows for passthrough sound from an external sound source with 3.5mm input. The notes are generated within the FPGA and then added to the signal from the external source before outputting both in stereo. Before the output, the sound and balance levels are adjustable from the keyboard. The settings for sound level and balance are shown on the VGA supported display as columns, depicting right and left volume and the played notes will be depicted by their corresponding note letters on the display as well.

The project is divided in three phases:

- Planning phase: Defining and planning the project designing the system.
- Implementation phase: Coding according to design specifications.
- Round of phase: Verify the project and present the result.

2. Design

This chapter gives an overview of the system and its various modules. Detailed description can be found in the Design Specification. The system consists of five major modules – Decode, Settings, VGA, and Audio Settings.

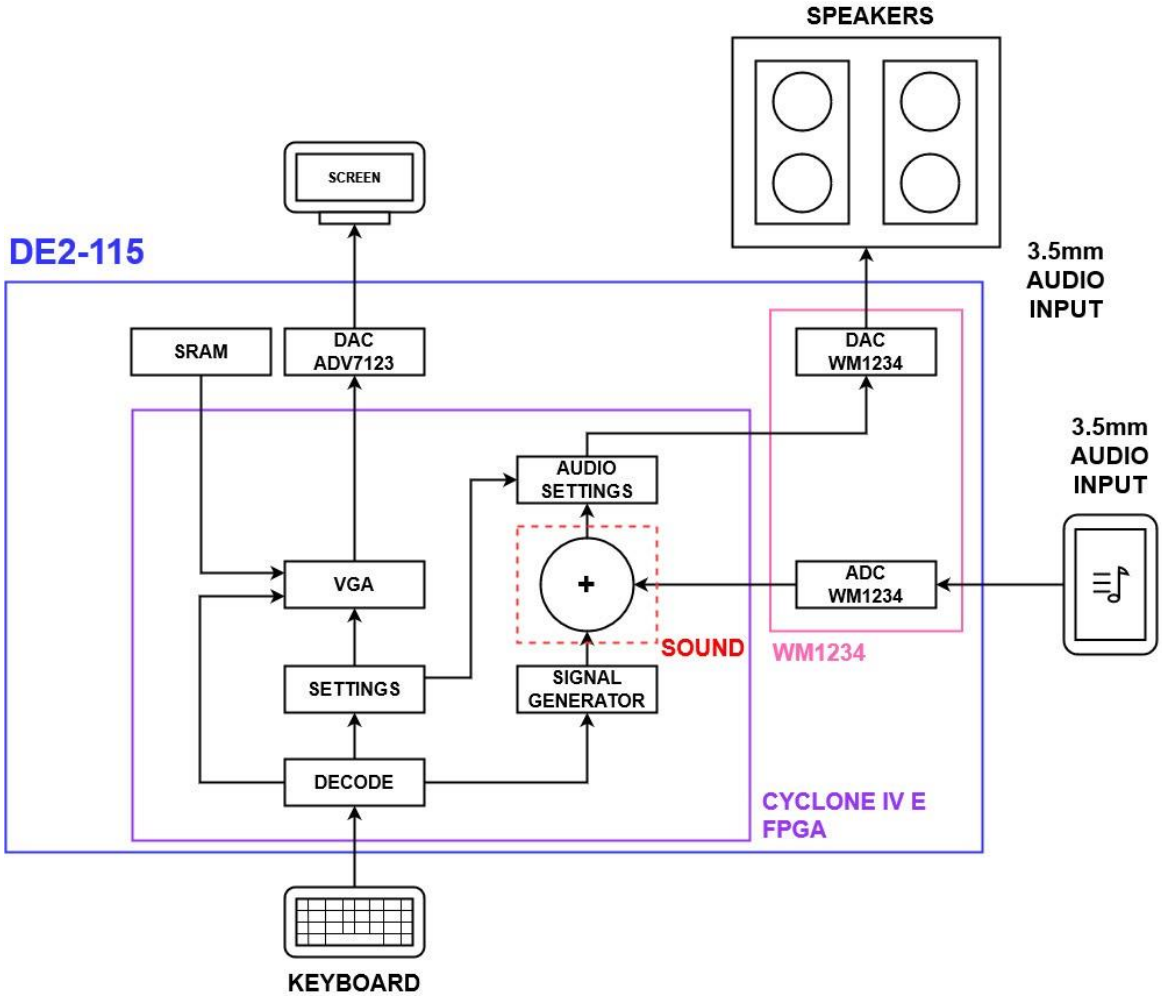


Figure 1: System overview

3. Communication

Under communications the audiovisual communication from the FPGA to the user is described. This is separated into audio and image.

3.1 Audio

As the keyboard is pressed, a keystroke is sent with information to the signal generator. This information is processed and turned into a playable signal. The possibility to instead use a 3.5 mm audio jack to feed a signal through from a mobile phone is available. After the signal is fed or is generated, it goes through some settings that give the possibility to pan the signal L/R. When the settings are chosen, the audio is fed to two speakers to output the sound.

3.2 Image

The image was delivered through VGA with a 640x480 screen. This is done through the FPGA. As soon as a key is pressed, it is sent to the FPGA and is displayed on the VGA screen as a note. There is implementation on the FPGA that continuously shows the desired pictures stored in the SRAM on the VGA- screen. This is done by looking at the settings, generating address and control signals to the SRAM, reading, and decoding the pixel data from the SRAM, generating the control signals for the VGA and video DAC (Digital to Analogue Converter). And finally sending the pixel data to the video DAC. All this is done through pipelining registers to ensure that the timing constraints are met.
Throughout the process, consistent communication is maintained with the 640x480 screen via the FPGA. Whenever a key is pressed, the corresponding note is displayed on the VGA screen, facilitated by the FPGA. The FPGA continuously displays the desired images stored in the SRAM on the VGA screen.
To facilitate communication with the screen, we review the settings, generate addresses, and control signals for the SRAM, read and decode the pixel data from the SRAM, and generate the control signals for the VGA and video DAC. Finally, we send the pixel data to the video DAC.
To ensure smooth operation throughout the project, all these processes are conducted through pipelining registers. This helps to maintain proper synchronization with the clock.

4. Module description

Decode- module:

The decoder will receive data from the PS/2 keyboard in series as make and break codes. Within the decoder those codes will be processed, and the necessary bits will be forwarded to the settings block, VGA block and signal generator blocks through a 12-bit vector called MIDI_Notes.

The decoder's subblocks simplify data handling and the transformation of signals. The scancode, representing the key pressed on the PS/2 keyboard, is received as the PS2_dat signal in a series. This scancode is then shifted into a parallel format within the decode block. The scancode is sent at keypress and repeats at a rate between 2 Hz and 30 Hz, and the break code is transmitted as soon as the key is released. The repeat rate is determined by a command sent to the keyboard, so this can vary as the product developed within this project does not send any commands to the keyboard. Once a whole make or break code is read it is put into a signal called Scancode, which then is read in a case statement. Where the information within Scancode sets bits within MIDI_Notes.

These keypresses correspond to different notes, as outlined in the table below.

Key	Note	Scancode Make	Scancode Break
Z	C	1A	F01A
S	C#	1B	F01B
X	D	22	F022
D	D#	23	F023
C	E	21	F021
V	F	2A	F02A
G	F#	34	F034
B	G	32	F032
H	G#	33	F033
N	A	31	F031
J	A#	3B	F03B
M	B	3A	F03A

The MIDI_Notes vector will then consist of all the possible notes and combinations of them, 12 bits in total, where each bit corresponds to one note (keypress). The value of that bit represents whether the note should be played or not and all twelve bits are sent to the VGA module and the Signal Generator.

Bit:	0	1	2	3	4	5	6	7	8	9	10	11
Value:	1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0
Note:	C	C#	D	D#	E	F	F#	G	G#	A	A#	B

The settings signal should contain information about volume and balance. For setting volume and balance the arrow keys on the keyboard are used. The make and break codes of those keys are read and put into Scancode. When any arrow keypress is read within scancode a case statement changes the Set signal. This signal is built in the same way as MIDI_Notes, but it consists of only four bits. Where each bits represent an action of raising the volume, lowering the volume or shifting the balance left or right.

Once those bits are set an if statements catches if any of those bits are now a one, and they were last set to zero. If so, the change is induced. This change is to the signals called Current_Volume and Current_Balance. Both those signals consist of 4 bits each, corresponding to 16 different values we can use for volume and balance.

In balance we decided to use just 15 of those values, to be able to have the average and median be the same.
We preferred the solution to just let whatever notes being pressed be played, up to all 12 at once, barring hardware limitations.

- Receives data from the PS/2 keyboard as make and break codes.
- Processes and forwards relevant bits to the Settings Block, VGA Block, and Signal Generator Block.
- Generates MIDI_Notes signal, which represents notes within the chromatic scale, with each bit indicating whether a note should be played.

Settings- module (Volume and balance):
The settings module is divided into two different submodules, balance, and volume. They work in a similar fashion, with the only difference being that the balance module differentiates between left and right channel. Both blocks receive the signal containing the digital sound as two signed 16-bit signals, LDAC and RDAC. Those channels are then multiplied with different factors depending on the settings information received from the decode block. The different factors are decided through a select statement, within which the factors to get a 2dB difference between the different volume levels are encoded. Since we multiply two 16-bit signed with each other, the result is put into a 32-bi- signed, and then truncated into a 16 bit again to fit the output.

The same is done in the balance control block, but using the lrsel signal and the balance setting to decide which output should be multiplied with what, or if it is just put through as is.

Signal generator- module:
The signal generator module receives a 12-bit vector from the decoder. The 12-bit vector corresponds to which keys of the keyboard are being pressed down. Then the signal generator will read the 12-bit vector and add the desired note frequency from our conversion table consisting of constants assigned frequency values for each note. It is important to note that the signal generator- module is designed to be able to receive a full 12-bit vector of all ones, resulting in generating a sound consisting of all frequency together with the phase to amplitude converter.

Code description of how to the signal generator operates includes:

- PSAC inputs/outputs: Several internal signals are used for the signal processing. Including x_"NOTE" and a_"NOTE" which represent phase accumulators and sine amplitudes for different tones.
- PSAC component: The code includes a component called PSAC (Phase to Sine Amplitude Converter) responsible for converting the phase information into sine amplitudes. It takes the notes as input and produces a_combined as output. Which are the notes combined.
- The signal output: The final left and right audio signals (LDAC and RDAC) are determined based on user settings, and the corresponding DACs are enabled (DAC_en).

Screen generation- module:

The screen generation block is responsible for managing the visual display on the connected VGA screen. The VGA screen has a resolution of 640x480 pixels and operates at a frame update frequency of 60Hz, resulting in 60 frames displayed per second.

It uses two critical synchronization signals:

- **hsync (horizontal synchronization)**: This signal is generated at the end of each line and ensures that the screen's display head moves to the next line.
- **vsync (vertical synchronization)**: This signal is generated at the end of the entire frame and is crucial for returning the display to the beginning of the screen.

Both hsync and vsync are active low, indicating that they go low to signify synchronization. Color information in the form of analog signals (R, G, B) is provided to a digital-to-analog converter (DAC) named ADV7123 to create the actual visual display. The VGA Block ensures that images are drawn from left to right, line by line, on the VGA screen, aligning with the screen's frame update frequency.

5. Challenges

Designing a system that involves interfacing with multiple hardware components is of course challenging. In this project the main challenge was understanding how code snippets from the laborations completed in the course could be used and modified in this system. Before this course, none of the project members had any prior VHDL coding experience, so learning a new language and developing a product was a challenge.

6. Experiences

Ghady: Personally, the most gratifying aspect of this endeavor was immersing myself in the problem-solving aspect of this project. However, there was an equal sense of satisfaction in the hands-on process of actualizing the midi keyboard. Our design involved a decent amount of planning and admittedly, we encountered a few issues that could not have been easily resolved without the help of our supervisor.

Hannes: For me the biggest takeaway from this project is the importance of forethought, or a well thought out design specification. Especially how much the hours put into the design and specification relieves the amount of work needed later in the project. As well as the ease of dividing the project into modules, where each member can get to work and know what is needed to combine the modules in the end. It makes for a more seamless work experience with much less hiccups along the way.

Kebba: I think that the most enjoyable part for me was learning VHDL, but another enjoyable part was designing and realizing digital circuits. Throughout my education I have had the pleasure of working on many projects and one of the things i enjoy the most is the process of creating something. Our design was very well thought out and while we of course stumbled upon some challenges (My_Fancy_Application took a while for me to understand...) The project went very smoothly. I do have to give special thanks to our supervisor who was of immense help throughout the project.

Aidin: The most enjoyable aspect of the project entailed acquiring proficiency in the VHDL programming language. Although distinct from prior programming languages I had encountered, it was promptly mastered. The process of crafting the VGA module and designing various animations proved to be highly engaging. Conversely, the amalgamation of diverse modules presented a greater challenge. Nevertheless, with dedicated and persistent efforts, the project was ultimately completed. I take satisfaction in the end product, as it is gratifying to interface with the MIDI keyboard and observe its functionality.

Martin: Throughout this project I have learned the importance of having an easy to read and well-structured code, to make sure other members of the group can read and understand the other group members' parts of the code well. Working on a larger project in a group it is necessary to divide the needed modules. And in the end put it together nicely, it helps a lot with professionally written code.

7. Improvements

If some students were to continue the product a suitable addition would be adding more keys to the product. From a software perspective some suitable improvements would be restructuring the “My_Fancy_Application“ code that is part of the signal generation module, splitting it into smaller pieces of code rather than a big block of code.
We designed the code so that every key can be pressed down and play a sound consisting of all notes together. The problem is that the keyboard has restrictions for how many keys can be pressed down at the same time. So, one improvement would be to firstly analyze the keyboard you are planning to use and figure out if there are any similar restrictions to the keyboard and plan around any restrictions. Even if our VHDL code works so all keys can be combined, it does not work in practice with the keyboard.

8. User Manual

8.1 Introduction

The MAGHK MIDI-Player is an easy-to-use musical instrument system! With it you can play your favorite tunes. This manual will help you get started so you can go ahead and jam those awesome songs!

8.2 Getting started

1. Open the application MAGHK MIDI on your Windows/MACOS device.
2. Connect a keyboard to the computer.
3. Rock on!

8.3 Playing music

Pressing Keys: Use the connected PS/2 keyboard to play music. Keys on the keyboard correspond to a musical note. As you press keys, you will hear the notes played through the connected speakers. You can press multiple keys at once to create harmonious melodies.

Keyboard button:	MIDI- note:
Z	C
S	C#
X	D
D	D#
C	E
V	F
G	F#
B	G
H	G#
N	A
J	A#
M	B

Table 1: Key-Note corresponding table

8.5 Adjusting sound

Volume: To adjust the volume, use the keyboard as follows:

- Press 'Up Arrow' key to increase the volume.
- Press 'Down Arrow' key to decrease the volume.

Balance: You can adjust the balance of sound between left and right channels:

- Press 'Left Arrow' key to shift the balance towards the left channel.
- Press 'Right Arrow' key to shift the balance towards the right channel

Keyboard button:	Setting:
Arrow up	Volume up
Arrow down	Volume down
Arrow left	Balance left
Arrow right	Balance right

Table 2: Key-Setting corresponding table

8.6 Visual display

The screen connected to the MAGHK MIDI-Player represents the system in an intuitive way.

As you press keys on the keyboard, corresponding musical notes will be displayed on the VGA screen. Enjoy watching your music come to life!

Manage your audio settings easily with panning and volume controls on the screen.

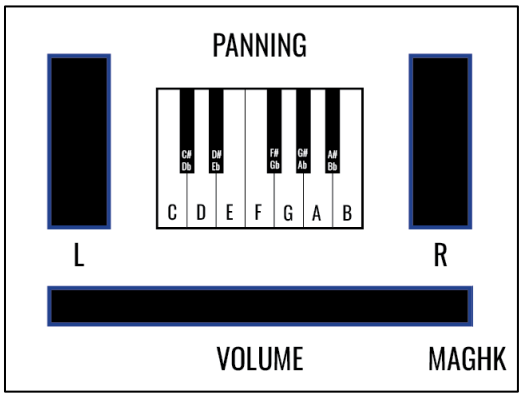


Figure 2: Visual display of system

9. Time summary report

The time was managed very well as can be seen the project was completed in far less time than anticipated. This is mostly thanks to the fact that the group was able to reuse modules from the laborations but also thanks to a well thought out circuit diagram and block descriptions.

Document	Expected workload (hours)	Actual Workload	Deadline
Requirement Specification v1.0	4h	3h	Week 37
Project Plan v0.1	10h	2h	Week 37
Design Specification v0.1	10h	4h	Week 37
Project Plan v1.0	4h	4h	Week 38
Design Specification v1.0	10h	5h	Week 38
Design Presentation	4h	2h	Week 39
Project Plan v2.0	4h	2h	Week 39
Design Specification v2.0	4h	2h	Week 39
Product development (VHDL)	100h	50h	Week 41
Project Report v0.1	10h	4h	Week 42
Project Report v1.0	10h	2h	Week 43
Final Presentation / demonstration	4h	2h	Week 43

Table 3: Expected workload vs actual workload

10. Project files

GitLab served as the platform of choice for our project due to its convenience in facilitating communication and collaboration among project members. The project repository was organized into two primary branches: "main" and "W.I.P" (Work in Progress). This division was implemented with the objective of segregating approved and finalized work within the "main" branch, while housing ongoing tasks and previous iterations within the "W.I.P" branch. The "main" branch featured a streamlined structure with fewer folders, promoting clarity and ease of navigation for end users. In contrast, exploring the "W.I.P" branch revealed a more extensive array of folders and a comprehensive history of various project components in earlier versions.