Course	Rev	Date	Students	Group	Page
TSUI03	2	2023-10-06	Ghady Al Haddad Aidin Jamshidi Hannes Fröberg Kebba Jeng Martin Castro Bildhjerd	2	1

Design Specification

Contents

1. Introduction

A MIDI-player is a device used to send MIDI signals, a standardized communication protocol for communication between electronic musical devices. In this project a MIDI-player named MAGHK shall be able to produce notes within one chromatic scale based on keypresses a PS/2 keyboard connected to it. The board will also allow for passthrough sound from an external sound source with 3.5mm input. The notes will be generated within the FPGA and then added to the signal from the external source before outputting both in stereo. Before the output, the sound level and balance shall be able to be adjusted from the keyboard.

The settings for sound level and balance will be 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.

2. Component list

This chapter specifies the components that are going to be used in the system as well as their purpose in the system.

No.	Component	Purpose
	VGA SCREEN	
	KEYBOARD	
	SPEAKER	

3. Block Diagram

The design begins in a top-down fashion, where the first objective is to create a block diagram which depicts how the different blocks of code shall be distributed.

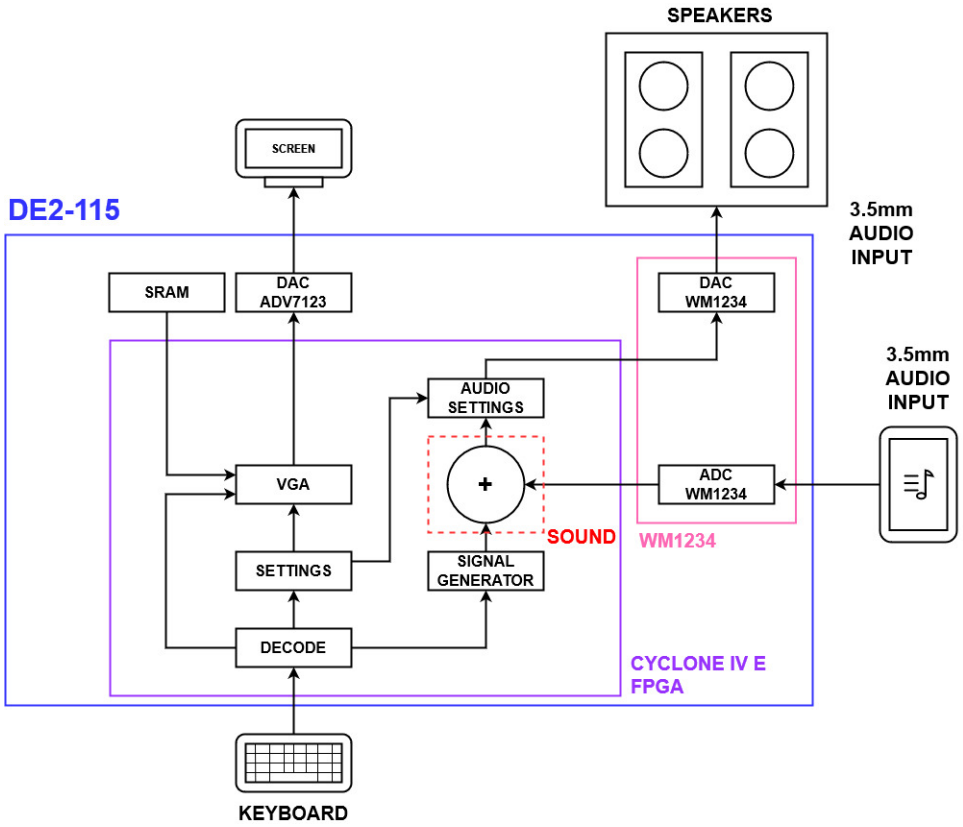

Figure 1 Overview of the system.

4. Communication

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

4.1. Audio

As the keyboard is pressed, a keystroke should be sent with information to the signal generator. This information will be 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 should exist besides the possibility to create the signal. After the signal is fed or is generated, it will go through some settings that will give the possibility to pan the signal L/R or just use mono sound. When the settings are chosen the audio will be fed to two speakers to output the sound.

4.2. Image

Throughout this project we will have steady communication with the 640x480 screen. This is done through the FPGA. As soon as a key is pressed, it gets sent to through the FPGA and is displayed on the VGA screen as a note. We will implement the FPGA that continuously shows the desired pictures stored in the SRAM on the VGA- screen. To do the communication to the screen we will look though the settings, generate address and control signals to the SRAM, read and decode the pixel data from the SRAM, generate the control signals for the VGA and video DAC. And finally send the pixel data to the video DAC. All this will be done through pipelining registers to ensure that the timing constraints are met.

Throughout this project, we will maintain consistent communication with the 640x480 screen via the FPGA. Whenever a key is pressed, the corresponding note will be displayed on the VGA screen, facilitated by the FPGA. We aim to implement an FPGA that continuously displays the desired images stored in the SRAM on the VGA screen.

To facilitate communication with the screen, we will review the settings, generate address, 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 will send the pixel data to the video DAC.

To ensure smooth operation throughout the project, all these processes will be conducted through pipelining registers. This will help maintain proper synchronization with the clock.

5. Blocks

In the following section the different blocks or modules of the block diagram will be delved into.

5.1. Decoder

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.

The MIDI_Notes signal must be able to contain 12 different values corresponding to notes within the chromatic scale, it may also need to contain 4 of those values at once, depending on how the signal generator block will handle multiple notes being played.

The VGA_Notes must transmit the same information to the VGA block to enable the visual representation of the notes on the screen. The settings signal should contain information about volume and balance changes, possibly requiring as few as 2 bits. For instance, ‘00’ could represent lower volume, ‘01’ could indicate balance to the right, ‘10’ could signify balance to the left, and ‘11’ could denote increased volume.

All these signals might need to be accompanied by an enable signal to inform the corresponding blocks that new information is available. Alternatively, this information could be included within the signal itself, and all information could be continuously relayed to the corresponding blocks.

Port	Type	Direction
Clk	Std_logic	In
Rstn	Std_logic	In
PS2_DAT	Std_logic	In
PS2_CLK	Std_logic	In
MIDI_Notes	Unsigned [11 downto 0]	Out
VGA_Notes	Same as MIDI_Notes, depending on number of notes painted on screen.	Out
Volume_level	STD_logic_vector(3 downto 0)	Out

Panning	STD_logic_vector(3 downto 0)	out

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 only once, and the break code is transmitted as soon as the key is released.

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 signal will then consist of all these notes, 12 bits in total, where each bit corresponds to one note. The value of that particular bit represents whether the note should be played or not.

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

The implementation is done with a case switch within decode, where the makecodes sets the appropriate bit to 1 and the breakcode sets it to 0. At most 4 of those notes shall be a 1 at the same time. So there needs to be a code structure that keeps track of how many bits are at 1, to not go over the limit. This will possibly be solved with a counter, to keep the signal out of the danger zone. We preferred the solution to just let whatever notes being pressed be played, up to all 12 at once, barring hardware limitations.

5.2. Settings

The settings block is designed to receive information from the Decode block regarding any user-requested changes in volume or balance. This information is then relayed to the Audio Settings, which has built-in volume control for both the left and right channels of the output. This is achieved through the DAC_Settings signal, which carries the register address (7 bits) and the data (9 bits).

The VGA block also needs to be aware of the current volume and balance settings. Therefore, a signal containing this information will be sent to the VGA block as well. This signal must include at least ten different levels for each channel, resulting in 100 possible combinations. These combinations need to be stored and transmitted to the VGA block. We have opted to use a 9-bit structure for this signal to maintain consistency with the data structure used by the Audio Settings.

Signal	Type	Direction
Clk	Std_logic	In
Rstn	Std_logic	In
Settings	Unsigned [1 downto 0]	In
Settings_EN	Std_logic	In
DAC_Settings	Unsigned [15 downto 0]	Out
VGA_Settings	Unsigned [8 down to 0]	Out

5.3. Audio Settings

5.4. VGA

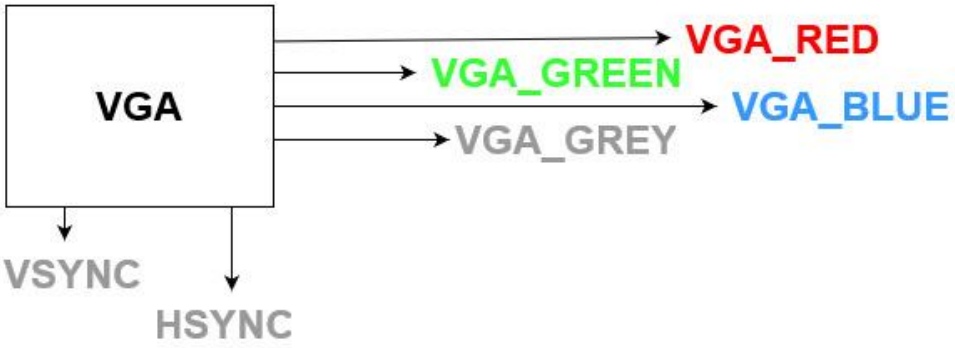

Figure 2

Signal	Type	Direction
Clk	Std_logic	In
Rstn	Std_logic	In
Vcnt	Std_logic	In
Vsync	Std_logic	Out
Hcnt	Unsigned (9 downto 0)	In
Hsync	Std_logic	Out
Pixcode	Unsigned (7 downto 0)	In
Ce	Std_logic	In
VGA_red	Unsigned (7 downto 0)	Out
VGA_green	Unsigned (7 downto 0)	Out
VGA_blue	Unsigned (7 downto 0)	Out
Vga_sync	Std_logic	Out
Blank	Std_logic	Out

Pixcode(7) = ‘0’ ger grayscale

Pixcode(7) = ‘1’ ger RGB

VGA_red <= Pixcode(6 downto 5)

VGA_green <= Pixcode(4 downto 2)

VGA_blue <= Pixcode(1 downto 0)

The screen that we have connected to the VGA port works with a resolution of 640 × 480 pixels, and a frame update frequency of 60 Hz, i.e., it displays 60 frames per second. The VGA port has two synchronization signals hsync and vsync, and three analogue colour signals (R, G, B). The color signals goes to a digital to analogue converter (DAC) named ADV7123. The image on the screen is drawn from left to right, line by line. hsync is the horizontal synchronization, which must be done after each line. The vsync is the vertical synchronization, which must be done after the entire image has been displayed in order to return to the beginning of the image. Hsync and vsync are active low.

5.5. Signal Generator

The signal generator will generate a corresponding waveform to the DAC converts the digital sine wave to an analog voltage signal. The analog voltage signal is sent to the audio output after that. The signal generation module then uses these MIDI note numbers to calculate the frequencies of the audio signals.

Forward sound: It passes the incoming sound directly to be added together with the 3.5mm.

Generate right: It generates and adds two sinusoids, 440 and 660 Hz, on the right channel when SW6 is ON. }

Generate left: It generates and adds two sinusoids, 440 and 550 Hz, on the left channel when SW7 is ON. }

MultiFunctionalAdaptor:

PORT EXPLANATION TABLE:

clk: The system clock signal.

rstn: The system reset signal.

SW: A switch input vector.

LEDR: An LED output vector.

lrsel: A signal that selects the left or right channel.

LADC: The left channel audio input signal.

RADC: The right channel audio input signal.

ADC_en: A signal that enables the audio inputs.

LDAC: The left channel audio output signal.

RDAC: The right channel audio output signal.

DAC_en: A signal that enables the audio outputs.

PORTS:

Clk, rstn	Std_logic	in
SW	Std_logic_vector (5to7)	in
LEDR	Std_logic_vector (to)	out
lrsel	Std_logic	in
LADC, RADC	Signed (15 downto 0)	out
ADC_en	Std_logic	in
DAC_en	Std_logic	out
MIDI_NOTES	Unsigned (15 downto 0)	in

INTERNAL SIGNALS:

ildac, irdac	Signed (15 downto 0)
x,xr, xl, xm	Unsigned (15 downto 0)
a,am,al,ar	Signed (15 downto 0)
sl, sr	Signed (15 downto 0)
noice	Std_logical_vector (15..0)
lack, rack	Unsigned (39..0)

CONSTANTS NOTE FREQUENCIES:

- 512 is the number of samples per cycle.
- 65536 is the maximum value of the phase accumulator.
- 50000000 is the clock frequency in Hertz.

A	Integer:=	295
A#	Integer:=	313
B	Integer:=	331

C	Integer:=	372
C#	Integer:=	394
D	Integer:=	417
D#	Integer:=	442
E	Integer:=	469
F	Integer:=	526
F#	Integer:=	557
G	Integer:=	591
G#	Integer:=	626

PFAC:

PORTS:

Clk	Std_logic	in
x	Unsigned (15..0)	in
a	Signed (15..0)	Out

6. Challenges

Designing a project that involves interfacing with multiple hardware components at the same time might be challenging; synchronization and timing could be among them. Coordinating input from the keyboard, generating the corresponding sound, and updating the display all in real-time can be tricky. We will need to carefully manage clocks and ensure that all the components are synchronized. VGA signal generation involves precise timing and control of the red, green, and blue color channels, as well as synchronization signals. Incorrect timing will likely lead to distorted or incorrect display output, especially since the display will need to be updated in real-time as buttons are pressed. Additionally, potential memory constraints may arise, requiring us to optimize data handling and storage to fit within the available memory space. Testing and debugging will probably be one of our biggest challenges, and having a structured approach that we can stick to is necessary to identify and fix issues.