ENSC 452
Final Project Individual Report

Piano Tiles - Video Game Hardware Accelerated Using Audio Signal Frequency Domain Analysis

Group 21
Justin Mateo
301393149
jmateo@sfu.ca

April 12, 2024

Table of Contents

List of Figures

1. Introduction

The purpose of this project was to design a game inspired by the Piano Tiles mobile application. A teaser of what the actual mobile application looks like is shown in Figure 1. This hardware version of the game seeks to replicate the engaging and rhythmic gameplay experience found in its software counterparts while leveraging the unique advantages of dedicated hardware. By creating a tangible, tactile interface for players to interact with, the project aims to enhance immersion and responsiveness compared to traditional software versions. The overarching goal is to demonstrate the functionalities:

1. Tile Generation Based on Song Played
2. Implement Synchronized Song Playback
3. Video Stream of Falling Tiles and Score Analyzer to process user click accuracy

Figure 1: Piano Tiles Mobile Application [1]

2. Review of Technical Literature Used

2.1 The Zynq Book Tutorials: For Zybo and Zedboard

This book really helped with the basics and using the borrowed hardware blocks such as the audio controller and VGA controller [1]. This book also went into working with interrupts which meant a lot. Examples of using both Vivado and Vitis in an actual working manner were displayed and were reassuring. This gave a clear idea of how to approach any type of project using these two softwares. This resource really helped with hardware/software integration, the tile generation system, interrupt handling, and the VGA/audio controller usage.

2.2 Fast Fourier Transform IP Product Guide by Xilinx

The documentation/guide about the FFT IP block provided by Xilinx really helped with understanding how the AXI protocol worked and what each of the inputs and outputs meant. Though this document is a lot of information in one location, if iterated through properly, a very good understanding of this hardware block can be established [2]. This resource really helped with the configuration and usage of the FFT block which was very unclear at first.

2.3 Using the AXI DMA in Vivado Web Tutorial

This web page really explained how the AXI DMA block worked. The idea of using a DMA block and interfacing through software was very unclear before reading through this. A step-by-step guide of setting up the hardware block and to then send data was shown in this web-page [3]. This resource really helped with, of course, the usage of the AXI DMA block and its connection with the FFT block, Zynq processing system and interrupts.

2.4 Others

Other resources, such as Piazza discussions with the course instructors, past forum posts on the Xilinx forums and Stack Overflow were very useful. Seeing what issues people are running into and if they are similar to me was very good to hear, as usually someone has walked them through the solution by then.

3. Personal Contributions

3.1 Project Partitioning

The partitioning of tasks for this project was mostly decided depending on the given milestones by the instructors. I worked on most of the hardware blocks and interfacing them with software and made sure that the appropriate functionality was working. Sina then worked more on the video side and some software, apart from the hardware button debouncer. This included loading and accessing audio tracks, creating menu screens, implementing tiles falling down the screen and computing score according to tiles on screen. Most of the functionalities above depend on the hardware and the interface with the hardware, such as hardware interrupts and handling for button presses and multi-core synchronization which I ensured worked. I also worked with sprite drawing which mostly was with number drawing.

3.2 Contributions

3.2.1 Tile Generation

This section includes three main components: the tile generator custom IP block, Xilinx's Fast Fourier Transform (FFT) IP block and Xilinx's AXI DMA block. The audio data had to have been transferred to the FFT IP block in a timely manner and in an AXI stream interface. Passing audio data by just simply writing to an address would not work as the hand-shaking protocol of AXI stream would not work properly due to timing. Configurations to the AXI DMA block were made to ensure that the data would be transferred using a "simple transfer" mode rather than "scatter gather" mode. This made it so that passing an array was very simple to execute. The data that is then streamed into the FFT block which is configured to be a 2048-point FFT. One caveat, however, is that the input and output of the FFT are 32-bits; the 16 most significant bits (MSB) being the imaginary part and the 16 least significant bits (LSB) being the real part. Since the audio data is all real, the audio data must be somehow compressed from 32-bits to 16-bits. This is just done by scaling using software and ensuring that the 16-MSB are all 0's. The FFT block will keep applying the 2048-point FFT on the input data until the AXI DMA block stops feeding data and a custom reset is applied. All this data from the FFT is being fed into the custom tile generator IP block which takes in and analyzes the first 1024 data points. This hardware block is custom-made to be compatible with AXI blocks such as the FFT block. This implies that the block has the capability to tell the FFT block when it is ready for more input data and know when the FFT block is ready with new data. Since this data is also composed of an imaginary and real part, the actual magnitude is found by computing the part of the Euclidean distance. For all iterations, the maximum magnitude and its corresponding bin is kept track of. When all 1024 computations are completed, the magnitude and bin are outputted which is accompanied by a done signal that can be used as an interrupt.

On the software-side of tile generation, the software contains an interrupt handler for the tile generator's done signal. Whenever the software is interrupted by this tile interrupt, a new tile is concatenated to the current tile array. This array of tiles includes information such as the magnitude and the corresponding bin. Further analysis on this array of tiles can be applied by comparing all magnitudes to a threshold; this is useful as we don't want all tiles being generated. These thresholds are computed real-time depending on the average of the magnitudes in the tile array.

3.2.2 Number Drawing

For this functionality, three things needed to be executed properly: sprite design, sprite storage, software for drawing. Sprite design included using a software called GIMP to easily control the pixel dimensions of the image. All digits from 0-9 were designed with white backgrounds and black font. These sprites then needed to be stored and accessed. The most simple way to do so was to store the hex-codes of each pixel in an array. A header file called numbers.h was used to easily access the pixels of the digits. To speed up this process, a Python script was written to write the header file. Finally some functions were written to simply take in an integer input which would then draw that integer on various areas on the screen. Different functions were used to draw the numbers in locations such as the current score in the game menu, high scores in the song menu, current and high score in the end game screen.

3.2.3 Interrupt Handling

Interrupts such as the button interrupts, tile generation interrupts and DMA interrupts. Learning to handle all types of interrupts connected to the interrupt request (IRQ) port of the Zynq Processing System was a task. The button interrupts were very similar to how it is handled in the Zynq Handbook [1]. However, since the interrupt had to be processed by the button debouncer unit, the block had to set attributes such that it was still an interrupt signal. The tile generator interrupt was a weird one to figure out as the actual instance of the tile generator was not defined like the GPIO, DMA or Timer blocks. For this, I realized that you can simply set the instance pointer as the interrupt system instance's pointer. Though to be able to do this, the interrupt ID had to have been inferred. It can be seen that the vector input into the IRQ is easily mapped to IDs starting at 61U, 62U and so on. Finally, the DMA interrupt was very similar to a basic GPIO interrupt set up.

3.2.4 Hardware Integration

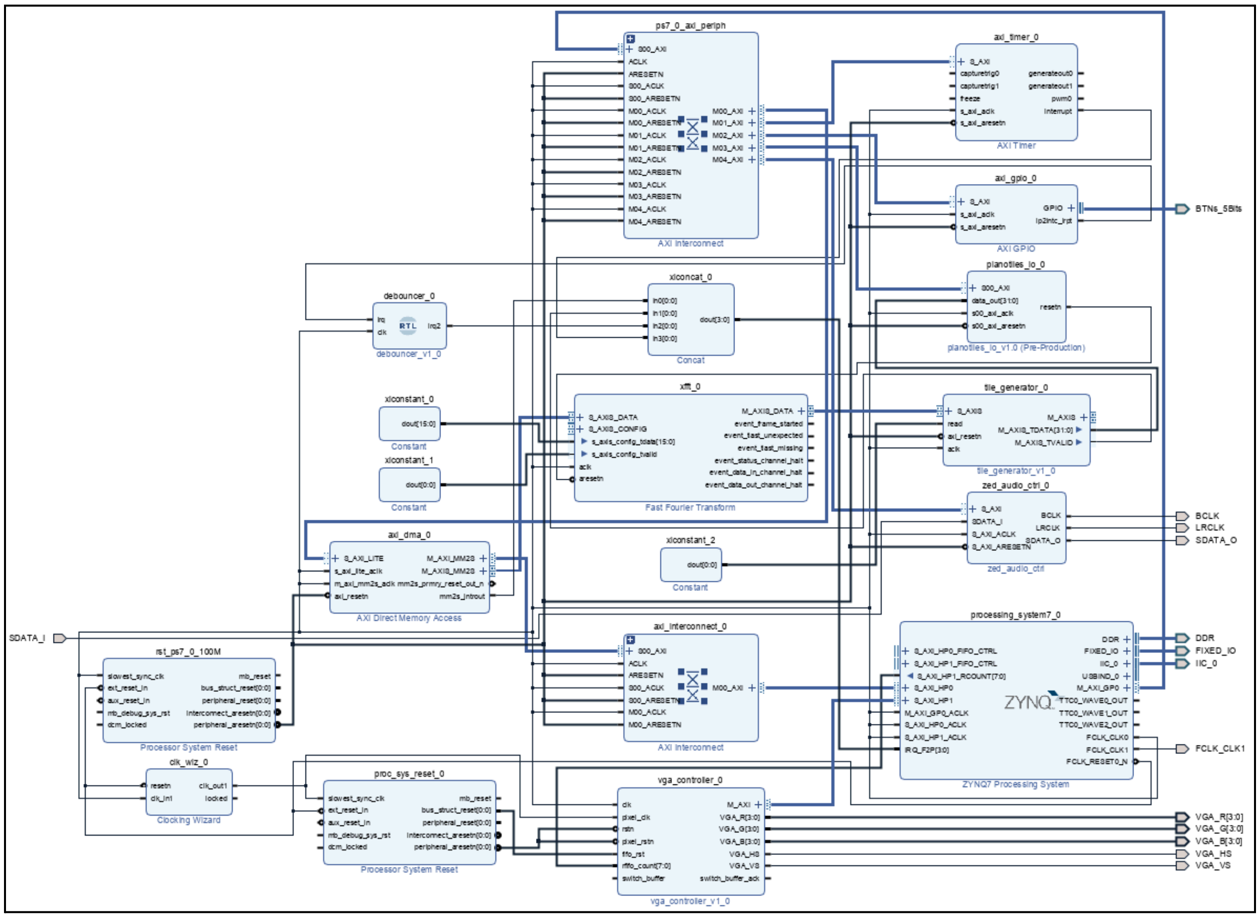


Figure 2: Vivado Block Design

This stage of the project was a tall task. Understanding how all AXI components are to be connected was a very important prerequisite before handling this. There was some important data that was difficult to understand how to access strategically. A workaround that I found was to connect all of these data signals into a custom IP block called Piano Tiles IO. This block took advantage of the slave registers that can be accessed in software. Input ports include the tile magnitude/bin and a resetn output port. These are important signals that need to be read or written to. Another component that needed to be understood was the processor system reset, as there basically needs to be one for each subsystem: video and audio. This is due to each having a different slowest clock. More knowledge about the ports on the Zynq Processing System was needed, such as the High Performance (HP) AXI ports; both the DMA and VGA controller had to be allocated for. To be able to fully understand this portion, completing both the audio and video tutorial was essential, plus many more posts on the Xilinx forums.

3.2.5 Software Integration

Definitely one of the more intensive portions of the project. Being given components from my partner and having to integrate the rest of the project was an arduous task. The software included modules for: display/menus, DMA interface, piano tiles I/O, interrupts, tiles and audio. Synchronizing and ordering the processes for each module in each main process was also very important. The source code for this is not the neatest but it definitely defines what the project should and can do. Learning to utilize two cores on the Zedboard was not too difficult. However, a big issue was the first attempt at implementing interrupt handling on core 1 as it seems to not be as intuitive as if it were to be implemented in core 0. Other than that hurdle, communication between the two cores is simple with just accessing the same address and interpreting correctly. Finally, ensuring that the software utilizes the custom hardware properly was essential to understand.

3.3 Tile Generator Algorithm Flow Chart

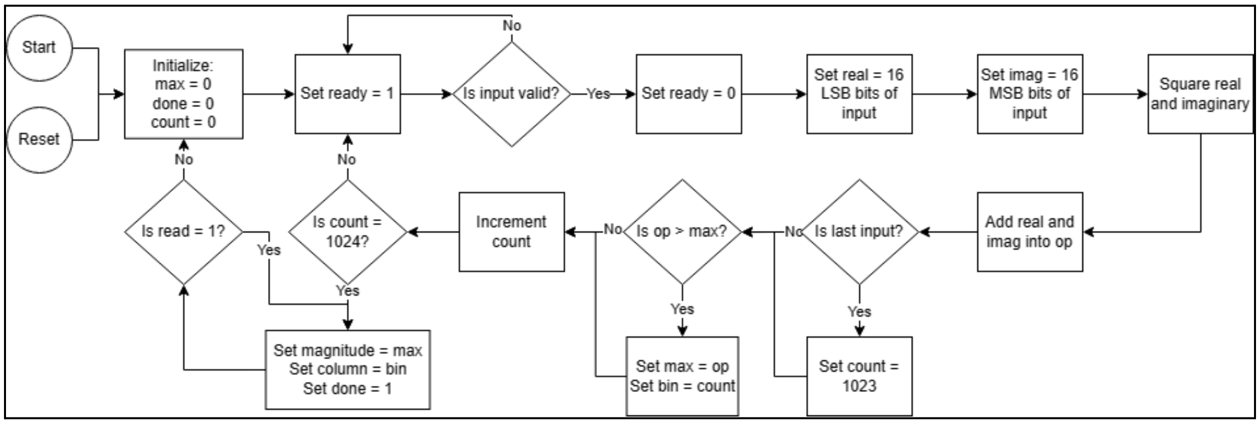

Start
Reset
Initialize: max = 0 done = 0 count = 0
Set ready = 1
No
Is input valid?
Yes
Set ready = 0
Set real = 16 LSB bits of input
Set imag = 16 MSB bits of input
Square real and imaginary
Add real and imag into op
Is last input?
No
Yes
Set count = 1023
Is op > max?
No
Yes
Set max = op Set bin = count
Increment count
Is count = 1024?
No
Yes
Set magnitude = max Set column = bin Set done = 1
Is read = 1?
Yes
No

Figure 3: Tile Generator Algorithm Flow Chart

3.4 Ensuring Ease of Integration

Ease of integration is essential to having a more straightforward integration process; especially when using the bottom-up design methodology. I made sure to communicate my design decisions with my partner to keep everyone up to date. Examples of which were to ensure that the output of my tile generation process can eventually feed his portion an array of integers mapping straight to which column. This was very important to clarify for him to get going with designing the software for tiles falling down. Also since I did most of the hardware design, I notified my partner that the hardware button debouncer must be fed an interrupt and output a debounced interrupt.

In terms of my design, I guaranteed that all hardware blocks I design were to easily connect to each other, whether it used an AXI interface or not. On the software side, I made certain that functions for the display/menus, DMA interface, piano tiles I/O, interrupts, tiles and audio had meaningful and easy to use API functions. These functions would only need parameters that completely define the method that is being executed. This made calling each of the functions in the main process make more sense and easy to order.

3.5 Hurdles Overcome

3.5.1 Tile Generation

The tile generation included working with the DMA, FFT and tile generator IP blocks. Learning how the FFT IP block and AXI interface works was a time-consuming task. However, this was crucial to make working with the block later on more effective. Knowing more about all the functionalities of the

FFT block made connecting it to the other IP block more understandable. Gaining knowledge about the AXI (stream) protocol was key to designing the tile generator IP block. The idea of handshaking had to be implemented with the tile generator and had to also be able to connect straight to the FFT block. The AXI input and output ports of the tile generator had to be manually configured to be compatible with other AXI peripherals such as the FFT block. I then had trouble with actually testing this little system; feeding data into the FFT block was a wall that was hit. I first attempted to mimic the AXI protocol with software, implementing the hand-shaking ideology, but I quickly realized this would totally not work. After some discussions with Chris, the use of an AXI DMA IP block was evident. Learning how to utilize the DMA block was very challenging. Not much documentation is up about the AXI DMA IP block, especially with the functionality I was looking for. The simple transfer mode gives me the option of simply sending an array given the array size through software. Most documentation that can be found online leads to a mode called scatter gather which did not apply to the use-case here.

3.5.2 Interrupt Handling

At first, the most knowledge I had about interrupts using Vitis was what the audio and video tutorial taught; button and timer interrupts. This part seemed really straight forward as there are built in or well-defined interrupt system set up functions for IP blocks such as the AXI GPIO and AXI Timer blocks. This was very confusing to apply to an interrupt signal that is neither of which and is simply a custom signal from a custom IP block. The done signal from the tile generator block and learning how to use it as an interrupt signal was very useful. This made the interrupt IDs, interrupt system initialization, set up and enabling make a lot of sense. Later connecting using the button interrupt that would be outputted by the button debouncer was very related to how to implement the tile generation interrupt.

3.5.3 Integration Process

This portion of the project was very intensive as a lot of components that were not really integrated together were all coming together at once. I personally knew how most of the code worked already so integrating my parts was not too difficult. However, when integrating my partner’s code, easy to understand API functions were not made so that took more time to understand. There was also a point where the interrupts stopped working at all but that was solely due to them being implemented on core 1. Implementation of interrupts on core 1 is not as intuitive as being in core 0. Finally, synchronizing the two cores and ensuring proper communication between the two was a bit tough at first.

3.6 Tools Used

3.6.1 Vivado 2020.2

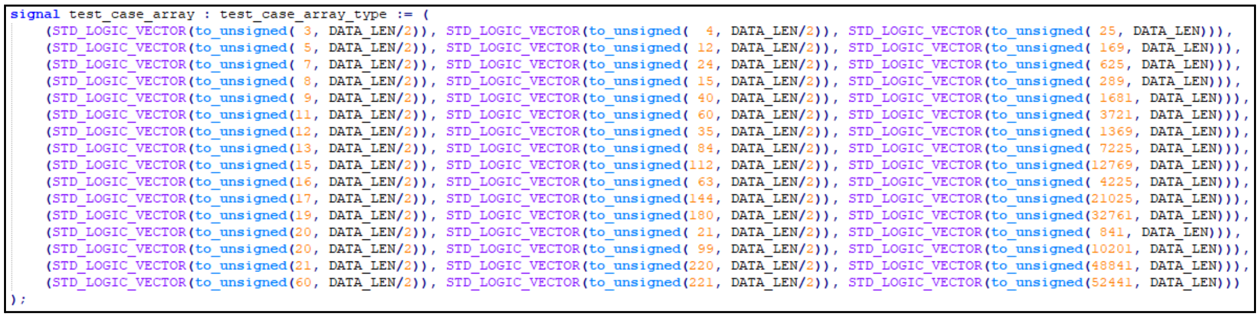
I learned a lot about Vivado. Creating custom AXI peripherals was very useful for creating the piano tiles I/O block and the tile generator IP block. I grew accustomed to the block design interface of the program and how connections between all IP blocks should be decided upon. Many smaller IP blocks such as the constant and concat IP blocks ensured that the blocks flowed with finesse. Learning how to place RTL blocks simply defined by .vhdl files was very useful especially for the button debouncer. IP block configurations were fleshed out; including the Zynq Processing System, FFT block, DMA block and clock wizard.

3.6.2 Vitis 2020.2

Vitis has shown me that it is capable of many things. Ranging from setting run configurations to store data before running to debugging software. Vitis is able to recognize the structure of the project given the hardware description. Setting up dual-core functionality is also something Vitis promotes nicely.

3.6.3 ModelSIM

ModelSIM was used to simulate the tile generator block. A tile generator testbench was created to do so. This test bench was created when the tile generator block was still generating a genuine Euclidean Distance. Figure 4 below displays the test cases that were run with the tile generator block. The first entry is the input real part, second is the input imaginary part and last is the expected output. The test ended with 100% accuracy so I knew that the block was working.



```
signal test_case_array : test_case_array_type := (
    (STD_LOGIC_VECTOR(to_unsigned( 3, DATA_LEN/2)), STD_LOGIC_VECTOR(to_unsigned(  4, DATA_LEN/2)), STD_LOGIC_VECTOR(to_unsigned( 25, DATA_LEN))),
    (STD_LOGIC_VECTOR(to_unsigned( 5, DATA_LEN/2)), STD_LOGIC_VECTOR(to_unsigned( 12, DATA_LEN/2)), STD_LOGIC_VECTOR(to_unsigned( 169, DATA_LEN))),
    (STD_LOGIC_VECTOR(to_unsigned( 7, DATA_LEN/2)), STD_LOGIC_VECTOR(to_unsigned( 24, DATA_LEN/2)), STD_LOGIC_VECTOR(to_unsigned( 625, DATA_LEN))),
    (STD_LOGIC_VECTOR(to_unsigned( 8, DATA_LEN/2)), STD_LOGIC_VECTOR(to_unsigned( 15, DATA_LEN/2)), STD_LOGIC_VECTOR(to_unsigned( 289, DATA_LEN))),
    (STD_LOGIC_VECTOR(to_unsigned( 9, DATA_LEN/2)), STD_LOGIC_VECTOR(to_unsigned( 40, DATA_LEN/2)), STD_LOGIC_VECTOR(to_unsigned( 1681, DATA_LEN))),
    (STD_LOGIC_VECTOR(to_unsigned(11, DATA_LEN/2)), STD_LOGIC_VECTOR(to_unsigned( 60, DATA_LEN/2)), STD_LOGIC_VECTOR(to_unsigned( 3721, DATA_LEN))),
    (STD_LOGIC_VECTOR(to_unsigned(12, DATA_LEN/2)), STD_LOGIC_VECTOR(to_unsigned( 35, DATA_LEN/2)), STD_LOGIC_VECTOR(to_unsigned( 1369, DATA_LEN))),
    (STD_LOGIC_VECTOR(to_unsigned(13, DATA_LEN/2)), STD_LOGIC_VECTOR(to_unsigned( 84, DATA_LEN/2)), STD_LOGIC_VECTOR(to_unsigned( 7225, DATA_LEN))),
    (STD_LOGIC_VECTOR(to_unsigned(15, DATA_LEN/2)), STD_LOGIC_VECTOR(to_unsigned(112, DATA_LEN/2)), STD_LOGIC_VECTOR(to_unsigned(12769, DATA_LEN))),
    (STD_LOGIC_VECTOR(to_unsigned(16, DATA_LEN/2)), STD_LOGIC_VECTOR(to_unsigned( 63, DATA_LEN/2)), STD_LOGIC_VECTOR(to_unsigned( 4225, DATA_LEN))),
    (STD_LOGIC_VECTOR(to_unsigned(17, DATA_LEN/2)), STD_LOGIC_VECTOR(to_unsigned(144, DATA_LEN/2)), STD_LOGIC_VECTOR(to_unsigned(21025, DATA_LEN))),
    (STD_LOGIC_VECTOR(to_unsigned(19, DATA_LEN/2)), STD_LOGIC_VECTOR(to_unsigned(180, DATA_LEN/2)), STD_LOGIC_VECTOR(to_unsigned(32761, DATA_LEN))),
    (STD_LOGIC_VECTOR(to_unsigned(20, DATA_LEN/2)), STD_LOGIC_VECTOR(to_unsigned( 21, DATA_LEN/2)), STD_LOGIC_VECTOR(to_unsigned( 841, DATA_LEN))),
    (STD_LOGIC_VECTOR(to_unsigned(20, DATA_LEN/2)), STD_LOGIC_VECTOR(to_unsigned( 99, DATA_LEN/2)), STD_LOGIC_VECTOR(to_unsigned(10201, DATA_LEN))),
    (STD_LOGIC_VECTOR(to_unsigned(21, DATA_LEN/2)), STD_LOGIC_VECTOR(to_unsigned(220, DATA_LEN/2)), STD_LOGIC_VECTOR(to_unsigned(48841, DATA_LEN))),
    (STD_LOGIC_VECTOR(to_unsigned(60, DATA_LEN/2)), STD_LOGIC_VECTOR(to_unsigned(221, DATA_LEN/2)), STD_LOGIC_VECTOR(to_unsigned(52441, DATA_LEN)))
);
```

Figure 4: Test Cases for Tile Generator Block

3.7 Ensuring Success

To ensure success, some tactics I used were to ensure that any module that I design that interfaces with hardware is also easily worked with. Easy to use API functions were made such as setResets, readMagnitude, transferData and so on. This made integrating software much easier than it would have been not implementing this method.

3.8 Design Methodology

My design methodology was a bottom-up approach. This meant that I would implement the smaller components which I would then connect together to make a sub-system. This sub-system would then be a component in the overall system. I actually looked into designing the whole project at first with full software using Python to give me a better idea of what to do. This did not really go anywhere as I could not get to something that I desired the resulting game to look like. This bottom-up approach includes starting with the tile generator block, then the FFT block and then to the DMA block. All these blocks would then be connected to create the tile generation system and that would be one little system in Piano Tiles' system.

3.9 Source Code Control

Keeping track of source code is essential, especially when working in groups or when migrating to different physical environments; home and the ENSC lab. GitHub was used to version control and keep coordinated with my partner. Another less reliable method was to simply keep files on OneDrive and a

USB stick. I could easily transfer Vivado projects between different systems, and set up new Vitis projects using the same source code.

3.10 Testing

Testing is very important to make sure individual subsystems are working correctly; ensuring this is true before integration is essential. Reiterating Section 3.6.3, ModelSIM was used to simulate the tile generator block with test cases. This testbench would iterate through all test cases: input specific data and compare the output to the expected output. Testing other portions of the project such as the actual tile generation on the software side was also something to note. The output of the tile generator would be 32-bits so ensuring that the magnitude value and bin values were somewhat reasonable was something to check. This was done by sometimes printing what the magnitude and bin value is through UART. Testing other features such as interrupts was a challenge as well. Button interrupts were thoroughly tested to make sure that each button was properly distinguished on the software side. The tile generation interrupts were tested by making sure that tile arrays were actually being generated. Further on in the integration process, testing each of the steps on integration was essential. Adding so many features into one can cause issues such as address spaces overlapping, changes in addresses and so on. More features such as access to audio data, menu displays and dual-core communication were tested by visuals/UART terminal output.

3.11 Modules I Wrote

1. *DMA Interface (dma_interface.c/h)*

   Includes DMA initialization, configuration and data transferring.
2. *Piano Tiles IO (fft_io.c/h)*

   Includes a read function for tile generator output, and a set function for personal reset.
3. *Interrupts (interrupts.c/h)*

   Includes generalized functions for easy-to-use calls for any type of interrupt.
4. *Major portions in Menus (menu.c/h)*

   Includes drawing each of the menus knowing their base address and adaptation of partner's code to work with the rest of the system.
5. *Tile Interface (tiles.c/h)*

   Includes a function that displays a summary of the generated tiles and will dynamically set thresholds depending on the input audio. These thresholds will determine if a tile would be displayed or not.
6. *Interrupt handling (in main.c/h)*

   Tile interrupt handler that will read slave register 0 from the tile generator AXI peripheral and will apply the bit mask accordingly to extract magnitude and corresponding bin. Will store this new tile depending on which song is chosen.

   Button interrupt handler will help traverse through the different modes (menus). Each button will have a different function in a different mode.
7. *Sprite/Number drawing (in main.c/h)*

   Is able to draw a positive integer that has no more than 6 digits. Positioning of this input depends on what mode the system is currently in. More functions are made to display lists of these numbers, for example, for the choose song menu for highscores.
8. *Core 0 Main and Helper functions (main.c/h)*

The main function for core 0 handles initializing all of the modules above and initializing certain variables that will define the current state of the system - mode. Will lead the user to a loop that will help them traverse through the different modes/functionalities of the game. Communicates with core 1 through a COMM_VAL to tell what song to play.

9. *Core 1 Main function (piano_tiles.c/h)*

    The main function for core 1 handles basically only audio. The certain audio data will be played depending on the COMM_VAL that is sent by core 0.

3.12 Milestones

My milestones include:

1. Creating a testbench for a tile generation hardware unit
2. Integrate the tile generation hardware unit with an FFT hardware block and software
3. Make tile generation occur a few seconds before the corresponding sound plays
4. Create number drawing function
5. Implement tile missing

All of these functionalities were successfully implemented and were integrated into the final game. Deeper analysis of each of these milestones are displayed in previous sections.

3.13 Lessons Learned

I have learned that working with Vivado and Vitis can be a real pain. There are certain versions with each software that have multiple bugs that people in hard-to-find forum posts have found solutions to. There are some cases where the simple fix to the issue was to upgrade the software to a more recent version, but that may lead to even more issues. One big bug that was very annoying was changing the Makefiles for custom AXI IP blocks that are deep in the auto-generated platform project. There were points where I had no idea what I was doing wrong where the simple fix was changing some ${OUTS} to ${$OUTS}.

4. Community Contribution

Unfortunately I had not participated in much community contribution. A very minimal amount was done, only asking questions and answering sometimes in the students' own Discord channel for ENSC 452.

4. Feedback to Xilinx

I had always run into the issue of having to look at the documentation about a hardware IP block that Xilinx had supplied. However, sometimes documentation may not clearly display what I am questioning. I would love more info blocks whenever I am in the re-customize IP window. I believe this is possible as a certain version of each IP block is saved in my system, so a corresponding file for the documentation should also be available. More setup with the documentation itself may be needed to easily map sections to certain options in the re-customize IP window. An example is this little portion from Figure 5 of the FFT block's re-customize IP window. An info button would be great to link to the actual documentation, or even when hovering over an option, an explanation of what that does.

5. Course Feedback

The course overall was very nicely structured. Anything that was not set in stone by the course schedule was talked about between the professor and the students for compromise. The timeline of the project worked pretty well as the break weeks (at least for me) landed on weeks with tougher tasks. However, I do believe that the project can be started earlier. Maybe the course content in the very beginning of the semester can be sped up to ensure this. I loved the "open" lab concept of the course as you have your own schedule to complete the milestone but are expected to finish it by a certain date. This kept me on track for the project as the milestones list was already a great way of piecing out the project. The course could be five credit hours, however, that would mean a few other courses should also be bumped up to five credit hours as well. I personally feel keeping it at four credit hours is totally fine, but some emphasis on the difficulty would be great. The course schedule for a Computer Engineering student at SFU already fulfills 150 credits in their degree so the amount of credits would not really matter. Maybe there should be a form to fill out to pursue this course while doing Capstone? I had chosen to do this and another course and did feel like being in two Capstone courses. Overall, lectures were great, but getting through the material faster would not be that bad. Going through more applied concepts such as actually using Vitis, I feel, would be more useful as this course does seem to emphasize the final project much more.

6. References

[1] L. H. Crockett, R. A. Elliot, M. A. Enderwitz, N. David, and R. W. Stewart, *The ZYNQ Book Tutorials: For Zybo and Zedboard*. Glasgow: Strathclyde Academic Media, 2015.

[2] “PG109 Fast Fourier Transform LogiCORE IP Product Guide,” AMD Technical Information Portal, https://docs.amd.com/r/en-US/pg109-xfft/event_data_in_channel_halt

[3] J. Johnson, “Using the AXI DMA in Vivado,” FPGA Developer, https://www.fpgadeveloper.com/2014/08/using-the-axi-dma-in-vivado.html/