ENSC 452
Final Project Group Report & User's Manual

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

Group 21

Sina Haghighi	Justin Mateo
301390048	301393149
sinah@sfu.ca	*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. Background

2.1 Fast Fourier Transform

The goal of Piano Tiles is to give the user the ability to choose a song to play and have appropriate tiles falling down the screen. The tiles should hit the bottom of the screen on beat to the song and the column the tile falls down should relate to the fundamental frequency or pitch of the song at that moment. To be able to achieve appropriate tiles generated for each song, there must be some way of frequency analyzing the audio data. The most common way of performing this type of analysis includes the use of the Fast Fourier Transform (FFT) which is computationally intensive; especially when performed on an *n* amount of songs of variable length. The FFT is a function that transforms an input that is in the time-domain to an output in the frequency-domain. With the frequency-domain representation of the input, information such as the fundamental frequency of the input can be found. Displayed in Figure 1 is an example of data represented in the time-domain with its respective frequency-domain representation.

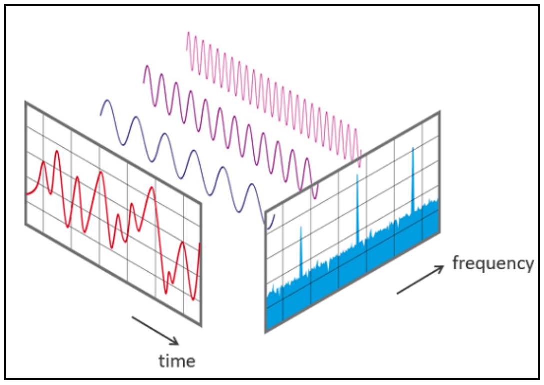


Figure 1: Fourier Transform of a Time Signal [2]

Note that the Fast Fourier Transform is an algorithm that computes the Discrete Fourier Transform (DFT) of a signal in a computationally efficient manner. The DFT is just the method of finding the discrete frequency-domain representation of a discrete time signal. Compare the DFT to the normal Fourier Transform that finds the continuous frequency-domain representation of a continuous time signal. The discrete fourier transform of an audio signal will consist of an n amount of frequency-bins which are each mapped to a magnitude value. These frequency-bins' frequency size (or resolution) can be realized solely knowing the input's sample rate (f_s) and window size - number of data points the FFT takes in (n). Due to the Nyquist limit, the output's sample rate will be half of the input's sample rate [3]. Also, since the output of the FFT will include an equal amount of positive and negative frequency bins, only half of the bins are allocated to positive frequencies. This leads to a frequency resolution being computed with Eq. 1.

$$Resolution = (f_s/2)/(n/2) = f_s/n \qquad \textit{(Eq. 1)}$$

2.2 Finding Fundamental Frequency

The fundamental frequency of a portion of audio is very important information to find as it is most likely related to the audio's pitch that the user would be hearing at that moment. Being able to correlate the tile that is falling down the screen to the pitch of the audio that the user is hearing can make the user experience better. To be able to find the fundamental frequency of subsections of the audio data, the FFT is performed on the subsections. The output of this FFT will include all frequency bins and their corresponding information of magnitude. This is perfect information to collect as the fundamental frequency is most likely in the frequency bin that has the greatest magnitude. There are some cases where the fundamental frequency is in fact not in the bin with maximum magnitude but the majority of the time it is. Therefore, Piano Tiles has been designed to find the fundamental frequency by simply finding the frequency bin with maximum magnitude.

2.3 Button Debouncing

Hardware buttons, particularly push buttons deal with voltage "bouncing" which is caused by vibrations in the physical button which oscillate between high and low voltage, or engaged and not engaged, found in Figure 2. This results in unintended activations of the button. In order to solve such issues, we implement a mechanism called a debouncer. The debouncer implements a threshold cycle during which any vibrations in the button within a certain time period are ignored, and once that period is over, whatever value that the button has will be registered. The button debouncing enables the button to perform precise clicks, used in our game for song selection, piano tile pressing, and menu interfacing.

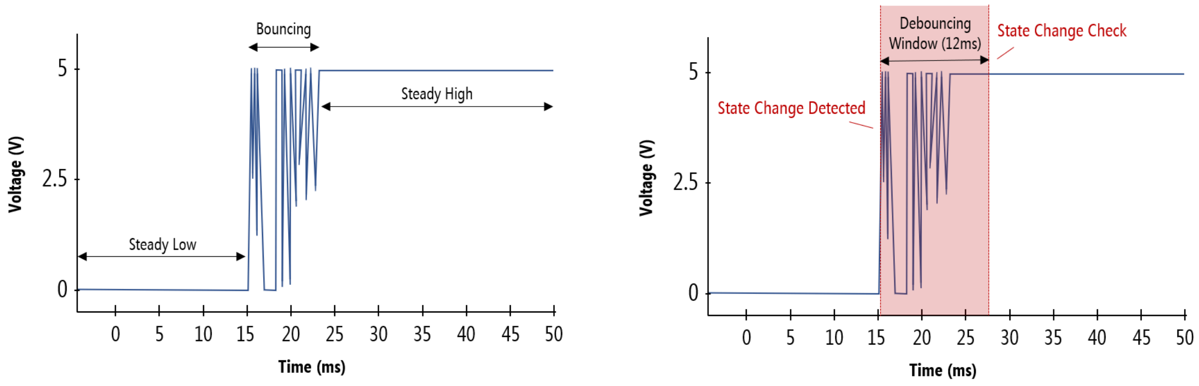


Figure 2: Button input Voltage Behaviour on Click [4]

2.4 Direct Memory Access

Accessing memory through software is very useful and easy to understand; however, it is inefficient. The idea of direct memory access (DMA) is to allow certain hardware blocks in the design to be able to access memory directly. The DMA hardware will act as the middle-person between memory and the respective hardware block. Advantages of DMA that Piano Tiles takes advantage of are less processor overhead and more efficient data transfer; there are many more advantages such as parallel processing and less power consumption. Requests for data are complicated to execute but the output data can be streamed to hardware very efficiently and with the appropriate protocol.

2.5 Audio Playback

The audio IP is a crucial part of this project in order to perform writes to the audio codec that will output through the AUX of the ZedBoard. Efficiently storing song data in DRAM and writing through to the codec were the core components to ensure success in this area. Particularly converting wav data to the appropriate sampling rate, encoding, and mono-stream enabled an efficient storage of data. The data was originally as wav files, which were then resampled at the codec's frequency, which is 44.1kHz. Upon resampling, the stereo data is fed into a mean compressor, which averages the stereo data into a mono-audio and encoded in 16 bit format. Once the data size is optimized, it passes through a wav to binary file conversion, the binary file is then stored in the project file directory and loaded into a blob.S through the assembler.

2.6 Video Display

For the video displaying, we pursued a VGA output from the board into a monitor with resolution of 1280 x 1024. The video display unit is responsible for communicating pixel values to the display at various addresses with an RGB value. Often, and in our case, programs will employ data structures which hold the pixel data to an image, or Sprites, which are then more easily written to the screen. The sprites that needed to be designed included numbers, tiles, and background menus. Upon designing the sprites, they are loaded into the run configurations of the program, which essentially store the sprite data to a memory location within the core's allocated memory.

3. System Overview

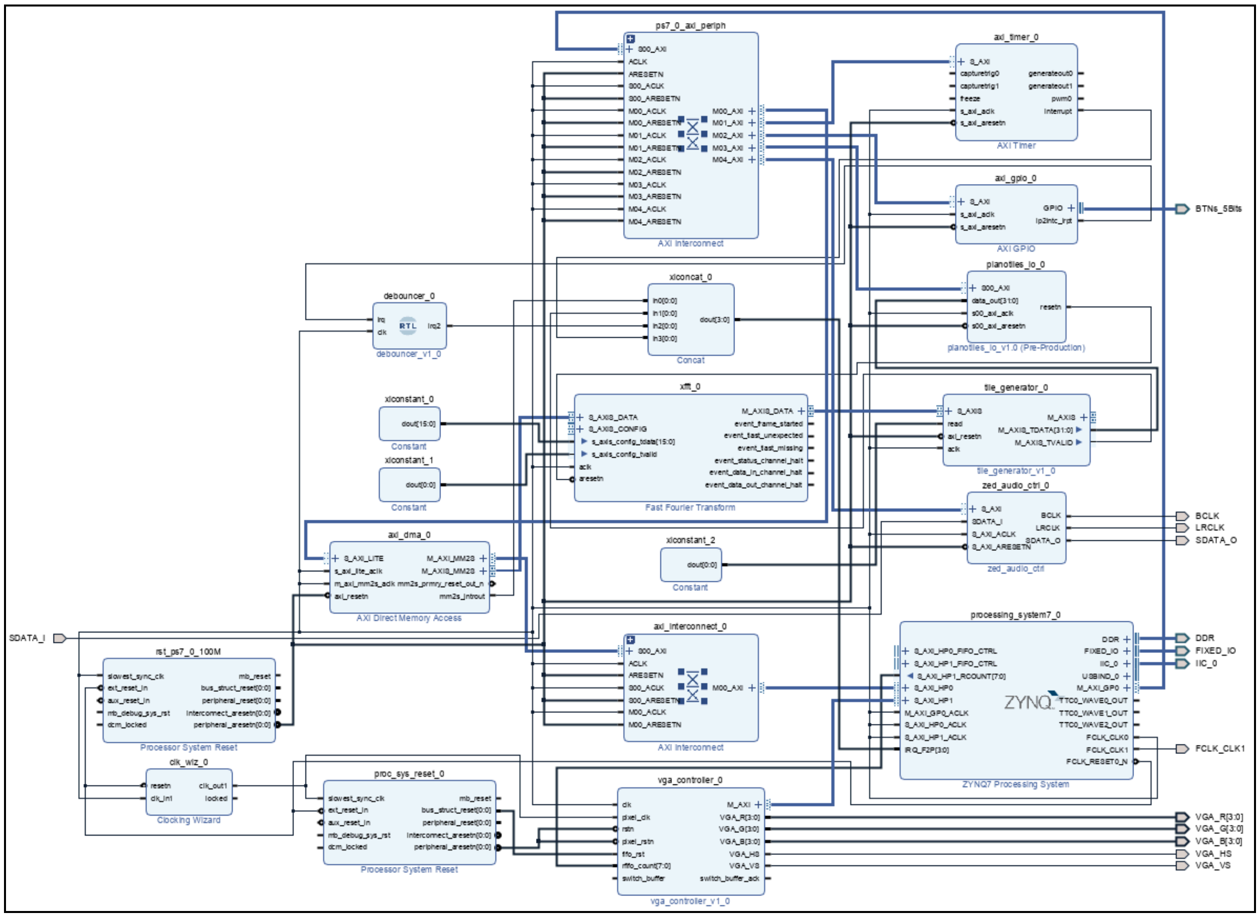


Figure 3: Vivado Block Design

3.1 System Block Diagram

The system block diagram that is displayed in Figure 4 encapsulates the high-level hardware blocks that are implemented in Piano Tiles' design. It can be seen that Xilinx's supplied IP blocks give the ability to directly access BRAM data and apply the fast fourier transform. This frequency-domain representation of the supposed audio data can then be further analyzed in software with core 0. The button driver and it's debounced interrupt signal can then also be analyzed by core 0. The VGA display handling is also performed by core 0. Audio is then handled by core 1. It can be seen that memory is shared between the two cores and this also gives the ability of having a communication value used for varying functionality.

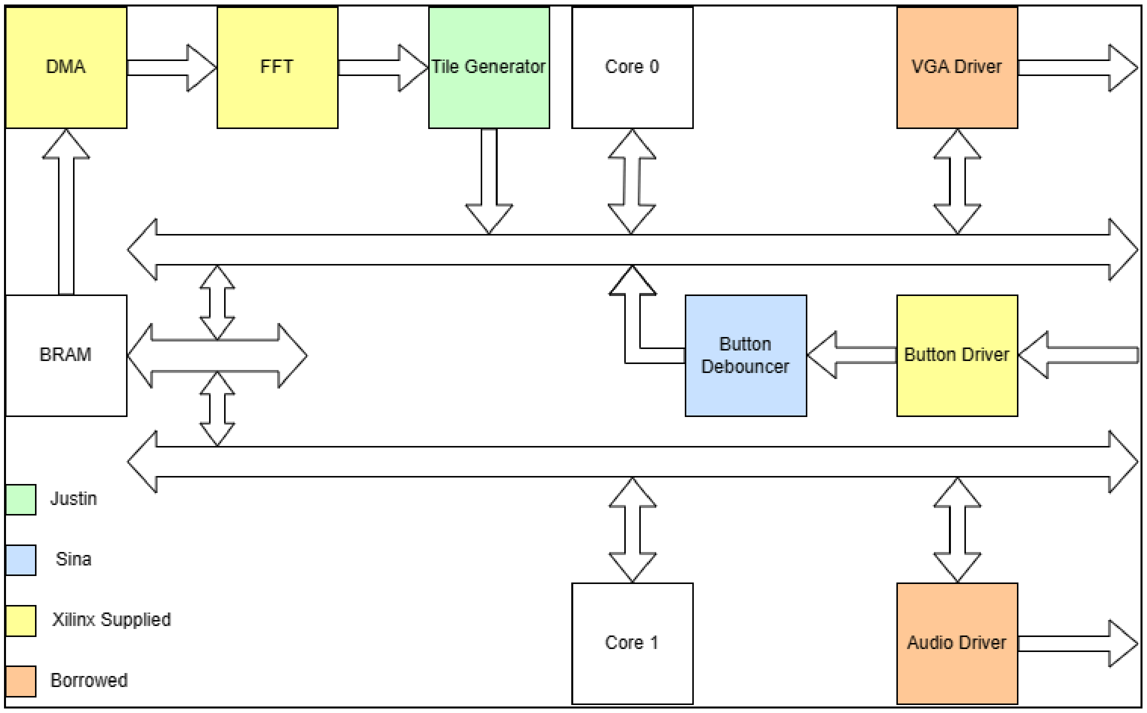
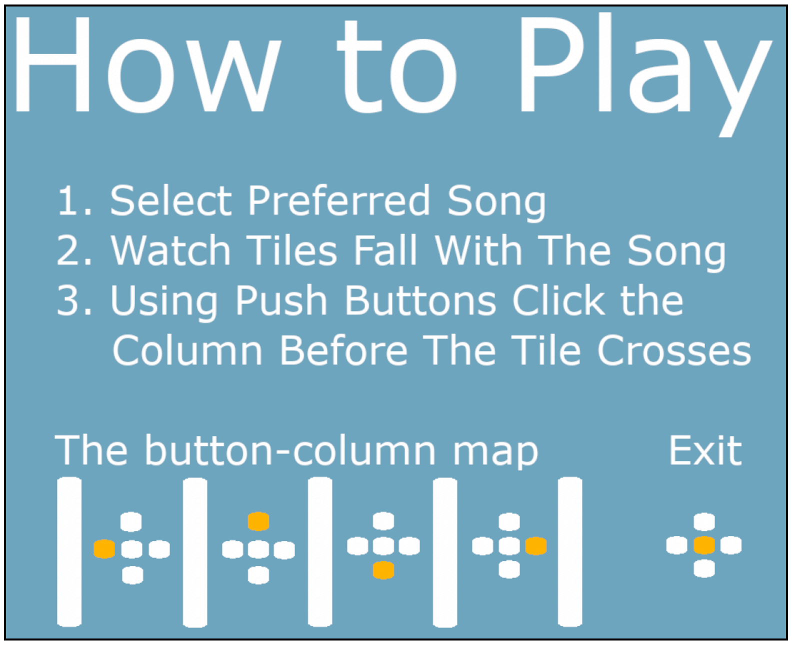


Figure 4: Piano Tiles System Block Diagram

3.2 User's Manual

Main Menu - On startup:

1. Quick Start:
    a. Quick start will get you right into the game play, where the first song is chosen and you begin playing
2. Select Song:
    a. This will navigate you to another page where you will be greeted with several song options including a Classic, Guitar, and Drum song
    b. Upon selecting the song, the program will begin the tile generation and gameplay with that song being played back
3. How To Play:
    a. This option serves as a brief overview of how the application works, including everything discussed in this manual, in addition to the ways the buttons are mapped to columns for gameplay, shown in Figure 5.



How to Play

1. Select Preferred Song
2. Watch Tiles Fall With The Song
3. Using Push Buttons Click the Column Before The Tile Crosses

Figure 5: How-To Start Menu in the Game

4. Game Play
    a. During game play, the piano tiles will fall from the top of the screen to the bottom of the screen
    b. The player's goal is to timely hit the piano tiles as they hit the bottom of the screen
    c. Score Breakdown is as follows:
        i. Premature Tile Hit (hitting the tile before it reaches the bottom) : -20
        ii. Tile Miss (the tile passes through the screen without a hit) : -50
        iii. Hitting the tile when it's at the bottom, the amount earned depends on how much of the tile is in frame:
            1. 1% - 20% in frame: +10
            2. 21% - 40% in frame: +20
            3. 41% - 60% in frame: +30
            4. 61% - 80% in frame: +40
            5. > 80% in frame: +50
    d. The current score is shown at the top of the screen
5. Score Menu:
    a. Upon finishing the game, a brief menu will be shown, including the final game score, as well as the high score that has been achieved for the specific song.
    b. If the players game score is better than the high score, it will update the high score
    c. After this menu is shown, the player will return to the Main Menu

3.3 IP Descriptions

3.3.1 Tile Generator

The tile generator block (tile_generator_0) uses the AXI stream interface type. It acts as a secondary peripheral to the FFT block and outputs the maximum magnitude with 32 bits and an interrupt for when the magnitude is valid. Other inputs include a *read*, *axi_resetn* and *aclk* which will be further explained later. The main function of this block is to take in the 1024 points of output data from the FFT block and realize which index (bin) was of highest magnitude. The input-data consisted of a 16-bit real and 16-bit imaginary part, so the magnitude was calculated with the Euclidean distance. The output-data consisted of a real magnitude and its corresponding bin number.

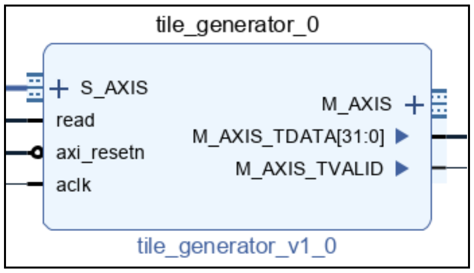


Figure 6: Tile Generator IP Block

3.3.2 Button Debouncer

The button debouncer block (debouncer_0) is an RTL block. It handles a 1-bit interrupt and performs debouncing by applying a debouncing window, implemented by a counter in the logic. Once the input (irq) is active, the program will apply a 10ms window (configured to be controlled by our clock frequency), once the window is complete, the final state of *irq* is passed through to the output (*irq2*).

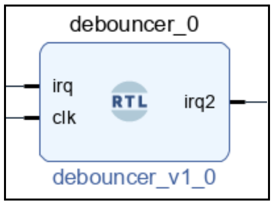


Figure 7: Button Debouncer IP Block

3.3.3 Piano Tiles IO

The piano tiles I/O block (pianotiles_io_0) is an AXI block that simply handles reading from and writing to certain addresses. This helps with reading the data feeding out from the tile generator through software which includes the magnitude and corresponding bin. This also helps with writing resets to the hardware manually through software.

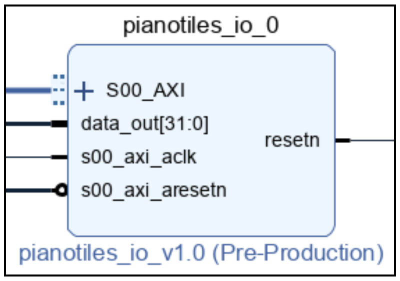


Figure 8: Piano Tiles IO IP Block

3.3.4 Zynq Processing System v5.5

The Zynq processing system block (processing_system7_0) handles communicating with almost all other hardware IP blocks. It handles receiving inputs from AXI peripherals through the high performance AXI ports, providing desired clock signals, resets and even access to certain hardware in the Zedboard. Most importantly, hardware interrupts can be accessed here.

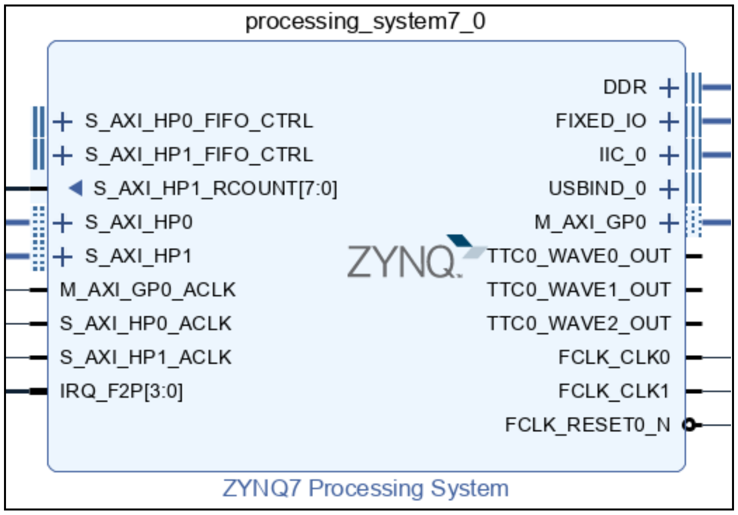


Figure 9: ZYNQ Processing System IP Block

3.3.5 Fast Fourier Transform v9.1

The FFT block (xfft_0) is an AXI stream peripheral that is able to receive AXI stream data and output AXI stream data. This hardware block is connected to the AXI DMA block through the S_AXIS port. This input data is what will be applied to the fast fourier transform. When the output is ready, the M_AXIS port will output the DFT of the input data.

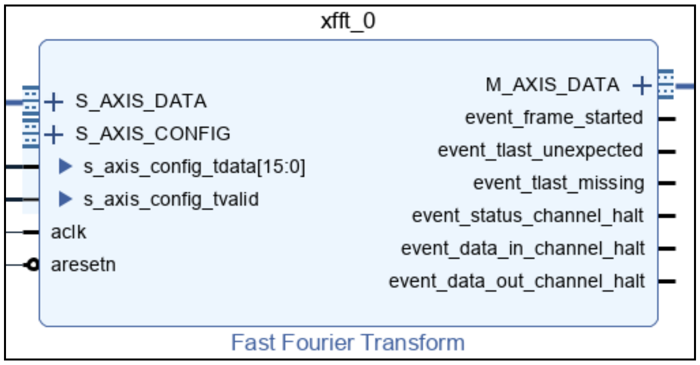


Figure 10: Fast Fourier Transform IP Block

3.3.6 AXI DMA v7.1

The AXI DMA block (axi_dma_0) is an AXI peripheral that is used to directly access memory. The DMA block will receive instructions from the Zynq processing system of what data to access. The requested data will then be outputted through the M_AXI_MM2S which is connected to the FFT block. The AXI DMA IP block also includes an mm2s_introut signal which can be used to notify the system when data transfer is completed.

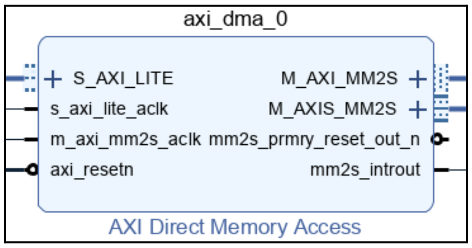


Figure 11: AXI DMA IP Block

3.3.7 Processor System Reset v5.0

The processor system reset(s) (proc_sys_reset_0 & rst_ps7_0_100M) is a hardware block that is used to ensure that all peripherals in the system are reset properly. The processor system reset is used to sync up the reset signals with corresponding clock signals. This is especially useful for systems like Piano Tiles as the slowest clock in the VGA portion is 108 MHz while the slowest clock in the audio portion is 100 MHz.

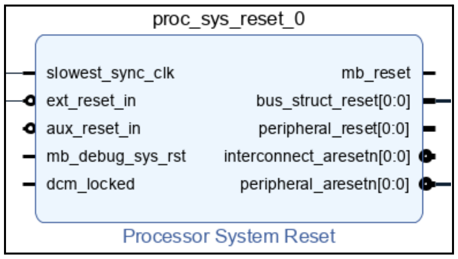


Figure 12: Processor System Reset IP Block

3.3.8 AXI GPIO v2.0

The AXI GPIO block (axi_gpio_0) is used to access the 5 buttons that are on the Zedboard. This AXI block is set up such that the GPIO port connects to the actual button hardware while the ip2intc_irpt port outputs an interrupt signal when a button press is executed. The GPIO. Realizing which button is actually pressed is done by reading a certain address in the AXI block itself.

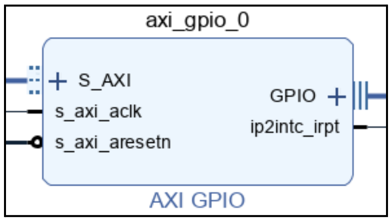


Figure 13: AXI GPIO IP Block

3.3.9 Clock Wizard v6.0

The clock wizard block (clk_wiz_0) is the hardware block that helps with achieving slower clock signals than what is readily available. This is used in the VGA portion of the hardware as the fabric's clock is set to 120 MHz but a 108 MHz clock is needed for the VGA controller's pixel clock.

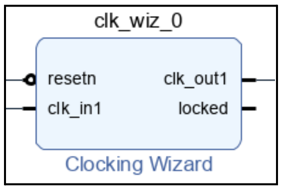


Figure 14: Clocking Wizard IP Block

3.3.10 AXI Interconnect v2.1

The AXI interconnect block(s) (axi_interconnect_0 & ps7_0_axi_periph) is what is used to allow the Zynq processing system to control multiple AXI peripherals, though not necessarily in parallel. The Zynq processing system is able to choose which peripheral to communicate with and be able to write to or read from each. Two were needed; one for the Zynq processing system to communicate with the various AXI peripherals and another one for the AXI DMA block to communicate with the Zynq processing system.

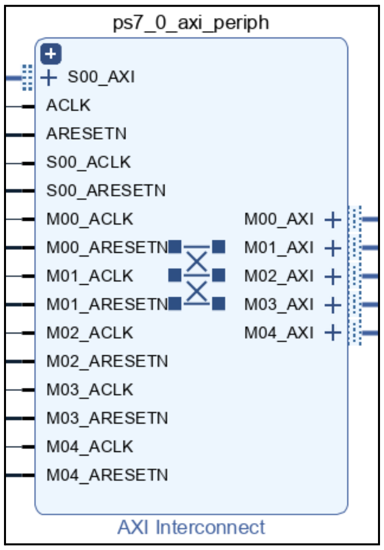


Figure 15: AXI Interconnect IP Block 1

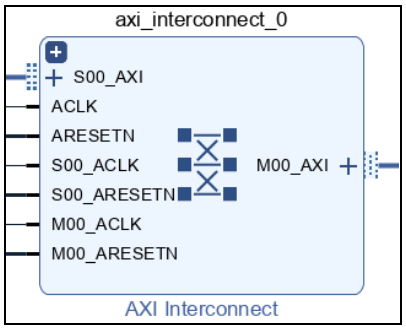


Figure 16: AXI Interconnect IP Block 2

3.3.11 Concat v2.1

The concat block (xlconcat_0) is able to concatenate multiple standard logic signals into one standard logic vector. This is especially useful for inputting multiple interrupts to the IRQ port on the Zynq processing system.

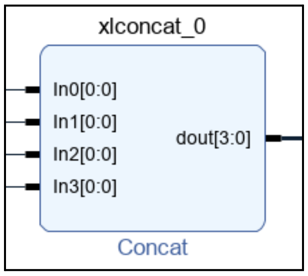


Figure 17: Concat IP Block

3.3.12 Constant v1.1

The constant block (xlconstant_i) is a hardware IP block that is able to output a constant standard logic vector. This is very useful for input ports that need to constantly be set as a specific value. For example this constant block in Figure 17 is to be connected to the configuration port of the FFT block. This configuration port on the FFT block must always be set to a certain value to ensure proper functionality.

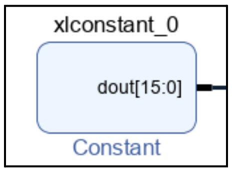


Figure 18: Constant IP Block

3.3.13 Zed Audio Control

The Zed audio control block (zed_audio_ctrl_0) is a AXI enabled hardware block that is able to output audio in SDATA_O, based on the SDATA_I which is the audio data. The BCLK is the base clock and the LRCLK is the left right clock for stereo audio.

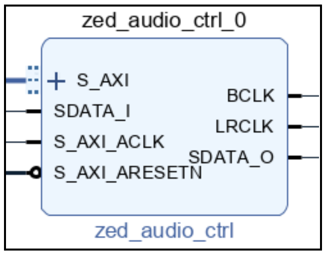


Figure 19: Zed Audio Control IP Block

3.3.14 VGA Controller

The VGA controller block (vga_controller_0) is a custom IP block which has a separate clock for pixels and overall block, this also means a separate rstn and pixel_rstn which are active low resets. The fifo_rst inputs the data in and rfifo_count is an std_logic_vector. Finally the switch buffer enables the double buffering of the input. The output is as a primary AXI peripheral with separate RGB colours, and a horizontal and vertical VGA output. In addition, for the switch buffer input there is an acknowledge signal sent.

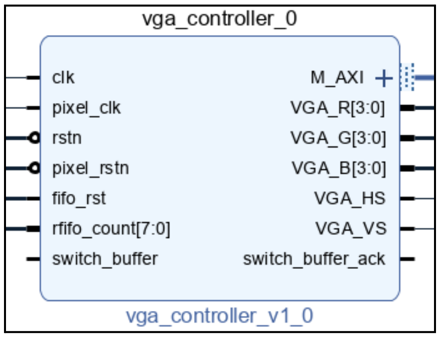


Figure 20: VGA Controller IP Block

4. Outcome

Results

Our program works quite well given the initial goals set out. We are able to effectively generate tiles using the FFT block to analyze songs and infer the frequencies bands (columns) at which each tile should generate. The audio is efficiently stored in our program and is able to be played back clearly, and in synchrony with the tiles being generated. The score analyzer, responsible for scoring how well a user is playing, works as expected, with the push-button inputs successfully debounced. The video controller properly displays the menus, including a getting started page, and a song selection page. On the game play page, the user is able to clearly see tiles scrolling down with very minimal distortion as it's being written to screen. Finally, the game score storing logic is implemented, allowing users to track their best scores for each song. Although some stretch goals would have been nice to implement, the overall program is fully implemented and intuitive to play.

Future Improvement

1. *Improved video streaming*

   Could have implemented double buffering throughout the program, particularly through the video streaming
2. *Appropriate frequency-bin distribution to game columns*

   The program could have used a quadratic distribution rather than a linear distribution for the frequency analysis, especially given that sounds are compressed in the sub 5kHz, this way there would be more spread of tiles across all columns rather than tiles being concentrated in one or two columns.
3. *Better current score updates*

   The current score that is being displayed in the actual game menu only updates when the user presses the button. This means that the displayed score is not always accurate as there are moments where the score is decremented due to missed tiles. The display function just needs to be implemented in a way that it can be called at any moment.
4. *Capability to play more songs*

   The functionality that is implemented in the source code only allows at most three songs to be used in the game. This can be solved by giving the user the ability to iterate through the song list rather than mapping a button to each song in the select song menu.

5. Blocks Description

This section further delves into the description of the hardware IP blocks that are chosen to be implemented in the system. The description of each block may include the actual functionality of the block and how the blocks are used.

5.1 Custom Blocks

5.1.1 Tile Generator

The tile generator IP block is designed to communicate with a primary AXI stream peripheral; more specifically, the FFT IP block. The output of the FFT block is known to be 32-bits long - the 16 MSB is the imaginary part and the 16 LSB is the real part. Both parts of this complex number are important to calculate the magnitude. The process of this tile generator block is easily described with the algorithm flow chart displayed in Figure 20.

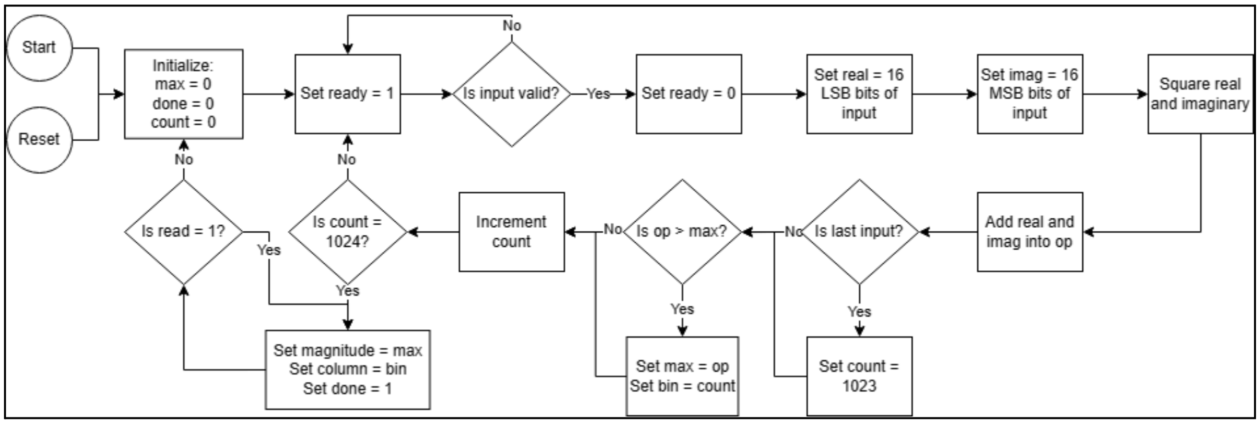

Start
Reset
Initialize: max = 0 done = 0 count = 0
Set ready = 1
Is input valid?
No
Yes
Set ready = 0
Set real = 16 LSB bits of input
Set imag = 16 MSB bits of input
Square real and imaginary
Add real and imag into op
Is last input?
Set count = 1023
Is op > max?
Set max = op Set bin = count
Increment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

Figure 21: Tile Generator Algorithm Flow Chart

To summarize, part of the process of calculating the Euclidean distance is applied to the complex number inputs. This is executed 1024 times while also keeping track of which bin included the maximum magnitude and its corresponding bin. Once this process is done, a signal is output which can then be used as an interrupt.

5.1.2 Button Debouncer

The Hardware Debouncer Block, efficiently tackles switch bounce issues typically associated with buttons. It stabilizes output signals, ensuring only clean transitions are recognized despite button erraticism. Utilizing debouncer filtering, it smooths button signals, with a specific debounce time of 10ms using a timer and managing with the processor clock frequency.

5.1.3 Piano Tiles IO

This custom hardware block is used to take advantage of how custom AXI peripherals are created. When a custom AXI peripheral is created, generated slave registers are included which can also be accessed for reading/writing through software. These slave registers were configured to be 32-bits long. This hardware block includes basic AXI ports plus an 32-bit input port and 1-bit output reset port.

The reset port is solely used to be able to reset certain hardware units in the system when desired. For example, this reset is connected to the reset port on the FFT block. The 32-bit input port is for the ability to read the magnitude and corresponding bin whenever a tile generation interrupt is to be handled. This 32-bit input is composed of 10 MSB for the bin number and 22 LSB for the magnitude. This data can be accessed by reading the 32 bits from the corresponding slave register. Then specific bit masks can be applied to extract the desired values.

5.2 Xilinx Supplied Blocks

5.2.1 Zynq Processing System v5.5

The Zynq processing system block is for controlling the whole system. The following configurations to the processing system have been applied: two high performance secondary AXI ports, one primary AXI port, a 4-bit interrupt port (IRQ), two clocks (120 MHz, 10 MHz) and finally access to hardware peripherals on the actual Zedboard (DDR, FIXED_IO, IIC and USB). This hardware block works with communicating with the VGA controller and AXI DMA block properly; data from each can be transferred to the processor. Four interrupts are handled by the Zynq processing system which include the tile generation unit, DMA block, hardware buttons and a timer. This timer was eventually not used in the implementation. The processing system is able to output different clocks of different speeds which can be configured. Finally, certain hardware on the Zedboard can be accessed through here as well. This is all configured through the re-customize IP window.

5.2.2 Fast Fourier Transform v9.1

The Fast Fourier Transform (FFT) block is an IP block that is provided by Xilinx. There are many ways that the block could be used and had to be configured properly to be used in Piano Tiles' implementation. The FFT block is used to analyze subsections of the input audio data to output the input's frequency-domain representation. This information is very useful as the fundamental frequency can be found. The main design choice made was that it would be a 2048-point FFT. Implications of a 2048-point FFT is that it will take in 2048 input points and produce 2048 output points. The input will be 2048 chunks of data that is composed of a real and imaginary part. Another design choice was the input and output data width to be 16-bits wide which means that an input/output point would be composed of a 16-bit real-part and 16-bit imaginary-part. The ordering of the information in each data point is important to realize as well; natural ordering was chosen which means that the 32 most significant bits are the real part, and the 32 least significant bits are the imaginary part. This is, again, all configured through the re-customize IP window.

5.2.3 AXI DMA v7.1

This hardware IP block gives the system the ability to directly transfer data to/from DDR to certain hardware blocks. The AXI DMA block has been configured to ensure that the data transfer mode is simple transfer. This type of transfer is to just simply stream data to a secondary AXI peripheral. This is essential for the FFT block as a huge amount of data is to be fed into it. Another positive of using simple transfer mode is that there are built in functions by Xilinx that can allow arrays to be transferred by just passing through the array and its length. This is very useful as the audio data is also in an array that is extracted using the blob.S process. Understanding how the DMA outputs data is essential to connect to the FFT block. The AXI protocol takes care of data transfer with the idea of hand-shaking.

5.2.4 AXI Interconnect v2.1

One AXI interconnect block is used by the Zynq processing system to control five AXI peripherals which include the AXI GPIO block, tile generator, piano tiles I/O, AXI DMA block and an AXI timer which is not used anymore. The other AXI interconnect is to allow the AXI DMA to communicate with the Zynq processing system. Since data only needs to be read from DDR and not written to, the M_AXI_MM2S port simply reads requested data from DDR.

5.2.5 Other Blocks

Section 3.3 describes the use of the following hardware blocks in great enough detail for understanding: processor system reset, AXI GPIO, clock wizard, concat and constant.

5.3 Borrowed Blocks

5.3.1 Zed Audio Control

A zed audio controller block is used to handle input audio data. Input audio is retrieved from DRAM and is fed into the output which consists of SDATA_O and output also bclk and LRCLK corresponding to the data block being ready, and control of stereo audio through LRCLK. The audio control is used to output songs to the auxiliary connector (headphones), and it accepts samples of data, encoded in 24 bits to process and output through its SDATA_O signal. In our program the entire audio data is fitted as binary in DRAM and the mono-audio is fed in 16-bit chunks concatenated together to form a song.

5.3.2 VGA Controller

The VGA controller is used to output RGB values, horizontal and vertical sync terms, and pixel location to an address that will be displayed on the VGA display. The inputs consist of a fifo stream with independent block and pixel clocks and resets, and can accept switch buffer signals to permit double buffering (buffer switching), the M_AXI output depicts the address to write the values to. The blocks is configured with image buffer addresses, horizontal and vertical polarity and pulse, the pixel write clock frequency and resolution of the display. Once configured, the input stream of the vga controller can properly display the input feed to the display at the correct rate, pixel, and colour.

6. Design Tree

An accompanying archive contains all the source code and related files utilized to construct the aforementioned modules. This archive includes a brief description of the contents of folders and the hierarchy of HDL files. Included in our submission is our design directory. Consisting of our code including both core's logic.

piano_tiles

- **Hardware**
  - **ip_repo**
    - **My_Debouncer_Core: debounce logic**
    - **vga_controller_ip: vga controlling logic**
    - **tile_generator_v4: tile generating logic**
    - **pianotiles_io: 32-bit memory device**
    - **Zed_audio_ctrl: audio controller logic**
  - **piano_tiles**
    - **piano_tiles_hardware.xsa: bitstream exported**
    - **piano_tiles_wrapper.xsa: bitstream exported**
- **Software**
  - **piano_tiles**
    - **src**
      - **main.c: contains main function for core0 to perform gameplay and visuals**
      - **menus: contains the sound**
        - ***.data**
      - **sources**
        - **blob.S: assembler code to load binary song data onto BRAM**
        - **Display.c: display write out driver**
        - **Dma_interface.c: direct memory access drivers**
        - **Fft_io.c: fast fourier transform drivers**
        - **Interrupts.c interrupt functionality just begun**
        - **menu.c: helper function for main.c to render images**
        - **tiles.c: play some EST Gee.**
        - **audio_binaries: directory containing the audio data as binary**
          - ***.bin**
  - **Piano_tiles_core1**
    - **src**
      - **Piano_tiles.c: contains main function for core1 to perform audio playback**
      - **sources**
        - **audio.c: audio driver for playback**
        - **blob.S: assembler code to load binary song data onto BRAM**

            - **Ip_functions.c: additional functions for writing and storing song data**
            - **audio_binaries: directory containing the audio data as binary**
                - ***.bin**
- **Piano_tiles_hardware**
    - **hw: the driver files for the hardware**
    - **Ps7-corteax9-0: core 2 hardware settings**
    - **ps7-cortex a9-1: core 1 hardware settings**

7. References

[1] “Get piano tiles 2030,” Microsoft Store, https://www.microsoft.com/en-us/p/piano-tiles-2030/9nn36v2t9h5j?activetab=pivot%3Aoverviewtab

[2] “Fast Fourier Transformation FFT - Basics,” NTi AUDIO, https://www.nti-audio.com/en/support/know-how/fast-fourier-transform-fft

[3] N. Ravanshad and H. Rezaee-Dehsorkh, “Ch. 12: Level-crossing sampling: principles, circuits, and processing for healthcare applications,” in *Compressive Sensing in Healthcare*

[4] “L3: Debouncing,” Physical Computing, https://makeabilitylab.github.io/physcomp/arduino/debouncing.html

[5] “Intellectual property,” AMD, https://www.xilinx.com/products/intellectual-property.html