DIGITAL AUDIO WORKSTATION

Final Project Individual Report

ENSC 452 Spring 2021
Group 7
April 16th, 2021

Nitish Mallavarapu
nmallava@sfu.ca

Contents

Introduction

In this report, I describe the role that I played as an individual on our final project where we designed both hardware and software components of an application that was to be run on the Zedboard. Kiel suggested to recreate a version of the 1992 Super Nintendo Entertainment System game Mario Paint Composer. This game is also formally known as a digital audio workstation (DAW). A DAW is an electronic device combined with application software used for editing and producing audio files. We committed to the idea for several reasons:

- It included audio and video components, which could be handled by separate cores.
- It had a high number of relatively small features, allowing for it to be more easily broken down.
- We judged it as sufficiently difficult for a course of this level.
- It was an idea that our group felt would be relatively enjoyable to design and program.

Modern DAWs are used for music, songs, speech, radio, television, soundtracks, podcasts, sound effects and nearly any other situation where complex recorded audio is needed. However, due to time constraints and skill level, we narrowed our design requirements to just the basics; Be able to place notes on a musical staff, have various instruments, playback the created song, and have a GUI to interact with all the necessary features. We split the work up into 6 milestones for each group member, with me focused on video components and Kiel focused on audio components.

Review of Technical Literature

AXI Timer v2.0 LogiCORE IP Product Guide

This IP core product guide provides a detailed overview of the underlying architecture of the AXI Timer. The relevant information I needed was how the timer uses the load value to count and produce an interrupt. Since the source is created by Xilinx, a long-standing company that has plenty of experience with FPGA design, I would say it is an objective piece of literature that provides accurate and high-quality information. Although the last revision to the document was October 2016, its still current information since a timer does not need any new features to it except for integration with new devices. I was able to use the information provided about the counter defaults and the masks that could be used to change these defaults to fix a problem with my software that resulted in the timer not generating interrupts at correct intervals.

Piazza and Xilinx forums

The piazza and Xilinx forums were the place to go when trying to get solutions for very specific problems that were not easy to understand. Since it is a student community, the experience level was quite low so further experimenting and discussions were still needed when trying to solve a problem. There was also a little bit of bias, so it was not a perfectly objective source since everyone has their own way of doing things. I was able to use quite a few of the already posted questions to solve problems I was having such as setting up the 2nd ARM processing core, uploading files to memory, and fixing Vitis compile issues. It also led me to a very useful source that I used which is described next.

FPGA Prototyping by VHDL Examples Textbook

The third piece of technical literature that I used was this textbook, suggested by a peer on piazza, written by Pong P. Chu. Dr. Chu has a PhD in computer engineering and has written 7 other texts. His book explains effective derivation of hardware and provides examples, as well as templates, that are general and can easily be integrated to construct large, complex systems. The specific portion on VGA controllers is what was most relevant to our project. He is very qualified in his area of expertise, so his materials are objective, but this textbook was published in 2008 so some of the material was outdated and newer ways of doing things in VHDL are available. I was able to use chapter 12 and chapter 13 to better understand how graphics worked using a VGA controller and how to print strings out to the display. This aided with the construction of our application GUI.

What I did

As mentioned before, the project was partitioned into audio and video components. Kiel focused on audio, while I focused on video, but I also ended up doing most of the PL design in Vivado. I followed a bottom-up design methodology, starting with base graphics, moving into note placement, then note playing, and finally additional control features. When we brought up the idea of using GitHub as a method of source control and sharing work between myself and Kiel, Zavier advised that just creating an archive from within the design tools and transferring that would be a much easier approach, so we followed the latter. A more detailed discussion of my contributions to the project is provided in the following milestone sections.

Milestone 1

Create the static background for the display and allow user to specify "commands" using terminal keyboard. Display "actions" to terminal.

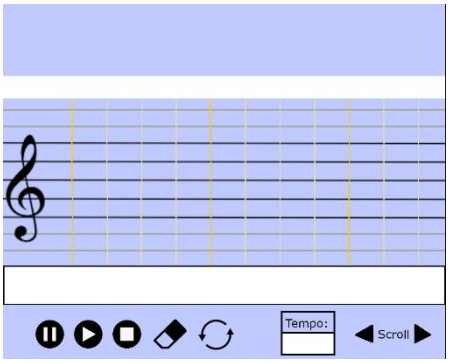


Figure 1: Static background v1

I decided to use GIMP to create the static background since it was free to use and lightweight. I combined a bunch of graphics found from google into what is seen in Figure 1. The original background was at a 480p resolution since I was still using the settings from the VGA tutorial. I also continued to use the same method of loading the image onto DDR and using its pointer to `memcpy()` to the image buffer address of the VGA core. I then programmed a simple while loop with `xil_printf()` and `scanf()` to simulate commands being run by the user. For this milestone, one thing I learned about Vitis was that it has a built-in serial terminal, so I stopped switching back and forth between Tera Term and Vitis. The hardware remained the same as the VGA tutorial. There was no need for testing this milestone as it was a simple goal and one can visually see if its working or not.

Milestone 2

Be able use the push buttons to highlight and "select" GUI buttons. Selected buttons will be indicated in the terminal menu (no submenus or actions will be currently generated). Use push button interrupts.

For this milestone, I needed to do a lot of research on how to configure the ZYNQ7 PS to enable the GIC, then use the Xilinx provided drivers to connect interrupt handlers to it. I also had to add the AXI GPIO IP core to the PL design and connect it to the buttons, although this was removed in a subsequent milestone as the button interfacing changed. To test if interrupts were working, I created a simple handler that would read from the memory mapped AXI GPIO peripheral and print out the value of the button that was pressed to the serial terminal. Once I accomplished that, I decided that I would show a button on the GUI being selected by drawing a red box around it. The biggest hurdle I had for this milestone was figuring out how to move the "selected" box around the screen. With the lack of documentation provided with the VGA core given to us, combined with a general lack of expertise with the Zedboard at the time, I could not figure out an elegant way to do this. The worst possible solution I came up with was to create a new image for each item on the GUI with a red box around it, load each image onto memory, and keep cycling through them based on which button was pressed. For the sake of meeting the submission deadline, I stuck to this method. Since the number of items to interact with on the GUI was going to increase for the next milestone, (136 notes that can be placed on a given page) this would not be feasible anymore and I already knew I would have to change it. I then modified the interrupt handler to write to the serial terminal which button was currently selected.

Something I learned about the Vitis tool during this milestone was how to use the memory monitor. Although it did not help much with this milestone, it came in handy in later milestones such as when I was dealing with reading and writing data between cores.

Milestone 3

Move the cursor around the staff and place "objects" or notes. All notes are of the same duration.

Realizing I needed a better implementation to draw graphics to the screen from the previous milestone, I decided to try and find a better VGA controller IP core that provided some documentation. I came across a few YouTube tutorials (links are provided in the group documentation) that taught me how to implement a VGA controller by putting together the following Xilinx provided IP cores: AXI Video Direct Memory Access, AXI4-Stream to Video Out, and the Video Timing Controller. It also included source code for using the drivers and some graphics processing functions that had an open-source license. I followed the tutorials to add and wire up all the hardware to the PL design in Vivado, then imported all the C files into the Vitis project. I carefully examined the code which gave me the knowledge that I needed to understand how drawing individual pixels worked using software.

```
imageWidth = 1920;
imageHeight = 1080;
numColor = 3;

i = imread('bg.bmp');
fileID = fopen('bg.txt', 'w');
for r = 1:imageHeight
    for c = 1:imageWidth
        for m = 1:numColor
            fprintf(fileID, "%d,", i(r, c, m));
        end
    end
end

fclose(fileID);
```

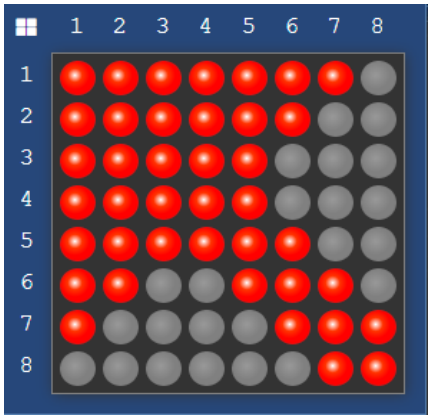
Figure 2: MATLAB script to convert images to integer RGB values.

One roadblock that I ran into was being able to store and display images. The method of saving GIMP files as .data, loading them to memory, and using a pointer did not work with the new hardware I was using. So instead, I created a MATLAB script that could convert image files into an integer array containing RGB values. I then stored this array in a .h file and referenced the array in the graphics drawing functions.

The biggest obstacle that I had to overcome for this milestone changing the way navigation around the GUI was implemented. The original idea was using a red selection box that would hover over a specific control. The logic was going to become very confusing and very long since you would need an if statement deciding where to go next for all the controls, which includes every grid position on the musical staff. I decided to take the name “cursor” at its face value and implement a real cursor that could scroll across the screen. My initial plan was to do a software implementation but remembering the requirements of at least 1 custom hardware module, I decided to design the cursor using Verilog. The design requirements were:

- Keep track of the mouse x-y coordinates.
- Use the center push button to click and the other push buttons to control which direction the mouse moved in
- Generate 2 different interrupts based on “mouse moving” or “mouse clicked”.
- Prevent the mouse from going past the borders of the screen.

A detailed description of the GUI cursor architecture is provided in the group report. I had to do extensive research on the AXI-4 Lite interface as I had never worked with anything like it before and learned how to use Vivado to create a custom AXI-4 Lite interfaced IP core. I had to also create additional debouncing hardware since the push buttons were being used to control the mouse. The debouncing sub-module also needed 2 variations to it since I needed the center button to only generate a single pulse and the other 4 buttons to generate a continuous high signal when held down. I tested the debounce hardware and the complete mouse hardware individually by using the built-in simulator in Vivado and forcing values to simulate button presses. Once the IP core was debugged, I integrated it with the PL design, wired up the push buttons and proceeded to create the software driver. After initializing the mouse in software, I created and tested the interrupt handlers by outputting the current position of the mouse to the serial terminal whenever the center button was pressed.




```
Bitmap: 0xC0E1733F1F1F3F7F
```

Figure 3: bitmap creator https://xantorohara.github.io/led-matrix-editor

The next problem I had was being able to make the mouse appear on the screen. The graphics software that I borrowed had 2 drawing functions, one that could draw rectangular images from pixel data, and one that could print characters from a bitmap. I originally tried creating a mouse sprite and using that to make the mouse appear on the screen, and then using the “mouse move” interrupt handler to constantly update the image. The problem with this was that I did not know how to get transparency to work, so there was a black box surrounding the mouse that followed it around since only rectangular images could be drawn. A creative solution that I came up with was to create a new item in the bitmap that resembled a mouse using a web app I found online, and then use the character printing function to draw the mouse to the screen. This automatically handled transparency using the bitmap data. After completely changing the design from the 2nd milestone, I finally started working on the actual goal of this milestone!

The note placing was done all in software, so no additional changes to the hardware design were needed for the rest of this milestone. I started this module's design by creating a struct for the note object and then a 3D array of these note objects. My thought process was that we would want to access an individual note by providing the page number, vertical staff position, and horizontal staff position as indices to the 3D array to access an individual note. During the final project demonstration Q&A, Dr. Shannon explained to us that we could have just stored the notes the way music is naturally played, as 2D arrays in contiguous blocks of memory. In hindsight, this would have been a more efficient and optimized way of doing things, but one learns from their mistakes and I will be more vigilant when it comes to creating data structures in the future. The next step was to be able to translate a mouse click to a grid position on the musical staff. I created a few helper functions to accomplish this. Once the conversion was done, I used the staff grid values to create a note and add it to the 3D, which for simplicity was hardcoded to only have 1 page. Now it was time for note graphics where I used the same method as graphics for the mouse. I added a character to the font bitmap and used the print character function to display a note object on the musical staff. I then used the debugger mode in Vitis to make sure that when a note was placed on the musical staff, that the appropriate grid position in the data structure contained the corresponding note object.

Milestone 4

Be able to play a created song; Display and shift a vertical bar that highlights which notes are being played for the current beat. Be able to play multiple concurrent notes.

I started this milestone with the scroller object. Since there were many additional features that still needed to be added, such as tempo, seeking, and play/pause functionality, I had to keep the big picture in mind. I designed a parametrized scroller object that considered all these possible features. Once the object was ready to use, I created a pink vertical bar to represent what beat of the song you were at. Then I needed some way to constantly update the bar at given intervals based on the tempo. I initially tried to use the xtime library provided by Xilinx to do a software implementation of keeping track of the time, but then remembered Exercise 2 from "The Zynq Book Tutorials" which used the AXI timer IP core to control scrolling LED's. I realized this was somewhat the same concept of updating data in set intervals so decided to go the AXI timer route instead as it would be a lot more efficient and logically made more sense. I added the core to the hardware design, wired it up, and then connected the timer interrupt handler to the GIC. I tested the interrupt by trying to print "hello world" to the serial monitor every 1 second, which meant that I needed to load the clock frequency into the timer, but it was completely off. It was printing the string much slower than 1/s. Only after struggling for a while assuming that something was wrong with my code did I finally decide to try and read the AXI timer documentation, where I found out that the AXI timer counts upwards by default! I then changed the initialization of the clock to include the mask for the down count option and the statements started printing at the right rate. From this point onwards, whenever I used a new Xilinx IP core, I made sure to read the documentation first.

The second part of this milestone was updating the graphics for the scroll bar using the interrupt handler. The graphics library that we borrowed did not have any erase functionality since it was a Connect 6 game where the resetting the board happened by just redrawing the original board. This worked fine for the author since a reset of the board only occurred once every so often. When I tried using this same method it was very slow since the whole background would be redrawn (including static parts that did not actually need to be erased) every time the scroller updated which could be as fast as once every 0.25s. I created my own erase function that was based off the original authors drawing function because. My erase function would only redraw the user defined portion of the background given the parameters img_offset_v, img_offset_h, size_v, size_h, which are visually shown in Figure 2 below.

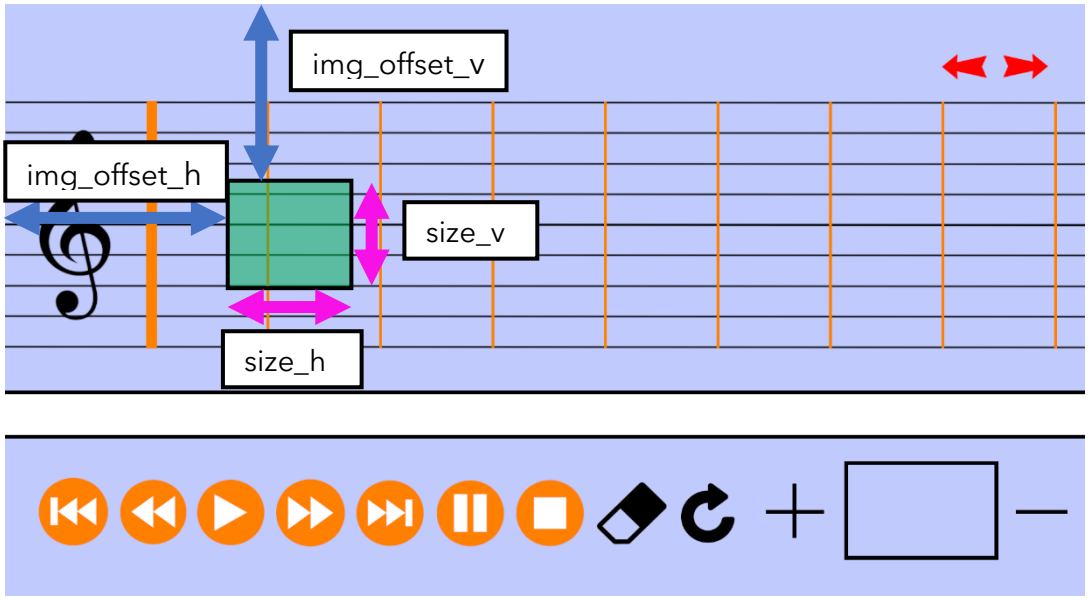


Figure 4: Parameters to the erase function. Green box will be set back to the original background.

Now instead of redrawing the whole background, only the portion of the background that the scroller covered would be redrawn, thus giving the illusion of the scroller being erased. This erase function also ended up coming in handy for later features such as updating the temp, updating the page number, and erasing notes. I then added the code that starts the timer whenever the play button is clicked and stops the timer whenever the pause button is clicked.

The final step was putting together both the scroller and note graphics and then playing the actual notes. All the components were already available, with only a few minor modifications being made to Kiel's note playing software; It was just a matter of getting it in the right drawing order and using Kiel's audio software to output the note to the audio controller. Figure 3 shows the order of the timer interrupt handler. Again, for simplicity, the max number of pages was kept to 1 for this milestone. The note playing and the graphics drawing were happening in the same core, which caused extra time to be padded in between each note being played due to the graphics processing. This is when we decided to implement the audio on a different core than the video but left it to milestone 6.

Figure 5: Order of operations within the timer interrupt function

Milestone 5

Implement multiple pages, be able to forward/rewind a song or restart the song. Update display accordingly and correctly play the song.

There were no hardware changes that were needed for this milestone. Changing the implementation to include multiple pages was as easy as increasing the `MAX_PAGES` macro. The hardest part of the scrolling feature was being able to implement the logic for multiple pages since there are a few corner cases for the drawing. I first brainstormed what these would be:

What should happen when…

- The scroller is already at the start and the user tries to rewind.
- The scroller is at the end of a page but not the end of the song and the user tries to fast forward.
- The scroller is at the beginning of a page but not the beginning of the song and the user tries to rewind.
- The scroller reaches the end of the song and the user tries to forward.

I decided to create a `updateScroller(int n_beats)` function that would cover all these cases. The parameter `n_beats` can either be a positive or negative value, that way when one of the 4 seeking buttons is pressed I just call the function with ±1 or ±4, which is then summed with the scrollers member variable `current_beat`. It can then work out whether the scroller needs to move to a new page or not, which is the last piece of information necessary to correctly draw everything to the screen. One important functionality that I wanted to have was that the page that the scroller was currently on and the page that the user wanted to edit did not have to be the same e.g., the scroller was paused at the first page but the user wants to edit the last page. To accomplish this, I used 2 separate page tracker variables, one for what page the scroller was currently on and one for the current page being edited. When the red arrows at the top right of the display are pressed, only the `edit_page` variable is changed, but when the scroller moves to a new page then both the `edit_page` and the `scroller.current_page` variables change. I also created a helper method `changePage()` whose sole purpose is to handle all the drawing when changing the page is needed. This helper method also handles the page number graphics at the top right of the screen. The complete algorithm flowchart for the `updateScroller` function is provided below.

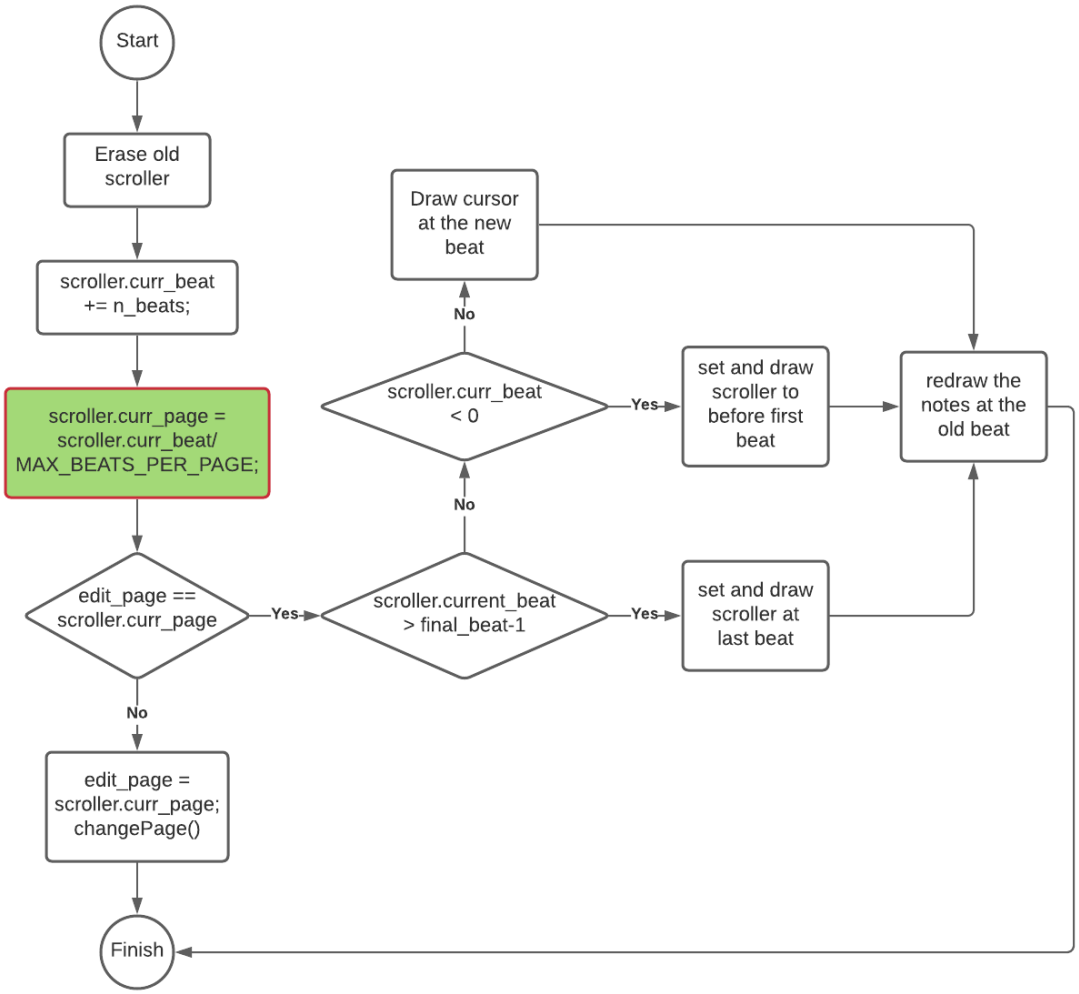


Figure 6: Algorithm flow chart for the seeking feature. Highlighted box is explained in paragraph below.

There are two important things to note about this algorithm.

1. In the highlighted box in Figure 4, a division is done between 2 integer values. I leveraged the fact that the remainder will be thrown away which causes the `scroller.curr_page` variable to change only when quotient changes even though the `scroller.curr_beat` variable changes.
2. There are no branches for what happens when the user tries to rewind past the initial beat or forward past the final beat AND a page change is required. This case is never possible as the max you can forward/rewind is 4 beats and there are 8 beats per page.

Milestone 6

Speed up and slow down song based on tempo setting set by the user.

My goal for this milestone was easy to accomplish since I created a parameterized design. All I had to do was code the plus and minus buttons to increase and decrease the tempo, which controls the speed at which the AXI timer generates interrupts. I also copied the code from the previous milestone to be able to display the tempo that was currently set.

I decided to help Kiel with 2 goals from his milestones. The first thing I did was the eraser feature to be able to delete specific notes. When a note is clicked, its boolean value `isUsed` is set to false, and the erase function from the graphics library is called on it.

The second thing I helped him with was setting up the other core to be able to output audio, that way the first core could focus on updating the graphics for the scrolling action. I followed Zavier's piazza post found here on how to do this. I then realized we needed some way to send the note data between the cores. I added two AXI BRAM controllers and a single Block Memory Generator to the PL design, wired it all up, exported the hardware to Vitis, and moved all the audio software to the second core. What I then did in the timer interrupt handler on the first core was create an integer array of length 17 to represent each note at a given beat. I looped through each note at the current beat and set the position of that element in the array to a 1 (is used) or 0 (is not used). I programmed this array to be written to memory by the one of the BRAM controllers and then read from memory by the second core, which uses the other BRAM controller. To control when the second core should wake up and read a new note, or when the first core can write a new note, I created a flag that can be polled by both the cores, which was an idea taken straight from the tutorial. While the second core has not received a new note yet, it constantly plays the old note that has already been received. One bug that arose from this was that when the pause or stop button was pressed, the old note would continue to play forever. I fixed this by also sending the `scroller.isPlaying` variable to the second core, and added some logic that decided if it should output anything to the audio controller or not. At this point, the timer interrupt handler, a critical component of the application, was fully complete; the complete algorithm flow chart is provided below.

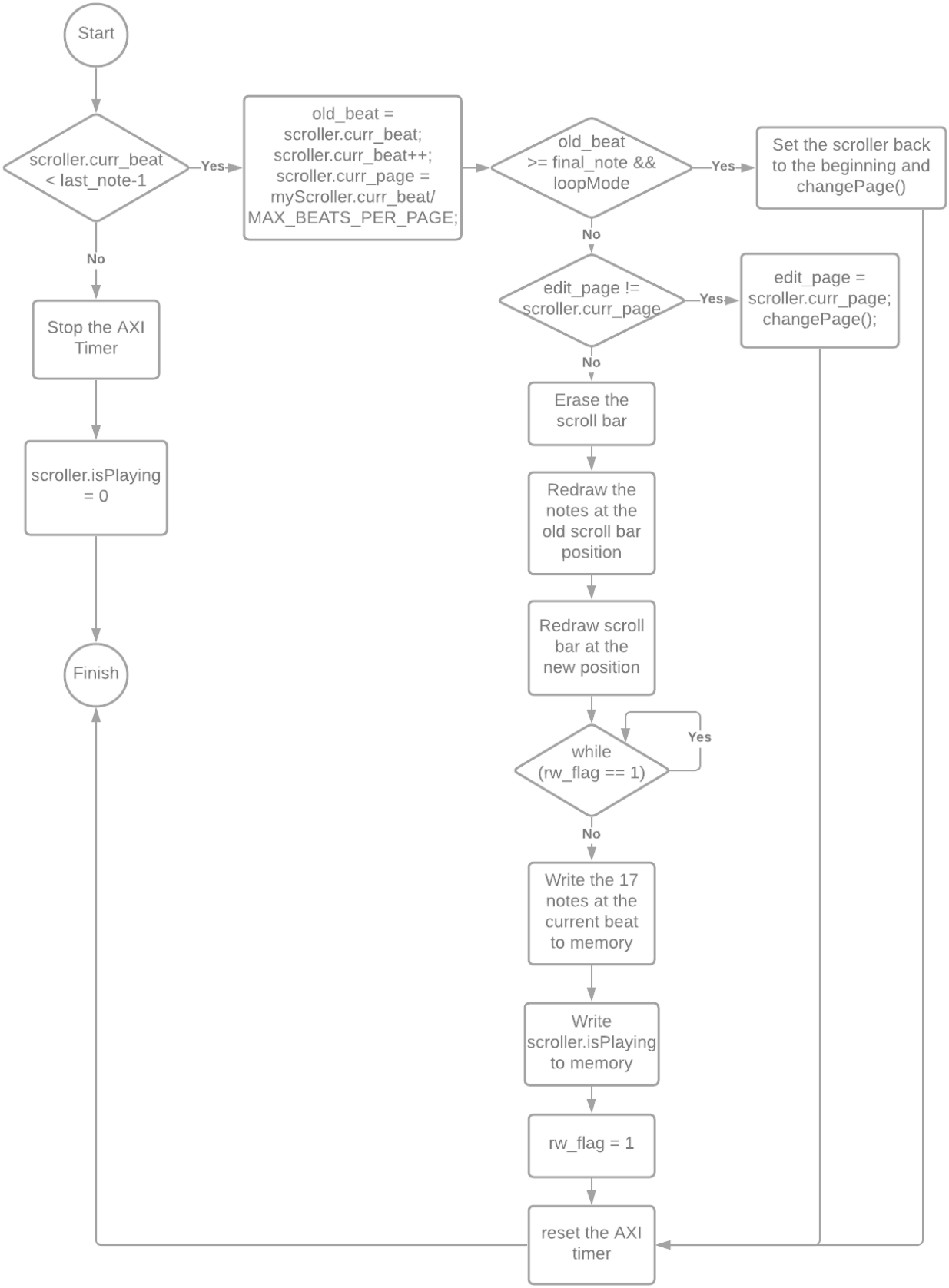


Figure 7: Timer interrupt handler algorithm flow chart

Community Contribution

Although I kept update to date with piazza posts, there were not many questions that I could answer, and those that I could already had answers. I did learn a lot from piazza though and I am glad it was there. I asked some questions when it came to graphics problems such as why I was getting screen tearing or how to draw individual sprites to the screen and made sure to update them with how I resolved the issue.

If there was one issue that I wish I had known the solution to from the start it would have been the makefile error with custom IP cores. I spent lots of time trying to find a solution and the error code that appears in Vitis is not very helpful at all. The forum post that I found online that worked for me is found here:

https://forums.xilinx.com/t5/Embedded-Development-Tools/Drivers-and-Makefiles-problems-in-Vitis-2020-2/td-p/1188742

I even had to save it as a bookmark on my web browser, because every time you update the hardware platform in Vitis the same makefile error reoccurs and must be fixed.

Feedback to Xilinx

I think the Xilinx tools were generally well designed. I remember how daunting Vivado looked when I first opened it but after a few tutorials from the "The Zynq Book" I realized its actually very easy to use and convenient. A lot of YouTube videos I watched were still using the Xilinx SDK and when I compared it to the Vitis IDE it simplified the design flow a lot. I really liked the fact that the whole hardware design flow in Vivado is laid out on the left in chronological order, making it easy to go back and forth between them. The block automation feature was also a nice to-have. The fact that the processing system in Vivado also gives you a visual diagram that you can click on to take you to certain settings was useful because it avoids the need to search through tabs to find what you are looking for. My least favourite part of tools was the fact that the latest version has some error with custom IP core, and you must fix it every time you need to update the hardware. I hope Vivado fixes this soon because it was a weird error to have, and hopefully future students do not struggle with it as well. If there was one feature that I wish the Vivado tool had, it would be turning the block design into a cleaner, printable system diagram. That would have helped with the group report or even for explaining things to others in general.

Course Feedback

Overall, I really enjoyed the class because it required pulling knowledge from a lot of previous courses and focused more on technical skills rather than theoretical knowledge. The project timeline worked perfectly and set reasonable expectations for us. The makeup weeks were also nice to have since when you are programming it can get quite frustrating and things do not always go the way you want them to. It is always nice to be able to have a second chance to show what you really know, and not lose marks even though you put the effort in. I liked the open lab concept and having modules due on a weekly basis. It not only gives us the freedom to structure our projects the way we see fit but also ensures we stay on track and focused instead of leaving it all to the last minute and submitting a low-quality product. I do not think that the course should be 5 credit hours if you still want people to enroll. The reason being that the requirements for graduation still stay the same, meaning everyone would just choose ENSC 450 as the constrained elective since it seems like less work being only 4 credits. The 1 extra credit would not do anything except discourage people from enrolling! The only other piece of constructive criticism I would have is to maybe try and connect the lectures to the Zedboard a little bit more. Other than that, I am glad I chose to take this course, and it was a very rewarding experience that has provided me with lots of resume enhancing skills. I will definitely be recommending ENSC 452 over ENSC 450 to my peers.