

Christopher Gutierrez

CECS 551 - Hardware/Software Codesign

Spring 2019

Final Project Report

Introduction

For this project, I will be using the Zybo Zynq-7000 SoC Development board for my embedded system. The initial plan for my system was to use the microSD and Audio Codec IPs on the Programmable Logic (PL) part of the board to create a rudimentary “beat maker.” On startup, the system would load sound files (mp3, wav, etc) from the microSD card into main DDR3 memory (also on PL). We would also use the dip switches on the board to control which track gets played. Multiple enabled switches would mean more than one track would play at a time, creating a “beat.” The push buttons would then be used to record, playback, and store the new track back into the microSD card. Due to time restraints, developing an mp3 or wav parser would not be possible.

The new plan for the system is to prerecord sounds using the codec, store them in the microSD card, and then reprogram the system’s top-level software application to load the sounds into memory on start-up. This way no parser would be necessary. Different dip switch combinations would store and playback different sounds. This is the current functionality of my system.

This project uses the Digilent Zybo DMA Audio demo available on their website as a base.

Operation

The first step in building my embedded system was to get the DMA Audio Demo working on Vivado 2018. To do so, one first opens the project in Vivado 2018. Before you move onto the SDK, you must upgrade the outdated IPs (the demo was written for Vivado 2016). Once that is done, you open the constraints file that was included with the demo. On lines 44 and 45, ports “iic\_scl\_io” and “iic\_sda\_io” are misspelled. The “iic” must be changed to uppercase in order for the synthesis to work. Once those issues are resolved, you can move on to adding the IP for the dip switches as outlined in lab 2 (AXI GPIO IP). After that, you synthesize the design and generate the bitstream. Once that is out of the way, you can export the hardware and launch the SDK.

The next step in the SDK is to create a Board Support Package (BSP) based on the exported hardware. Then, you import the DMA application project that was provided by the demo project. Make sure to resolve any file inclusion issues before continuing. Once you no longer have any errors, you can connect the board and test the demo to make sure it is working properly. Lastly, make changes in the following files that correspond to the changes in Figures 1 to 5:

In the application project, we edit only 4 of the files. Those four files can be found on my [GitHub repository](https://github.com/gutierc2/CECS-561-Audio-Project-Files). Replace the corresponding files in the project.

Since we are going to have more recordings, we’re going to need to increase the heap size. You can do so by right clicking the application project and clicking on “generate linker script”. Change the heap value to 40 MB.

Once the changes have been made, you can once again connect the board and run the application on the project. The functionality is the same as the demo, with the exception that you can have multiple recordings by enabling different combinations of switches. Additionally, the tracks will be saved onto the microSD card. The next time you start the application, the tracks will be automatically loaded into memory for playback.

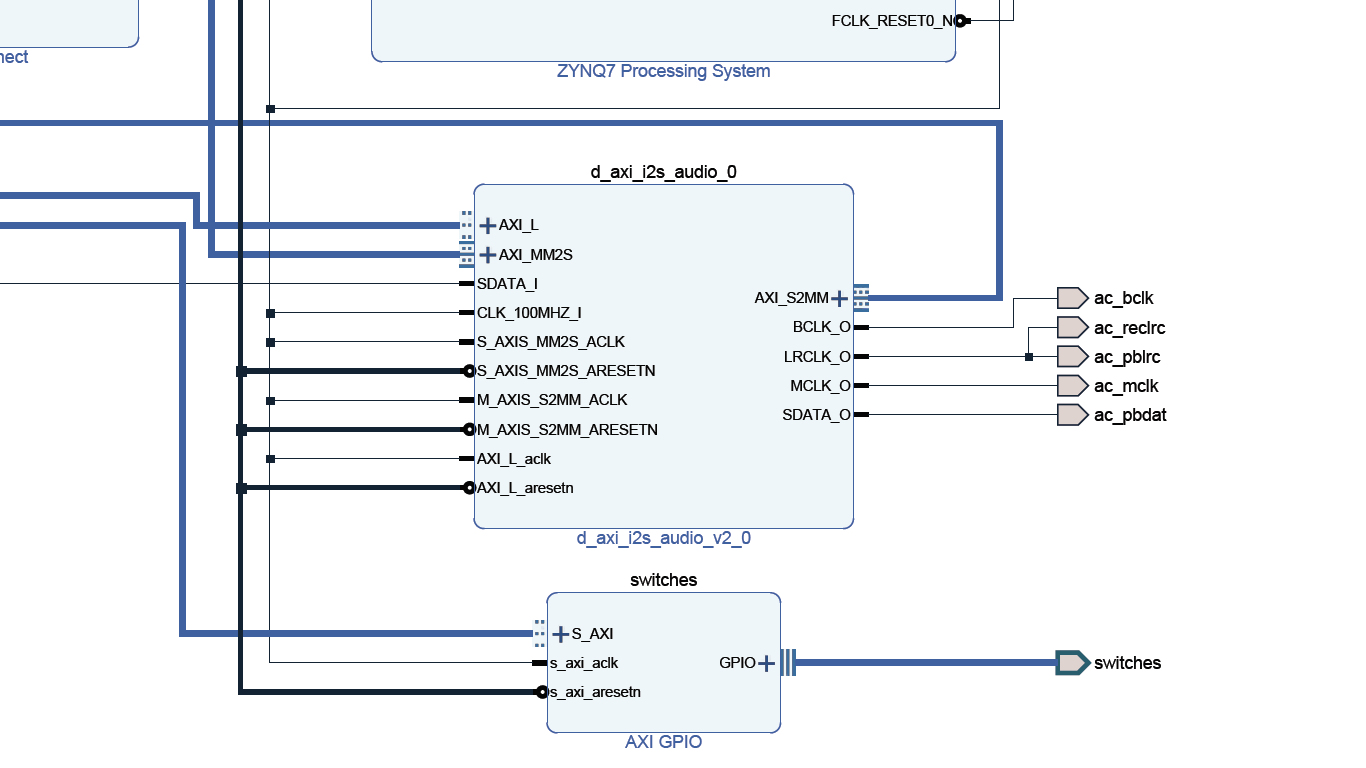
**For a more detailed explanation on how to build this project, see my lab manual (also on my repo).**

**For a video demo of the project, please visit** [**https://youtu.be/\_57vpqX5fZY**](https://youtu.be/_57vpqX5fZY)

Theory

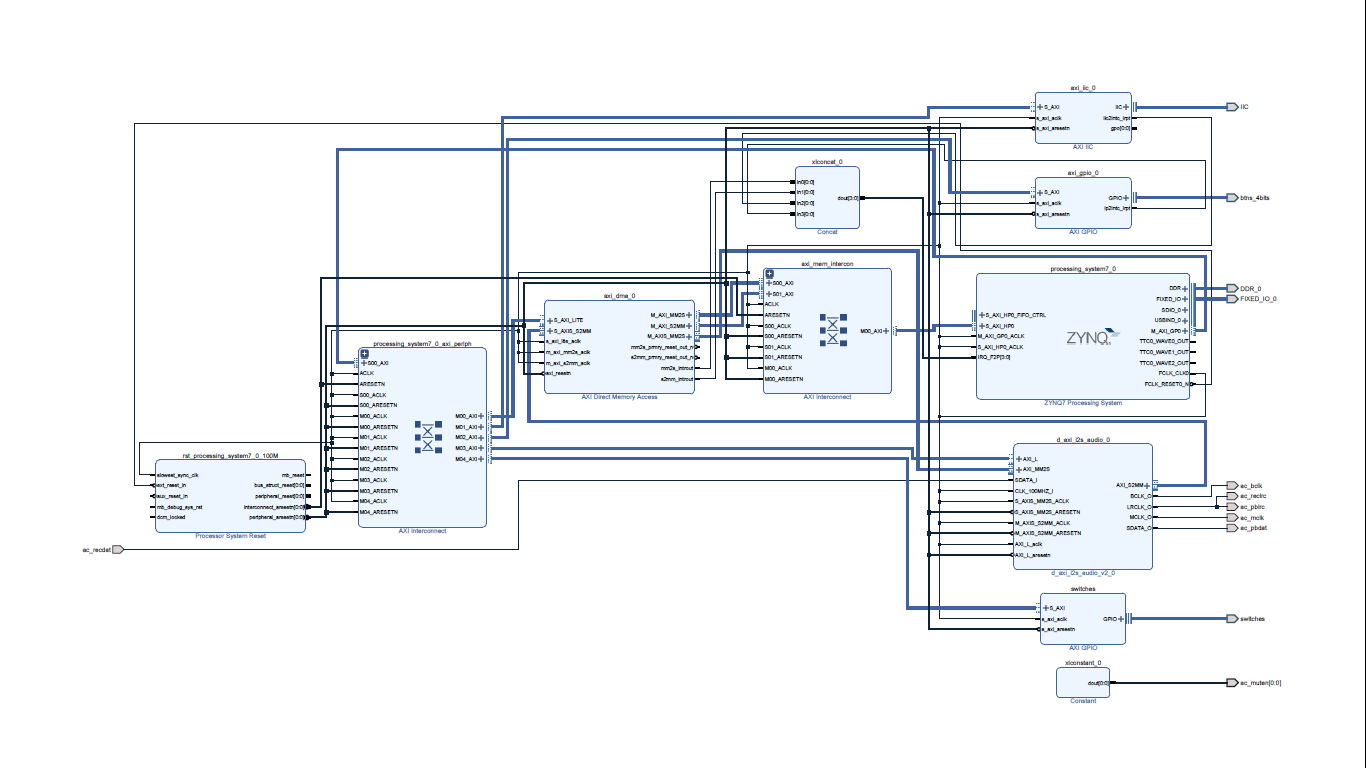
The main new hardware component I am using (that we have not used in class) is the Audio Codec as well as the microSD slot on the Zybo board. The microSD slot is enabled by default on the board’s Processing System (PS). As for the Audio Codec, I am using the IP provided by the demo (see Figure 6 below). Three of the major signals on this block are the AXI\_S2MM, AXI\_MM2S, and AXI\_L signals. The first two take care of streaming data to and from the codec and main DDR memory. AXI\_L are AXI Interconnect “handshake” signals.

Figure 6: Audio codec block on block diagram



On the software side, as we’ve already seen in the Operation section, we use the dynamic memory allocation provided by the C standard library to allocate enough space on memory for each recording. We also use the ‘xilffs’ (Xilinx Fat File System) library which contains functions necessary to make use of the microSD card.

Hardware Source Code

Figure 7 below shows the complete block diagram for my design

The design is nearly identical to the one for Digilent’s audio demo with the exception of the added switches IP on the bottom right of the diagram.

Software Source Code

The software is largely the same as in the demo project, except for the changes we made to the four files as outlined in the Operation section of this report. As previously mentioned, the four files we modified (‘audio.h’, ‘audio.c’, ‘demo.h’, ‘demo.c’) can be viewed and/or downloaded from my [GitHub repository](https://github.com/gutierc2/CECS-561-Audio-Project-Files).

Conclusion

The first big hurdle when working on this project was getting the demo to work. Combing through several forums and doing my own troubleshooting eventually resulted in success. The second hurdle came in the form of attempting to understand how the audio demo actually works. Digilent provides us with the functionality, but little to no documentation related to the audio drivers included in the demo. Again, through various forums and manual troubleshooting I gradually gained a modest understanding of the codec and its drivers. The last big hurdle was figuring out how to allocate space in memory. What offset could I add to the base memory address such that I wasn’t accessing information I shouldn’t be accessing? Once I realized I could use dynamic memory allocation for this issue, the rest was a matter of implementing the logic. An additional challenge after this was finding information on how to mount and communicate with microSD card. Once I figured out how to enable the Fat File System library in the project’s board support package, I had to figure out what the functions were to mount the microSD card as well as open, read, write, and close a file. There are several online resources available to help complete this task. Future work will include experimentation on how to get sounds to play on top of each other and storing the result into the microSD card for future playback.