Course Name: Embedded Systems 2

Course Number and Section: **14:332:xxx:xx**

Experiment: Final

Instructor: Luis Garrido

Date Performed: 12/19/23

Date Submitted: 12/19/23

Submitted by: Atharva Pandhare 203003207

GitHub Link: https://github.com/Embedded-Systems-2-Fall-2022/ee493-final-project-AtharvaPan265

Electrical and Computer Engineering Department

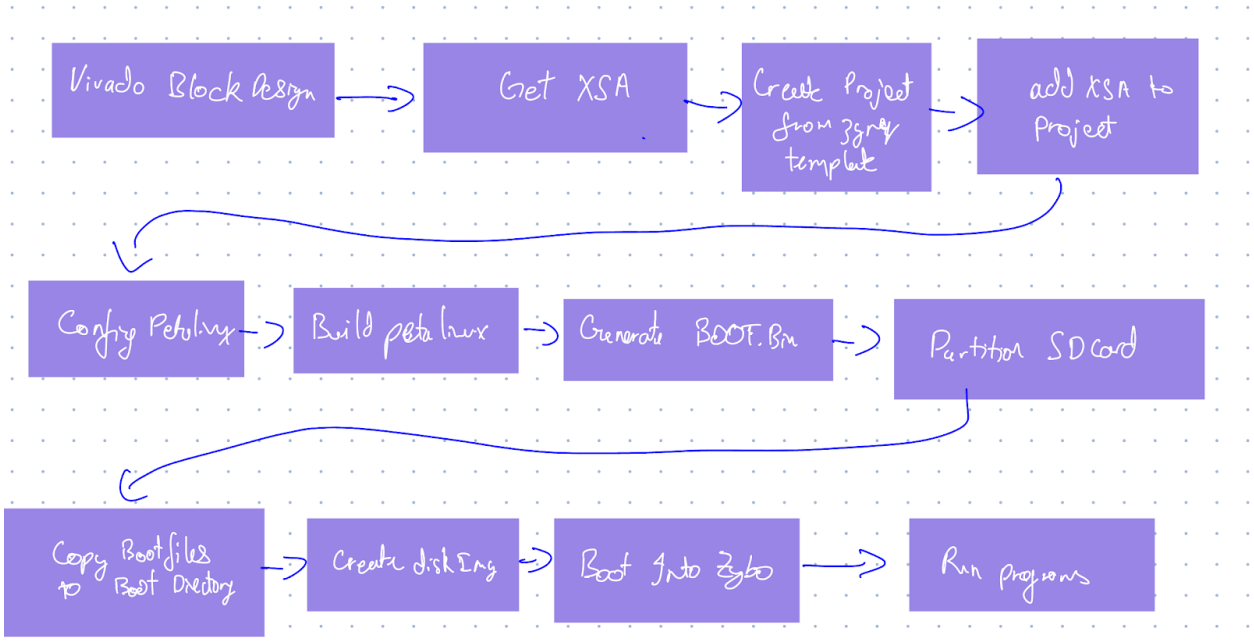
School of Engineering

Rutgers University, Piscataway, NJ 08854

1. Purpose

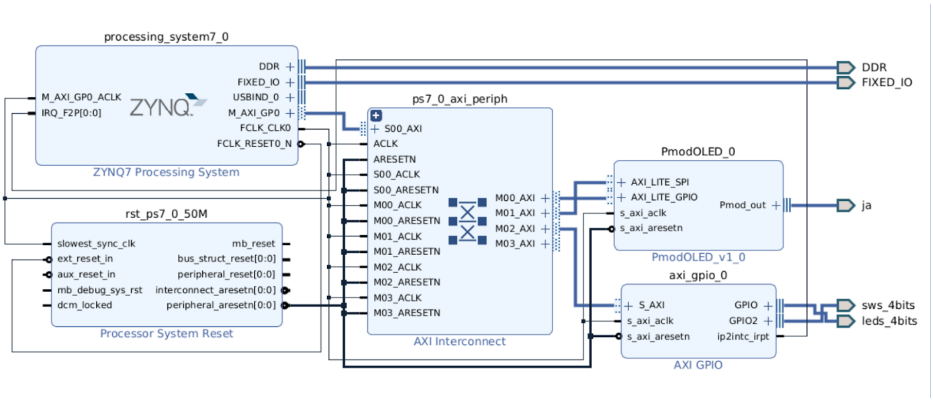
In my final Project, I decided to tackle a Petalinux-based project. Initially, I planned on using PMODs, but as I outlined in the report, there were many difficulties in getting them to work. Through the project, I learned a lot about the ins and outs of the beta Linux build process and how various components of Linux work. I tried to set up a file system and SSH as a working environment for Python with any necessary packages. Using these tools, I created a script that enables a CLI interface with ChatGPT. As a little gimmick, the Zybo board's LEDs also turn on when the servers respond.

2. Theory of operation

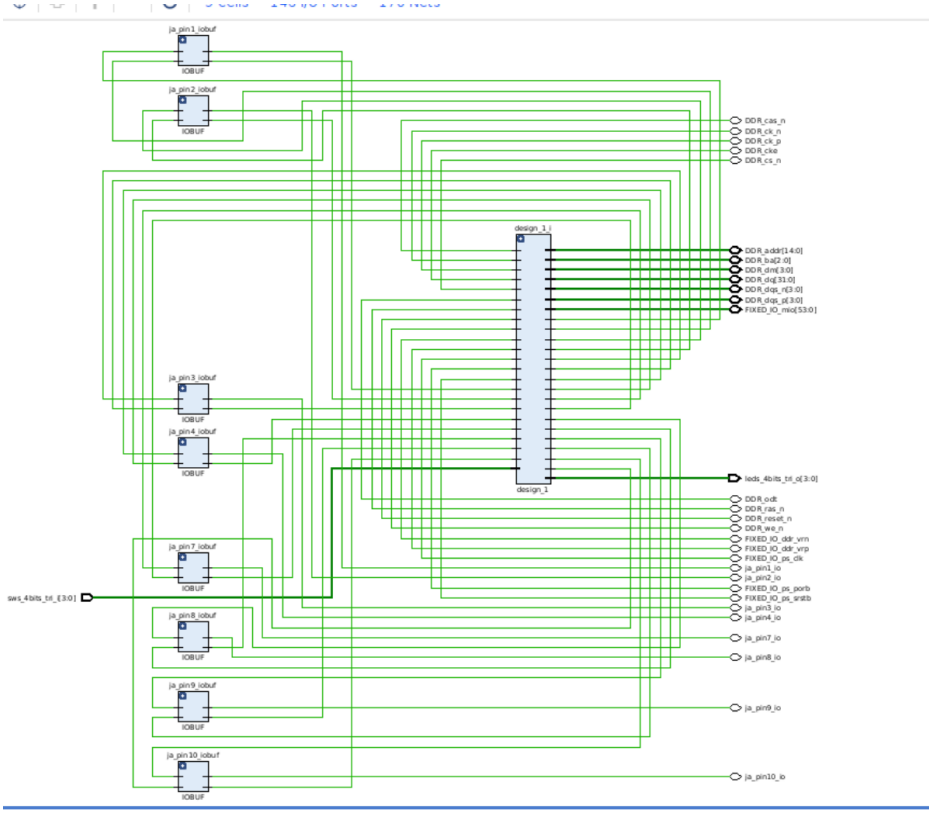


3. Vivado

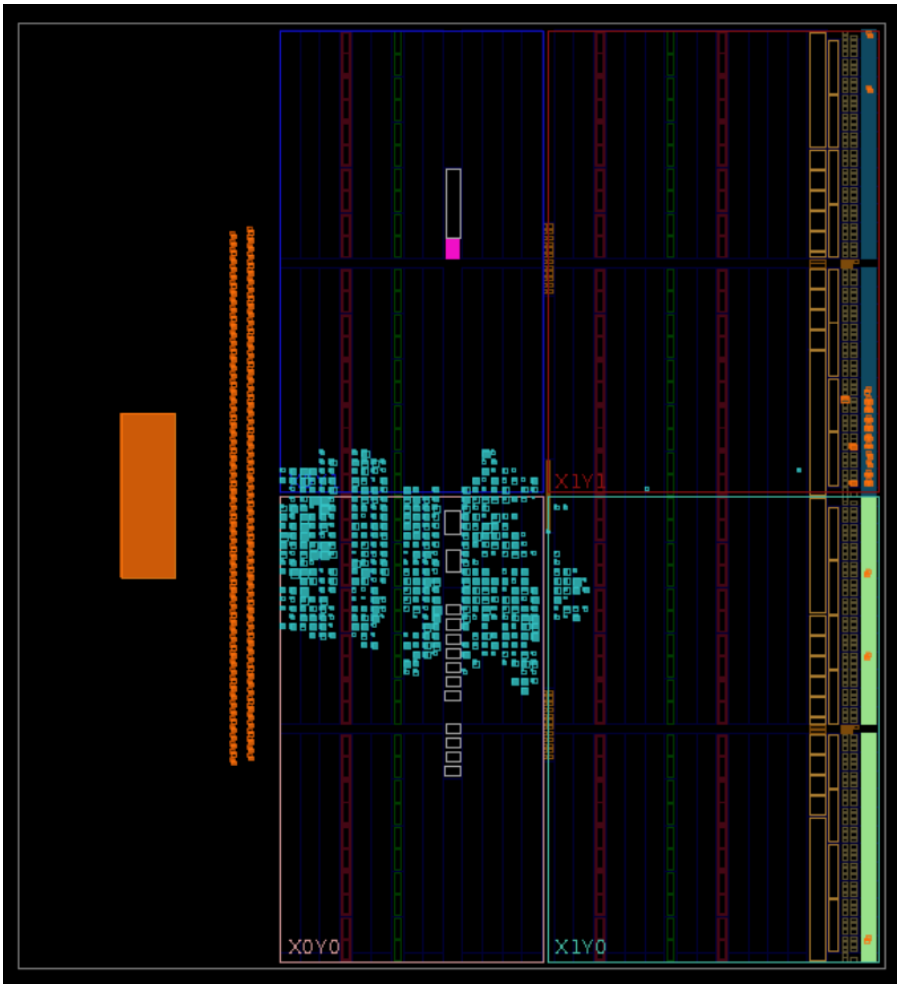
    a. Block Design



b. Elaboration Schematic



c. Device Layout



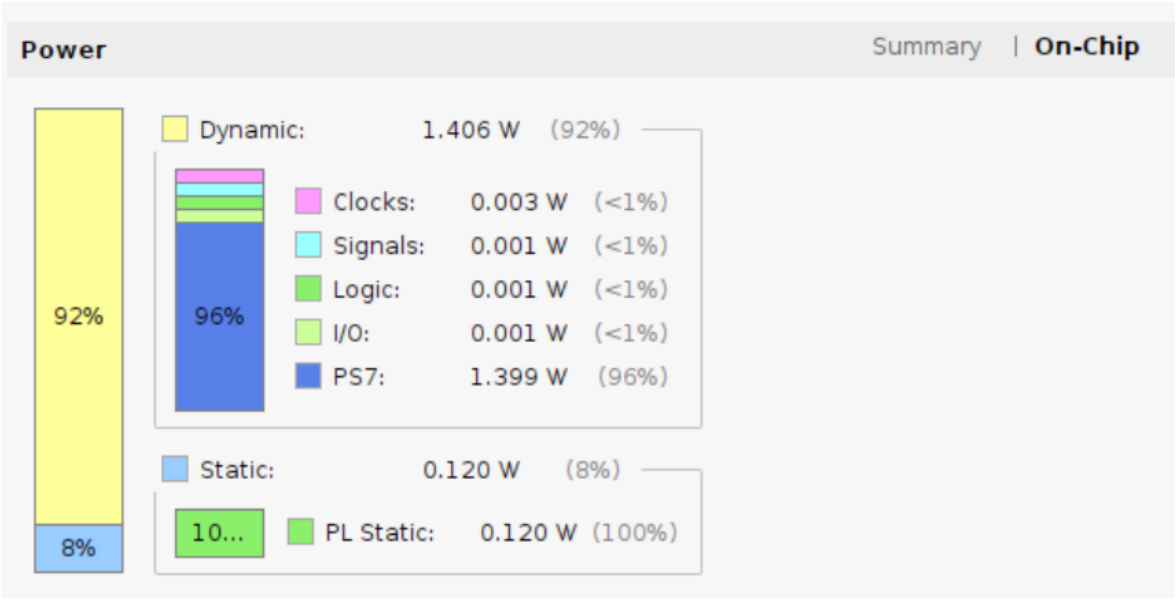
d. Post_Synthesis Utilization

Utilization **Post-Synthesis** | Post-Implementation

Graph | **Table**

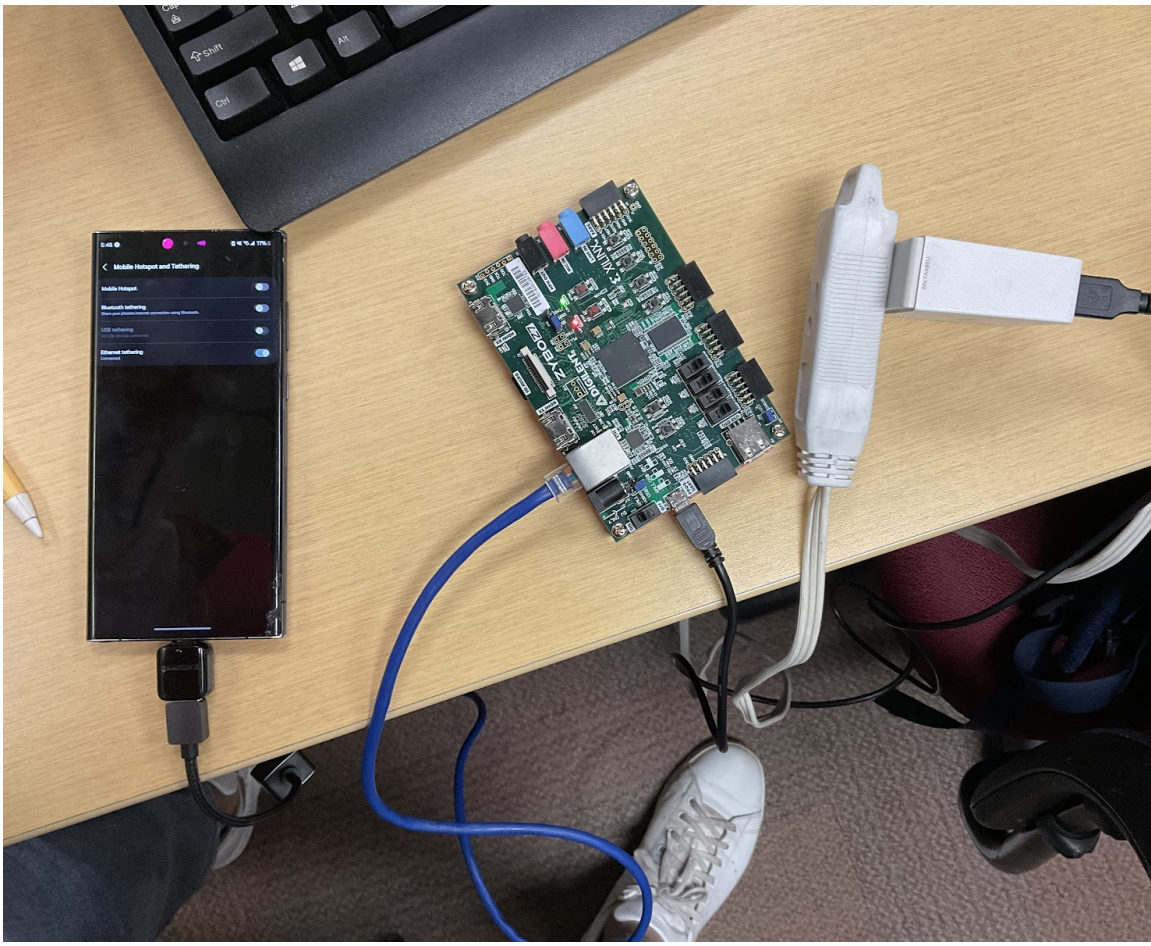
Resource	Estimation	Available	Utilization %
IO	16	100	16.00

e. On-Chip Power Graph



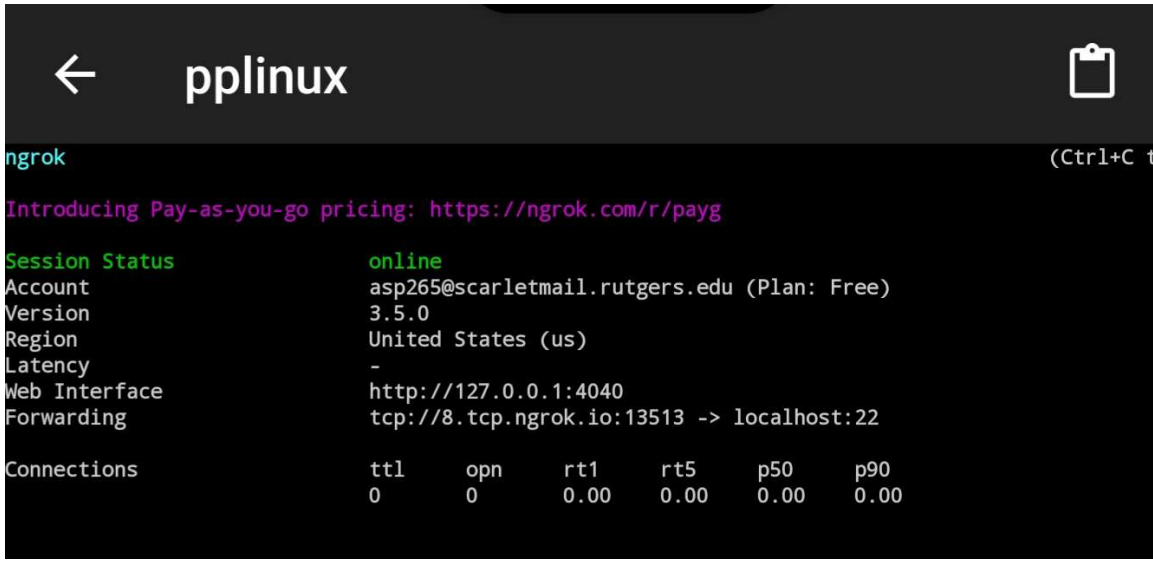
f. Demo

i. Setup



ii. Ethernet SSH

iii. SSH Server



iv. gpt.py

I wrote a ChatGPT chatbot that runs in the Linux CLIon the Zybo, which you can shh into from anywhere.

4. Difficulties encountered

Given our experience with Petalinux builds, I knew this would be a rough experience. Going into this, I knew very minimally about the Petalinux environment. First of all, I had to get a Petalinux build to work. My initial plan was to use Sftp to get a file to print on an OLED. However, as I worked on it, I realized that interfacing with PMODs would not be feasible in the time frame of the assignment.

a. Petalinux-Build

In the labs, I could never complete a build without any issues, and even if I could, I would never be able to get this to work if I built it on the school computers. I had to get Petalinux tools to work on my personal computer; I set up a Virtual machine and installed the tools needed. This allowed me to complete hours of building within 20 minutes. This enabled me to test and retest many times, which really helped me set up SSH properly.

b. Python Working Environment

I wanted to get a Python working environment going on the Petalinux install. I had to do this by entering petalinux-config and enabling the Python libraries. This solves many issues at once, allowing you to get Python packages, including pip and venv, which are essential to setting up a working environment.

c. Internet connectivity

Now that I have pip, I have the power to download packages, which is useless without a working internet connection. I figured I could use my phone to tether ethernet via the USB port to enable internet access. This gave me the power to connect to the world, and tools like wget, curl, git, and pip became very usable. This also enables me to do Sftp.

d. Root File System

I ran into the issue where the onboard memory would be overloaded to store these packages. For this, I needed to set up a RootFS. To do this, I had to take a few steps. First, I need to make 2 partitions in my SD card, one bootable 1Gb FAT32 partition, and the rest of the SD can be an ext4 partition. Next, in petalinux-config, you first need to set the file system type to SD, and then you need to build. After building, you have to run the following commands

- “sudo umount <ext4 partition location>”
    - Unmounts the partition
- “sudo dd if=images/linux/rootfs.ext4 of=<ext4 partition location>”
    - Creates a disk image using the rootfs.ext4 file generated by petalinux tools
- Sync
    - Syncs cached data to permanent storage
- sudo resize2fs <ext4 partition name>
    - Resizes the partition to the rest of the drive
- sync

After this, you can just copy the BOOT.bin, boot.scr, and image.ub to the bootable partition.

e. SSH

The Zybo, by default, enables Drobears SSH, which doesn't interact nicely with OpenSSH, which is a more common SSH protocol. This caused issues where I would not be able to connect. To get around this, I need to go back into petalinux-config, disable all dropbear packages, and enable the OpenSSH ones. This solved that issue. Now I can connect to to devices via the ethernet cable using SSH.

Now that I have SSH, a working internet connection, a Python working environment, and a storage file system, it is time to upgrade from Minicom to something better. I wanted to expose my Zybo's TCP port so I could connect from anywhere because that would be cool. To do this I used a service called Ngrok which exposed a port to the internet. This allowed me to ssh into the Zybo wirelessly.

f. PMODs

I tried to get PMODs to work, but I was not able to, I tried a way using the UIO drivers, but I could not find the settings for that. I also then tried to hardcode everything for it, but Xilinx has more proprietary drivers than numbers in pi, so that did not work.

g. MISC

As I mentioned, I wanted to get Sftp working, but I did get it working; once you enable the network packages OpenSSH on the Zybo, you can then use the Sftp tools, I did not end up using it, but it is capable.

I also enabled systemd, which allows for better startup process management and faster booting; after some research, I found out that it is not a good thing if you need a minimal application as it is more code and takes more space than the default, i.e., systemv(which actually dates back from OG UNIX days). I wanted to set up the Ngrok server on boot, but since I didn't pay for a static hostname, I need to check the numbers anyway, so I don't need this.

5. Takeaways/Conclusion

This Final Project has been one of my life's most fun learning experiences. With my difficulties and goals, I had to work in different ways to get my project done.

a. Documentation

I got really comfortable with reading documentation; I spent a lot of time on UG1141, reading the Xilinx wiki, and scouring the Xilinx forums. This was an enjoyable experience I found

myself genuinely enjoying the process where I would go into documentation looking for a specific topic and find something else to look into.

b. Petalinux tools

I learned much about the Petalinux tools and how and where to look for specific settings and packages. I think one of the coolest things I took away from this is knowing that I can build using the template, meaning I could have used the latest version of Petalinux tools and Vivado. I learned about the FPGA manager and how it allows us to switch bitstreams from within Petalinux. I think this, rather than being a learning experience, has been an exercise in exposure to the world of petalinux

c. Linux

Going into this project, I thought I, having been a Linux main for almost a decade, would know a lot about Linux, but I was wrong. I learned new tools every step of the way; when trying to get SSH working, I learned about the SSH config and known hosts files, which I needed to keep clearing as I kept running into issues when connecting after I restarted ngrok. I also got really comfortable with Vim, which helped me the next day when I had to use vim to code my final project due to technical difficulties.

In conclusion, this project was a very fun exploration of Petalinux. Learning about the different ways to work in this environment gave me a better understanding of the applications of FPGAs and the constraints placed on engineers developing these and working with them. I wish I could get the PMODs working, as that would be very cool. Going forward, if I had more time, I would definitely sit down and focus entirely on PMODs, especially now that I have a better grasp on petalinux tools.