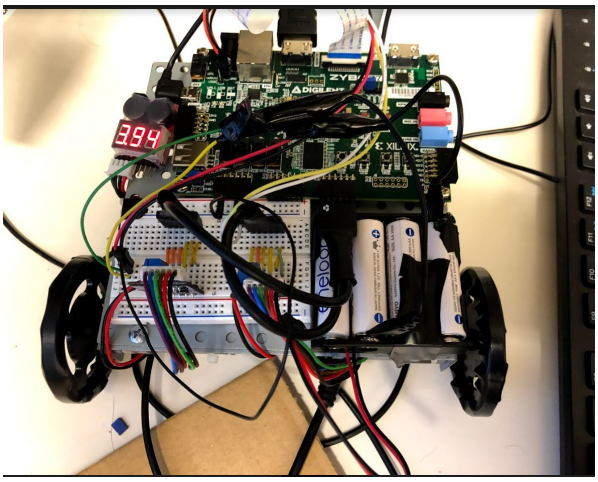
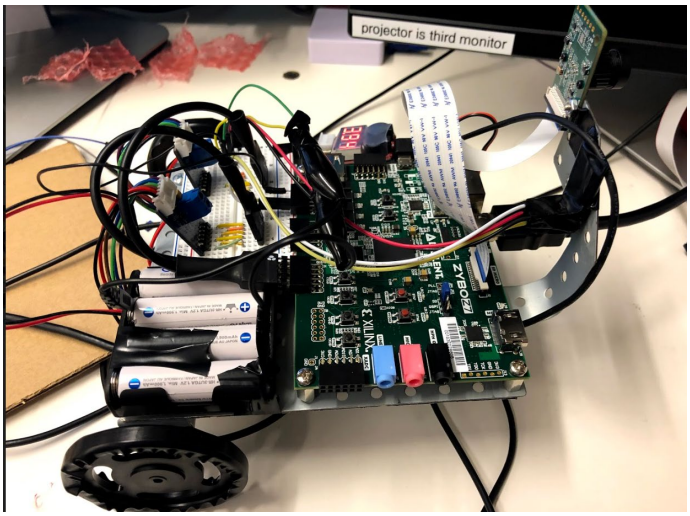
Robot Project Report
Group B-2





Project goal:

The goal of this project was to create a remote controlled robot. The robot would be controlled via some interface that ran on a computer or potentially a smartphone. Communication with the robot would be through wifi. In order to navigate while controlling the robot, a video feed of the camera on the robot would be sent the controller.

Camera:

For this project we used the Pcam 5C camera module. Getting the camera to work was the most straight forward part of this project. However it wasn't without its difficulties. We took full advantage of demo project offered on the digilent Pcam 5C webpage. This demo project was nice because it came with a fully configured camera hardware pipeline that included debayering, gamma correction, different video resolutions, fully configured vdma, and more. The only problem was that the hardware was too big to fit on the zybo z7-10. Our first approach to solving this was to delete as many unnecessary IP-cores as we could. Things like the gamma corrections, rgb to ycbcr, and hdmi output were thrown out. This still wasn't enough to fit the project onto the board. After some digging online, we found a forum post where a Digilent staff member had found a way to significantly shrink the size of the project just by disabling the debug features of one of the IP cores.

Wifi:

Our first plan was to get wifi working by using a wifi pmod chip and using linux to interface with it via some driver. After doing some quick research we learned that this was not feasible since there were no drivers for our chip. We did however find IP cores that would allow us to create a socket connection all in bare metal. This solution seemed nice since there were already example projects that would quickly show us how to get started using the wifi. Nothing is ever as easy as it seams. We ran into a lot of problems using the IP and sample projects. After installing an earlier version of Vivado, we were finally able to test out the sample project and we were met with a blank terminal. After a few hours of debugging we weren't able to figure out much. It appeared that the sample program would enter an endless loop. Some signals would be sent to the wifi module but then it would stop. We eventually decided to use the ESP-01. This chip was much easier to use since you can interface with it just as you would any other uart device. The speeds we could achieve were much slower since this chip used uart as opposed to the original pmod chip which used SPI, but it was better than nothing.

After we got the ESP-01 wifi chip working, the only other problem we had with it was how to power it. We ran into a problem when the camera hardware was loaded onto the zybo that running a program would draw more current that our power supply could supply. This would cause the board to reset. We eventually resolved this by using the rechargeable eneloop batteries to directly power the wifi. These batteries were conveniently 1.2V each. This means we could use 3 of them for 3.6V which is just inside the operating range of the ESP-01.

Finally we had wifi working and now integrating it into the project was straight forward. On the robot side, we could simply read and write data from the socket through uart functions that we were already familiar with from MP-4. On the controller side, we used python so writing and receiving data from the socket was very easy.

Wheels:

Getting the wheels turning took a lot of time spent debugging, but in the end turned out to be fairly straight forward. During our initial research we found H-Bridge drivers that would

allow us to easily control the motors in software. However, when getting these to work we ran into 2 problems. The first problem was that we first tested it out using the same hardware as the camera pipeline. This was a problem because when regenerating the camera pipeline hardware, the VHDL wrapper would not regenerate and thus the and thus the pins we constrained for the pmod H-Bridge would not actually be set. Thinking that the drivers just didn't work, we tried to implement a different solution using AXI timers to generate our own PWM signal. This didn't work for the same reason as the H-bridge drivers. After many hours we eventually realized the problem and manually regenerated the VHDL wrapper.

Now the drivers were working fine. We could call a function which would turn any wheel the direction, speed, and distance we wanted. The other problem arises when we went to use these drivers in the same project with the camera code. The camera drivers were C++ and the H-bridge drivers were C. This lead to linking errors because we were calling C functions from C++ code. The solution to this was to just surround the included C library headers in an extern "C" block which lets the linker know to use C function calling semantics.

Controller Interface:

The final part of this project was creating a desktop application that we can use to interface with the robot and control it. To accomplish this we made a simple python program. This program used the pygame library to display the the video received from the robot and took input from the arrow keys to go left, right, forward, and backwards. The video resolution that we display is 639 by 479. There is one less column and row than 680 by 480 because the camera hardware pipeline blanks out the top row and left column. It takes around 25 seconds to receive the full picture in grayscale at this resolution which comes out to about 96 Kilobits per second . Because it is so slow, we decided to have it send a single frame at the request of the user.

Resources used in this project:

https://reference.digilentinc.com/learn/programmable-logic/tutorials/zybo-z7-pcam-5c-demo/start

https://reference.digilentinc.com/learn/programmable-logic/tutorials/pmod-ips/2018.2