Lab Report for Lab 6

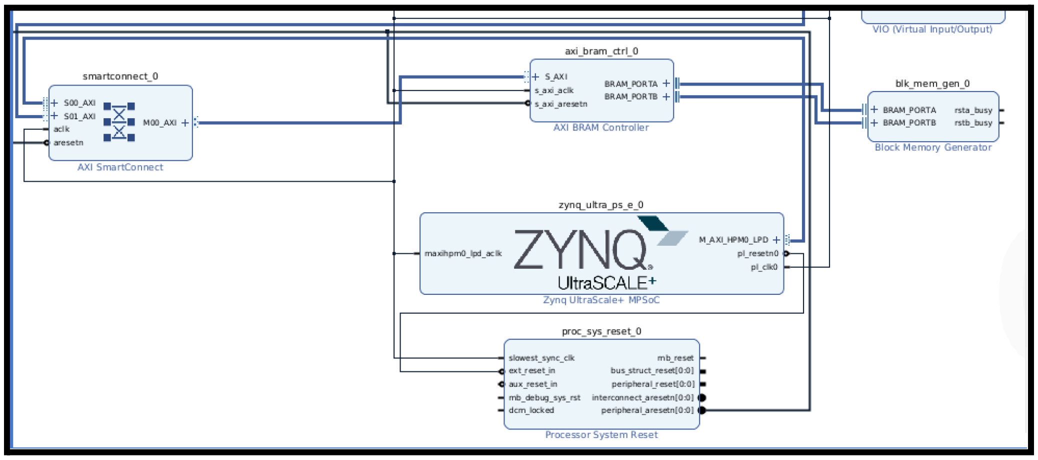
Source Code Link: https://github.com/imezabu/cs391r1-lab6.git

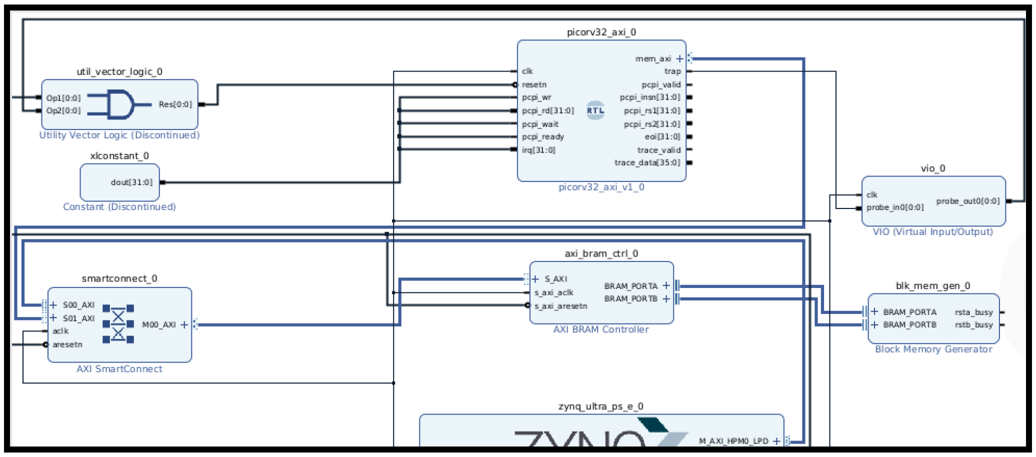
Lab6 1.1

Approach

For 1.1, our group followed the instructions outlined in the getting started guide to ultimately implement the block design for our pico32rv module as shown below. Each wire was chosen to connect the correct input to its related output all throughout the block design. To make sure that our resulting bitstream from the block design was generated, we checked our directory to see if the file was in the correct path.

Code:





Testing:

Testbench Code:

```
debug_nets.ltx
final_pico_wrapper.bit
final_pico_wrapper.ltx
```

(Excerpt(s) of the testbench(es) for this problem.)

Testing Approach:
Our testing approach was to just generate the bitstream and then connect it to the FPGA board. We just followed the instructions in the Getting Started pdf. For testing the values of simple_test, we had to use busybox devmem as opposed to devmem2 since it kept stalling and giving us bus errors.

Results:

Waveforms (Group): n/a
There were no waveforms for this portion of the lab.

Outcomes:

We were able to get everything running and were able to properly generate the bitstream to then be run and connected to our board. However, we ran into some issues when initially generating everything. When we first started it took over 30+ minutes to generate our first bitstream, and we had to kill the process and re-run it. Additionally, when first trying to run everything, we had a lot of errors due to incorrect wire connections. This led us to reconnect all the wires and ports once again. After killing the initial process and re-running it we were finally able to generate a proper bitstream after 10 mins and with more threads.

Lab6 1.2

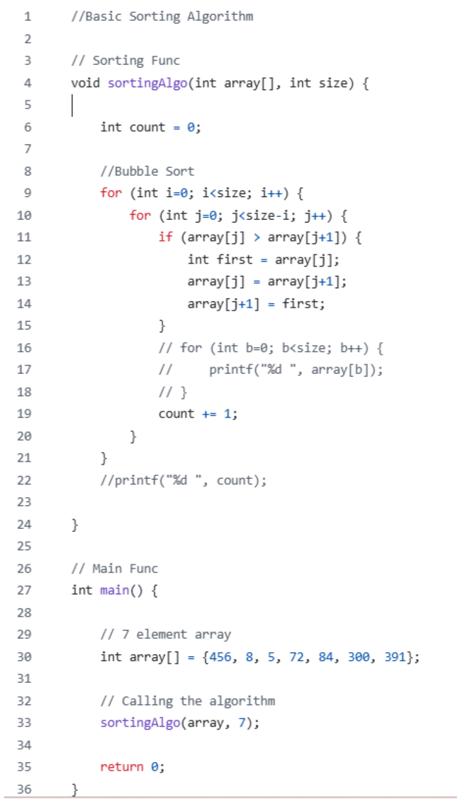
Approach

This portion only involves running tests with our FPGA board and connecting/loading up the bitstream to the FPGA board. Our approach was to just mainly the Getting Started Guide and understanding the board itself along the way. Lots of trial and error. We had to use SCP to send the bitstream and c files over to our FPGA board. From there, we ran lpd_32bit.sh, flashed the bitstream onto the pico, compiled our files, and finally loaded them onto the fpga board. Connecting to it via vivado allowed us to start and stop our program using our probes.

Code: n/a

Testing:

Testbench Code:



```c
//Basic Sorting Algorithm

// Sorting Func
void sortingAlgo(int array[], int size) {

    int count = 0;

    //Bubble Sort
    for (int i=0; i<size; i++) {
        for (int j=0; j<size-i; j++) {
            if (array[j] > array[j+1]) {
                int first = array[j];
                array[j] = array[j+1];
                array[j+1] = first;
            }
            // for (int b=0; b<size; b++) {
            //     printf("%d ", array[b]);
            // }
            count += 1;
        }
    }
    //printf("%d ", count);

}

// Main Func
int main() {

    // 7 element array
    int array[] = {456, 8, 5, 72, 84, 300, 391};

    // Calling the algorithm
    sortingAlgo(array, 7);

    return 0;
}
```

Our own C file:

(Excerpt(s) of the testbench(es) for this problem.)

Testing Approach:

- Using minicom -> whenever we minicommed we couldn't differentiate which port to use -> correct one was 1
- Couldn't get any text to pop up -> had to press reset on each port until port usb1 gets a response ->prompted to login
- Couldn’t type anything or take any inputs -> had to mess around with the settings – changed a hardware setting -> disable hardware flow control
- All of your .s file weren’t properly downloading from blackboard -> had to make our own and copy and paste it; didn’t have the right permissions
- Had to figure out the correct way to use SCP to copy over files
- Devmem was not installed, installed busybox and had to prepend everything with “busybox”
- Couldn’t get the probes to properly appear -> had to redo block design so they get detected and regenerated the bitstream

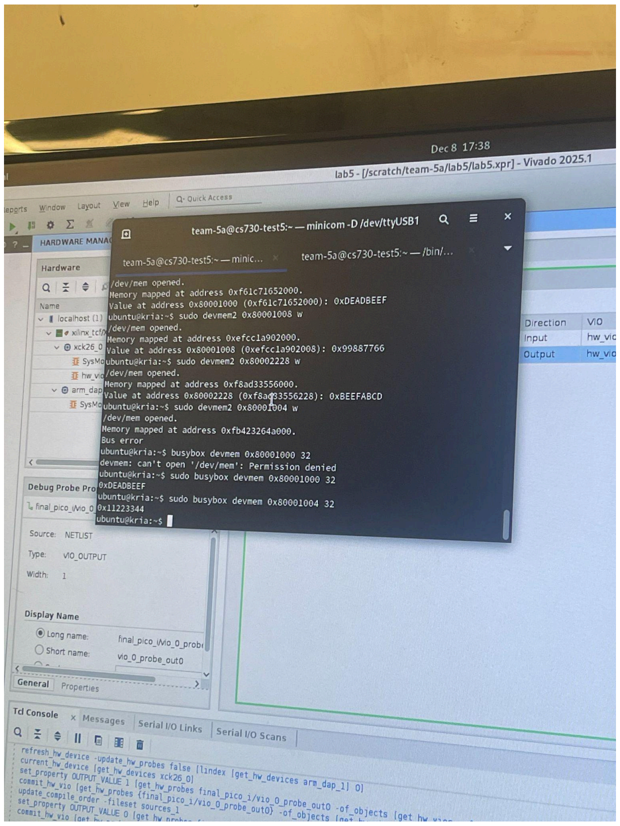
- Program ran but then it kept hanging -> had to get new firmware
- Devmem2 sometimes had bus errors -> use busybox devmem

We worked to follow the instructions very directly from the slides, but on many occasions had to debug issues that occurred even when following these instructions. Aside from these issues, we were able to directly follow the steps from Chapter 2 and Chapter 3 of the getting started guide, and were able to get everything running (loading the binary file to Bram, connecting to the FPGA board, running it on the fpga, and testing for the correct values).

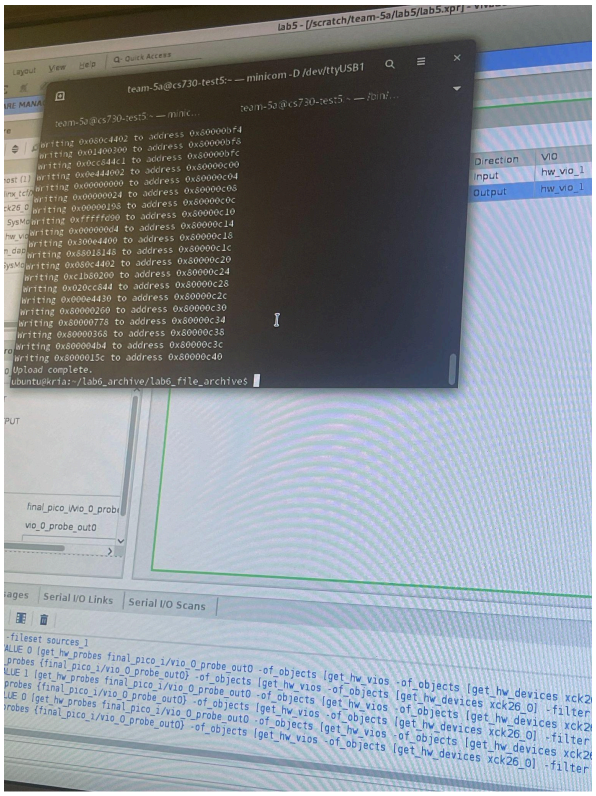
<u>Results:</u>

Waveforms (Group): n/a
There were no waveforms for this portion.

Outcomes:
We were able to verify the writes by running the provided commands, at times having to alter the command somewhat due to issues encountered earlier. Happily, we confirmed that we were reading all the correct values that were listed in the instructions.



```
/dev/mem opened.
Memory mapped at address 0xf61c71652000.
Value at address 0x80001000 (0xf61c71652000): 0xDEADBEEF
ubuntu@kria:~$ sudo devmem2 0x80001008 w
/dev/mem opened.
Memory mapped at address 0xefcc1a902000.
Value at address 0x80001008 (0xefcc1a902008): 0x99887766
ubuntu@kria:~$ sudo devmem2 0x80002228 w
/dev/mem opened.
Memory mapped at address 0xf8ad33556000.
Value at address 0x80002228 (0xf8ad33556228): 0xBEEFABCD
ubuntu@kria:~$ sudo devmem2 0x80001004 w
/dev/mem opened.
Memory mapped at address 0xfb423264a000.
Bus error
ubuntu@kria:~$ busybox devmem 0x80001000 32
devmem: can't open '/dev/mem': Permission denied
ubuntu@kria:~$ sudo busybox devmem 0x80001000 32
0xDEADBEEF
ubuntu@kria:~$ sudo busybox devmem 0x80001004 32
0x11223344
ubuntu@kria:~$
```



```
Writing 0x080c4402 to address 0x80000bf4
Writing 0x01400300 to address 0x80000bf8
Writing 0x0cc844c1 to address 0x80000bfc
Writing 0x0e444002 to address 0x80000c00
Writing 0x00000000 to address 0x80000c04
Writing 0x00000024 to address 0x80000c08
Writing 0x00000198 to address 0x80000c0c
Writing 0xfffffd90 to address 0x80000c10
Writing 0x000000d4 to address 0x80000c14
Writing 0x300e4400 to address 0x80000c18
Writing 0x88018148 to address 0x80000c1c
Writing 0x080c4402 to address 0x80000c20
Writing 0xc1b80200 to address 0x80000c24
Writing 0x020cc844 to address 0x80000c28
Writing 0x000e4430 to address 0x80000c2c
Writing 0x80000260 to address 0x80000c30
Writing 0x80000778 to address 0x80000c34
Writing 0x80000368 to address 0x80000c38
Writing 0x800004b4 to address 0x80000c3c
Writing 0x8000015c to address 0x80000c40
Upload complete.
ubuntu@kria:~/lab6_archive/lab6_file_archive$
```

Additionally, we were able to upload our own C file, which ran successfully.

<u>Reflections:</u>

In reflection, this portion of the lab posed a fair deal of challenges. The directions were largely clear, however, when things failed or we encountered an error, it was largely on us to try to investigate some reasonably unfamiliar problems on our own to try to continue our work. Thankfully, after one entirely progress halting issue where our FPGA kept hanging and wouldn't let us test our writes, we were provided a solution to update the FPGA's firmware, which did resolve our issue, allowing us to complete the lab. Other than these key challenges, it was neat to go from wiring a virtual block design, to implementing all our virtual design work to a physical device, and running binary compiled C code on it with success.

<u>Individual Contributions:</u>

Beren Akpinar: For lab 6, I mainly helped create our block design and ensure that all wires were connected correctly. After creating the block design, I generated the bitstream and debugged any errors we got in this step to progress further along in the getting started guide. Lastly, I also helped my teammates in trying to understand why our FPGA board wouldn't connect after resetting it on the last step.

Ruby Chen: For lab 6 I mainly helped with making the block design and getting through the initial first two chapters of the Getting Started pdf. I also helped when it came to generating the bitstream, loading the binary files, and running the test programs. However, most of this lab was done in a group at one computer so we all contributed that way. Then I helped with setting up the repo and documentation.

Michael Ross: For lab 6, I assisted with the initial block design creation, assisting in debugging issues with further setup steps, and assisted with documentation in this report.

Isaac Meza: For lab 6, I assisted with loading the bitstream to the FPGA board, compiling and loading the binaries, and getting the design detected by vivado as well as running the actual program on the board.