Anthony Humay, Andrew Musk, Clayton Green

M152A Lab 6

Mastorakis

Lab 2 Report

**Introduction**

In this lab, we built a sequencer / adder that utilizes four 16-bit registers that can be used as operands or result storage. On our Nexys 3 board, the 8 slider switches are used to represent 8 bit instructions to our program, where a switch turned on represented a high bit. The sequencer reads this information in, and used the following convention to decode the instruction:

Slider switches:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

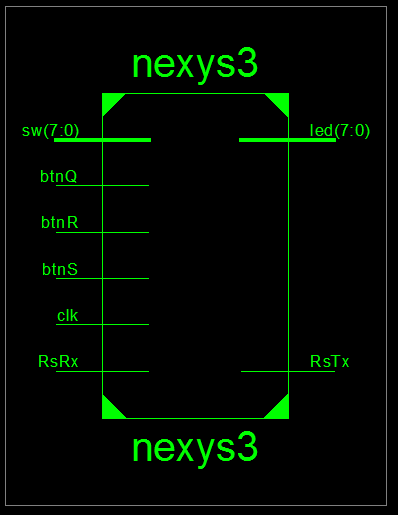
Corresponding positions decoded as:

|  |  |  |  |
| --- | --- | --- | --- |
| Type of instruction (SEND, MULT, PUSH, ADD) | Register to use | Register to use | Register to use |

Once these switches are set desirably by the user, the user can send this to the sequencer by using the PUSH ‘opcode’ and pressing a button on the Nexys 3 board. To display this value on putty, a separate opcode SEND was used along with which register to display.

What we had to do was implement one of these opcodes, MULT, which multiplied two register values and stored the result into another register. In addition, we created a new button to send information without having to first setting the SEND ‘opcode’. Another task included adding file reading functionality, so files with pre-written 8 bit instructions could be loaded and run. Finally, we cleaned up the display when a register was sent to putty, adding which register’s value it was being shown.

**Design Description**



*Figure 1.1: Design schematic of our project.*

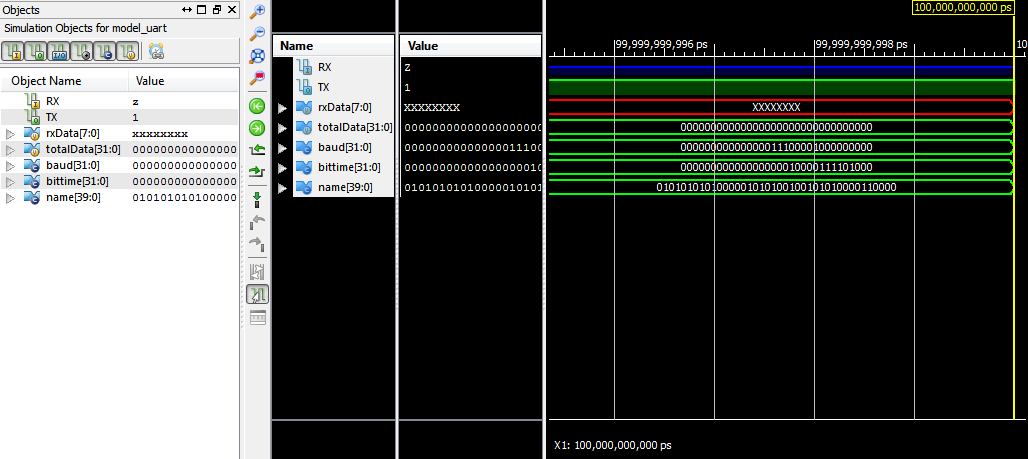
The addition and multiplication operations were implemented in the their respective modules, seq\_add and seq\_mult. These were then instantiated in the seq\_alu module, which handled arithmetic operations. The seq.v module was the ‘main’ module (along with nexys3.v) where seq\_alu.v was instantiated. uart\_top and nexys3.v was where the display modifications were made.

For the “An Easier Way to Load Sequencer Program” task, we wrote seq.code, which is just a text file full of 8 bit binary that details the first 10 numbers of the Fibonacci sequence.

When testing the file loading functionality, we changed the testbench tb.v so that it would read in the file accordingly. If we were not testing this functionality, we used a slightly modified version of the original tb.v that was originally given to us.

**Simulation Documentation**

We tested all the core functionality first - this primarily meant the four opcodes of PUSH, SEND, MULT, and ADD. We tested various permutations of those commands, and made sure that the according registers were correctly updated and correctly changed. Checking the appropriate registers became extra important when we changed the UART output so that it showed which one was being displayed. Testing this actually revealed a glitch where we had switched the first and second register and it was therefore displaying incorrectly.



*Figure 1.2: Simulation objects from iSim.*

The first bug that we ran into during our simulation was related to our multiplier function. The issue with this was that we created the mult module and added it to the seq\_alu.v but we didn’t change seq.v. The next issue we ran into in simulation was not setting the simulation run time high enough. When set to around 10000000 our program was not producing any output. After changing this to 100000000000 our program began to display output. Another issue we ran into was having source files where the path changed between labs after uploaded to google drive. When working on the fibonacci numbers, we were having issues as we forgot to change the first line of the source file to reflect the number of lines in the file.

**Conclusion**

In this lab, we implemented a sequencer and adder that took eleven inputs: eight slider switches that represented a eight bit instruction, a button to execute the instruction, and another button to automatically execute the SEND instruction. We then implemented control logic of the instructions, including adding multiplication functionality. This control logic modified the four available registers, which could then be sent to the universal asynchronous receiver-transmitter.

We had a bit of difficulty initially figuring out exactly what the text file was supposed to hold for the Fibonacci, but the example in the section above made it clear after further inspection. In addition, we were unsure which files we had to modify for Fibonacci. We eventually realized it was merely the file that was being passed in.

Overall the lab was straightforward and clear, but it would have been helpful to spend more time on metastability and more general theory as it relates to noise in circuits.