

CS M152A: Introductory Digital Design

Laboratory

**Lab #2**

**Sequencer**

Name: Kenny Chan, Kenny Luu, Kyle Reidy

UID: 004769092, 104823244, 104839297

Lab Section: 5

Due Date: May 9, 2018

**Introduction**

For this lab, we were provided with prewritten sequencer verilog code and a task to add new features to the code. This sequencer came with simple operations that most other ISA’s have, including the ability to push immediate data onto registers, add operands, and retrieve data from its registers to be displayed through UART.

Our tasks were to implement additional support for multiplication, improve the UART output, add a new button to push data onto registers, improve the UART once again, implement an additional way to quickly run commands, and use the improved sequencer to display fibonacci numbers.

We used the Nexys3 board to upload our sequencer code onto its memory. Once working, we used Putty and iSim to simulate and interpret the sequencer’s UART outputs. Doing so allowed us to debug our sequencer and make sure that all functionalities were working as intended.

**Design Description**

The following figure describes the high-level implementation of the sequencer. The commands are sent to the control logic through the onboard push buttons and slider switches on the Nexys3 board. From the control logic, signals are sent to the registers, the ALU, and the UART. Depending on which machine code instruction is indicated by the slider switch, different operations will be performed. From the USB UART, the output of the send can be viewed using a terminal.

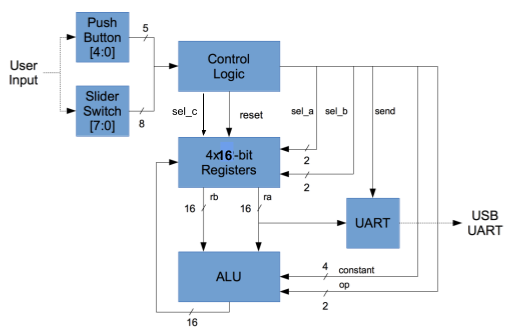


Diagram of Sequencer Implementation from the Lab Manual.

If a push instruction is indicated by the value of the slider switch, sel\_a from the control logic selects which register to left shift and push a new value. The ALU then left shifts the register by four bits, and writes the constant indicated by the control signal as the four least significant bits.

If the instruction indicated was instead an add or multiply command, the control logic would signal which registers to input as operands for the ALU, using sel\_a and sel\_b. The ALU would perform its operation (an add or multiply) depending on the control signal indicated by op and write to the register indicated by sel\_c.

Whenever a send command is issued, the control logic indicates which register to use as input to the onboard UART using sel\_a. If the send signal is asserted, the UART on the Nexys3 board will output the register data to our terminal.

**Tasks**

For this lab, we were were assigned seven tasks. Three of the tasks related to the simulation side of the project, and the other four pertained to the implementation aspect. Our changes to the design did not add any significant changes to the datapath shown in the figure.

Our first task was to execute some test instructions on the FPGA board, and examine the output. Using our prior experience, we easily built the project. Then we decoded the instructions into corresponding 8-bit code and inputted using the slider switch.

Our second task was to implement the multiplication instruction to the ALU. In verilog, this was as simple as replicating the addition function of the ALU and addition components of the modules, but exchanging ‘+’ with ‘\*’. We also added a new ALU op input of 2’b10 to tell the ALU when to perform a multiplication.

Our third task was to improve the UART output by printing four bytes in one line, rather than printing a byte per line. This part of the lab involved understanding the receiving end of the UART protocol. To achieve cleaner UART output, we modified the code within the testbench file to behave differently. This required creating a buffer to store characters as they were received and printing this buffer when a ‘\n’ character was received. This part of the lab proved quite difficult, as understanding the UART semantics was no simple task.

Our fourth assignment was to implement a new SEND button. To accomplish this task, we first examined the .ucf file and uncommented an additional button on the Nexys3 board. This allowed us to use the additional button from the available PushButton[0:4] shown on the figure. This new SEND button acted the same as the regular command button. Therefore we replicated the code from the original reset and command button and debouncer for our new button. Inside the control logic shown on the figure, we included the new SEND button as an input, which would perform a SEND operation and skip the process of checking the opcode. It does this by triggering the UART module to start output the register data.

Our fifth task was to improve the UART output even further by indicating the register as well as the data value. However, instead of editing the testbench files, we made changes to the actual UART module on the sequencer. This part of the lab involved an understanding of the send mechanism. Once again, the UART syntax provided some challenges, in terms of figuring out exactly which characters needed to be transmitted and how this could be achieved.

Our sixth task was to implement an easier way to quickly send commands to the sequencer when running the testbench simulation. Instead of executing a static testbench within the testbench, we want to execute instructions provided in a text file. To do this, we used the $fopen, $fscanf, and $feof functions included in verilog. We used $fopen to access the text file with all of our commands written into it. After opening the file, we used $fscanf to extract the number of commands in the file, which is indicated by the first line. After that, we used a while loop to continually extract commands using $fscanf and run the proper task based on the opcode. We checked the condition of the while loop by using $feof to see whether we have reached the end of the file.

The final task was to design a sequence of instructions to calculate the values of the first ten fibonacci numbers. To begin, we pushed the value 1 into two registers and printed them using send. Then we added these values, stored in a third register, and printed that. Then we repeat this process by adding the latest two register, and outputting the result ten times.

**Simulation Documentation**

Throughout this lab, we tested our implementations using the iSim software. Especially after we had improved the UART output in part two of the lab, the simulations provided by this program proved to be invaluable tools. When we started the lab, our simulation was able to output every instruction it executed to the log, every character it received, and more. After improving the output, the simulation could log four bytes at a time, which was very helpful, as this was the full value stored in a register.

Using the simulation, we tested many aspects of our implementation and every command available to be executed (send, push, add, and multiply). We were careful to test edge cases as well, such as overflow during addition and multiplication, to ensure our implementation still behaved properly.

**Conclusion**

As was noted before, understanding the intricacies of the UART protocol proved to be one of the most challenging parts of this lab for us. While much of the lab was rather straightforward, part two alone took us over two hours as we struggled to understand the meaning and methods behind the code provided in the model\_uart.v file. Eventually, with some help from the TA, we were able to figure out how to correctly implement this part of the lab, by poring over the provided code repeatedly and by testing and failing many implementations. The merits of struggling with a problem and figuring it out for oneself are evident, but a provided explanation of UART and perhaps how it functions within verilog may be greatly beneficial for the understanding of this lab.

Overall, we consider this lab to be a success. We learned about some of the intricacies of mapping components of the board to data structures in our software. We became more comfortable with reading source code, understanding it, and applying these provided techniques to new scenarios. Our implementation worked on all counts, and we will be able to apply what we have learned to future labs.