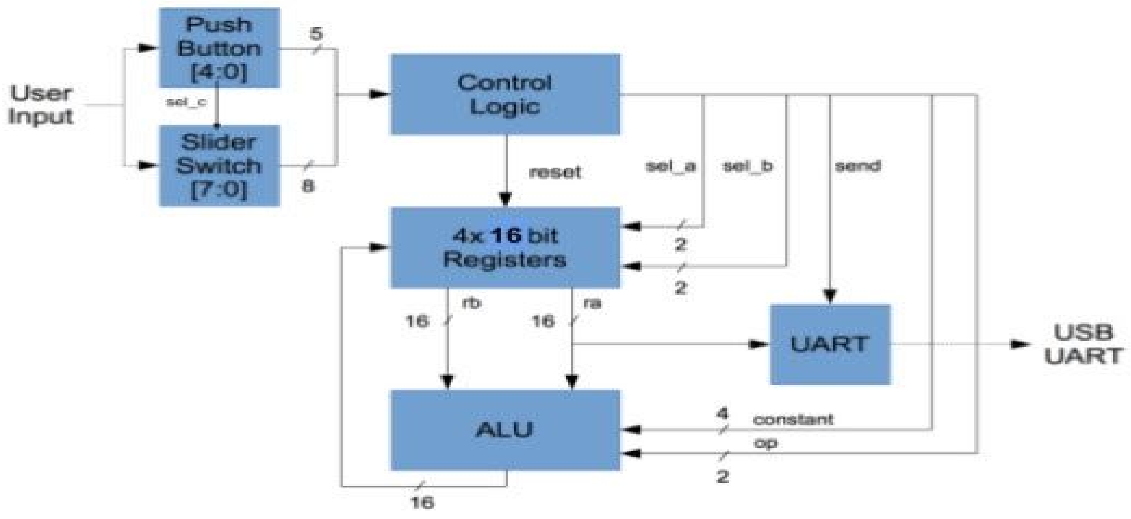
CSM152A - Lab 2 Report

Group Members:
Grant Roberts, 405-180-937
Ryan Brennan, 605-347-597
Yu Gao, 804-957-843

Section 5
TA: Kuo
Date Demoed: 02/05/20

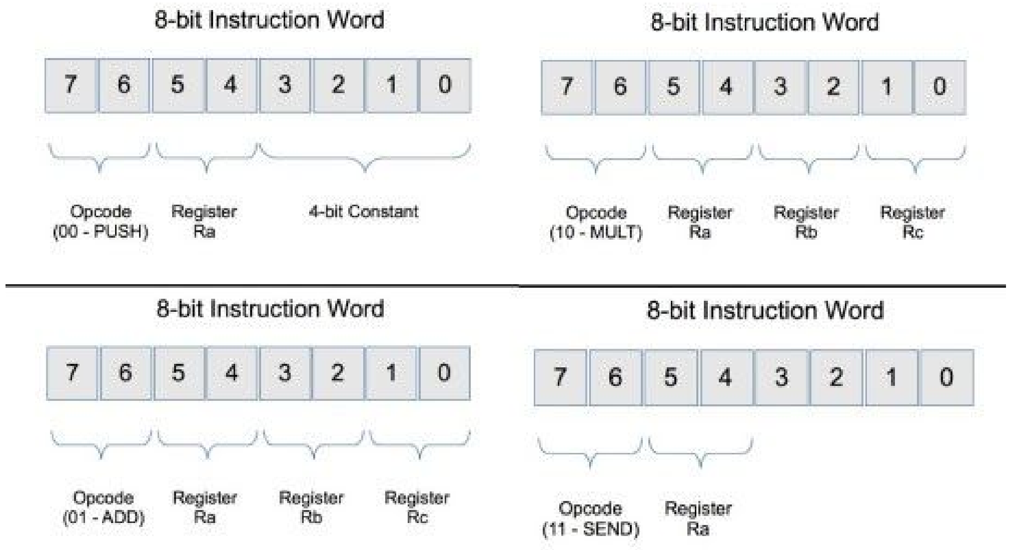
1) Introduction

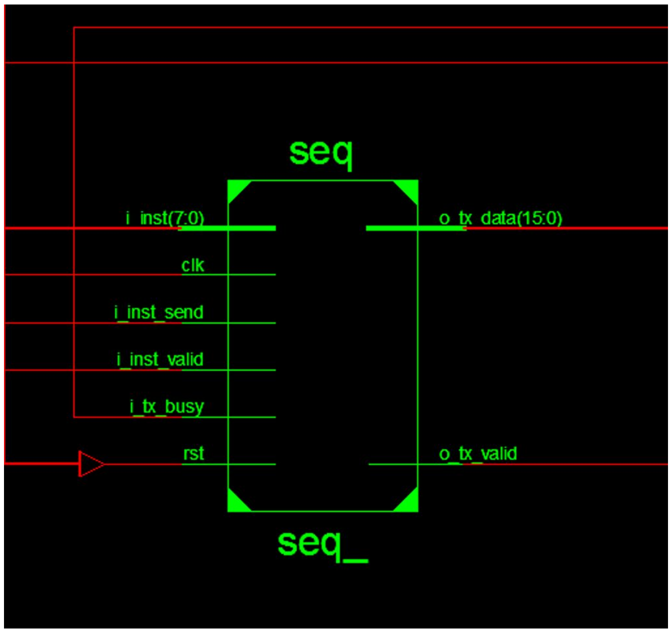
Our objective for this lab was to create and modify a sequencer as an example of a small FPGA project. Using various combinations of the eight switches and five buttons we were able to implement different ways of adding, pushing, and multiplying, as well as sending our results from these operations to a serial monitor with the onboard UART port. Shown below is a diagram of our sequencer design, taken from the lab manual.



1.1) Sequencer operation

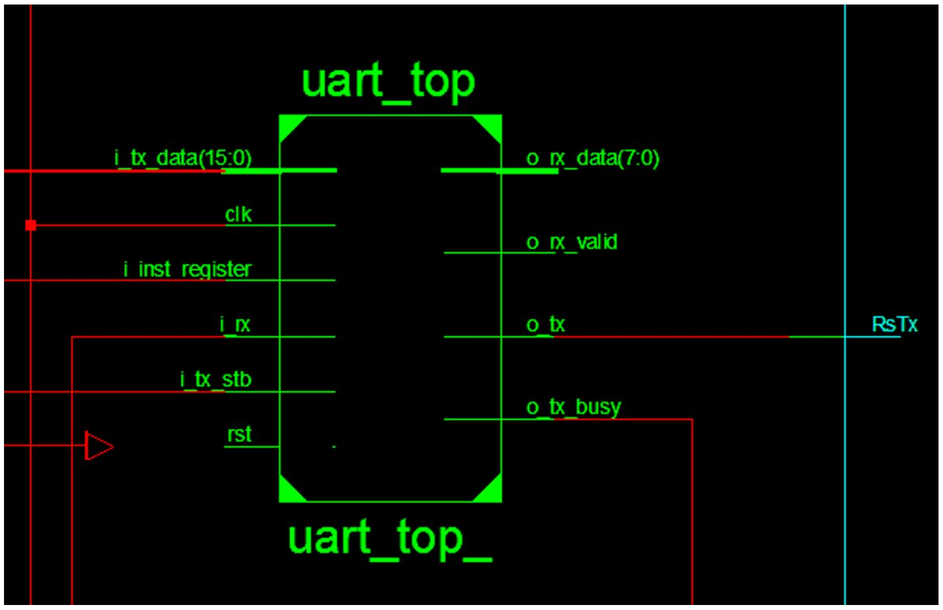
The sequencer was operated by a user entering in instructions through opcodes using 8 switches and push buttons. Each switch represents a single bit instruction: down for 0 and 1 for up. The center button was the push and execute button while the right push button was used as a reset. The majority of this lab is to build upon this sequencer and redesign certain features such as a separate send button.

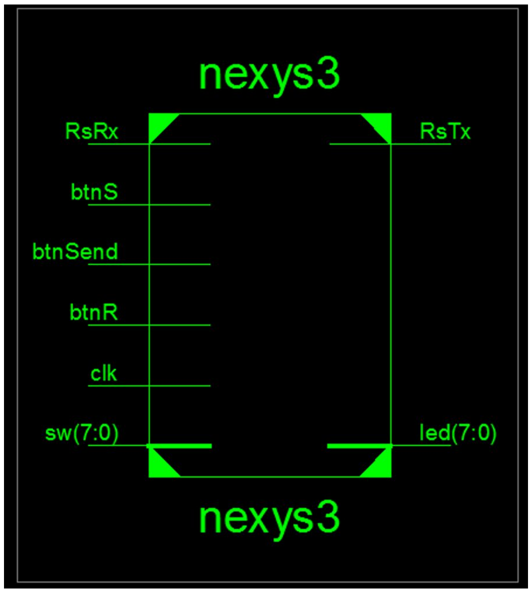




2) Implementation

The first task was to build our project from the given source files and map the pins for our buttons and switches to the variable names we'll refer to them by in our program. Specifically, in our .ucf file we uncommented the lines pertaining to switches 0-7, called sw<0> through sw<7>, the middle button btnS, and the right button btnR. We then made sure our opcodes were specified to 00 for push, 01 for add, 11 for send, that our nexys3.v file was configured to recognize the positive edge input from both of our buttons, and compiled our project. The position of switches 6 and 7 were to be interpreted as our two-bit opcodes and the remaining switches specified one or more operands, btnS would be used to execute the instruction specified by the switch positions, while btnR would be used to reset all of the register contents to 0.

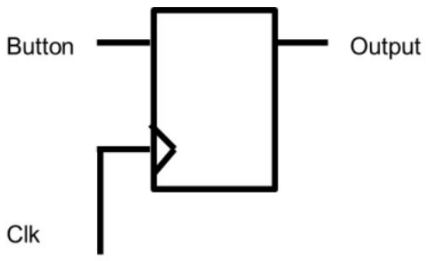


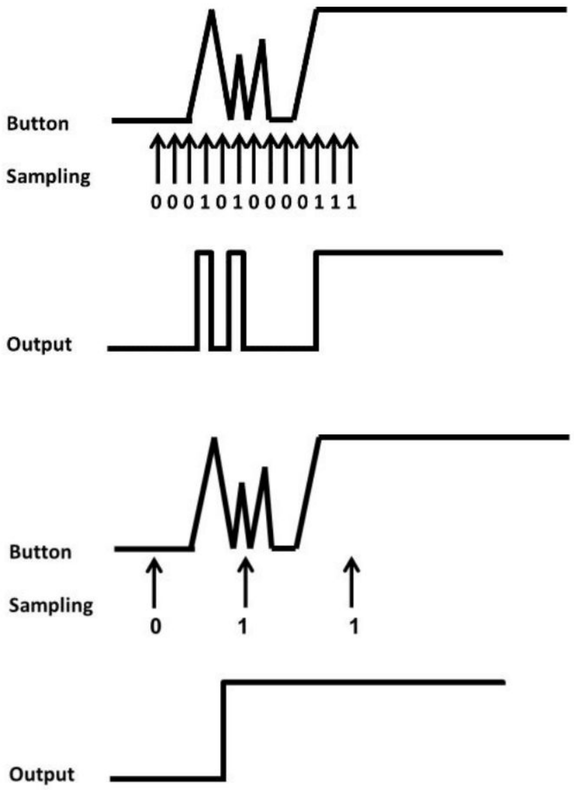


Multiply Operation
Next, we wanted to implement a multiplication function and began by encoding it as the two-bit opcode 10 in our sequencer definitions. We used the add module as a template, created a new module called seq_mul.v and simply changed the operation from addition to multiplication.

A Separate SEND Button
Then, instead of the send function being a designated opcode to be specified through the use of our switches, we were tasked with implementing the send function as a button instead of a series of switch positions. This button would send the contents of the register specified by the switch positions to our uart module for output. We enabled a button for "Send" instruction at location "A8" in .ucf file. As the source file shows, we applied the same way of debouncing the button to the new button. The basic idea is using a slower clock signal to down sample the button signal. Then a new reg "inst_send" will only be asserted when the previous button signal is 0 and the current one is 1. Finally, in the "seq" module, it has a new input signal "i_inst_valid" from "nexys3" module. Since the original button doesn't need to keep the "Send" functionality, we replaced the line
"assign o_tx_valid = i_inst_valid & inst_op_send & ~i_tx_busy;"
with
"assign o_tx_valid = i_inst_send & inst_op_send & ~i_tx_busy;". Hence all necessary editions to the source file are done.

Downsampling





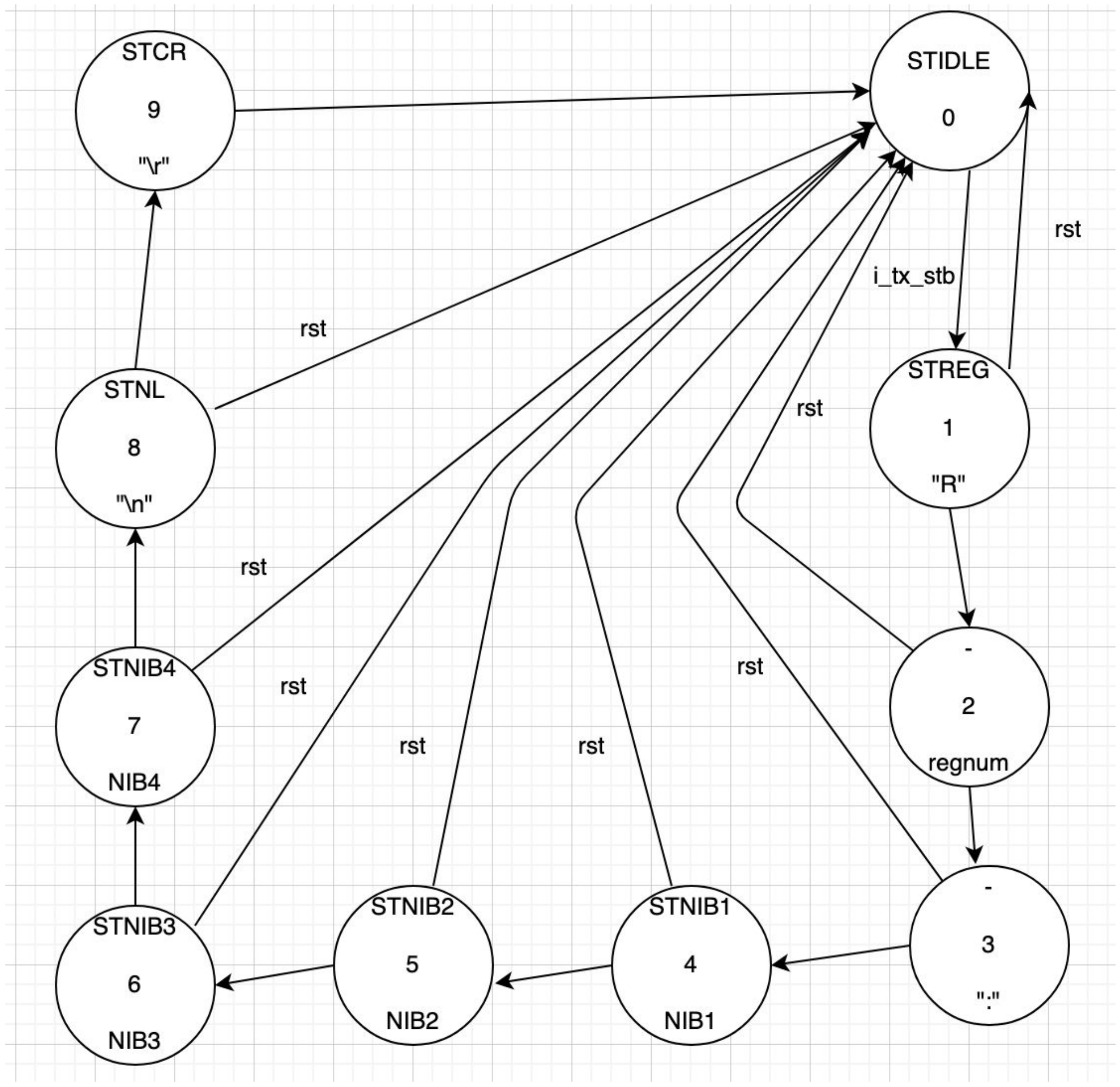
One issue introduced by the buttons is the generation of noise due to the physical properties of the button's contacts when it is pressed. In order to filter out this noise and ensure that we only generate one button signal for each button press we use a process called downsampling. The basic premise is to use a flip flop as shown above on the left and set the clock frequency low enough as to filter out the noise and only sample the button press once. The input without the use of downsampling is shown above right and with downsampling is shown above left.

Nicer UART Output

In the next section we wanted a nicer way to display our UART output, with the register number and its contents on the same line in the format RXX:YYYY where the the x's represent the two-bit register number and the y's represent its contents. We implemented this via a finite state machine within uart_top.v that would, through a series of case statements, transition from an idle state to assigning our output to be '**R**', then to the register number, followed by a '**:**', the contents of the register, a newline and finally a carriage return. After this cycle was complete it would return back to its idle state. Our new states are shown below:

```
  parameter stIdle = 0;
  parameter st_R   = 1;
  parameter st_RN  = 2;
  parameter st_colon = 3;
  parameter stNib1 = 4;
```

```
parameter stNL   = uart_num_nib+4;
parameter stCR   = uart_num_nib+5;
```



Notes: We encountered a problem that the Serial port didn't do character return and start a new line. The reason is the register "state" has only 3 bits, which can only store 7 states at maximum. Since we increased the states by 3, the total number of states is 9. To solve this, we need to expand "state" to 4 bits, so that it can store 9 states.

An Easier Way to Load Sequencer Program

In this step, we edited the test bench to load instructions from a file "seq.code". The first line contains the number of instructions, and start from the second line to the end are instructions. Each instruction has eight bits that represent 8 switches. After reading the first line using instruction “$readmemb("seq.code", in_ram)”, we saved everything into a matrix with size 1024 by 8, becasue the file is up to 1024 long, and each instruction is 8 bits long. We then used a for loop to loop through the instruction, and call the function “tskRunInst(in_ram[i])”.

Fibonacci Numbers

The last task is outputting the first ten of fibonacci numbers “0, 1, 1, 2, 3, 5, 8, 13, 21, 34”. Here is the instruction list and the comment of each instruction:

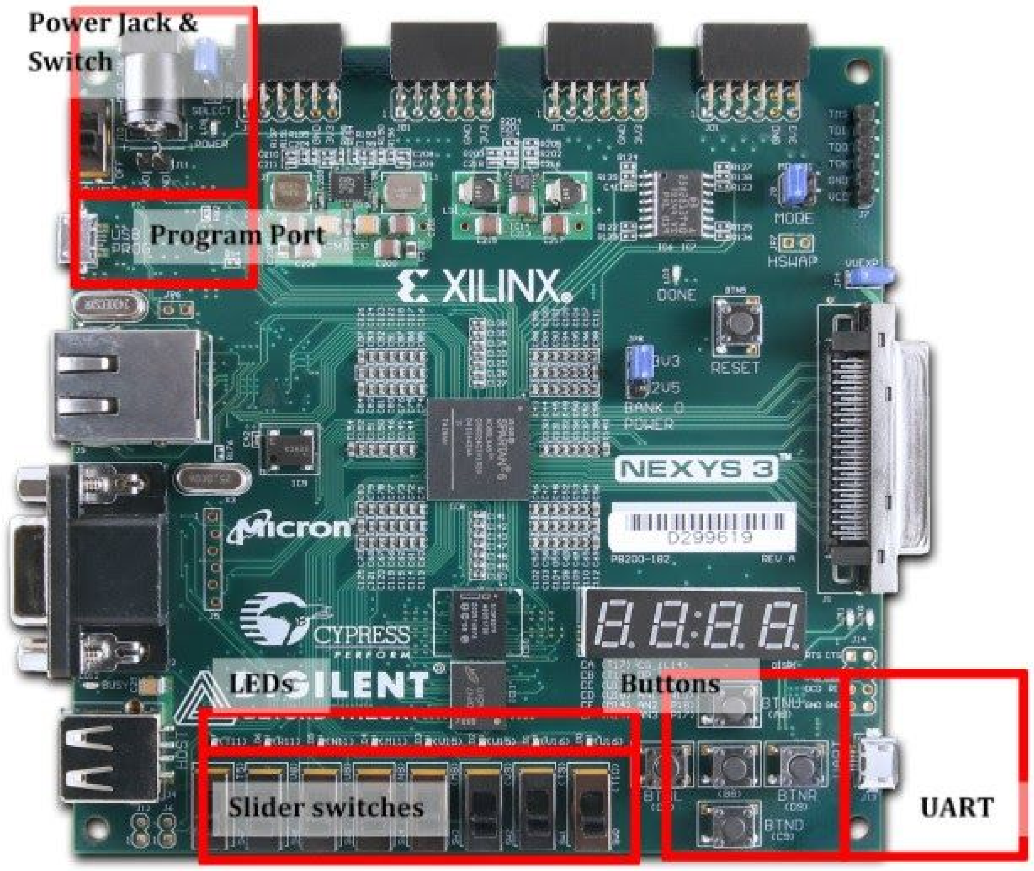
```
10100           // 20 instructions
11000000        // Send R0 = 0
00000001        // Push R0 = 1
11000000        // Send R0 = 1
00010001        // Push R1 = 1
11010000        // Send R1 = 1
01000110        // Add R2 = R0 + R1 = 2
11100000        // Send R2 = 2
01011000        // Add R0 = R1 + R2 = 3
11000000        // Send R0 = 3
01100001        // Add R1 = R0 + R2 = 5
11010000        // Send R1 = 5
01000110        // Add R2 = R0 + R1 = 8
11100000        // Send R2  = 8
01011000        // Add R0 = R1 + R2 = 13
11000000        // Send R0 = 13
01100001        // Add R1 = R0 + R2 = 21
11010000        // Send R1 = 21
01000110        // Add R2 = R0 + R1 = 34
11100000        // Send R2 = 34
```

Notes: we encountered a problem that the testbench didn’t show the “UART” output. The reason is that in the task “tskRunInst”, which only simulates a push action for “btns”. Since we added a new button for “Send” instruction and disabled the “btns” for “Send” instruction, the “Send” instruction is never executed. We then modified the task “tskRunInst” to :

```
“   task tskRunInst;
      input [7:0] inst;
      begin
        $display ("%d ... Running instruction %08b", $stime, inst);
        sw = inst;
        #1500000 btnS = 1;
        #3000000 btnS = 0;

        #3000000 btnSend = 1;
        #3000000 btnSend = 0;
```

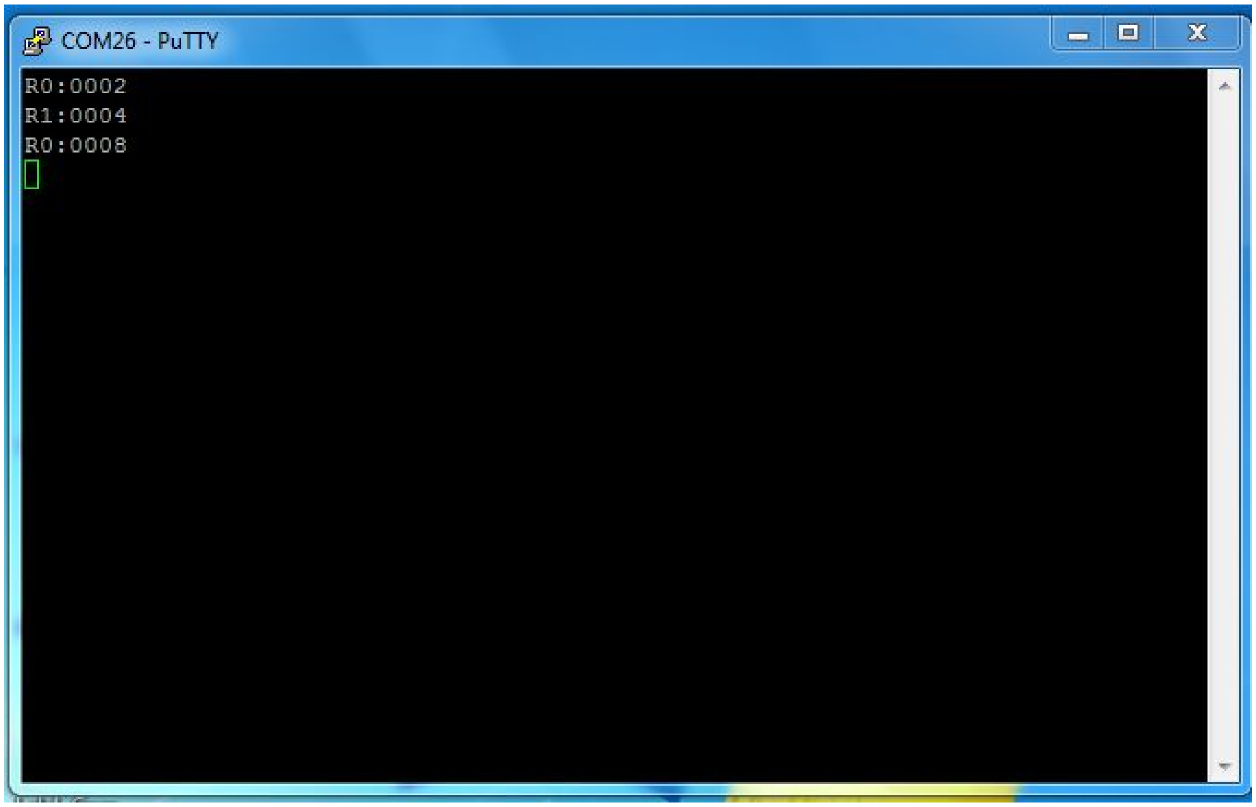
```
    end
  endtask // ”
```



3) Testing

Testing our project while adding various button functionality mainly consisted of connecting our UART port to the PC and using Putty to view the output. We tested various combinations of addition and multiplication for our first two modules and ensured that the send, execute, and reset buttons all functioned properly. To test the add and multiply modules we sent values to two registers and performed the arithmetic operation, outputting to UART at each step.

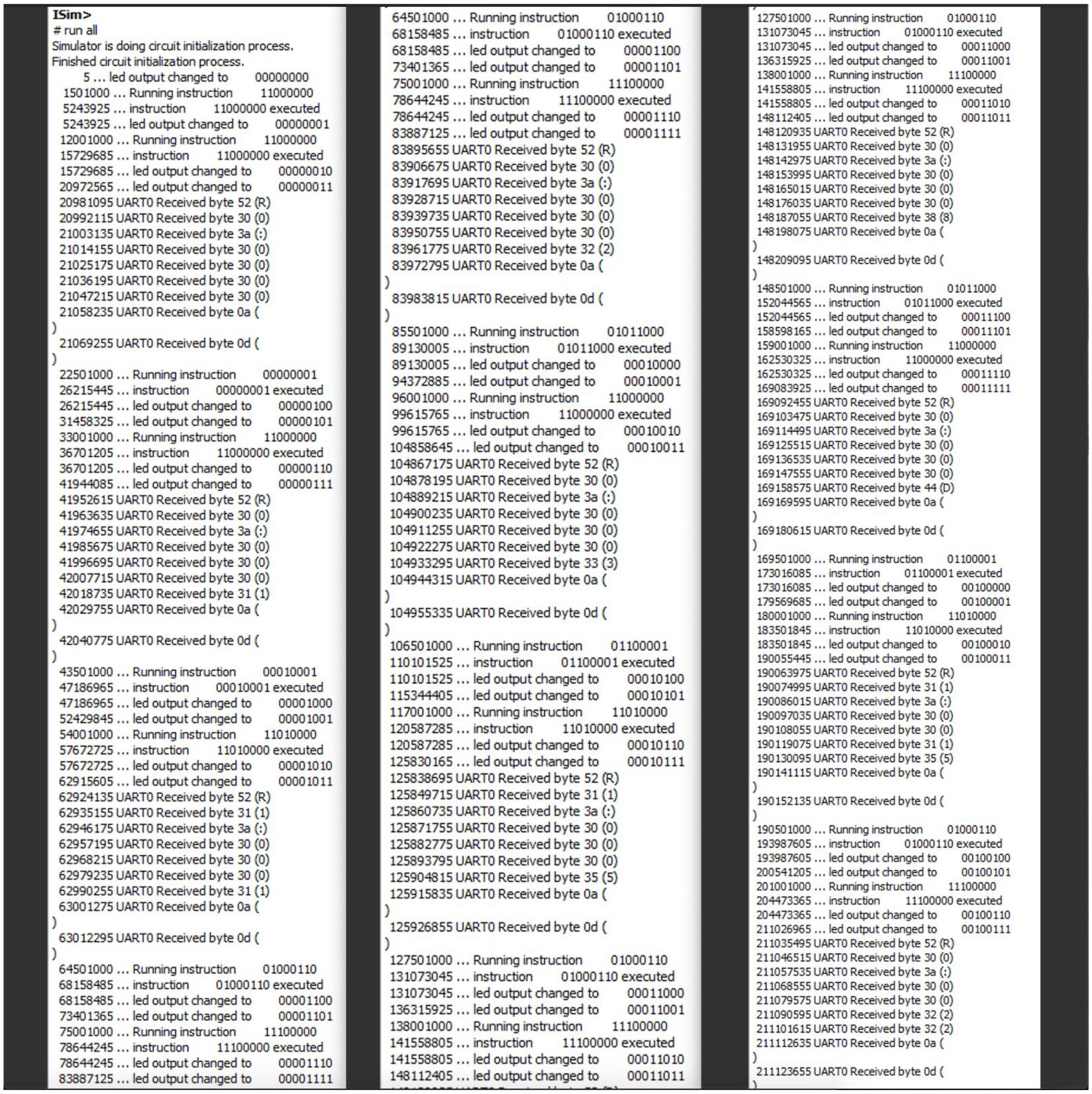
Here is the screenshot of putty to show our multiplication is working with the formatted UART output.



```
R0:0002
R1:0004
R0:0008
```

Loading the instructions from a text file and then modifying them to output a Fibonacci sequence were both tested via a testbench file and console output. Using our tb.v file we printed the 8-bit instruction executed, the register number, and the contents of the register for each instruction.

This is the screenshot of the first ten Fibonacci number:



```
ISim>
# run all
Simulator is doing circuit initialization process.
Finished circuit initialization process.
        5 ... led output changed to        00000000
  1501000 ... Running instruction        11000000
  5243925 ... instruction        11000000 executed
  5243925 ... led output changed to        00000001
 12001000 ... Running instruction        11000000
 15729685 ... instruction        11000000 executed
 15729685 ... led output changed to        00000010
 20972565 ... led output changed to        00000011
 20981095 UART0 Received byte 52 (R)
 20992115 UART0 Received byte 30 (0)
 21003135 UART0 Received byte 3a (:)
 21014155 UART0 Received byte 30 (0)
 21025175 UART0 Received byte 30 (0)
 21036195 UART0 Received byte 30 (0)
 21047215 UART0 Received byte 30 (0)
 21058235 UART0 Received byte 0a (
)
 21069255 UART0 Received byte 0d (
)
 22501000 ... Running instruction        00000001
 26215445 ... instruction        00000001 executed
 26215445 ... led output changed to        00000100
 31458325 ... led output changed to        00000101
 33001000 ... Running instruction        11000000
 36701205 ... instruction        11000000 executed
 36701205 ... led output changed to        00000110
 41944085 ... led output changed to        00000111
 41952615 UART0 Received byte 52 (R)
 41963635 UART0 Received byte 30 (0)
 41974655 UART0 Received byte 3a (:)
 41985675 UART0 Received byte 30 (0)
 41996695 UART0 Received byte 30 (0)
 42007715 UART0 Received byte 30 (0)
 42018735 UART0 Received byte 31 (1)
 42029755 UART0 Received byte 0a (
)
 42040775 UART0 Received byte 0d (
)
 43501000 ... Running instruction        00010001
 47186965 ... instruction        00010001 executed
 47186965 ... led output changed to        00001000
 52429845 ... led output changed to        00001001
 54001000 ... Running instruction        11010000
 57672725 ... instruction        11010000 executed
 57672725 ... led output changed to        00001010
 62915605 ... led output changed to        00001011
 62924135 UART0 Received byte 52 (R)
 62935155 UART0 Received byte 31 (1)
 62946175 UART0 Received byte 3a (:)
 62957195 UART0 Received byte 30 (0)
 62968215 UART0 Received byte 30 (0)
 62979235 UART0 Received byte 30 (0)
 62990255 UART0 Received byte 31 (1)
 63001275 UART0 Received byte 0a (
)
 63012295 UART0 Received byte 0d (
)
 64501000 ... Running instruction        01000110
 68158485 ... instruction        01000110 executed
 68158485 ... led output changed to        00001100
 73401365 ... led output changed to        00001101
 75001000 ... Running instruction        11100000
 78644245 ... instruction        11100000 executed
 78644245 ... led output changed to        00001110
 83887125 ... led output changed to        00001111
```

```
 64501000 ... Running instruction        01000110
 68158485 ... instruction        01000110 executed
 68158485 ... led output changed to        00001100
 73401365 ... led output changed to        00001101
 75001000 ... Running instruction        11100000
 78644245 ... instruction        11100000 executed
 78644245 ... led output changed to        00001110
 83887125 ... led output changed to        00001111
 83895655 UART0 Received byte 52 (R)
 83906675 UART0 Received byte 30 (0)
 83917695 UART0 Received byte 3a (:)
 83928715 UART0 Received byte 30 (0)
 83939735 UART0 Received byte 30 (0)
 83950755 UART0 Received byte 30 (0)
 83961775 UART0 Received byte 32 (2)
 83972795 UART0 Received byte 0a (
)
 83983815 UART0 Received byte 0d (
)
 85501000 ... Running instruction        01011000
 89130005 ... instruction        01011000 executed
 89130005 ... led output changed to        00010000
 94372885 ... led output changed to        00010001
 96001000 ... Running instruction        11000000
 99615765 ... instruction        11000000 executed
 99615765 ... led output changed to        00010010
104858645 ... led output changed to        00010011
104867175 UART0 Received byte 52 (R)
104878195 UART0 Received byte 30 (0)
104889215 UART0 Received byte 3a (:)
104900235 UART0 Received byte 30 (0)
104911255 UART0 Received byte 30 (0)
104922275 UART0 Received byte 30 (0)
104933295 UART0 Received byte 33 (3)
104944315 UART0 Received byte 0a (
)
104955335 UART0 Received byte 0d (
)
106501000 ... Running instruction        01100001
110101525 ... instruction        01100001 executed
110101525 ... led output changed to        00010100
115344405 ... led output changed to        00010101
117001000 ... Running instruction        11010000
120587285 ... instruction        11010000 executed
120587285 ... led output changed to        00010110
125830165 ... led output changed to        00010111
125838695 UART0 Received byte 52 (R)
125849715 UART0 Received byte 31 (1)
125860735 UART0 Received byte 3a (:)
125871755 UART0 Received byte 30 (0)
125882775 UART0 Received byte 30 (0)
125893795 UART0 Received byte 30 (0)
125904815 UART0 Received byte 35 (5)
125915835 UART0 Received byte 0a (
)
125926855 UART0 Received byte 0d (
)
127501000 ... Running instruction        01000110
131073045 ... instruction        01000110 executed
131073045 ... led output changed to        00011000
136315925 ... led output changed to        00011001
138001000 ... Running instruction        11100000
141558805 ... instruction        11100000 executed
141558805 ... led output changed to        00011010
148112405 ... led output changed to        00011011
```

```
127501000 ... Running instruction        01000110
131073045 ... instruction        01000110 executed
131073045 ... led output changed to        00011000
136315925 ... led output changed to        00011001
138001000 ... Running instruction        11100000
141558805 ... instruction        11100000 executed
141558805 ... led output changed to        00011010
148112405 ... led output changed to        00011011
148120935 UART0 Received byte 52 (R)
148131955 UART0 Received byte 30 (0)
148142975 UART0 Received byte 3a (:)
148153995 UART0 Received byte 30 (0)
148165015 UART0 Received byte 30 (0)
148176035 UART0 Received byte 30 (0)
148187055 UART0 Received byte 38 (8)
148198075 UART0 Received byte 0a (
)
148209095 UART0 Received byte 0d (
)
148501000 ... Running instruction        01011000
152044565 ... instruction        01011000 executed
152044565 ... led output changed to        00011100
158598165 ... led output changed to        00011101
159001000 ... Running instruction        11000000
162530325 ... instruction        11000000 executed
162530325 ... led output changed to        00011110
169083925 ... led output changed to        00011111
169092455 UART0 Received byte 52 (R)
169103475 UART0 Received byte 30 (0)
169114495 UART0 Received byte 3a (:)
169125515 UART0 Received byte 30 (0)
169136535 UART0 Received byte 30 (0)
169147555 UART0 Received byte 30 (0)
169158575 UART0 Received byte 44 (D)
169169595 UART0 Received byte 0a (
)
169180615 UART0 Received byte 0d (
)
169501000 ... Running instruction        01100001
173016085 ... instruction        01100001 executed
173016085 ... led output changed to        00100000
179569685 ... led output changed to        00100001
180001000 ... Running instruction        11010000
183501845 ... instruction        11010000 executed
183501845 ... led output changed to        00100010
190055445 ... led output changed to        00100011
190063975 UART0 Received byte 52 (R)
190074995 UART0 Received byte 31 (1)
190086015 UART0 Received byte 3a (:)
190097035 UART0 Received byte 30 (0)
190108055 UART0 Received byte 30 (0)
190119075 UART0 Received byte 31 (1)
190130095 UART0 Received byte 35 (5)
190141115 UART0 Received byte 0a (
)
190152135 UART0 Received byte 0d (
)
190501000 ... Running instruction        01000110
193987605 ... instruction        01000110 executed
193987605 ... led output changed to        00100100
200541205 ... led output changed to        00100101
201001000 ... Running instruction        11100000
204473365 ... instruction        11100000 executed
204473365 ... led output changed to        00100110
211026965 ... led output changed to        00100111
211035495 UART0 Received byte 52 (R)
211046515 UART0 Received byte 30 (0)
211057535 UART0 Received byte 3a (:)
211068555 UART0 Received byte 30 (0)
211079575 UART0 Received byte 30 (0)
211090595 UART0 Received byte 32 (2)
211101615 UART0 Received byte 32 (2)
211112635 UART0 Received byte 0a (
)
211123655 UART0 Received byte 0d (
)
```

4) Conclusion

This was our first project working closely with the hardware of the Nexys3 board, specifically its switches, buttons and I/O ports. Each module represented its own challenges and requirements. We first had to implement the missing multiply instruction. We closely followed the add module that was already implemented as a guideline, this required creating a seperate .v file. We then had to create a new send button, as the current design required the user to implement an OP code of 11 to be able to send. This is a poor design when we have other buttons we have at our disposal. The UART output was a little ambiguous as well as to which register was actually sending the values. So we added more states to be able to print the register number. We ran into the most issues with this section as we had trouble implementing the correct state transitions. This completed the hardware part of the lab and we continued on to the sequencer program and the Fibonacci program. We implemented the sequence program by loading the seq.code file into our testbench. The Fibonacci program also caused some issues for us as we removed the original action for sending an instruction by our new send button. Once we figured this out it printed as expected. Overall this lab was an interesting exercise to understand how to modify existing code, add new modules of our own, interface with the board's hardware and learn more about some of its I/O protocols.