**CS M152A / EE M116L**

**Lab 3: Adder Multiplier Sequencer**

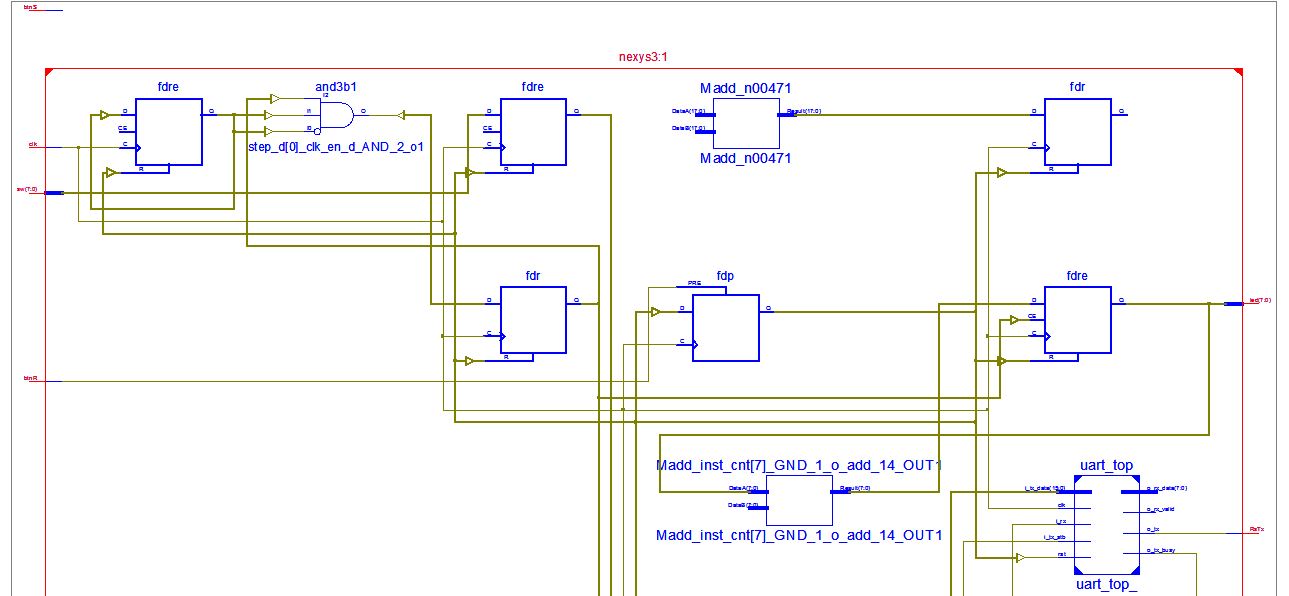
**Scott Shi**

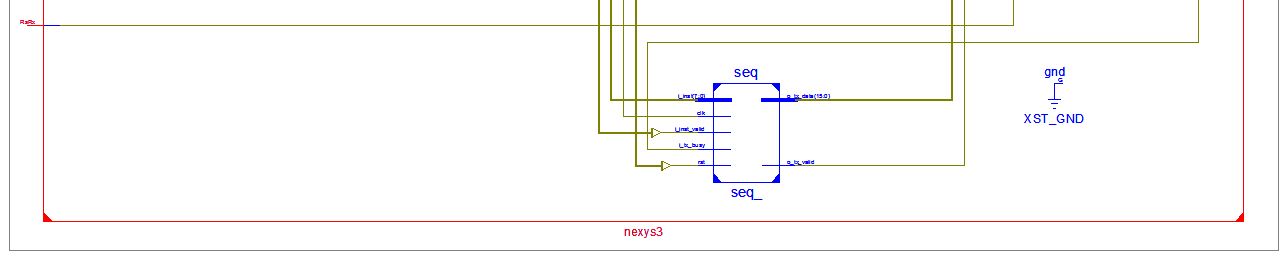
**Minh Le**

**INTRODUCTION**

In this lab, we play with the already provided code for us that allows the FPGA to be programmed as a mini finite state machine. The functionality it allows us to do is to put values in registers, add, multiply, and send them to the terminal. Although the usage of the complete program is quite simple (just flip switches to represent binary strings and press buttons to send instructions), the core of the lab revolves around understanding the code given to us and also making a few modifications to it. Our understanding of the code can be demonstrated by the questions we answered below. And the modifications were demoed in class.

**SCHEMATIC OF MODULE**



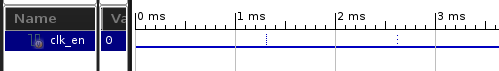


**Clock Enable (nexys3.v)**

1. What does ‘clk\_en’ do ?

‘clk\_en’ is a signal that used as a clock divider. After a certain amount of ticks on the clock, clk\_en will be high.

2. Add ‘clk\_en’ to the waveform, find the periodicity of this signal.



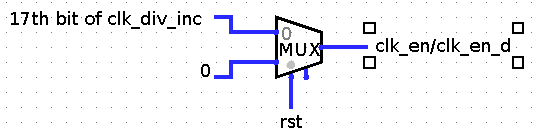
Here are two instances of clk\_en.

The code denotes that there’s a 763 Hz timing signal for the clock enable. This means that it has a period of 1/763 = a period of 1.31 ms / cycle.

3. What is the value of ‘clk\_dv’ during the clock cycle that ‘clk\_en’ is high?

The value of ‘clk\_dv’ during the clock cycle in which ‘clk\_en’ is high is 0.

4. Schematic



**Instruction Valid (nexys3.v)**

1. Write down the first simulation time interval (number) which expression:

inst\_vld = ~step\_d[0]&clk\_en\_d evaluates to 1. (line 100 of nexys3.v)

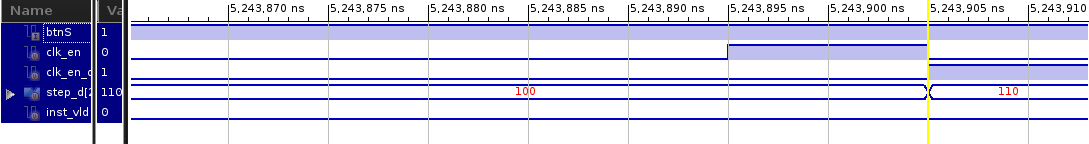
The first simulation time interval in which the expression evaluates to 1 is

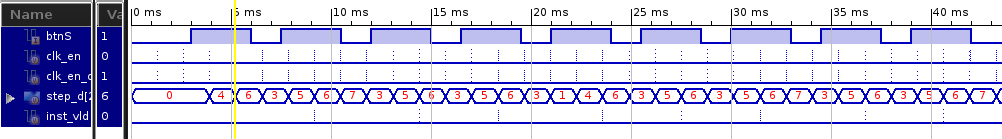
[5,243,915ns , 5,243,925ns] -- it spans 10 nanoseconds.

2. What is the purpose of clk\_en\_d signal in the above expression? Why not use clk\_en?

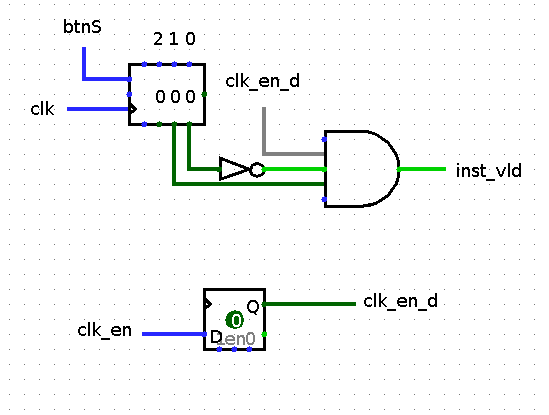
Button trigger is stored when clk\_en is high. We want to use the new values of step\_d that stores btnS whenever on the frequency of clk\_en but by then it is low. This delay uses the value of step\_d when clk\_en is high since clk\_en\_d delays by only 1 clock cycle.

3. Add ‘clk\_en’, ‘step\_d[1], step\_d[0], btnS, clk\_en\_d, and inst\_vld and capture timing relationship.





4. Schematic



**\*\*\*\*\*\*\*\*\*\*\***

Model\_uart.v

\*\*\*\*\*\*\*\*\*\*\*

reg [7:0] rxData;

reg [7:0] temp[3:0];

integer ind;

event evBit;

event evByte;

event evTxBit;

event evTxByte;

reg TX;

initial

begin

TX = 1'b1;

ind = 0;

end

always @ (negedge RX)

begin

rxData[7:0] = 8'h0;

#(0.5\*bittime);

repeat (8)

begin

#bittime ->evBit;

//rxData[7:0] = {rxData[6:0],RX};

rxData[7:0] = {RX,rxData[7:1]};

end

->evByte;

if(rxData != "\n" && rxData != "\r")

begin

temp[ind] = rxData;

ind = ind + 1;

end

else if(rxData == "\r")

begin

$display("%d %s recieved bytes(%s%s%s%s)", $stime, name, temp[0], temp[1], temp[2], temp[3]);

ind=0;

end

end

task tskRxData;

output [7:0] data;

begin

@(evByte);

data = rxData;

end

endtask // for

task tskTxData;

input [7:0] data;

reg [9:0] tmp;

integer i;

begin

tmp = {1'b1, data[7:0], 1'b0};

for (i=0;i<10;i=i+1)

begin

TX = tmp[i];

#bittime;

->evTxBit;

end

->evTxByte;

end

endtask // tskTxData

endmodule // model\_uart

**Register File (seq\_rf.v)**

1. What is the line of code where a register is written a non-zero value? Is this combinational or sequential logic?

A register is written a non-zero value in line 33.

The code is sequential logic block since it is a nonblocking assignment in an always block depending on clk.

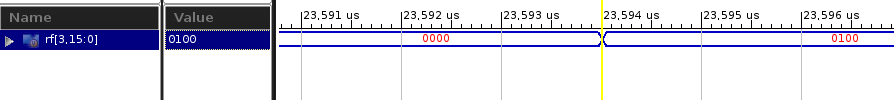
2. Find lines of code where register values are read out from register file. Sequential Combinational? If you were to manually implement the readout logic, what kind of logic elements would you use?

Values are read out from register file in lines 35-36..

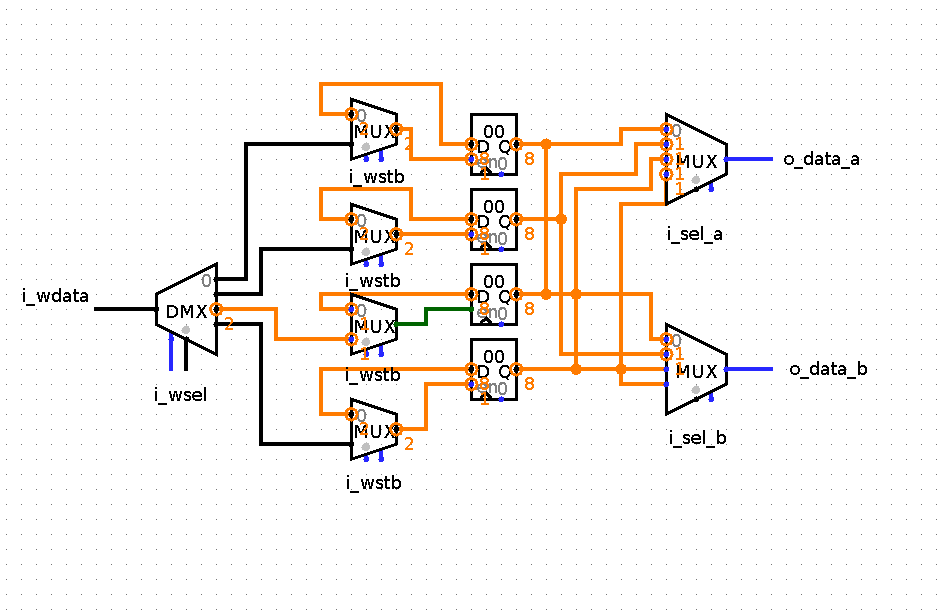
The code is combinational since they are wires.

If I needed to manually implement the readout logic, I would use combinational logic elements because there is no need to store values in extra hardware that is required to implement sequential logic. These values are already stored in an internal register.

3. Capture a waveform that shows first time register 3 is written w/ non-zero value



4. Draw a circuit diagram of the register file block.



**An Easier Way to Load Sequencer Program**

1. Identify the part of tb.v where the instructions are sent to the UUT.

Whenever tskRunInst is called, the 8 bits of instructions are sent to UUT to exectute.

2. Which user tasks are called in this process?

The tasks that call tskRunInst are tskRunPUSH, tskRunMULT, tskRunADD, and tskRunSEND.

**Reading from seq.code instead**

reg [7:0] instr\_array [0:1023];

$readmemb("seq.code", instr\_array);

for (i = 1; i <= instr\_array[0]; i = i + 1)

begin

tskRunInst(instr\_array[i]);

End

\*\*\*\*\*\*\*\*\*\*\*\*\*

Workshop insns

\*\*\*\*\*\*\*\*\*\*\*\*\*

1001

00000100  
00000000  
00010011  
10000110  
01100011  
1100  
1101  
1110  
1111

\*\*\*\*\*\*\*\*\*\*\*\*\*

Fibonacci insns

\*\*\*\*\*\*\*\*\*\*\*\*\*

10100

00000001  
00010001  
01000110  
01011011  
1100  
1101  
1110  
1111  
01101100  
01110001  
01000110  
01011011  
1100  
1101  
1110  
1111  
01101100  
01110001  
1100  
1101

**CONCLUSION**

In this lab, we got more practice with sequential and combinational code. We were able to identify, understand, and draw schematics for the many variables used in this mini-ALU code. A challenge we ran into when doing this lab was figuring out the necessity for clk\_en\_d in the line of code to evaluate “inst\_vld”. We thought that since it was non-blocking and sequential, it made sense to use clk\_en since its value would be offsetted by 1 anyways. Then we realized it made more sense to use clk\_en\_d because we want to know when clk\_en resets rather than when it’s high. All of our findings were described above in answering the questions.