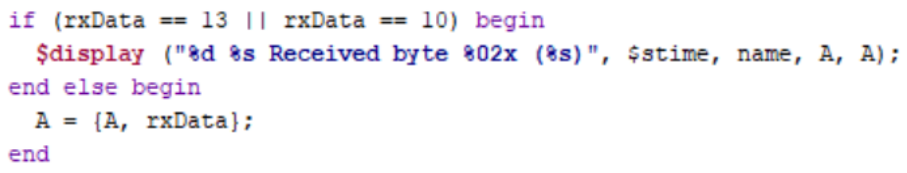
Myra Yamazaki
Haoxuan Li

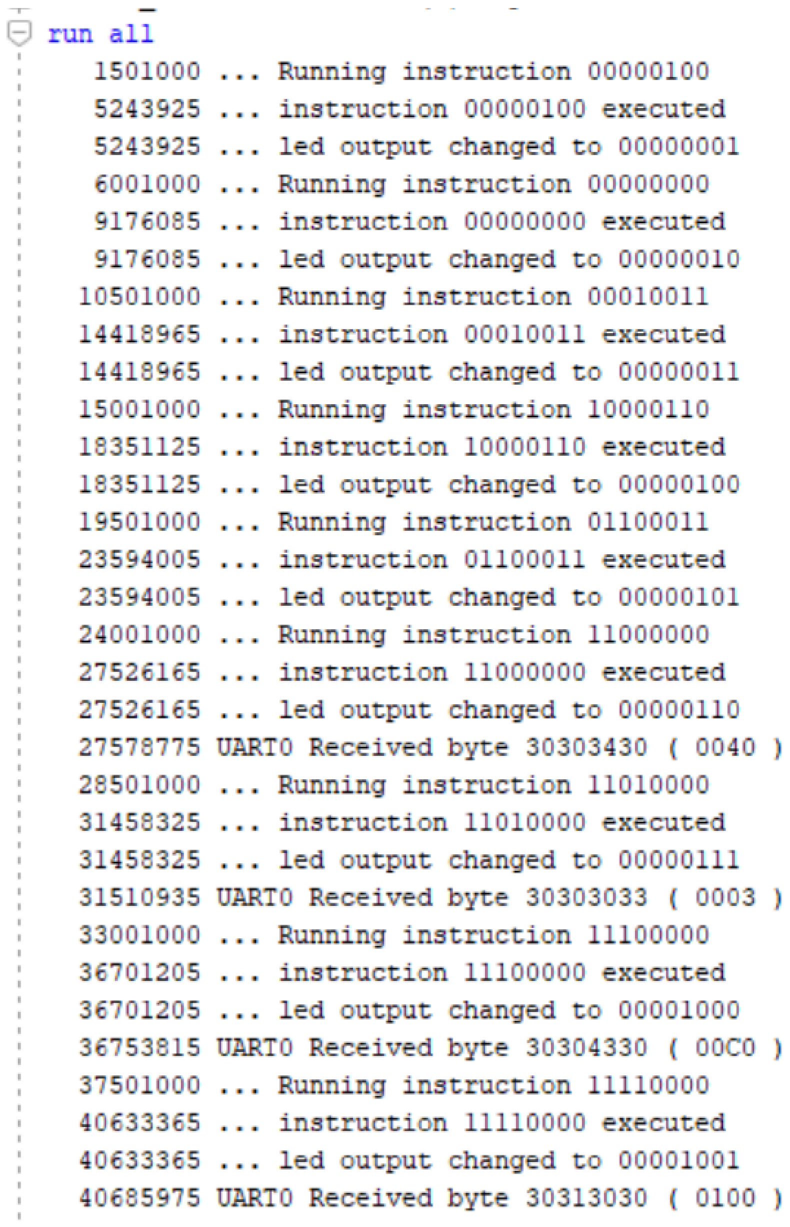
Lab 1

Workshop 2



```
if (rxData == 13 || rxData == 10) begin
  $display ("%d %s Received byte %02x (%s)", $stime, name, A, A);
end else begin
  A = {A, rxData};
end
```

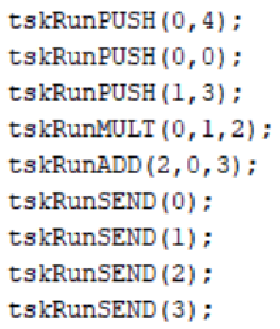
1. Identify the part of the tb.v where the instructions are sent to the UUT.



```
run all
   1501000 ... Running instruction 00000100
   5243925 ... instruction 00000100 executed
   5243925 ... led output changed to 00000001
   6001000 ... Running instruction 00000000
   9176085 ... instruction 00000000 executed
   9176085 ... led output changed to 00000010
  10501000 ... Running instruction 00010011
  14418965 ... instruction 00010011 executed
  14418965 ... led output changed to 00000011
  15001000 ... Running instruction 10000110
  18351125 ... instruction 10000110 executed
  18351125 ... led output changed to 00000100
  19501000 ... Running instruction 01100011
  23594005 ... instruction 01100011 executed
  23594005 ... led output changed to 00000101
  24001000 ... Running instruction 11000000
  27526165 ... instruction 11000000 executed
  27526165 ... led output changed to 00000110
  27578775 UART0 Received byte 30303430 ( 0040 )
  28501000 ... Running instruction 11010000
  31458325 ... instruction 11010000 executed
  31458325 ... led output changed to 00000111
  31510935 UART0 Received byte 30303033 ( 0003 )
  33001000 ... Running instruction 11100000
  36701205 ... instruction 11100000 executed
  36701205 ... led output changed to 00001000
  36753815 UART0 Received byte 30304330 ( 00C0 )
  37501000 ... Running instruction 11110000
  40633365 ... instruction 11110000 executed
  40633365 ... led output changed to 00001001
  40685975 UART0 Received byte 30313030 ( 0100 )
```

2. Which user tasks are called in this process?

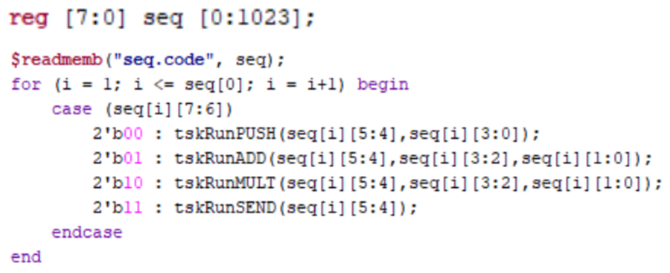
   PUSH, PUSH, PUSH, MULT, ADD, SEND, SEND, SEND, SEND



```
tskRunPUSH(0,4);
tskRunPUSH(0,0);
tskRunPUSH(1,3);
tskRunMULT(0,1,2);
tskRunADD(2,0,3);
tskRunSEND(0);
tskRunSEND(1);
tskRunSEND(2);
tskRunSEND(3);
```

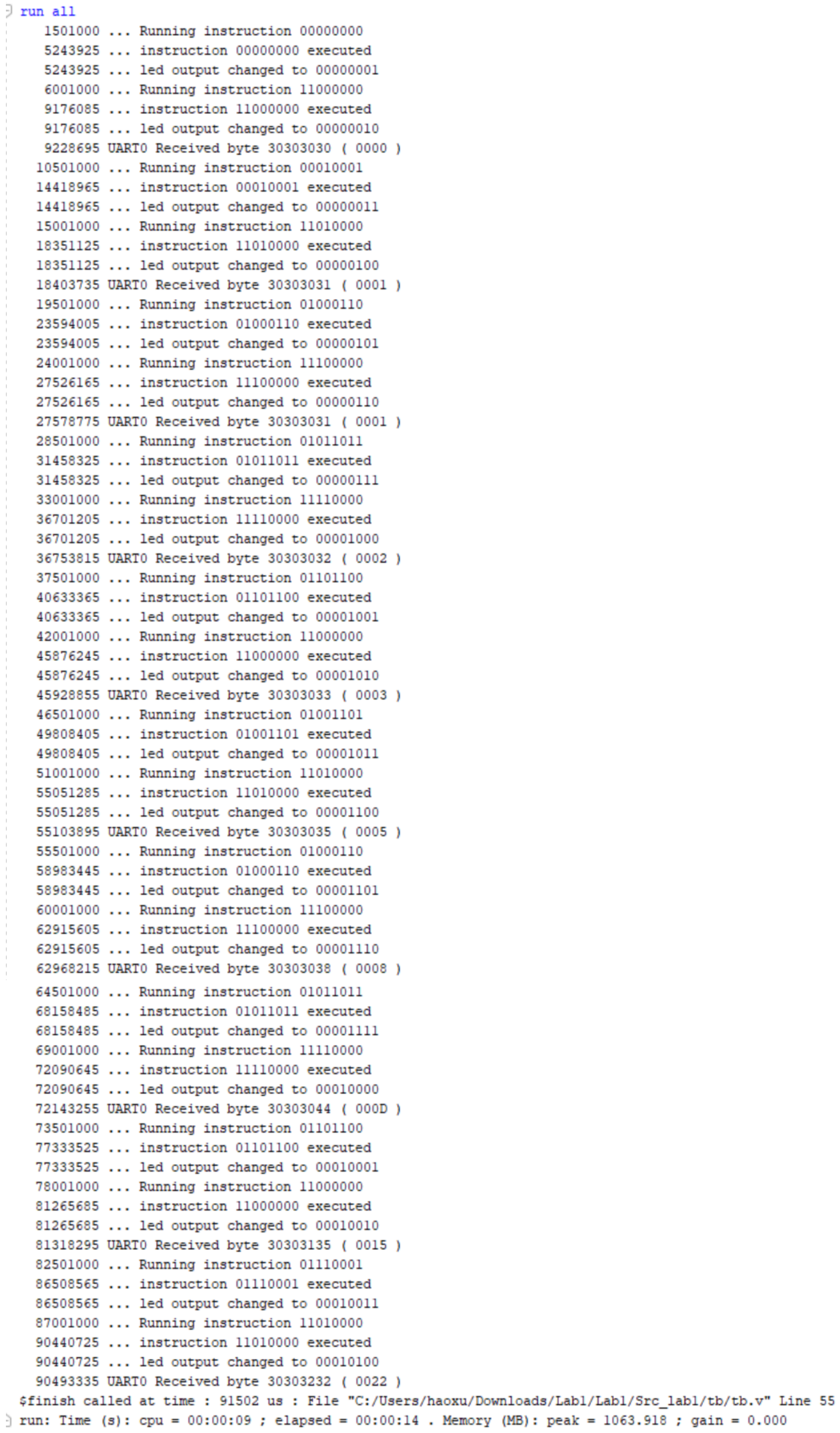
Fibonacci Numbers

R0 = 0	00000000
print R0	11000000
R1 = 1	00010001
print R1	11010000
R0+ R1 = R2 = 1	01000110
print R2	11100000
R1 + R2 = R3 = 2	01011011
print R3	11110000
R2 + R3 = R0 = 3	01101100
print R0	11000000
R0 + R3 = R1 = 5	01001101
print R1	11010000
R0 + R1 = R2 = 8	01000110
print R2	11100000
R1 + R2 = R3 = 13	01011011
print R3	11110000
R2 + R3 = R0 = 21	01101100
print R0	11000000
R3 + R0 = R1 = 34	01110001
print R1	1101000



```
reg [7:0] seq [0:1023];

$readmemb("seq.code", seq);
for (i = 1; i <= seq[0]; i = i+1) begin
    case (seq[i][7:6])
        2'b00 : tskRunPUSH(seq[i][5:4],seq[i][3:0]);
        2'b01 : tskRunADD(seq[i][5:4],seq[i][3:2],seq[i][1:0]);
        2'b10 : tskRunMULT(seq[i][5:4],seq[i][3:2],seq[i][1:0]);
        2'b11 : tskRunSEND(seq[i][5:4]);
    endcase
end
```

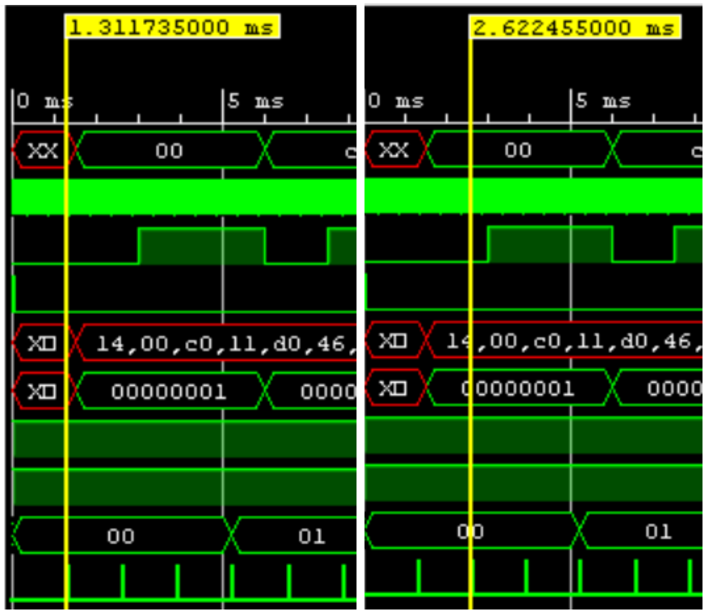


```
run all
    1501000 ... Running instruction 00000000
    5243925 ... instruction 00000000 executed
    5243925 ... led output changed to 00000001
    6001000 ... Running instruction 11000000
    9176085 ... instruction 11000000 executed
    9176085 ... led output changed to 00000010
    9228695 UART0 Received byte 30303030 ( 0000 )
   10501000 ... Running instruction 00010001
   14418965 ... instruction 00010001 executed
   14418965 ... led output changed to 00000011
   15001000 ... Running instruction 11010000
   18351125 ... instruction 11010000 executed
   18351125 ... led output changed to 00000100
   18403735 UART0 Received byte 30303031 ( 0001 )
   19501000 ... Running instruction 01000110
   23594005 ... instruction 01000110 executed
   23594005 ... led output changed to 00000101
   24001000 ... Running instruction 11100000
   27526165 ... instruction 11100000 executed
   27526165 ... led output changed to 00000110
   27578775 UART0 Received byte 30303031 ( 0001 )
   28501000 ... Running instruction 01011011
   31458325 ... instruction 01011011 executed
   31458325 ... led output changed to 00000111
   33001000 ... Running instruction 11110000
   36701205 ... instruction 11110000 executed
   36701205 ... led output changed to 00001000
   36753815 UART0 Received byte 30303032 ( 0002 )
   37501000 ... Running instruction 01101100
   40633365 ... instruction 01101100 executed
   40633365 ... led output changed to 00001001
   42001000 ... Running instruction 11000000
   45876245 ... instruction 11000000 executed
   45876245 ... led output changed to 00001010
   45928855 UART0 Received byte 30303033 ( 0003 )
   46501000 ... Running instruction 01001101
   49808405 ... instruction 01001101 executed
   49808405 ... led output changed to 00001011
   51001000 ... Running instruction 11010000
   55051285 ... instruction 11010000 executed
   55051285 ... led output changed to 00001100
   55103895 UART0 Received byte 30303035 ( 0005 )
   55501000 ... Running instruction 01000110
   58983445 ... instruction 01000110 executed
   58983445 ... led output changed to 00001101
   60001000 ... Running instruction 11100000
   62915605 ... instruction 11100000 executed
   62915605 ... led output changed to 00001110
   62968215 UART0 Received byte 30303038 ( 0008 )
   64501000 ... Running instruction 01011011
   68158485 ... instruction 01011011 executed
   68158485 ... led output changed to 00001111
   69001000 ... Running instruction 11110000
   72090645 ... instruction 11110000 executed
   72090645 ... led output changed to 00010000
   72143255 UART0 Received byte 30303044 ( 000D )
   73501000 ... Running instruction 01101100
   77333525 ... instruction 01101100 executed
   77333525 ... led output changed to 00010001
   78001000 ... Running instruction 11000000
   81265685 ... instruction 11000000 executed
   81265685 ... led output changed to 00010010
   81318295 UART0 Received byte 30303135 ( 0015 )
   82501000 ... Running instruction 01110001
   86508565 ... instruction 01110001 executed
   86508565 ... led output changed to 00010011
   87001000 ... Running instruction 11010000
   90440725 ... instruction 11010000 executed
   90440725 ... led output changed to 00010100
   90493335 UART0 Received byte 30303232 ( 0022 )
$finish called at time : 91502 us : File "C:/Users/haoxu/Downloads/Lab1/Lab1/Src_lab1/tb/tb.v" Line 55
run: Time (s): cpu = 00:00:09 ; elapsed = 00:00:14 . Memory (MB): peak = 1063.918 ; gain = 0.000
```

Workshop 1

Clock Dividers

1. Add clk_en to the simulation's waveform tab then run the simulation again. Use the cursor to find the periodicity of this signal (you can select the signal and use arrow keys to reach the exact edges). Capture a waveform picture that shows two occurences of clk_en, and include it in the lab report.



   Period is 1.311735ms

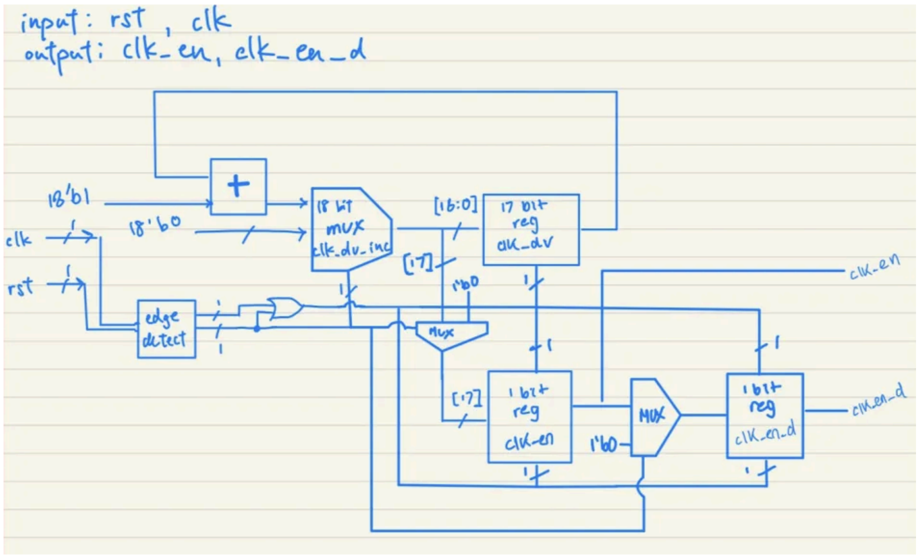
2. A duty cycle is the percentage of one period in which a signal or system is active: D = T/P * 100 %, where D is the duty cycle, T is the interval where the signal is high, and p is the period. What is the exact duty cycle of clk_en signal?

   D = (1*10^-5/1.31173)*100 = 0.00076%

3. What is the value of clk_dv signal during the clock cycle that clk_en is high?

   17'b00000000000000000

4. Draw a simple schematic/diagram of signals clk_dv, clk_en, and clk_en_d signals. It should be a translation of the corresponding Verilog code.



Debouncing

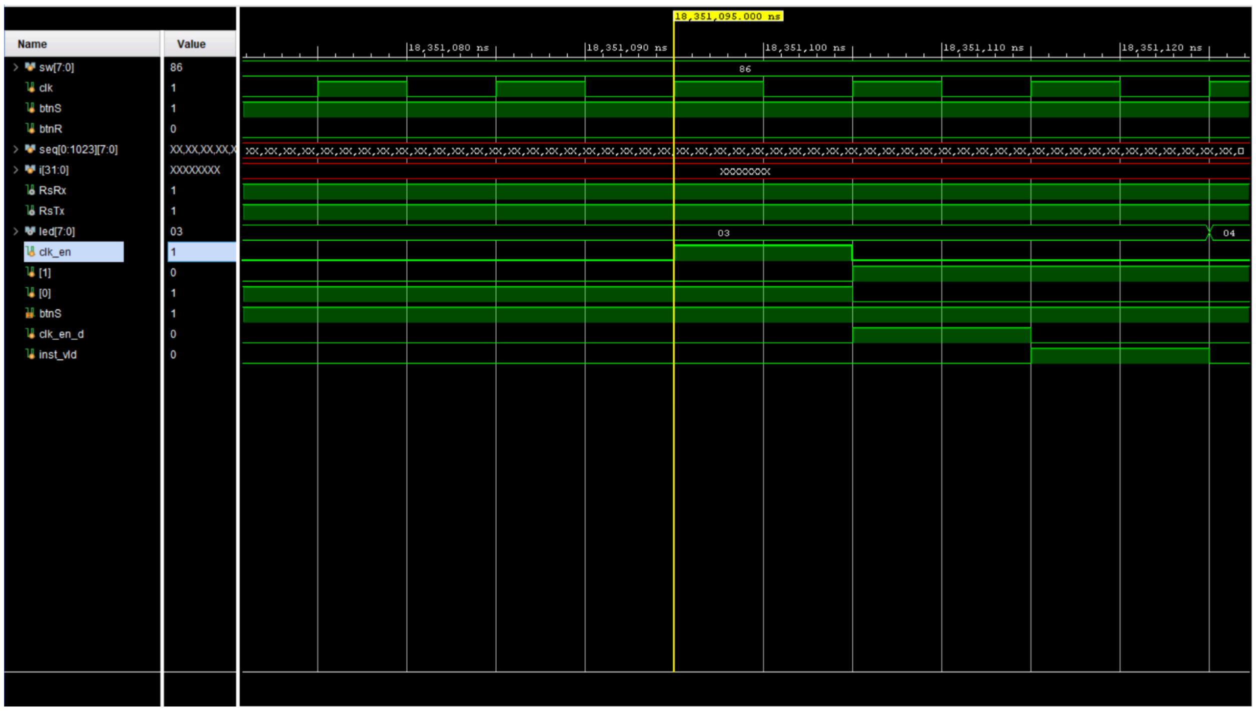
1. What is the purpose of clk_en_d signal when used in expression ~step_d[0] & step_d[1] & clk_en_d? Why don't we use clk_en?

   clk_en_d is a delayed version of clk_en that checks the validity of previously inputted instructions. Implementing clk_en instead would be incorrect as inst_vld would be unable to validate the instruction to a button press since the step_d register would updated on clk_en. clk_en_d is to ensure that we do not send double instructions on a button press.

2. Instead of clk_en <= clk_dv_inc[17], can we do clk_en <= clk_dv[16], making the duty cycle of clk_en 50%? Why?

   The expression clk_en <= clk_dv[16] would mean that clk_en is directly derived from the 17th bit of clk_dv. If this bit changes with every clock cycle, it would result in a clock enable signal with a 50% duty cycle. This means that the clock enables signal transitions from 0 to 1 and from 1 to 0 on every clock cycle. Using clk_dv_inc[16] could potentially incur the unwanted input noise from buttons. clk_dv_inc[17] was chosen such that there would be a higher chance of ignoring the initial and proceeding debounce of the buttons.

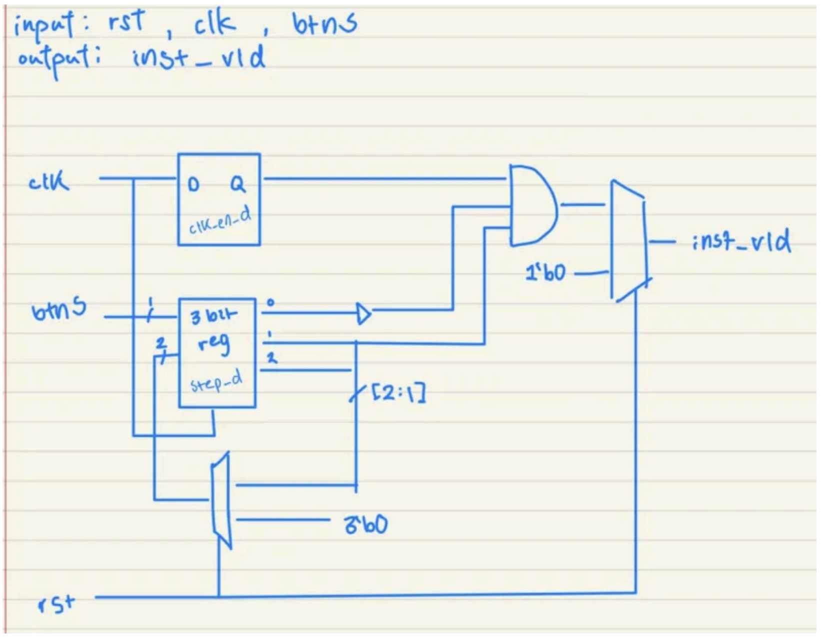
3. Include waveform captures that clearly show the timing relationship between clk_en, step_d[1], step_d[0], btnS, clk_en_d, and inst_vld.



Name	Value
sw[7:0]	86
clk	1
btnS	1
btnR	0
seq[0:1023][7:0]	XX,XX,XX,XX,X
i[31:0]	XXXXXXXX
RsRx	1
RsTx	1
led[7:0]	03
clk_en	1
[1]	0
[0]	1
btnS	1
clk_en_d	0
inst_vld	0

4. Draw a simple schematic/diagram of signals above. It should be a translation of the corresponding Verilog code.

input: rst, clk, btnS
output: inst_vld



Register File

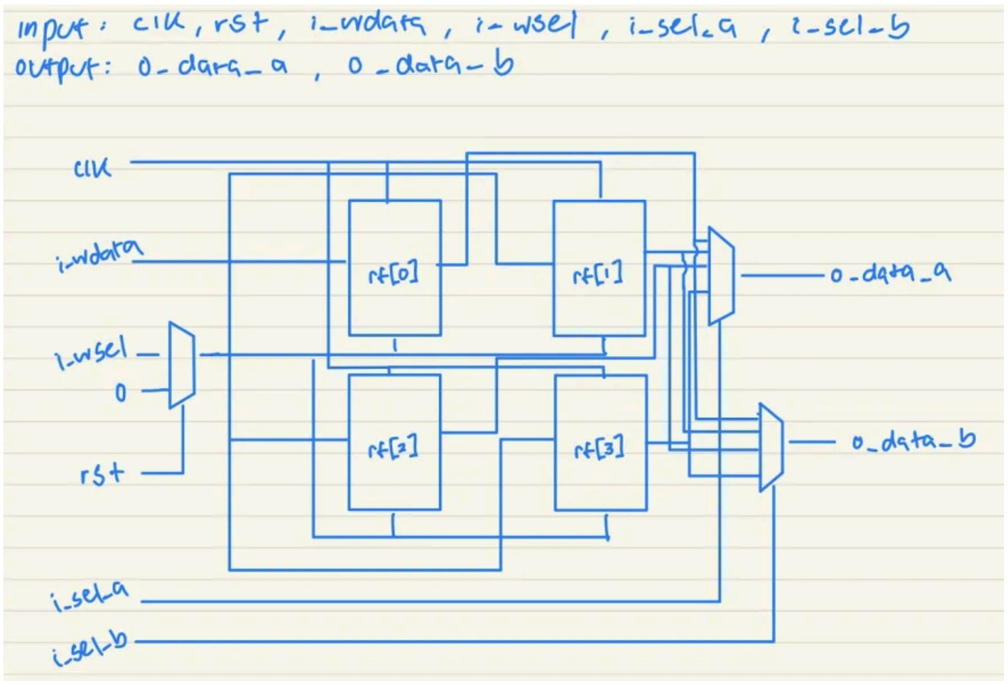
1. Find the line of code where a register is written a non-zero value. Is this sequential logic or combinatorial logic?

rf[i_wsel] <= i_wdata;
Sequential logic because D Flip-flops use sequential logic to store memory on a clock edge. We know this because the code is located inside an *always @ (posedge clk)* block.

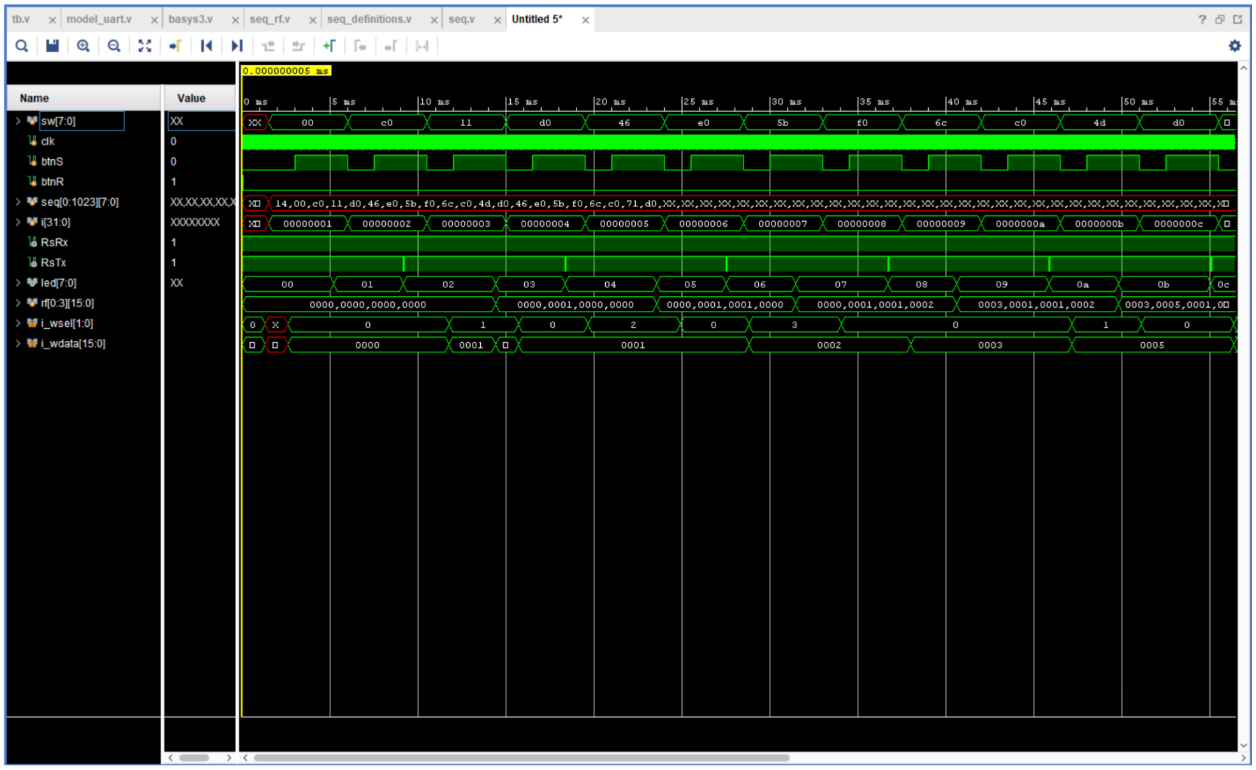
2. Find the lines of code where the register values are read out from the register file. Is this sequential or combinatorial logic? If you were to manually implement the readout logic, what kind of logic elements would you use?

assign o_data_a = rf[i_sel_a];
assign o_data_b = rf[i_sel_b];
Combinational logic, these registers are not under any clock signal and therefore can be selected with a multiplexer with a select signal for each.

3. Draw a circuit diagram of the register file block. It should be a translation of the corresponding Verilog code.


input: clk, rst, i_wdata, i_wsel, i_sel_a, i_sel_b
output: o_data_a, o_data_b
clk
i_wdata
i_wsel
0
rst
rf[0]
rf[1]
rf[2]
rf[3]
o_data_a
o_data_b
i_sel_a
i_sel_b

4. Capture a waveform that shows the first time register 3 is written with a non-zero value.


tb.v
model_uart.v
basys3.v
seq_rf.v
seq_definitions.v
seq.v
Untitled 5*
0.000000005 ms
Name
Value
0 ms 5 ms 10 ms 15 ms 20 ms 25 ms 30 ms 35 ms 40 ms 45 ms 50 ms 55 ms
sw[7:0] XX | XX 00 c0 11 d0 46 e0 5b f0 6c c0 4d d0
clk 0
btnS 0
btnR 1
seq[0:1023][7:0] XX,XX,XX,XX,X | 14,00,c0,11,d0,46,e0,5b,f0,6c,c0,4d,d0,46,e0,5b,f0,6c,c0,71,d0,XX,XX,XX,XX,XX,XX,XX,XX,XX,XX,XX,XX,XX,XX,XX,XX,XX,XX,XX,XX,XX,XX,XX,XX,XX,XX,XX,XX,XX,XX,XX
i[31:0] XXXXXXXX | 00000001 00000002 00000003 00000004 00000005 00000006 00000007 00000008 00000009 0000000a 0000000b 0000000c
RsRx 1
RsTx 1
led[7:0] XX | 00 01 02 03 04 05 06 07 08 09 0a 0b 0c
rf[0:3][15:0] | 0000,0000,0000,0000 0000,0001,0000,0000 0000,0001,0001,0000 0000,0001,0001,0002 0003,0001,0001,0002 0003,0005,0001,0
i_wsel[1:0] | 0 X 0 1 0 2 0 3 0 1 0
i_wdata[15:0] | 0000 0001 0001 0002 0003 0005