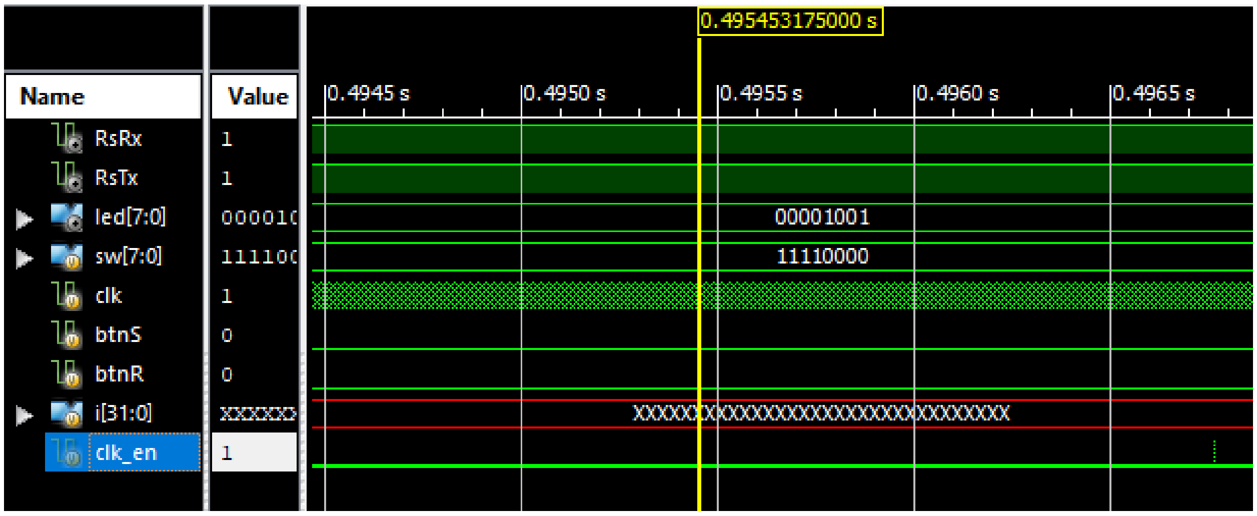
Group 6:
Sari Abu-Hamad
Adarsh Chilkunda
Drake Cote

Workshop 1:

Clock Dividers

1. Add clk_en to the simulation's waveform tab and 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 occurrences of clk_en, and include it in the lab report. Indicate the exact period of the signal in the report.

- Period: =0.496763895000 - 0.495453175000 = 0.00131072
- 

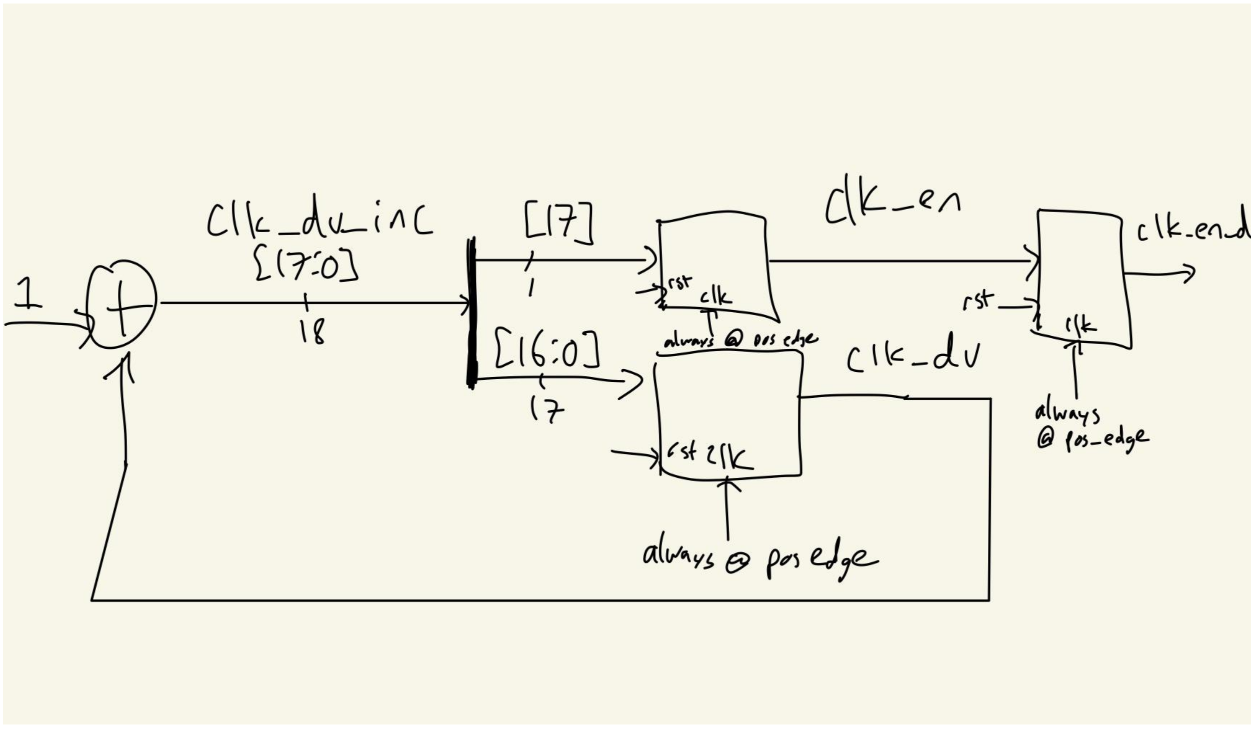
2. A duty cycle is the percentage of one period in which a signal or system is active: $D = T/P \times 100\%$, where D is the duty cycle, T is the interval where the signal is high, and p is the period. What is the exact duty cycle of clk_en Signal?

- 0.496763905000 - 0.496763895000 = T = 1 x 10^-8
- D = T/P x 100% = 1 x 10^-8 / 0.00131072 * 100% = 7.62939 x 10^-4

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

00000000000000000

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

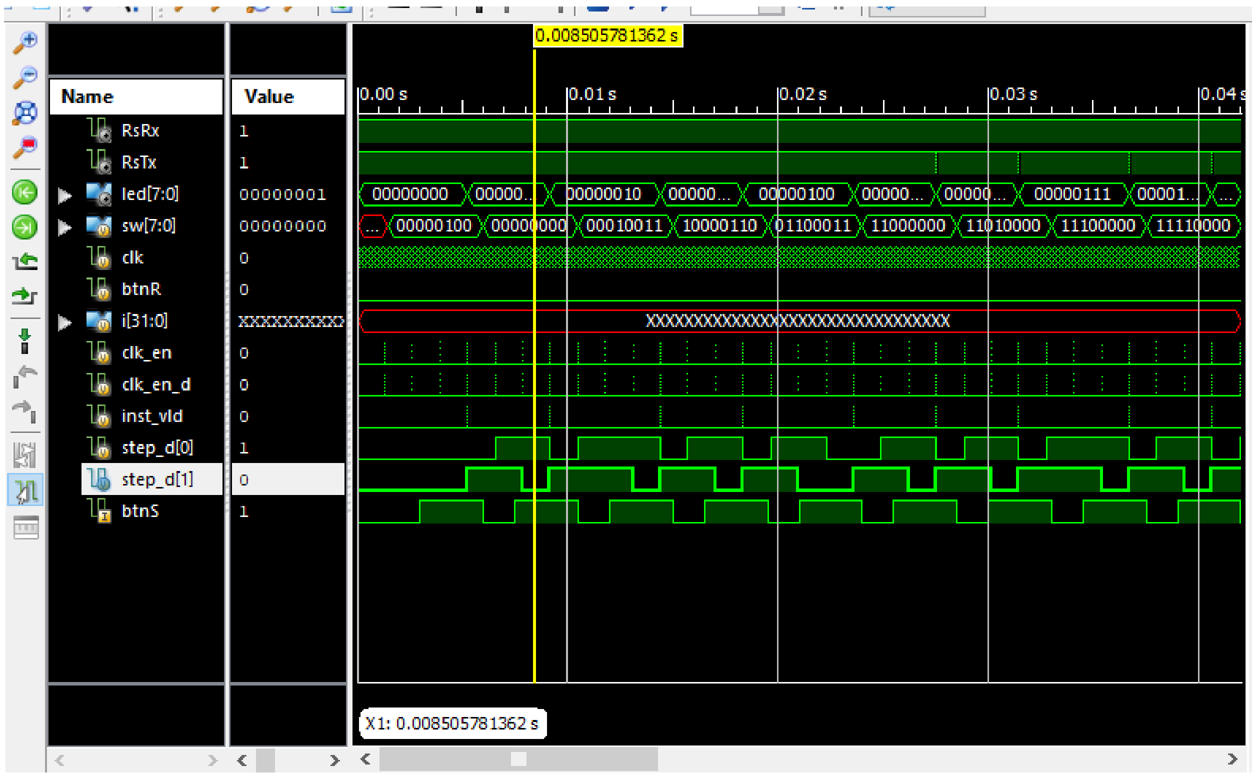
1. What is the purpose of clk_en_d signal when used in expression ~step_d[0] & step_d[1] & clk_en_d? Why don't we use clk_en?

The purpose of using the clk_en_d signal is to ensure inst_vld gets set to the correct value on the rising edge of the clock. If we set inst_vld to clk_en, it would read the wrong value because clk_en would not yet be high at the instant the clock goes high. Therefore, we use clock_en_d which stores the value of clock_en with a delay to ensure proper assignment.

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?

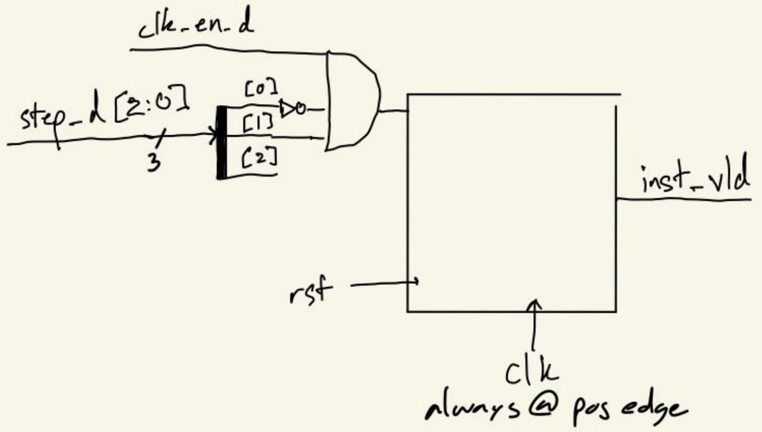
Yes since clk_dv_inc is assigned to clk_dv + 1, and thus when clk_dv overflows, the highest bit of clk_dv_inc (17) will be 1. The next cycle, clk_dv will overflow to 0, so clk_dv_inc will be 0 + 1 = 1 and the highest bit will be 0. This allows clk_en to be on for one clk cycle period, making the duty cycle 50%.

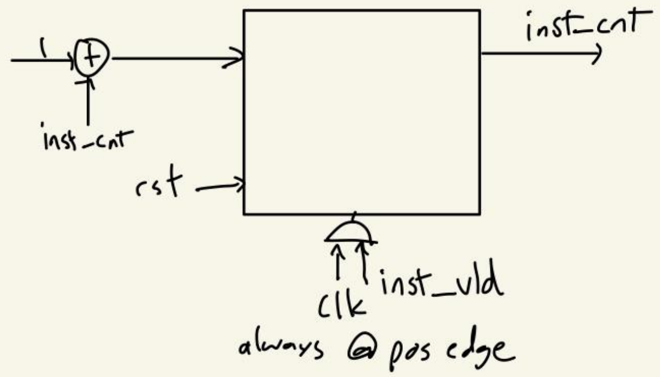
3. Include waveform captures that clearly show the timing relationship between clk_en, step_d[1], step_d[0], btnS, clk_en_d, and inst_vld.

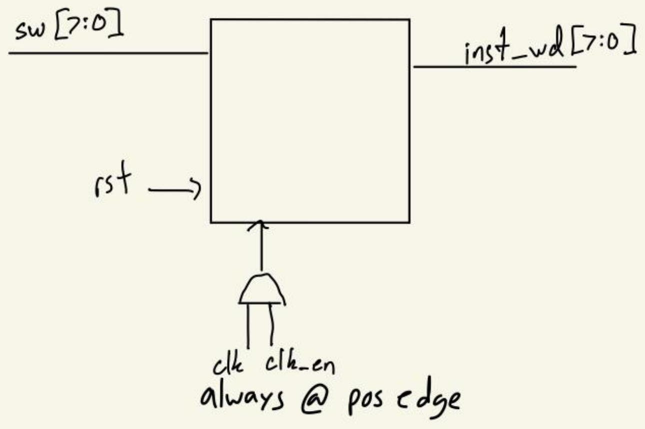


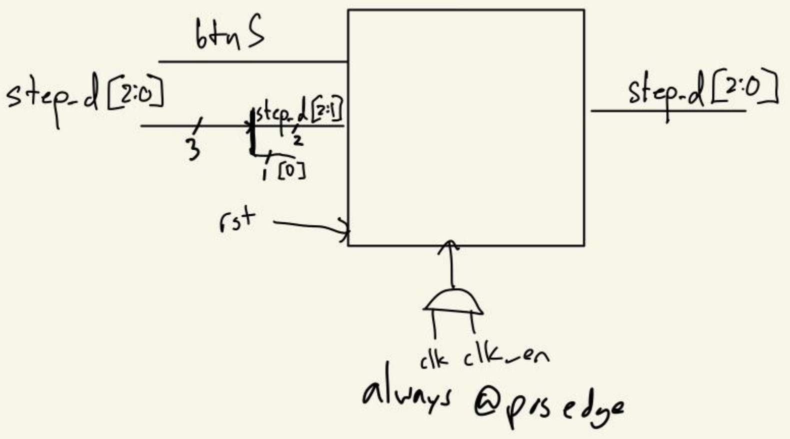
Name	Value
RsRx	1
RsTx	1
led[7:0]	00000001
sw[7:0]	00000000
clk	0
btnR	0
i[31:0]	XXXXXXXXXX
clk_en	0
clk_en_d	0
inst_vld	0
step_d[0]	1
step_d[1]	0
btnS	1

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


clk_en_d
step_d [2:0]
3
[0]
[1]
[2]
inst_vld
rst
clk
always @ pos edge


1
+
inst_cnt
inst_cnt
rst
clk
inst_vld
always @ pos edge


sw [7:0]
inst_wd [7:0]
rst
clk
clk_en
always @ pos edge


btnS
step_d [2:0]
3
step_d [2:1]
2
1 [0]
step_d [2:0]
rst
clk
clk_en
always @ pos edge

Register File

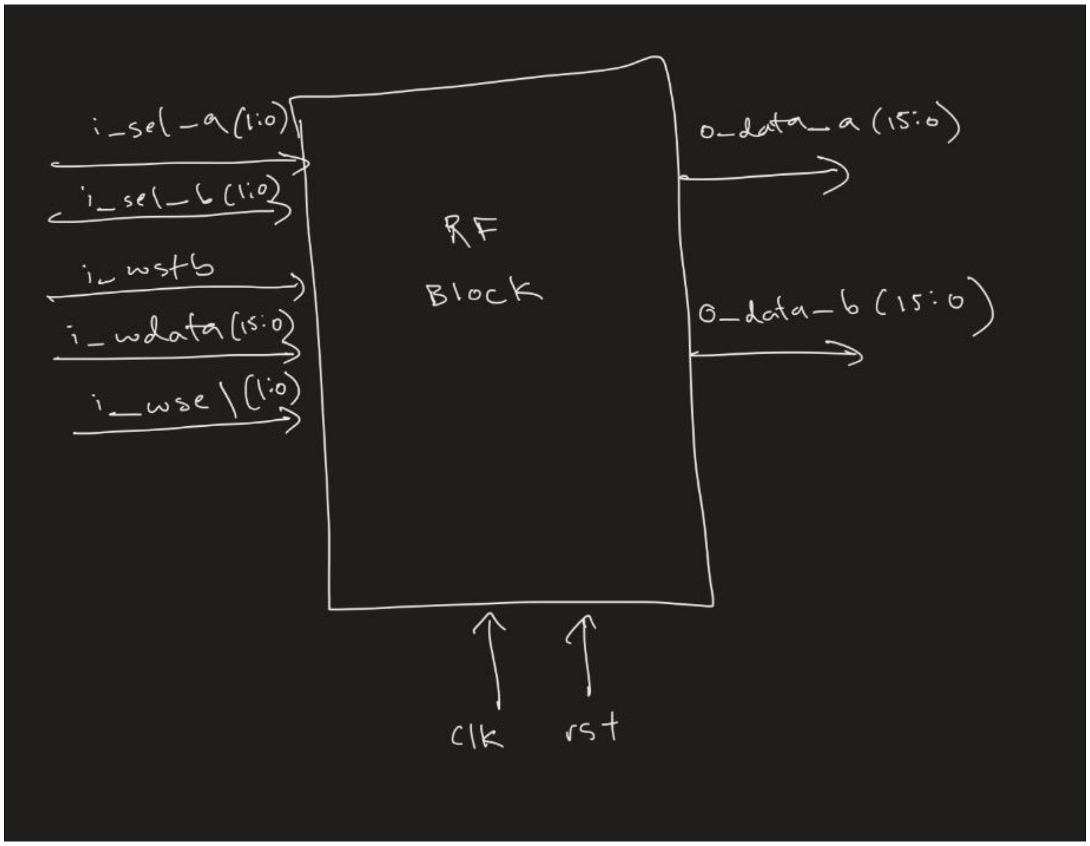
1. Find the line of code where a register is written a non-zero value. Is this sequential logic or combinatorial logic?

On line 33, register i_wsel gets set to i_wdata. This is sequential logic, because it is in an always @ (posedge clk) block. This only occurs when the clock is high.

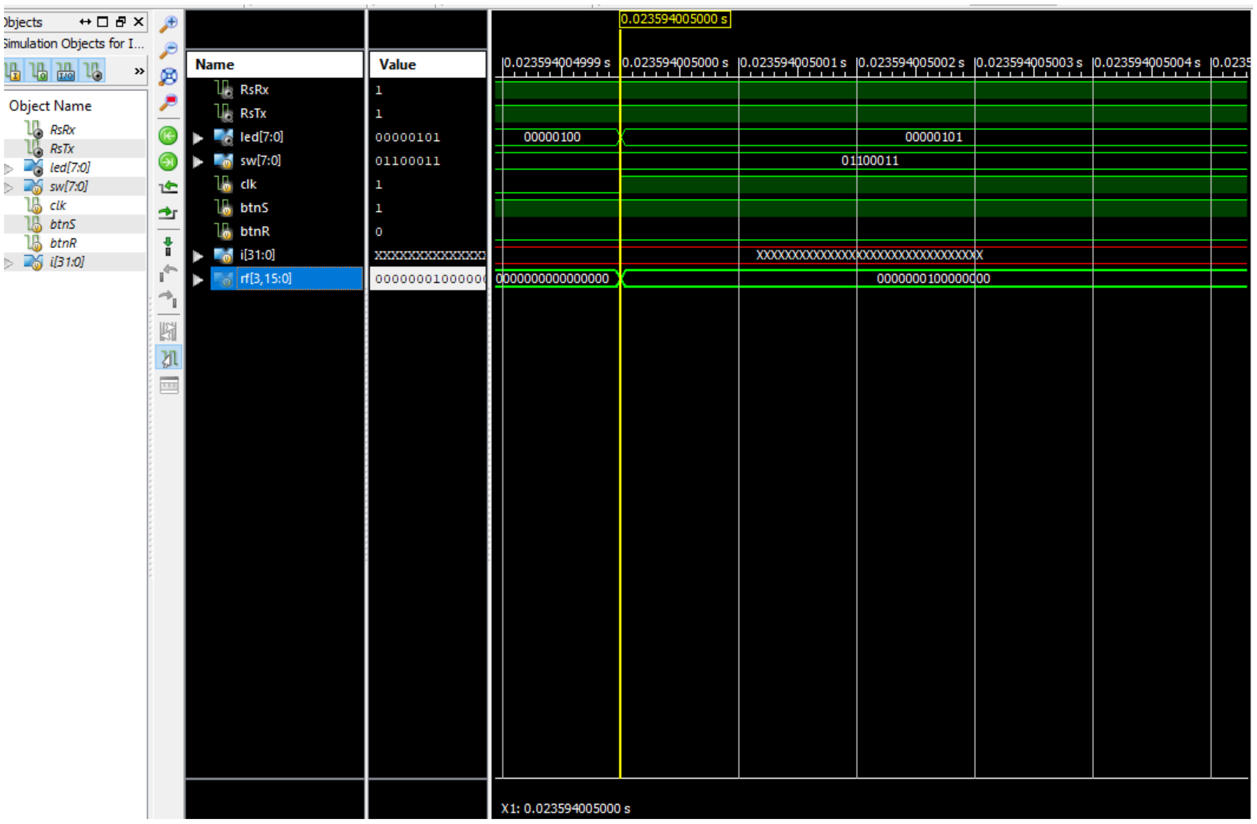
2. Find the lines of code where the register values are read out from the register file. Is this sequential or combinatorial logic? If you were to manually implement the readout logic, what kind of logic elements would you use?

The register values are read out on lines 35 and 36. This is combinational logic (it's not in the always @ posedge clk block and so doesn't depend on clock signals). If we were to manually implement the readout logic, we would use wires since o_data_a and o_data_b are assigned the values in the register file. We would also use multiplexers to select the registers from the register file to be read out.

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



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



Workshop 2

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

The instructions are sent to the UUT in the “initial” block of tb.v after the #1500000 wait; this occurs on lines 29-41 for the source code where all of the tskRun tasks are run.

2. Which user tasks are called in this process?

The user tasks tskRunPUSH, tskRunMULT, tskRunADD, and tskRunSEND are called in this process in order to send specific instructions to the UUT.