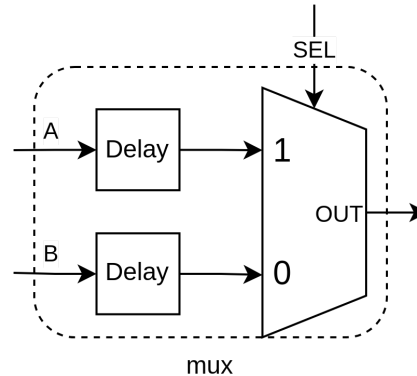


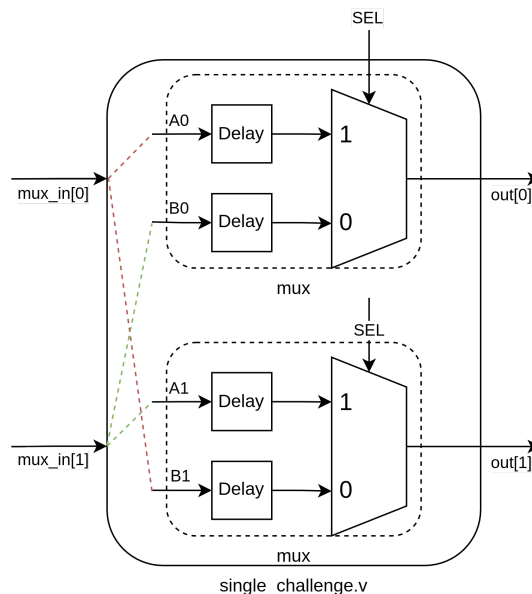
Part 1

Task 1-2: Creating an arbitrary PUF

1. The lower block used is the MUX with the process_delay, which was provided with the handout.



2. To take in a single challenge bit, another module named "singel_challenge.v" was created, which had the below-mentioned I/Os. The mux module was called twice within this module.
 - a. Input: 2-bit array "mux_in"
 - b. Output: 2-bit array "out"



- c. A test bench was created to evaluate the module, which provided the working proof of this module.

```

cen598@cse420-VirtualBox:~/Documents/project3/CSE598_Project3_Handout/project3_part1/modelsim$ ./run_test tb_single_challenge
#
# do load.do
# ../
# Errors: 0, Warnings: 0

# vsim -voptargs="+acc" -batch -quiet tb_single_challenge -do "run -all; quit" -L 220model_ver
# Start time: 00:05:37 on Oct 21,2023
#
# run -all
# TIME 1, Challenge: 0, MUX_IN 1: 0, MUX_IN 0: 1, OUT1: 1, OUT0: x
# TIME 2, Challenge: 0, MUX_IN 1: 0, MUX_IN 0: 1, OUT1: 1, OUT0: 0
# TIME 3, Challenge: 0, MUX_IN 1: 0, MUX_IN 0: 1, OUT1: 1, OUT0: 0
# TIME 4, Challenge: 0, MUX_IN 1: 0, MUX_IN 0: 1, OUT1: 1, OUT0: 0
# TIME 5, Challenge: 0, MUX_IN 1: 0, MUX_IN 0: 1, OUT1: 1, OUT0: 0
# TIME 6, Challenge: 0, MUX_IN 1: 0, MUX_IN 0: 1, OUT1: 1, OUT0: 0
# TIME 7, Challenge: 0, MUX_IN 1: 0, MUX_IN 0: 1, OUT1: 1, OUT0: 0
# TIME 8, Challenge: 0, MUX_IN 1: 0, MUX_IN 0: 1, OUT1: 1, OUT0: 0
# TIME 9, Challenge: 0, MUX_IN 1: 0, MUX_IN 0: 1, OUT1: 1, OUT0: 0
# TIME 10, Challenge: 0, MUX_IN 1: 1, MUX_IN 0: 0, OUT1: 1, OUT0: 0
# TIME 11, Challenge: 0, MUX_IN 1: 1, MUX_IN 0: 0, OUT1: 0, OUT0: 0
# TIME 12, Challenge: 0, MUX_IN 1: 1, MUX_IN 0: 0, OUT1: 0, OUT0: 1
# TIME 13, Challenge: 0, MUX_IN 1: 1, MUX_IN 0: 0, OUT1: 0, OUT0: 1
# TIME 14, Challenge: 0, MUX_IN 1: 1, MUX_IN 0: 0, OUT1: 0, OUT0: 1
# TIME 15, Challenge: 0, MUX_IN 1: 1, MUX_IN 0: 0, OUT1: 0, OUT0: 1
# TIME 16, Challenge: 0, MUX_IN 1: 1, MUX_IN 0: 0, OUT1: 0, OUT0: 1
# TIME 17, Challenge: 0, MUX_IN 1: 1, MUX_IN 0: 0, OUT1: 0, OUT0: 1
# TIME 18, Challenge: 0, MUX_IN 1: 1, MUX_IN 0: 0, OUT1: 0, OUT0: 1
# TIME 19, Challenge: 0, MUX_IN 1: 1, MUX_IN 0: 0, OUT1: 0, OUT0: 1
# TIME 20, Challenge: 1, MUX_IN 1: 0, MUX_IN 0: 1, OUT1: 0, OUT0: 1
# TIME 21, Challenge: 1, MUX_IN 1: 0, MUX_IN 0: 1, OUT1: 0, OUT0: 0
# TIME 22, Challenge: 1, MUX_IN 1: 0, MUX_IN 0: 1, OUT1: 0, OUT0: 1
# TIME 23, Challenge: 1, MUX_IN 1: 0, MUX_IN 0: 1, OUT1: 0, OUT0: 1
# TIME 24, Challenge: 1, MUX_IN 1: 0, MUX_IN 0: 1, OUT1: 0, OUT0: 1
# TIME 25, Challenge: 1, MUX_IN 1: 0, MUX_IN 0: 1, OUT1: 0, OUT0: 1
# TIME 26, Challenge: 1, MUX_IN 1: 0, MUX_IN 0: 1, OUT1: 0, OUT0: 1
# TIME 27, Challenge: 1, MUX_IN 1: 0, MUX_IN 0: 1, OUT1: 0, OUT0: 1
# TIME 28, Challenge: 1, MUX_IN 1: 0, MUX_IN 0: 1, OUT1: 0, OUT0: 1
# TIME 29, Challenge: 1, MUX_IN 1: 0, MUX_IN 0: 1, OUT1: 0, OUT0: 1
# TIME 30, Challenge: 1, MUX_IN 1: 1, MUX_IN 0: 0, OUT1: 0, OUT0: 1
# TIME 31, Challenge: 1, MUX_IN 1: 1, MUX_IN 0: 0, OUT1: 1, OUT0: 1
# TIME 32, Challenge: 1, MUX_IN 1: 1, MUX_IN 0: 0, OUT1: 1, OUT0: 0
# TIME 33, Challenge: 1, MUX_IN 1: 1, MUX_IN 0: 0, OUT1: 1, OUT0: 0
# TIME 34, Challenge: 1, MUX_IN 1: 1, MUX_IN 0: 0, OUT1: 1, OUT0: 0
# TIME 35, Challenge: 1, MUX_IN 1: 1, MUX_IN 0: 0, OUT1: 1, OUT0: 0
# TIME 36, Challenge: 1, MUX_IN 1: 1, MUX_IN 0: 0, OUT1: 1, OUT0: 0
# TIME 37, Challenge: 1, MUX_IN 1: 1, MUX_IN 0: 0, OUT1: 1, OUT0: 0
# TIME 38, Challenge: 1, MUX_IN 1: 1, MUX_IN 0: 0, OUT1: 1, OUT0: 0
# TIME 39, Challenge: 1, MUX_IN 1: 1, MUX_IN 0: 0, OUT1: 1, OUT0: 0
# ** Note: $stop : ../tb/tb_single_challenge.v(45)
# Time: 40 ps Iteration: 0 Instance: /tb_single_challenge
# Break in Module tb_single_challenge at ../tb/tb_single_challenge.v line 45
# quit

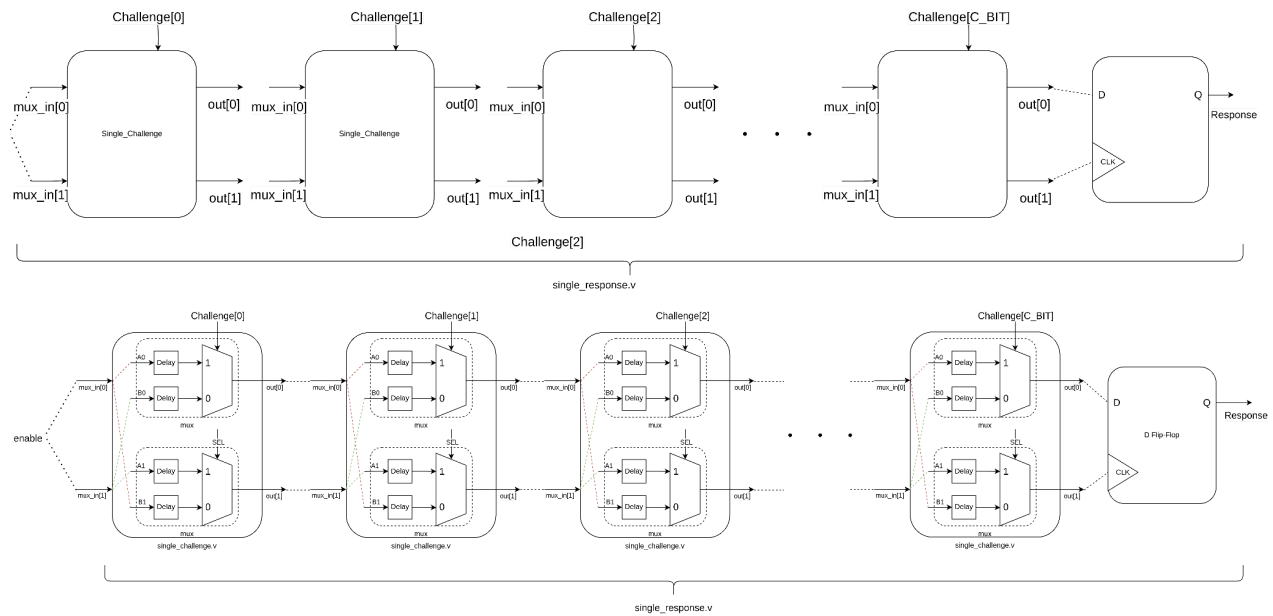
```

3. Next, the single_challenge module was called C_BITS times and was cascaded with a D-flip flop to obtain a single_response module.
 - a. Input: C_BIT(the challenge bits required), Delay (parsed to the single_challenge module)
 - b. Output: A single bit response

```

1 module single_response #(
2
3     parameter C_BITS = 4, // Challenge Bits
4     parameter [2*4:C_BITS-1:0] DELAY = 32'd12 //1 MUX needs 4bits, 8 MUX needs 32 bits
5
6 ) (
7     input [C_BITS-1:0] challenge, // The PUF challenge input
8     input enable, // Enable the PUF circuit
9     input reset,
10    output reg resp // The PUF response output, should
11 );
12
13 wire [2*C_BITS-1:0] mux_out;
14
15 single_challenge #(
16     .DELAY(DELAY[7:0])
17 ) single_challenge_0(
18     .challenge(challenge[0]), // The PUF challenge input
19     .mux_in(enable,enable),
20     .out({mux_out[1],mux_out[0]})
21 );
22
23 genvar i;
24
25 generate
26     for(i=1; i<C_BITS; i=i+1) begin
27         single_challenge #(
28             .DELAY(DELAY[8*(i+1)-1:8*i])
29         ) single_challenge_inst(
30             .challenge(challenge[i]), // The PUF challenge input
31             .mux_in({mux_out[(i*2)-1],mux_out[(i*2)-2]}),
32             .out({mux_out[(i*2)+1],mux_out[(i*2)]})
33         );
34     end
35 endgenerate
36
37 //D Flip-flop block
38
39 always @(posedge mux_out[2*C_BITS-1] or posedge reset)
40 begin
41     if (reset)
42         resp <= 1'b0;
43     else
44         resp <= mux_out[2*(C_BITS-1)];
45     end
46 end
47
48 endmodule

```



- c. The above image describes the single_response module.
- d. A test bench, "tb_single_response", was created to verify the module which produced the following result.
 - i. For this test bench, C_BIT=2 was considered along with a known 16-bit delay array was parsed, to verify the obtained and expected response easily.

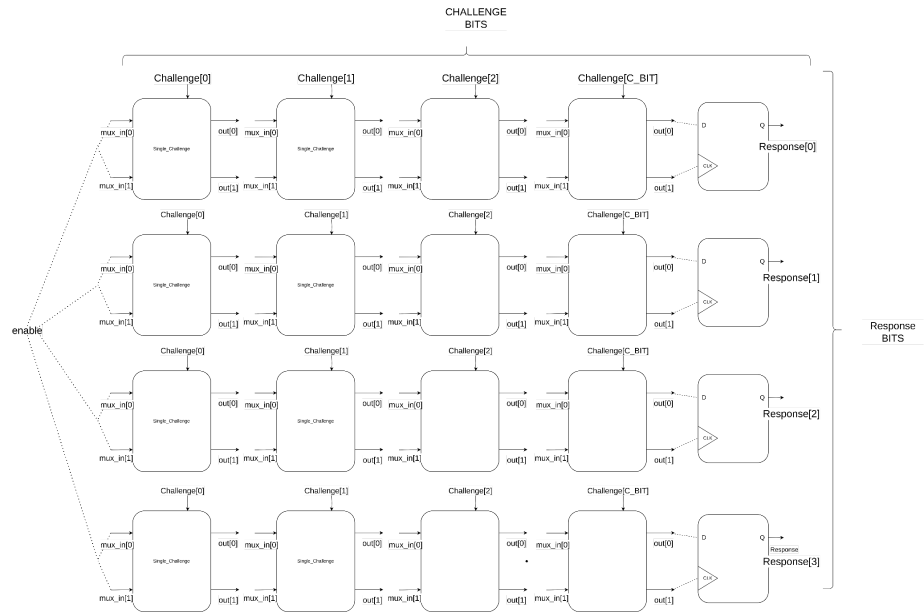
```

cen598@cse420-VirtualBox:~/Documents/project3/CSE598_Project3_Handout/project3_part1/modelsim$ ./run_test tb_single_response
#
# do load.do
# ../
# Errors: 0, Warnings: 0

# vsim -voptargs="+acc" -batch -quiet tb_single_response -do "run -all; quit" -L 220model_ver
# Start time: 23:11:28 on Oct 14,2023
#
# run -all
# Challenge: 00, Response: 0
# Challenge: 01, Response: 0
# Challenge: 10, Response: 1
# Challenge: 11, Response: 1
# ** Note: $stop      : ../tb/tb_single_response.v(98)
# Time: 248 ps Iteration: 0 Instance: /tb single response
# Break in Module tb_single_response at ../tb/tb_single_response.v line 98
# quit
# End time: 23:11:29 on Oct 14,2023, Elapsed time: 0:00:01
# Errors: 0, Warnings: 0
cen598@cse420-VirtualBox:~/Documents/project3/CSE598_Project3_Handout/project3_part1/modelsim$

```

4. For creating an arbitrary PUF, the single_response module was called R_BITS times.
 - a. The image below is the code for the arbiter_puf module
 - b. For testing, a 4x4 configuration was set in the test bench, and the delay array was generated using the provided Python script.



```

1 module arbiter_puf #(
2     parameter C_BITS = 4, // Challenge Bits
3     parameter R_BITS = 4, // Response Bits
4     parameter [2*4*C_BITS:R_BITS-1:0] DELAY = 4'd12 // Random Delay Values
5 ) (
6     input          reset, // Active-high reset to set all registers to 0
7     input          enable, // Enable the PUF circuit
8     input [C_BITS-1:0] challenge, // The PUF challenge input
9     output [R_BITS-1:0] resp // The PUF response output
10 );
11
12 //-----
13 // Your Code Here.
14 //-----
15 genvar i;
16
17 generate
18     for (i = 0; i < R_BITS; i = i + 1) begin
19         single_response #(
20             .C_BITS(C_BITS),
21             .DELAY(DELAY[32*(i+1)-1:32*i])
22         ) single_response_inst (
23             .challenge(challenge),
24             .enable(enable),
25             .reset(reset),
26             .resp(resp[i])
27         );
28     end
29 endgenerate
30
31 //-----
32 endmodule

```

```

cen598@cse420-VirtualBox:~/Documents/project3/CSE598_Project_Handout/project3_part1/modelsim$ ./run_test tb_arbiter_puf
#
# do load.do
# ../
# Errors: 0, Warnings: 0
#
# vsim -voptargs="+acc" -batch -quiet tb_arbiter_puf -do "run -all; quit" -L 220model_ver
# Start time: 00:16:45 on Oct 21,2023
#
# run -all
# Challenge: 0000, Response: 1110
# Challenge: 0001, Response: 1110
# Challenge: 0010, Response: 1101
# Challenge: 0011, Response: 1111
# Challenge: 0100, Response: 1111
# Challenge: 0101, Response: 0011
# Challenge: 0110, Response: 0110
# Challenge: 0111, Response: 1110
# Challenge: 1000, Response: 0111
# Challenge: 1001, Response: 0010
# Challenge: 1010, Response: 0010
# Challenge: 1011, Response: 0110
# Challenge: 1100, Response: 0110
# Challenge: 1101, Response: 1101
# Challenge: 1110, Response: 1101
# Challenge: 1111, Response: 0001
# ** Note: $stop : ../tb/tb_arbiter_puf.v(85)
# Time: 1504 ps Iteration: 0 Instance: /tb_arbiter_puf
# Break in Module tb_arbiter_puf at ../tb/tb_arbiter_puf.v line 85
# quit
# End time: 00:16:46 on Oct 21,2023, Elapsed time: 0:00:01
# Errors: 0, Warnings: 0
cen598@cse420-VirtualBox:~/Documents/project3/CSE598_Project_Handout/project3_part1/modelsim$

```

Task 3: Harvester PUF

- A Python script name, part1task3.py (within the modelsim directory) was created to harvest responses from a 4-challenge, 32-response bit configuration. This python would run 16 time and parse the newly generated delay param to the harvester to for the computing hamming distance in Task4.

```
gen598@cse420-VirtualBox:~/Documents/project3/CSE598_Project3_Hand
# do load.do
# ../
# Errors: 0, Warnings: 0

# vsim -voptargs="+acc" -batch -quiet tb_harvest_crp -do "run -all
# Start time: 00:31:32 on Oct 21,2023
#
# run -all
# Challenge: 0000, Response: 000011100001100010110110101110
# Challenge: 0001, Response: 101011110111000101110110011110
# Challenge: 0010, Response: 1011111001110100000111110110101
# Challenge: 0011, Response: 11001110000101000010111100001111
# Challenge: 0100, Response: 1111011011100001000111011101111
# Challenge: 0101, Response: 1101011010100001100011101000011
# Challenge: 0110, Response: 0101111010110001101110001110110
# Challenge: 0111, Response: 101111111110001110111000111110
# Challenge: 1000, Response: 111100001110010101101010100111
# Challenge: 1001, Response: 0111000010101011111010111000010
# Challenge: 1010, Response: 0111000010101011111000111010010
# Challenge: 1011, Response: 0011001111010011111000011110110
# Challenge: 1100, Response: 00101001000010101111001101010110
# Challenge: 1101, Response: 0011100101011101011100110111100
# Challenge: 1110, Response: 10111001010011000101001110001101
# Challenge: 1111, Response: 01111000000011000110101110000001
# ** Note: $stop : ../tb/tb_harvest_crp.v(87)
# Time: 1504 ps Iteration: 0 Instance: /tb_harvest_crp
# Break in Module tb_harvest_crp at ../tb/tb_harvest_crp.v line 87
# quit
# End time: 00:31:32 on Oct 21,2023, Elapsed time: 0:00:00
# Errors: 0, Warnings: 0
gen598@cse420-VirtualBox:~/Documents/project3/CSE598_Project3_Hand
```

```
#!/usr/bin/python3
import sys
import subprocess
import csv

response_list = []

with open('table.csv','w',newline='') as file:
    writer=csv.writer(file)

    for i in range(16):
        subprocess.call(["./delay.py", "4", "32", "delay_params.v"])
        result = subprocess.run(["./run_test", "tb_harvest_crp"], capture_output=True
                                ,text=True).stdout
        # print(result.split('# ** Note')[0].split('Response: '))
        data = result.split()
        # print(data)
        # print("Responses...:")
        response = [data[i+1] for i, word in enumerate(data) if word == 'Response:']
        response_list.append(response)
        writer.writerow(response)

print(response_list)
```

- The harvested 16 challenge-response pairs were stored in a CSV file name table.csv.

Task 4: Computing the Hamming distance

- For calculating the hamming distance between for both single-chip and multi-chip, a python file name part1task4.py created

- Which reads the table.csv file to get the 16 columns, each column representing a single harvest
- Single-chip hamming distance was calculated by counting the difference in the positional bits of unique pair of response within a single column(that is $^{16}C_2$ total combinations) and averaging gave the result.
 - Each line prints the single-chip hamming distance for a single harvest.

```
cen598@cse420-VirtualBox:~/Documents/project3/CSE598_Project3_Handout/project3_part1/modelsim$ ./part1task4.py
0.4901041666666667
0.4763020833333335
0.4958333333333335
0.465625
0.50390625
0.4885416666666667
0.4958333333333335
0.4875
0.4763020833333335
0.4713541666666667
0.49609375
0.4856770833333333
0.48359375
0.4703125
0.4880208333333335
0.4916666666666667
cen598@cse420-VirtualBox:~/Documents/project3/CSE598_Project3_Handout/project3_part1/modelsim$
```

Part 2

Task 1: Creating a RO_basic block

- The Basic structure of the RO block is a NAND Gate cascaded with 8 NOT Gates in the

```
1 module RO_basic #(
2     parameter not_gate = 9,
3     parameter DELAY=1)
4 (
5     input enable,
6     output RO_out
7 );
8 //-----
9 // Your Logic Here
10 //-----
11
12 (*keep*) wire [not_gate:0] ring_io;
13
14 nand nand_1(ring_io[0], enable, ring_io[not_gate-1]);
15
16 genvar i;
17 generate
18     for(i = 0; i < not_gate; i = i + 1) begin: not
19         (*keep*) not not_inst(ring_io[i+1],ring_io[i]);
20         // (*keep*) assign #(DELAY) ring_io[i+1] = ~ring_io[i];
21     end
22 endgenerate
23
24 assign RO_out = ring_io[not_gate]; // RO basic module with
25 // assign RO_out = ring_io[not_gate]; // RO basic module wi
26 //(*keep*) not not_out(RO_out, ring_io[not_gate]); // RO_bas
27 // inverted outout
28
29 //-----
30 endmodule
```

- For testing using the “./run_test” script additional parameters were provided

- “not_gate” param is the total number of inverters within the RO_basic module.

- It is essential that the total inverter(including the NAND) remains an odd number.

- The “Keep” command is used to maintain those odd numbers of inverters, otherwise the compiler would try to cancel out an even number of inverted to minimise the no of resources for optimisation.

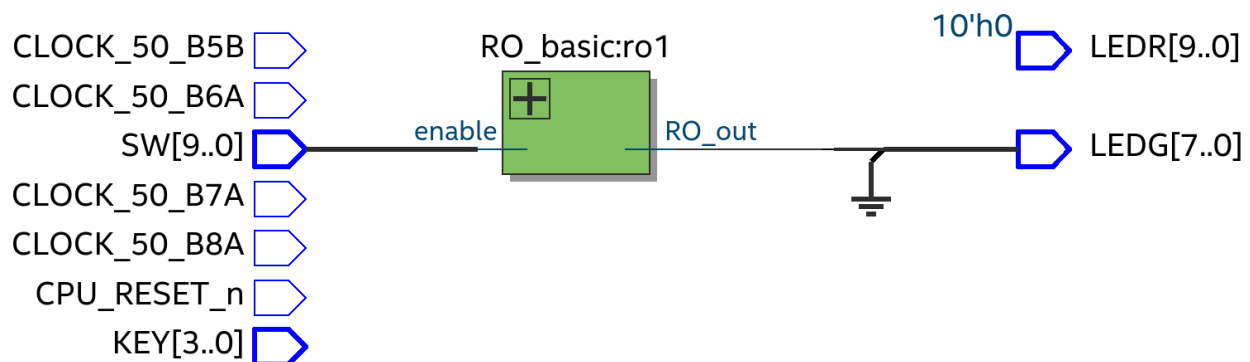
```

1 module cyclone_v_top (
2     ///////////////////////////////////////////////////
3     input          CLOCK_50_B5B,
4     input          CLOCK_50_B6A,
5     input          CLOCK_50_B7A,
6     input          CLOCK_50_B8A,
7
8     ///////////////////////////////////////////////////
9     input          CPU_RESET_n,
10
11    ///////////////////////////////////////////////////
12    input          [3:0] KEY,
13
14    ///////////////////////////////////////////////////
15    output         [7:0] LEDG,
16
17    ///////////////////////////////////////////////////
18    output         [9:0] LEDR,
19
20    ///////////////////////////////////////////////////
21    input          [9:0] SW
22
23 );
24 //-----
25 // Your Logic Here
26 //-----
27
28 RO_basic #(
29     .not_gate(9),
30     .DELAY(1)) ro1(SW[0], LEDG[0]);
31
32

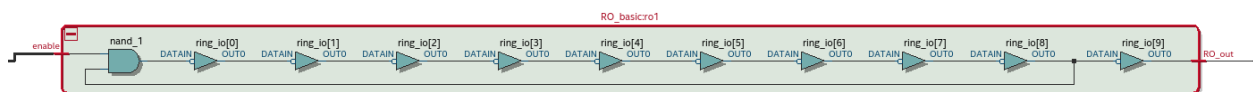
```

- For testing and verification purposes, the first methods done was deploying the RO_basic module on to the FPGA.
 - It was done by calling the module from the cyclone_v_top script as shown in the figure below.
 - Switch and LED were configured to set the input and the output respectively.

- The generated RTL schematic looks as shown in the figure below.
 - A total number of 9 inverters were used for the ring oscillator



- However an additional NOT gate was cascaded with the ring oscillator for fully utilising all the 10 LUT within a single cell.



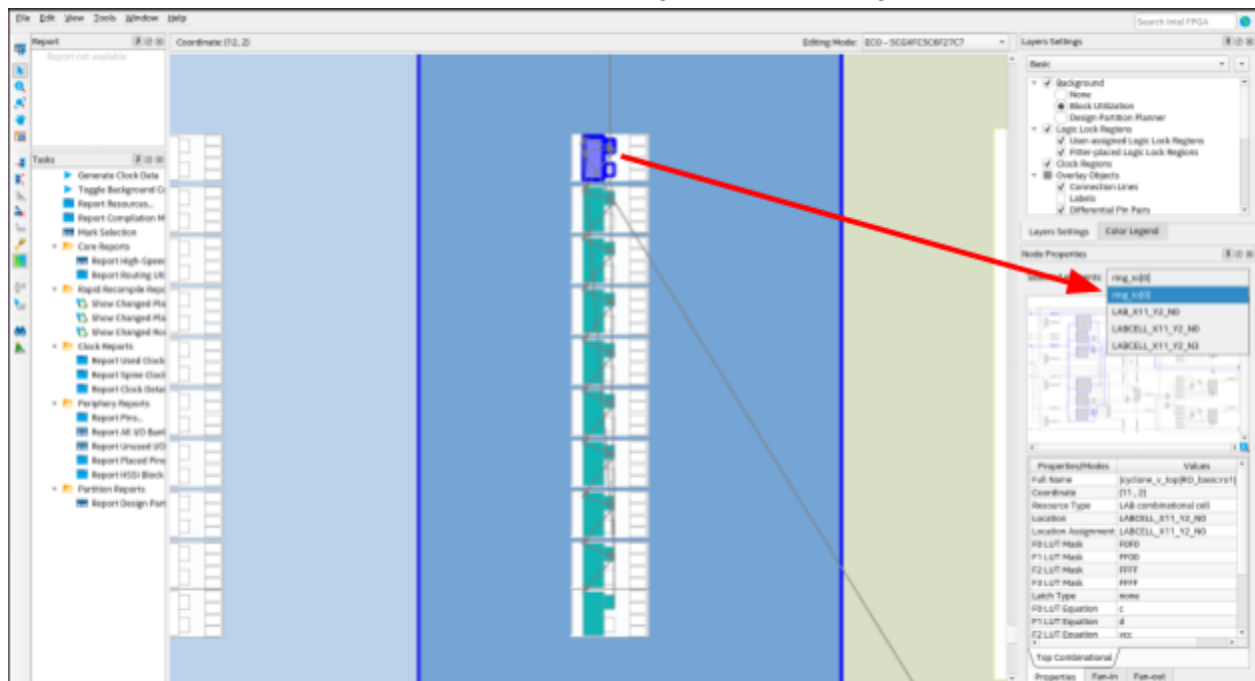
- The placement was achieved by changing the example.tcl file
 - running it using the Tools → ‘tcl script’ after it has been compiled
 - Making sure that the already compiled and place wire were removed from the assignment editor.

```

1 set_location_assignment LABCELL_X11_Y2_N0 -to "R0_basic:ro1|ring_io[0]"
2 set_location_assignment LABCELL_X11_Y2_N6 -to "R0_basic:ro1|ring_io[1]"
3 set_location_assignment LABCELL_X11_Y2_N12 -to "R0_basic:ro1|ring_io[2]"
4 set_location_assignment LABCELL_X11_Y2_N18 -to "R0_basic:ro1|ring_io[3]"
5 set_location_assignment LABCELL_X11_Y2_N24 -to "R0_basic:ro1|ring_io[4]"
6 set_location_assignment LABCELL_X11_Y2_N30 -to "R0_basic:ro1|ring_io[5]"
7 set_location_assignment LABCELL_X11_Y2_N36 -to "R0_basic:ro1|ring_io[6]"
8 set_location_assignment LABCELL_X11_Y2_N42 -to "R0_basic:ro1|ring_io[7]"
9 set_location_assignment LABCELL_X11_Y2_N48 -to "R0_basic:ro1|ring_io[8]"
10 set_location_assignment LABCELL_X11_Y2_N54 -to "R0_basic:ro1|ring_io[9]"

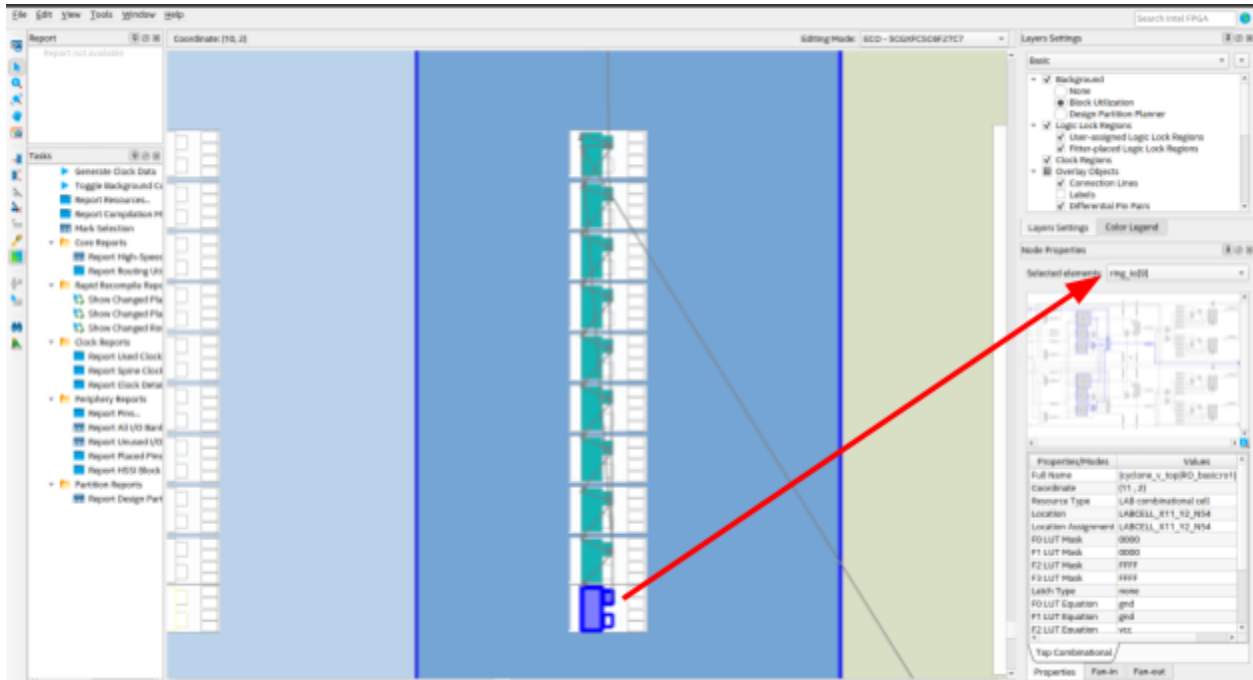
```

- This results in a column RO_basic module implementation as seen below, where the first LUT with the ALM cell can be seen as being the first (i.e. ring_io[0]) wire.



- Whereas the last LUT, can be seen pointed towards its wire representation that is, the ring_io[9].

NOTE: the full utilisation of the LAB cell can be seen within the two images of the chip-planner.



```

module tb_RO_basic();

parameter not_gate = 8;
parameter DELAY = 1;

reg          enable;
wire         RO_out;

RO_basic #(
    .not_gate(not_gate),
    .DELAY(DELAY)
) RO_basic_inst (
    .enable(enable),
    .RO_out(RO_out)
);

always #1 $display("TIME %g, enable: %b, RO_out: %b", $time, enable, RO_out);

initial begin
    enable = 1'b0;

    #20
    enable = 1'b1;

    #100
    $stop;
end

endmodule

```

- For testing the RO_basic block, a test bench name, "tb_RO_basic.v" was created.
- From the output, the oscillation of the output can be observed → verifying the module.

```

# TIME 68, enable: 1, RO_out: 1
# TIME 69, enable: 1, RO_out: 1
# TIME 70, enable: 1, RO_out: 1
# TIME 71, enable: 1, RO_out: 1
# TIME 72, enable: 1, RO_out: 1
# TIME 73, enable: 1, RO_out: 1
# TIME 74, enable: 1, RO_out: 1
# TIME 75, enable: 1, RO_out: 1
# TIME 76, enable: 1, RO_out: 0
# TIME 77, enable: 1, RO_out: 0
# TIME 78, enable: 1, RO_out: 0
# TIME 79, enable: 1, RO_out: 0
# TIME 80, enable: 1, RO_out: 0
# TIME 81, enable: 1, RO_out: 0
# TIME 82, enable: 1, RO_out: 0
# TIME 83, enable: 1, RO_out: 0
# TIME 84, enable: 1, RO_out: 1
# TIME 85, enable: 1, RO_out: 1
# TIME 86, enable: 1, RO_out: 1
# TIME 87, enable: 1, RO_out: 1
# TIME 88, enable: 1, RO_out: 1
# TIME 89, enable: 1, RO_out: 1
# TIME 90, enable: 1, RO_out: 1
# TIME 91, enable: 1, RO_out: 1
# TIME 92, enable: 1, RO_out: 0
# TIME 93, enable: 1, RO_out: 0
# TIME 94, enable: 1, RO_out: 0
# TIME 95, enable: 1, RO_out: 0
# TIME 96, enable: 1, RO_out: 0
# TIME 97, enable: 1, RO_out: 0
# TIME 98, enable: 1, RO_out: 0
# TIME 99, enable: 1, RO_out: 0
# TIME 100, enable: 1, RO_out: 1
# TIME 101, enable: 1, RO_out: 1
# TIME 102, enable: 1, RO_out: 1
# TIME 103, enable: 1, RO_out: 1
# TIME 104, enable: 1, RO_out: 1
# TIME 105, enable: 1, RO_out: 1
# TIME 106, enable: 1, RO_out: 1
# TIME 107, enable: 1, RO_out: 1
# TIME 108, enable: 1, RO_out: 0
# TIME 109, enable: 1, RO_out: 0
# TIME 110, enable: 1, RO_out: 0
# TIME 111, enable: 1, RO_out: 0
# TIME 112, enable: 1, RO_out: 0
# TIME 113, enable: 1, RO_out: 0
# TIME 114, enable: 1, RO_out: 0
# TIME 115, enable: 1, RO_out: 0
# TIME 116, enable: 1, RO_out: 1
# TIME 117, enable: 1, RO_out: 1
# TIME 118, enable: 1, RO_out: 1
# TIME 119, enable: 1, RO_out: 1
** Note: $stop : ../tb/tb_RO_basic.v(29)
# Time: 120 ps Iteration: 0 Instance: /tb_RO_basic
# Break in Module tb_RO_basic at ../tb/tb_RO_basic.v line 29
# quit
# End time: 15:35:23 on Oct 20, 2023, Elapsed time: 0:00:00
# Errors: 0, Warnings: 0

```

Task 2

- “Counter_group.v” was filled with 2 MUXs (16:1) followed by a counter for respective MUXs.
 - A single MUX→ counter looks like

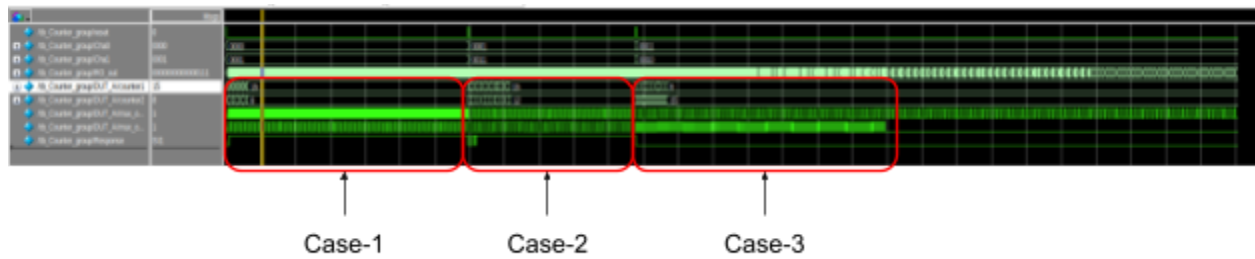
```
17 // MUX 1 implementation
18 assign mux_out_0 = R0_out[Cha0];
19
20
21 //Counter 1 implementation
22 always @(posedge mux_out_0 or posedge reset) begin
23     if (reset) begin
24         counter1 <= 4'h0; //Reseting the counter
25     end
26     else if (counter1 < 4'hf) begin
27         counter1 <= counter1 + 1; //increment the counter whenever mux_out
           available
28     end
29
30 end
```

- Same was implemented for the second MUX followed by another counter

```
32 // MUX 2 implementation
33 assign mux_out_1 = R0_out[Cha1];
34
35
36 //Counter 2 implementation
37 always @(posedge mux_out_1 or posedge reset) begin
38     if (reset) begin
39         counter2 <= 4'h0; //Reseting the counter
40     end
41     else if (counter2 < 4'hf) begin
42         counter2 <= counter2 + 1; //increment the counter whenever mux_out
           available
43     end
44
45 end
```

- Both the counters count till 4 bits, that is for 15 ticks and whichever finishes first produces either 0 or 1.
 - It is important to emulate the 16 different RO_basic modules that oscillated at a different frequency for a race condition between both the counters.
 - This can either be done by creating square-wave with different period so that, with the select bit parse to the MUX different conditions can be presented and verification of the counter_group be done.
 - The other was by using the vsim (a fellow coursemate, named Katy, helped me with setting up the vsim and showing how it's done)
- The image below describes the 3 boxes as 3 cases
 - Case 1 and 2: where counter 1 is faster, can be seen as the waveforms are more denser resulting in the response being 1

- While case 3 has 2nd counter running faster then the first counter, resulting in a 0-bit response.



Task 3 & 4:









- An intermediate module named, puf_single.v was created
 - Input: reset, 4-bit for select of MUX 1 and MUX 2
 - Output: single response bit..



















```

11 wire [15:0] R0_out;
12
13 generate
14   genvar i;
15   for (i = 0; i < 16; i = i + 1)
16     begin: R0_basic_int
17       R0_basic #(
18         .not_gate(9),
19         .DELAY(1))
20       (
21         .enable(~reset),
22         .R0_out(R0_out[i])
23       );
24     end
25 endgenerate
26
27 Counter_group counter_grp_inst(
28   .reset(reset),
29   .Cha0(Cha0),
30   .Cha1(Cha1),
31   .R0_out(R0_out),
32   .Response(Response)
33 );
34

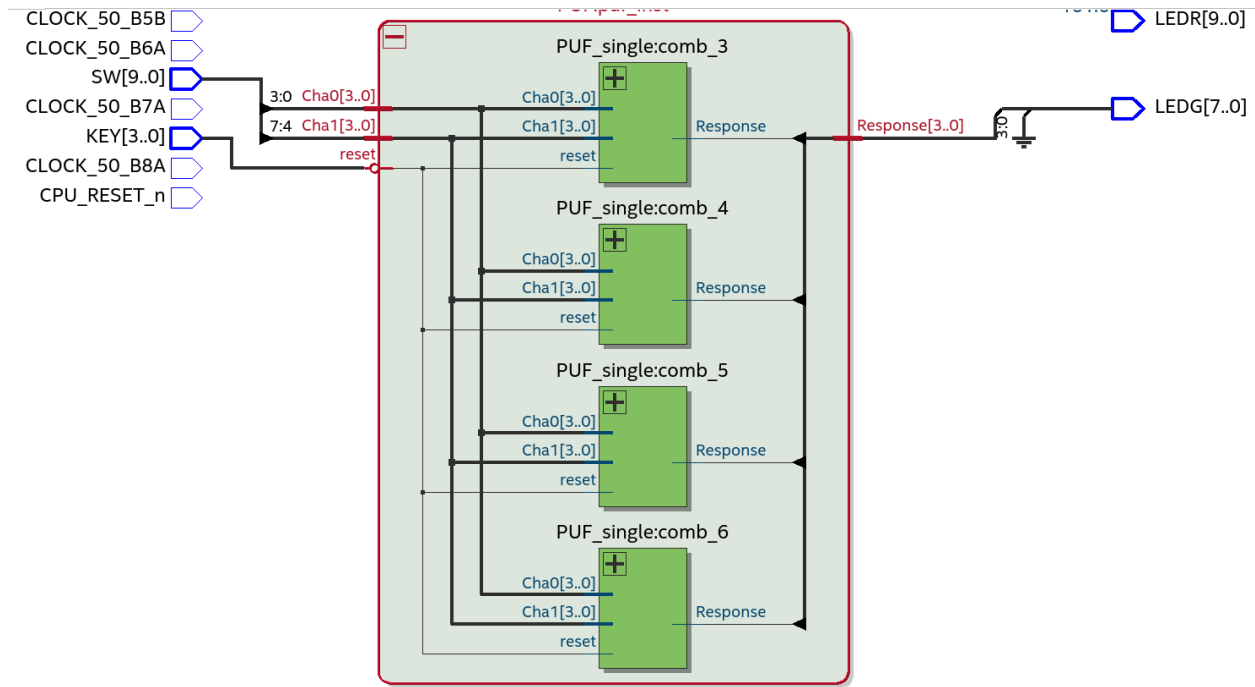
```

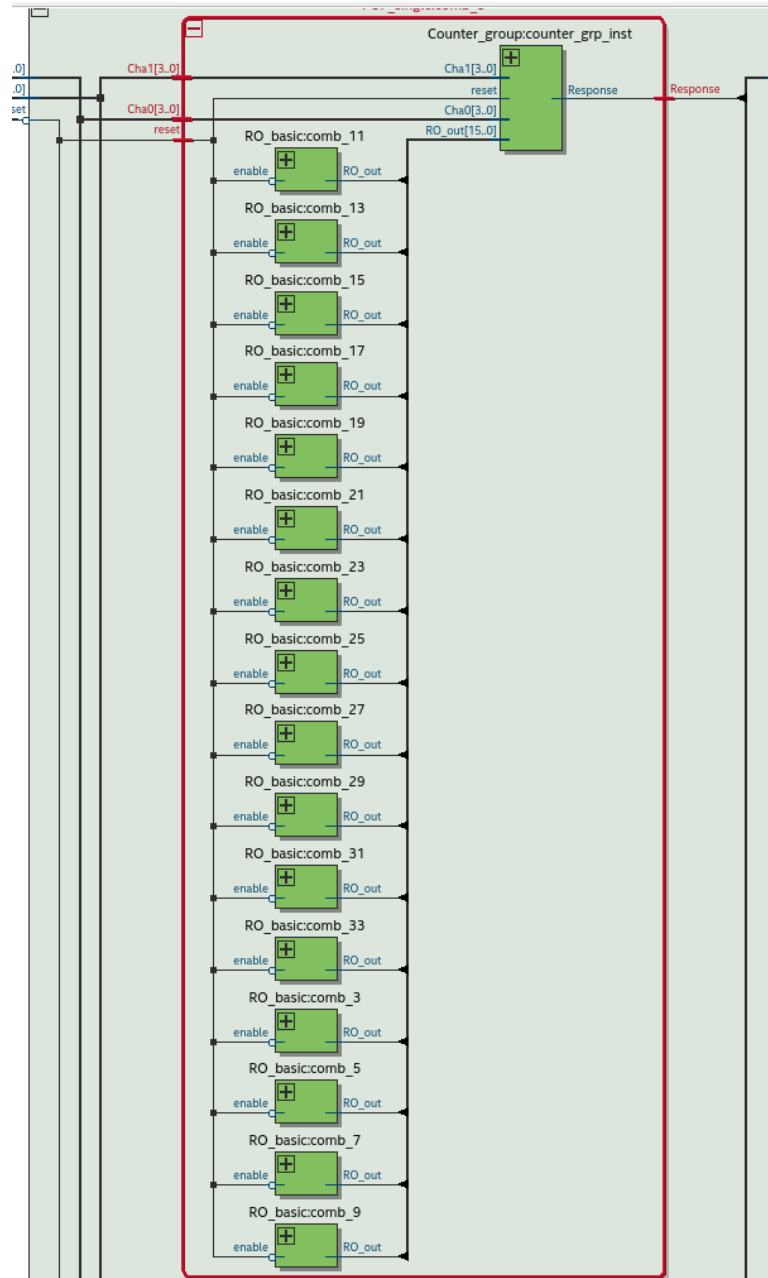
- This module was called 4 times in the puf.v script to create RO PUF with 4 response bit, using a generate function block. The image below shows the hierarchy view of the process.

Entity:Instance	
 Cyclone V: 5CGXFC5C6F27C7	
▼  cyclone_v_top 	
▼  PUF:puf_inst	
▶  PUF_single:comb_3	
▶  PUF_single:comb_4	
▶  PUF_single:comb_5	
▶  PUF_single:comb_6	

▼  PUF_single:comb_3	
 RO_basic:comb_3	
 RO_basic:comb_5	
 RO_basic:comb_7	
 RO_basic:comb_9	
 RO_basic:comb_11	
 RO_basic:comb_13	
 RO_basic:comb_15	
 RO_basic:comb_17	
 RO_basic:comb_19	
 RO_basic:comb_21	
 RO_basic:comb_23	
 RO_basic:comb_25	
 RO_basic:comb_27	
 RO_basic:comb_29	
 RO_basic:comb_31	
 RO_basic:comb_33	
 Counter_group:counter_grp_inst	

- In total creating a total of 64 RO_basic modules.





- A TCL file was generated for placing all the RO module using a python script named part2task4.py. **(failed)**

NOTE: I was able to generate a RO PUF, and test it generation in the chip-planner and through RTL diagram but got stuck at the tcl placement and couldn't solve it within deadline.