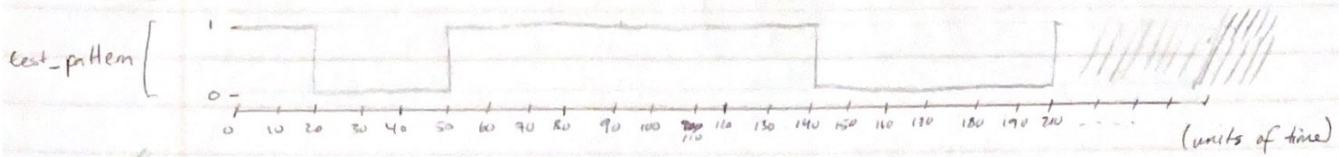
ECE426 HW3 CHASE LOTITO

Q1 INITIAL BLOCK STATEMENT.

Draw a timing diagram for the "test_pattern" signal.

(A)



(B) The timing diagram is going to be the exact same as the assignment of test_pattern is still delayed after the assignment operator (=).

"COPY TIMING DIAGRAM"

(C) Using the non-blocking assignment operator (<=), we can give each test_pattern separately. However, each time delay lines up with part A and part B, so we again get the exact same timing diagram. (ex. #20 0, #30 1 takes #50, #50 1 is the same)

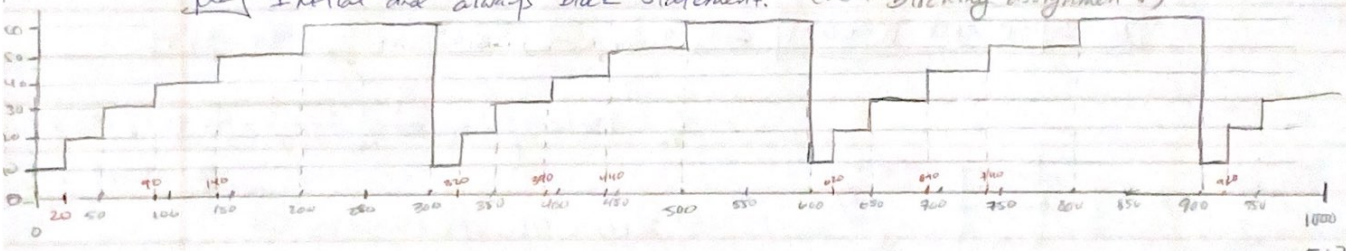
$\Delta t = \frac{1000-0}{n}$

$10 = \frac{1000-0}{n}$

$n = 100$

"COPY TIMING DIAGRAM"

Q2 Initial and always block statement. (non-blocking assignments)

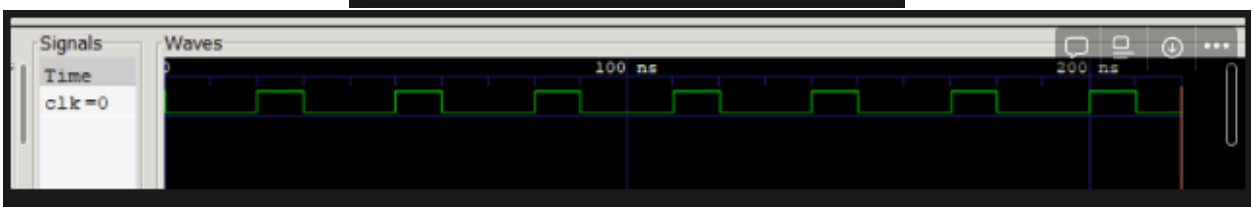


[E]

Q3

Part A:

```
/*
Chase Lotito - SIUC - SP2024
ECE426 - Chao Lu
HW3 - Q3 - Clock Generation
*/

`timescale 1ns / 1ns

// Work done here
module clock();

// Duty Cycle Parameters
parameter TON = 10;
parameter TOFF = 20;

// let me see the signals!
initial begin : GTKWAVE
    $dumpfile("clock.vcd");
    $dumpvars(0, clk);
end

// I/O, Reg, Wire
reg clk;

// Setting clock duty cycle
always begin
    clk = 0;
    #TOFF clk = 1;
    #TON clk = 0;
end

integer i;
always @(posedge clk) begin
    #200 $finish;
end

endmodule
```



This clock has TON=10ns and TOFF=20ns, and here the duty cycle is 33.3%. The only way to get a 50% duty cycle is to make TON = TOFF.

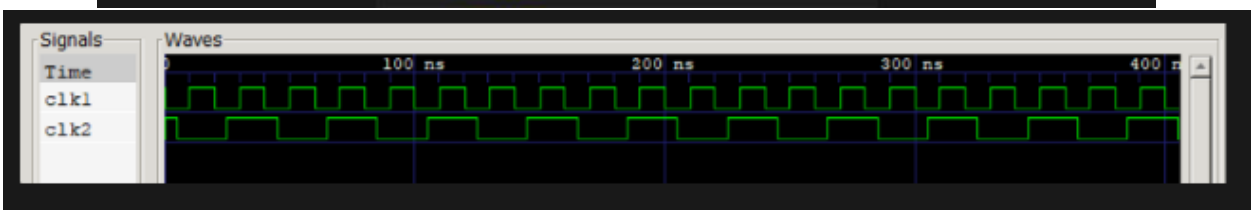
B

```
7   `timescale 1ns / 1ns
8
9   // Work done here
10  module clock();
11
12  // Duty Cycle Parameters
13  parameter TON = 10;
14  parameter TOFF = 20;
15  parameter cycles = 10;
16
17  // let me see the signals!
18  initial begin : GTKWAVE
19      $dumpfile("clock.vcd");
20      $dumpvars(0, clk);
21  end
22
23  // I/O, Reg, Wire
24  reg clk;
25
26  initial begin
27      // Setting clock duty cycle
28      repeat(cycles) begin
29          clk = 0;
30          #TOFF
31          clk = 1;
32          #TON
33          clk = 0;
34      end
35  end
36
37  endmodule
```

C

Q3 > dbClk.v

```
/*
Chase Lotito - SIUC - SP2024
ECE426 - Chao Lu
HW3 - Q3 - Clock Generation
*/

`timescale 1ns / 1ns

// Work done here
module DoubleClock();

// Duty Cycle Parameters (50%, 10 cycles)
parameter TON1 = 10;
parameter TOFF1 = 10;

parameter TON2 = 20;
parameter TOFF2 = 20;

parameter cycles1 = 20;
parameter cycles2 = 10;

parameter phase = 5;

// let me see the signals!
initial begin : GTKWAVE
    $dumpfile("clock2.vcd");
    $dumpvars(0, DoubleClock);
end

// I/O, Reg, Wire
reg clk1, clk2;

// CLOCK 1
initial begin
    // Setting clock duty cycle
    repeat(cycles1) begin
        clk1 = 0;
        #TOFF1
        clk1 = 1;
        #TON1
        clk1 = 0;
    end
end

// CLOCK 2
initial begin
    // Setting clock duty cycle
    clk2 = 1;
    #phase
    repeat(cycles2) begin
        clk2 = 0;
        #TOFF2
        clk2 = 1;
        #TON2
        clk2 = 0;
    end
end

endmodule
```



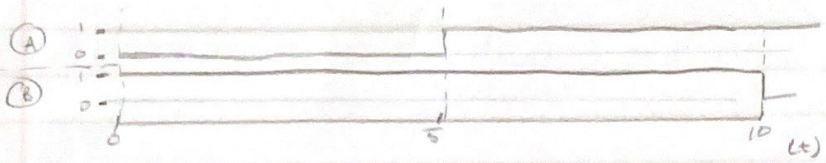
[10.4] Short answer questions

(A) Why is it better to use case statement over nested if-else?

Case statements synthesize as multiplexers that give no priority to input cases, and nested if else statements synthesize as unplicated multiple-level combinational logic w/ priorities. This means case-statement circuits have the smaller propagation delay as the logic is done in parallel.

#5 A=1,
#5 B=0,

(B) Wave forms of A and B if initially A=0, B=1?



(C) Explain the difference btwn == and === operators.

== is the logic equality operator, and === is the case equality operator. Both test equality.

They differ in how they deal with unknown (x) and high impedance states (z).

== treats "x equality z" as unknown (x).
(i.e., x==z → x and equal!)

=== treats all 4 logic states separately.
(i.e. x===z → FALSE)