Lab 1

Task 1

Simulating a basic 8-bit binary counter

```
module counter #( //file name and module name must be the same
    parameter WIDTH = 8 // specify counter bit width
)(
    //interface signals
    input logic                 clk, //clock
    input logic                 rst, //reset
    input logic                 en,  //counter enable
    output logic [WIDTH-1:0]    count  //count output
);

//always_ff specifies a clocked circuit. This is an asynchronous reset
always_ff @ (posedge clk, posedge rst)
    if (rst) count <= {WIDTH{1'b0}}; //non-blocking assignment (asynchronou
s).
    // {} is used to form WIDTH bits of 0
    else count <= count + {{WIDTH-1{1'b0}}, en};

endmodule
```

Note that we use '<=', which is a non-block assignment as it means that all registers would be updated simultaneously on the clock edge and not sequentially

Additionally, note that the concatenation operator { } creates (WIDTH-1) number of zeros, and is constructed with en, effectively extending en (1 bit) to match WIDTH bits.

```
/*
Mandatory header files.
Note the name Vcounter.h is for the module counter
*/

#include "Vcounter.h"
#include "verilated.h"
#include "verilated_vcd_c.h"

int main(int argc, char **argv, char **env) {
   int i; // coutns the number of clock cycles to simulate
   int clk; //clk is the module clock signal


Verilated::commandArgs(argc, argv);
   //Instantiate the counter module as Vcounter.This is the DUT
   Vcounter* top = new Vcounter;

   /*
   initialise trace dump
   Turn on signal tracing, and tell verilator to dump the waveform data
   to counter.vcd file
   */

   Verilated::traceEverOn(true);
   VerilatedVcdC* tfp = new VerilatedVcdC;
   top→trace(tfp, 99);
   tfp→open("counter.vcd");

   /*
   Initialise simulation inputs
   Sets initial signal values before simulation
   Top is the name of the top-level entity
   Only the top-level signals are visible
   */
```

```
    top→clk = 1;
    top→rst = 1;
    top→en = 0;

    /*
    run simulation for many clock cycles

    The for loop is where the simulation happens
    i counts the clock cycles
    */
    for (i=0; i<300; i++){
        //change rst and en signls during simulation. rst = 1 for i = 0, 1, 15. rst = 0
for all other cycles
        top →rst = (i < 2) | (i==15);
        top → en = (i > 4); //en turns on from i = 5 onwards

        //dump variables into VCD file and toggle clock
        for (clk=0; clk<2; clk++){
                // time is in ps. tfp is responsible for for recording all values
                // during the simulation
            tfp→dump(2*i+clk);
            top→clk = !top→clk; //toggle clock
            top→eval(); //evaluate model again with new clock value
        }

        /*
        shifted by 1 as rst and en code block is after the for loop.
        Therefore, evaluation of the clock is done before the reset and enable is
        evaluated again
        */


        if (Verilated::gotFinish()) exit(0); //exit if finish signal received

    }
```

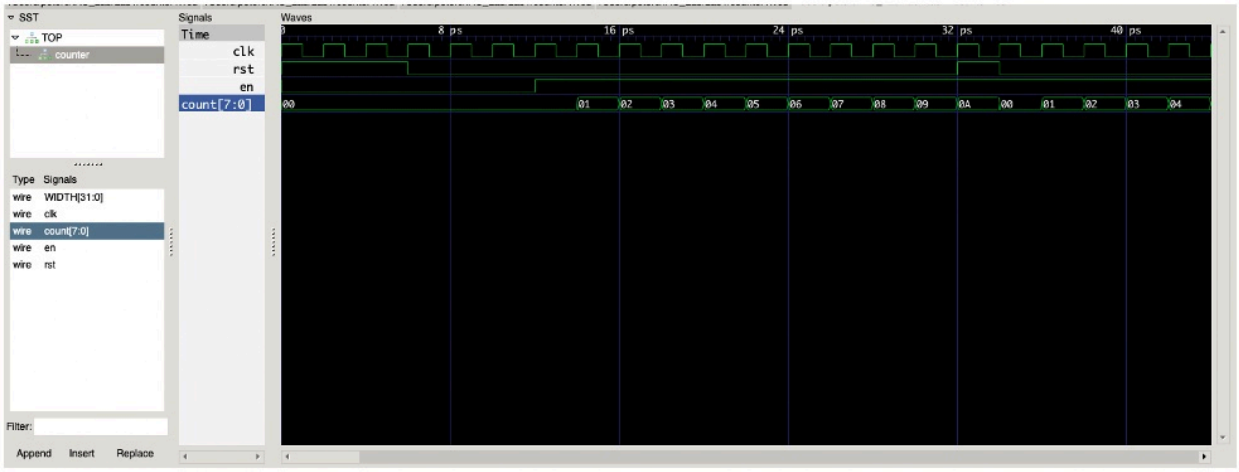
```
    tfp -> close(); //close VCD file
    exit(0); //exit program
}
```

Note that time is in ps as most clocks are in MHz - GHz range, so picosecond resolution helps ensure accurate edge alignment.

However, the time axis's unit does not matter as the circuit is clock-based, not time-based

Moreover, time is in ps because Verilator uses ps as the default time unit

GTK wave output

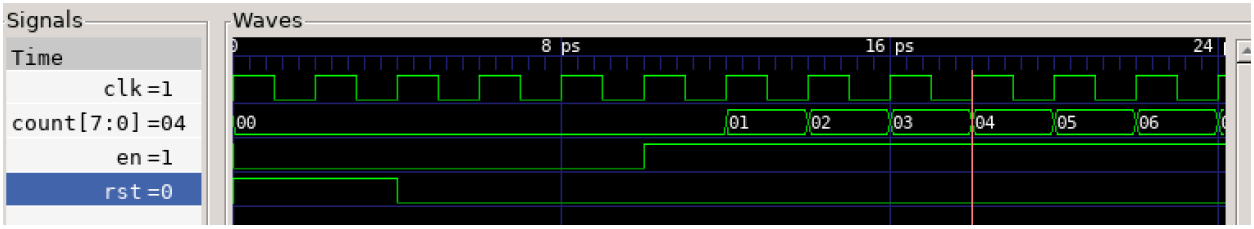


Test yourself challenges:

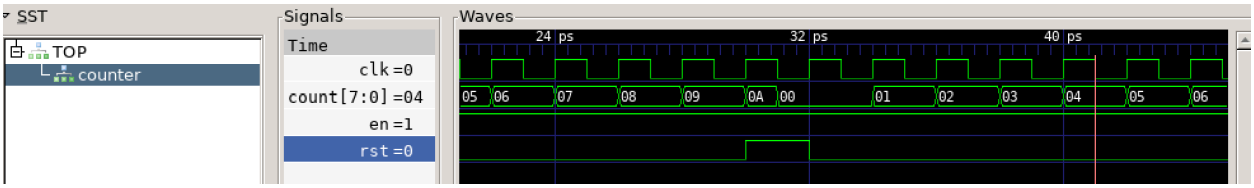
1. Modify the testbench so that you stop counting for 3 cycles once the counter reaches 0x9, and then resume counting. You may also need to change the stimulus for *rst*.
    a. You just need to modify the en and rst values in the for loop in the testbench (modify ‘i’)
2. The current counter has a synchronous reset. To implement asynchronous reset, you can change line 11 of counter.sv to detect change in *rst* signal. (See notes.)\

a. Change the counter.sv code to "always_ff @ (posedge clk, posedge rst)" instead to look for a positive edge in rstThere is a slight bug in Verilog with the implementation of asynchronous reset, where it doesn't reset instantly. This is why asynchronous functions are avoidedThe bug may be due to the fact that res changes on clock cycle, which usually never happens

3. Make these modification, compile, and run. Examine the waveform with GTKwave and explain what you see.
    a. You can see from the GTK wave that en=1 on the 5th clock poscycle (i=0, i=1...) The count then increases on the next clock cycle when en=1.



The asynchronous bug is as follows, where reset happens the next half cycle



Task 2

Linking Verilator simulation with Vbuddy

```
/*
Mandatory header files.
Note the name Vcounter.h is for the module counter
*/

#include "Vcounter.h"
#include "verilated.h"
#include "verilated_vcd_c.h"
```

```
#include "vbuddy.cpp"  // include vbuddy code

int main(int argc, char **argv, char **env) {
    int i; // counts the number of clock cycles to simulate
    int clk; //clk is the module clock signal


Verilated::commandArgs(argc, argv);
        // Instantiate the counter module as Vcounter. This is the DUT
    Vcounter* top = new Vcounter;

    /*
    initialise trace dump
    Turn on signal tracing, and tell verilator to dump the waveform data
    to counter.vcd file
    */

    Verilated::traceEverOn(true);
    VerilatedVcdC* tfp = new VerilatedVcdC;
    top→trace(tfp, 99);
    tfp→open("counter.vcd");


    //init vbuddy
    if (vbdOpen()!=1) return(-1);
    vbdHeader("Lab 1: Counter");

    /*
    Initialise simulation inputs
    Sets initial signal values before simulation
    Top is the name of the top-level entity
    Only the top-level signals are visible
    */
    top→clk = 1;
    top→rst = 1;
```

```
top→en = 0;

/*
run simulation for many clock cycles

The for loop is where the simulation happens
i counts the clock cycles
*/
for (i=0; i<300; i++){
    // change rst and en signls during simulation.
    // rst = 1 for i = 0, 1, 15. rst = 0 for all other cycles
    top →rst = (i < 2) | (i==15);
    top→en = vbdFlag(); //en turns on when the button on vbuddy is pressed

    //dump variables into VCD file and toggle clock
    for (clk=0; clk<2; clk++){
        tfp→dump(2*i+clk);
        //time is in ps.
        //tfp is responsible for for recording all values during the simulation
        top→clk = !top→clk; //toggle clock
        top→eval(); //evaluate model again with new clock value
    }

    /*
    //display count value on vbuddy
    vbdHex(4, (int(top→count) >> 16) & 0xF);
    vbdHex(3, (int(top→count) >> 8) & 0xF);
    vbdHex(2, (int(top→count) >> 4) & 0xF);
    vbdCycle(i+1); //display cycle number
    */

    vbdPlot(int(top→count), 0, 255); //plot count value on vbuddy display


    /*
    shifted by 1 as rst and en code block is after the for loop.
```

```
        Therefore, evaluation of the clock is done before the reset
        and enable is evaluated again
        */


        if (Verilated::gotFinish()) exit(0); //exit if finish signal received

    }

    vbdClose(); //close vbuddy
    tfp -> close(); //close VCD file
    exit(0); //exit program
}
```