COUNTERS IN VERILOG

LAB # 10

Fall 2023

CSE-304L

Computer Organization & Architecture Lab

Submitted by: **AIMAL KHAN**

Registration No.: **21PWCSE1996**

Class Section: **A**

"On my honor, as student of University of Engineering and Technology, I have neither given nor received unauthorized assistance on this academic work."

Student Signature: [illegible]

Submitted to:

Dr. Bilal Habib

Monday, January 29, 2024

Department of Computer Systems Engineering

University of Engineering and Technology, Peshawar

ASSESSMENT RUBRICS COA LABS

LAB REPORT ASSESSMENT				
Criteria	**Excellent**	**Average**	**Nill**	**Marks Obtained**
1. Objectives of Lab	All objectives of lab are properly covered [Marks 10]	Objectives of lab are partially covered [Marks 5]	Objectives of lab are not shown [Marks 0]	
2. MIPS instructions with Comments and proper indentations.	All the instructions are well written with comments explaining the code and properly indented [Marks 20]	Some instructions are missing are poorly commented code [Marks 10]	The instructions are not properly written [Marks 0]	
3. Simulation run without error and warnings	The code is running in the simulator without any error and warnings [Marks 10]	The code is running but with some warnings or errors. [Marks 5]	The code is written but not running due to errors [Marks 0]	
4. Procedure	All the instructions are written with proper procedure [Marks 20]	Some steps are missing [Marks 10]	steps are totally missing [Marks 0]	
5. OUTPUT	Proper output of the code written in assembly [Marks 20]	Some of the outputs are missing [Marks 10]	No or wrong output [Marks 0]	
6. Conclusion	Conclusion about the lab is shown and written [Marks 20]	Conclusion about the lab is partially shown [Marks 10]	Conclusion about the lab is not shown [Marks0] [Marks 0]	
7. Cheating			Any kind of cheating will lead to 0 Marks	
Total Marks Obtained: __________ Instructor Signature: _____________________				

Counters in Verilog

Objectives:

- Implement counters in Verilog
- Implement controlled and uncontrolled asynchronous counters
- Implement RAM

Tasks:

Task 1: Write a Verilog code to implement 4-bit counter

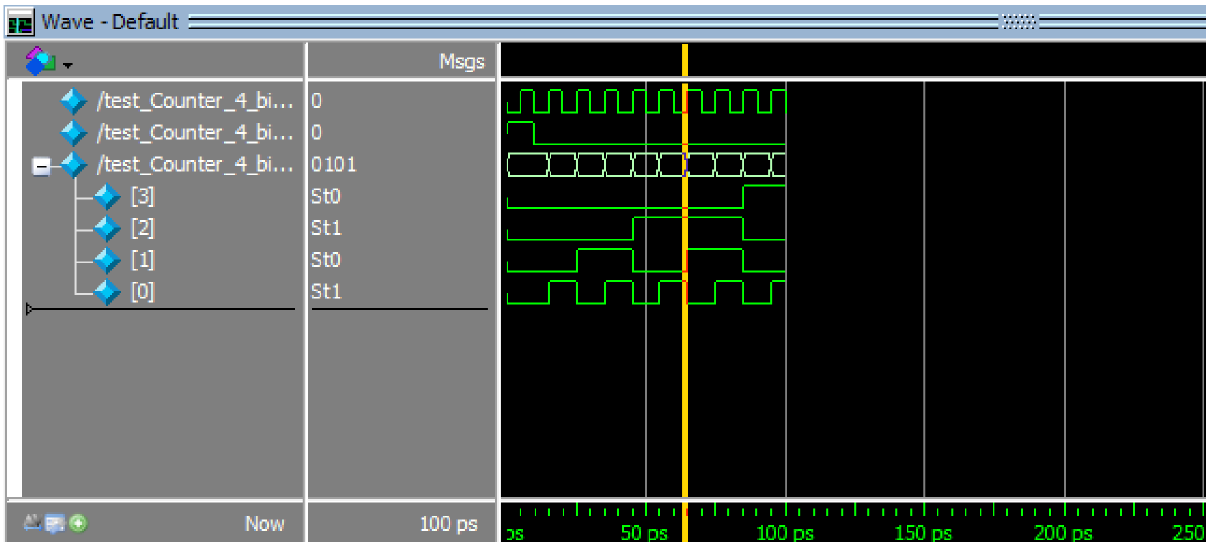
DUT Code:

```
module Counter_4_bit(
    input wire clock,
    input wire reset,
    output reg [3:0] count
);
    always @(posedge clock or posedge reset)
    begin
        if(reset) begin
            count <= 4'b0000;
        end else begin
            count <= count +1;
        end
    end
endmodule
```

Test Code:

```
module test_Counter_4_bit();
    reg clk;
    reg rst;
    wire [3:0] count;

    Counter_4_bit counter(
        .clock(clk),
        .reset(rst),
        .count(count)
    );

    always begin
        #5
        clk = ~clk;
    end

    initial begin
        clk = 0;
        rst = 0;

        // Reset the counter initially
        rst = 1;
        #10 rst = 0;
        #20;

        $display("Time = 0, Count = %b", count);
        #10;
        $display("Time = 10, Count = %b", count);
        #10;
        $display("Time = 20, Count = %b", count);
        #10;
        $display("Time = 30, Count = %b", count);
    end
endmodule
```

Output:



Task 2: Write a Verilog code to implement 8-bit counter

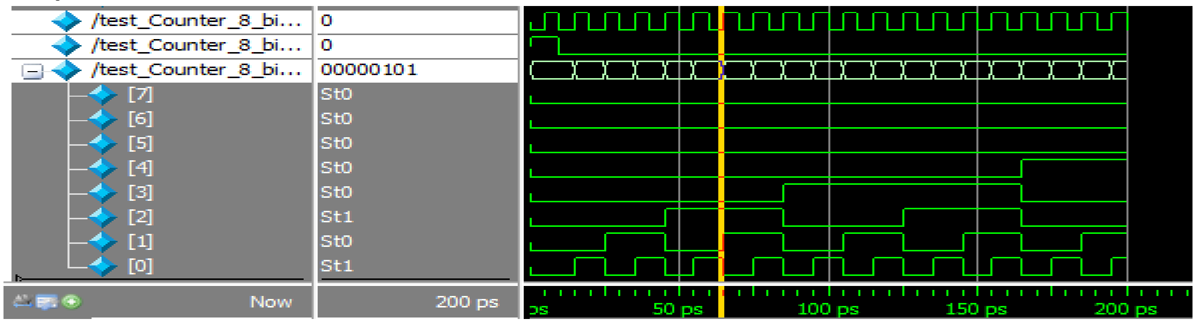
DUT Code:

```
module Counter_8_bit(
    input wire clock,
    input wire reset,
    output reg [7:0] count
);

    always @(posedge clock or posedge reset)
    begin
        if(reset)
            count <= 8'b00000000;
        else
            count <= count +1;
    end
endmodule
```

Test Code:

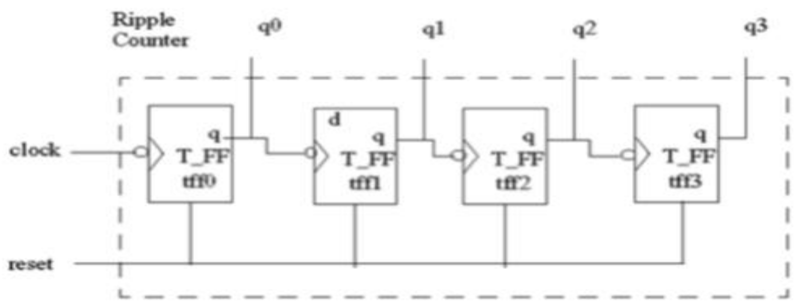
```
module test_Counter_8_bit();
    reg clk;
    reg rst;
    wire [7:0] count;

    Counter_8_bit counter(
        clk, rst, count
    );

    always begin
        #5 clk = ~clk;
    end

    initial begin
        clk = 0;
        rst = 0;

        rst = 1;
        #10 rst = 0;

        #20;
        $display("Time = 0, Count = %b", count);
        #10;
        $display("Time = 10, Count = %b", count);
        #10;
        $display("Time = 20, Count = %b", count);
        #10;
        $display("Time = 30, Count = %b", count);

        #10;
        $display("Time = 30, Count = %b", count);
        #10;
        $display("Time = 30, Count = %b", count);
        #10;
        $display("Time = 30, Count = %b", count);
        #10;
        $display("Time = 30, Count = %b", count);
    end
endmodule
```

Output:

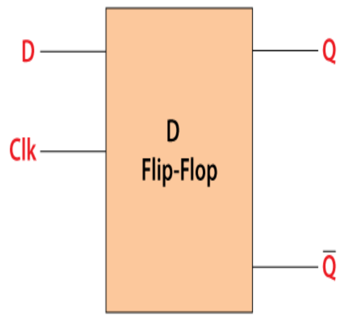


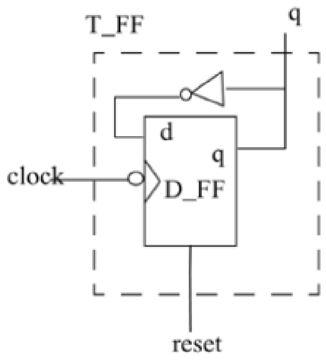
```
/test_Counter_8_bi...   0
/test_Counter_8_bi...   0
/test_Counter_8_bi...   00000101
    [7]                 St0
    [6]                 St0
    [5]                 St0
    [4]                 St0
    [3]                 St0
    [2]                 St1
    [1]                 St0
    [0]                 St1
Now                     200 ps
```

Task 3: Implement 4 BIT Uncontrolled Asynchronous UP COUNTER in Verilog using T FLIP FLOP.



Block Diagram





Circuit Diagram

DUT Code:

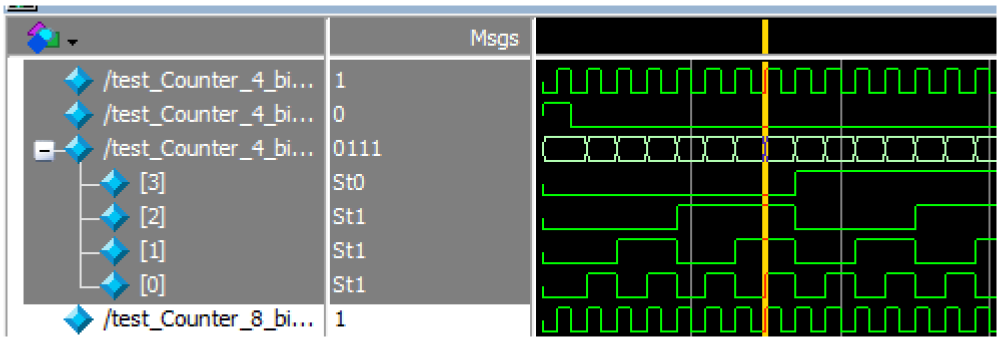
```
module T_Uncontrolled_Async_Counter_4_bit(
    input wire clock,
    input wire reset,
    output reg [3:0] count
);
    reg [3:0] t_signals;

    always @(posedge clock or posedge reset) begin
        if (reset) begin
            t_signals <= 4'b0000; // Reset T signals to 0 when reset
is high
        end else begin
            t_signals <= t_signals + 1; // Increment T signals on
each clock cycle
        end
    end

    T_Flip_Flop tff0 (.T(t_signals[0]), .clock(clock),
.Q(count[0]));
    T_Flip_Flop tff1 (.T(t_signals[1]), .clock(clock),
.Q(count[1]));
```

```
    T_Flip_Flop tff2 (.T(t_signals[2]), .clock(clock),
.Q(count[2]));
    T_Flip_Flop tff3 (.T(t_signals[3]), .clock(clock),
.Q(count[3]));

endmodule
```

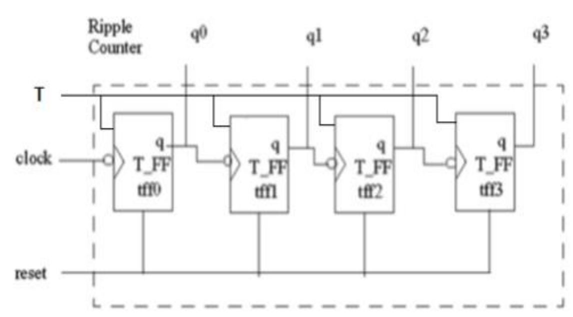
Test Code:

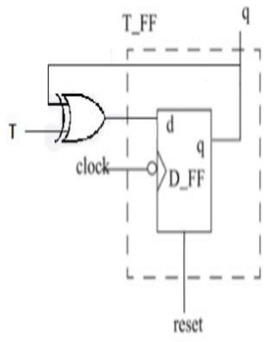
```
module test_T_Uncontrolled_Async_Counter_4_bit();
    reg clk;
    reg rst;
    wire [3:0] count;

    T_Uncontrolled_Async_Counter_4_bit uut (
        .clock(clk),
        .reset(rst),
        .count(count)
    );

    always begin
        #5 clk = ~clk;
    end

    // Testbench initialization
    initial begin
        clk = 0;
        rst = 0;

        // Reset the counter initially
        rst = 1;
        #10 rst = 0;

        // Wait for some clock cycles and observe the counter
        #20;

        $display("Time = 0, Count = %b", count);
        #10;
        $display("Time = 10, Count = %b", count);
        #10;
        $display("Time = 20, Count = %b", count);
        #10;
        $display("Time = 30, Count = %b", count);
    end
endmodule;
```

Output:



Task 4: Implement 4 BIT Controlled Asynchronous UP COUNTER in Verilog using T FLIP FLOP.





DUT Code:

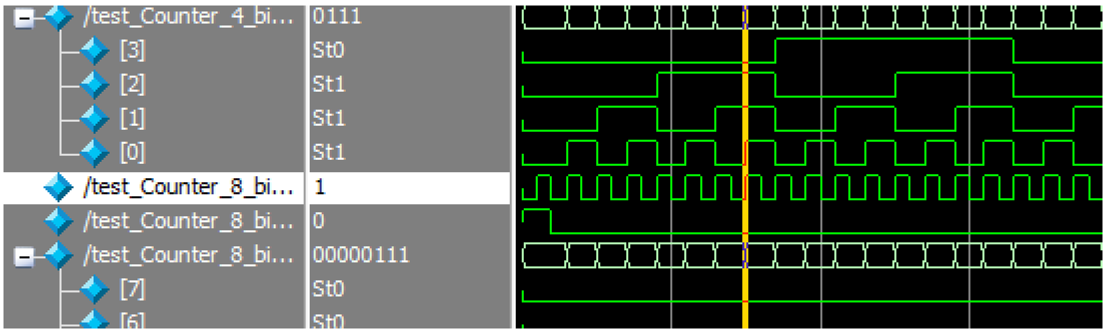
```
module T_Controlled_Async_Counter_4_bit(
    input wire clock,
    input wire reset,
    input wire enable,
    output reg [3:0] count
);

    reg [3:0] t_signals;

    always @(posedge clock or posedge reset) begin
        if (reset) begin
            t_signals <= 4'b0000; // Reset T signals to 0 when reset 
is high
        end else if (enable) begin
            t_signals <= t_signals + 1; // Increment T signals on 
each clock cycle when enable is high
        end
    end

    T_Flip_Flop tff0 (.T(t_signals[0]), .clock(clock), 
.Q(count[0]));
    T_Flip_Flop tff1 (.T(t_signals[1]), .clock(clock), 
.Q(count[1]));
```

```
    T_Flip_Flop tff2 (.T(t_signals[2]), .clock(clock),
.Q(count[2]));
    T_Flip_Flop tff3 (.T(t_signals[3]), .clock(clock),
.Q(count[3]));

endmodule
```

Test Code:

```
module test_T_Controlled_Async_Counter_4_bit();
reg clk;
reg rst;
reg enable;
wire [3:0] count;

// Instantiate the controlled asynchronous up-counter module
T_Controlled_Async_Counter_4_bit uut (
    .clock(clk),
    .reset(rst),
    .enable(enable),
    .count(count)
);

// Clock generation
always begin
    #5 clk = ~clk; // Toggle the clock every 5 time units
end

// Testbench initialization
initial begin
    clk = 0;
    rst = 0;
    enable = 1; // Enable the counter

    // Reset the counter initially
    rst = 1;
    #10 rst = 0;

    // Wait for some clock cycles and observe the counter
    #20;

    // Test the counter for several clock cycles
    $display("Time = 0, Count = %b", count);
    #10;
    $display("Time = 10, Count = %b", count);
    #10;
    $display("Time = 20, Count = %b", count);
    #10;
    $display("Time = 30, Count = %b", count);

    // Finish simulation
    $finish;
end
endmodule
```

Output:



```
/test_Counter_4_bi...   0111
  [3]                   St0
  [2]                   St1
  [1]                   St1
  [0]                   St1
/test_Counter_8_bi...   1
/test_Counter_8_bi...   0
/test_Counter_8_bi...   00000111
  [7]                   St0
  [6]                   St0
```

Task 5: Implement 16x8 RAM in Verilog.

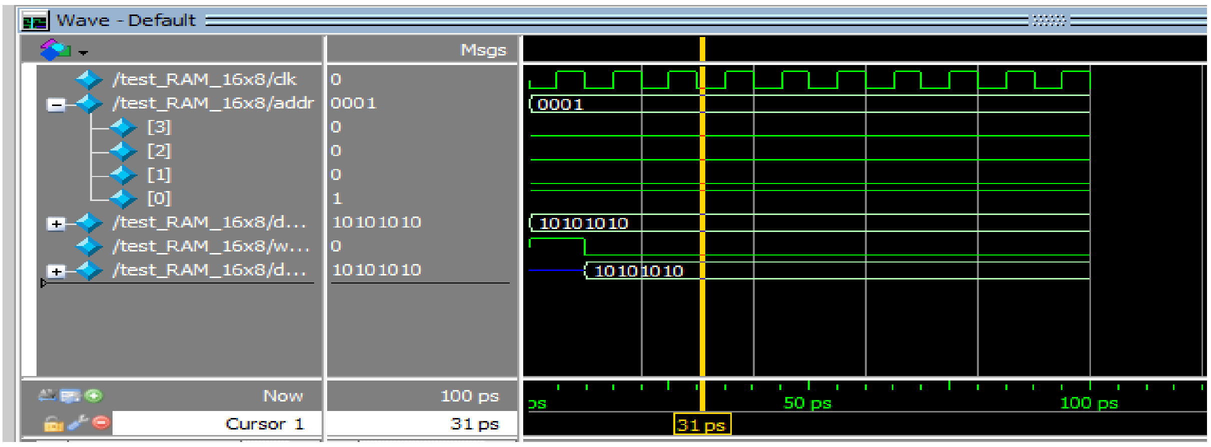
DUT Code:

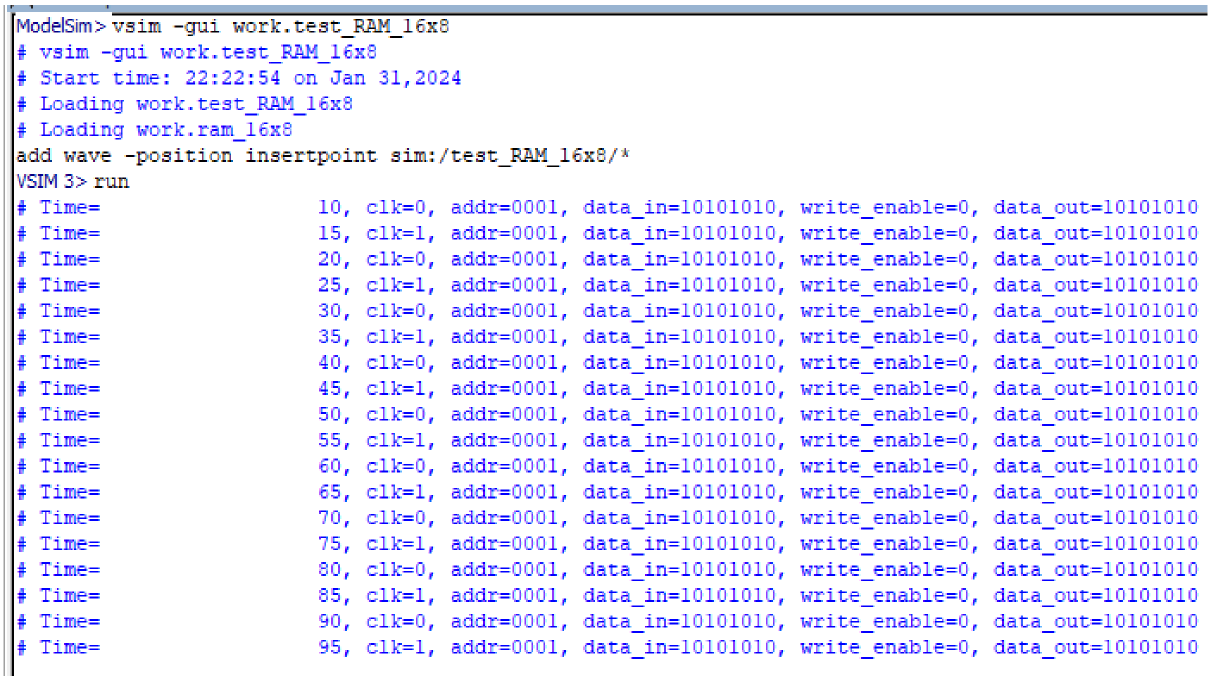
```
module RAM_16x8 (clk, addr, data_in, write_enable, data_out);
  input wire clk;
  input wire [3:0] addr;
  input wire [7:0] data_in;
  input wire write_enable;
  output wire [7:0] data_out;

  reg [7:0] memory [0:15]; // 16 words by 8 bits

  // Read operation
  assign data_out = (write_enable) ? 8'bzzzz_zzzz : memory[addr];

  // Write operation
  always @(posedge clk) begin
    if (write_enable)
      memory[addr] <= data_in;
  end

endmodule
```

Test Code:

```

// Testbench for 16x8 RAM
module test_RAM_16x8;

  // Inputs
  reg clk;
  reg [3:0] addr;
  reg [7:0] data_in;
  reg write_enable;

  // Outputs
  wire [7:0] data_out;

  // Instantiate the RAM module
  ram_16x8 uut (
    .clk(clk),
    .addr(addr),
    .data_in(data_in),
    .write_enable(write_enable),
    .data_out(data_out)
  );

  // Clock generation
  always begin
    #5 clk = ~clk;
  end

  // Initial block
  initial begin
    // Initialize inputs
    clk = 0;
    addr = 4'b0000;
    data_in = 8'b0000_0000;
    write_enable = 0;

    // Apply write operation to store data
    addr = 4'b0001;
    data_in = 8'b1010_1010;
    write_enable = 1;
    #10 write_enable = 0;

    // Read from the RAM
    addr = 4'b0001;
    write_enable = 0;

    // Monitor signals
    $monitor("Time=%t, clk=%b, addr=%b, data_in=%b, write_enable=%b,
data_out=%b", $time, clk, addr, data_in, write_enable, data_out);

    // Run simulation for 100 clock cycles
    #100 $finish;
  end

endmodule
```

Output:





```
ModelSim> vsim -gui work.test_RAM_16x8
# vsim -gui work.test_RAM_16x8
# Start time: 22:22:54 on Jan 31,2024
# Loading work.test_RAM_16x8
# Loading work.ram_16x8
add wave -position insertpoint sim:/test_RAM_16x8/*
VSIM 3> run
# Time=                   10, clk=0, addr=0001, data_in=10101010, write_enable=0, data_out=10101010
# Time=                   15, clk=1, addr=0001, data_in=10101010, write_enable=0, data_out=10101010
# Time=                   20, clk=0, addr=0001, data_in=10101010, write_enable=0, data_out=10101010
# Time=                   25, clk=1, addr=0001, data_in=10101010, write_enable=0, data_out=10101010
# Time=                   30, clk=0, addr=0001, data_in=10101010, write_enable=0, data_out=10101010
# Time=                   35, clk=1, addr=0001, data_in=10101010, write_enable=0, data_out=10101010
# Time=                   40, clk=0, addr=0001, data_in=10101010, write_enable=0, data_out=10101010
# Time=                   45, clk=1, addr=0001, data_in=10101010, write_enable=0, data_out=10101010
# Time=                   50, clk=0, addr=0001, data_in=10101010, write_enable=0, data_out=10101010
# Time=                   55, clk=1, addr=0001, data_in=10101010, write_enable=0, data_out=10101010
# Time=                   60, clk=0, addr=0001, data_in=10101010, write_enable=0, data_out=10101010
# Time=                   65, clk=1, addr=0001, data_in=10101010, write_enable=0, data_out=10101010
# Time=                   70, clk=0, addr=0001, data_in=10101010, write_enable=0, data_out=10101010
# Time=                   75, clk=1, addr=0001, data_in=10101010, write_enable=0, data_out=10101010
# Time=                   80, clk=0, addr=0001, data_in=10101010, write_enable=0, data_out=10101010
# Time=                   85, clk=1, addr=0001, data_in=10101010, write_enable=0, data_out=10101010
# Time=                   90, clk=0, addr=0001, data_in=10101010, write_enable=0, data_out=10101010
# Time=                   95, clk=1, addr=0001, data_in=10101010, write_enable=0, data_out=10101010
```

Reference:

To view my codes, please refer to my GitHub Account.

Conclusion:

In summary of this lab, I have learned how to implement simple counters, controlled and uncontrolled asynchronous T counters. I also learned how to implement RAM in Verilog. This lab will help in my future endeavors.

The End.