

FPGA INTERNSHIP ASSIGNMENT

Submitted By : Sankaliya Mirali

Submitted To : Anoushka ma'am

Topic : Flip-Flop & Counter Coding

1. D Flip-Flop

Explanation:

This waveform shows the behavior of a D flip-flop where the output depends on the clock signal. When the clock is active, the output follows the input value. When the clock becomes inactive, the flip-flop stores the last value and ignores further changes in the input. This allows the circuit to hold data for later use.

EDA playground

New Run Save Copy

KnowHow WEBINARS Anatomy of an Embedded Linux System FREE • 1 Hour MAR 06, 2026 REGISTER NOW Resources Community Help Playgrounds Profile

Doulos does not endorse training material from other suppliers on EDA Playground.

▼ Languages & Libraries

Testbench + Design SystemVerilog/Verilog UVM / OVM None Other Libraries None OVL SVUnit

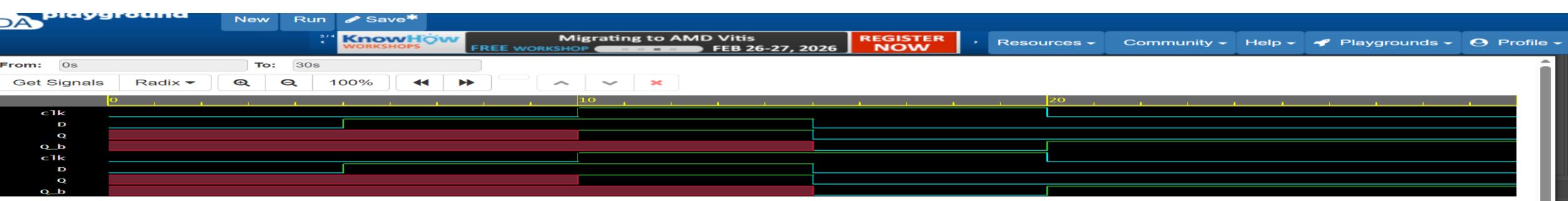
Enable TL-Verilog Enable Easier UVM Enable VUnit

▼ Tools & Simulators Icarus Verilog 12.0 Compile Options -Wall -g2012 Run Options Run Options Use run.bash shell script Open EPWave after run

```
1 module tb;
2 reg D, clk;
3 wire Q, Q_b;
4
5 D_latch uut(D, clk, Q, Q_b);
6
7 initial begin
8   $dumpfile("dump.vcd");
9   $dumpvars(0, tb);
10
11   D = 0; clk = 0;
12   #5 D = 1;
13   #5 clk = 1;
14   #5 D = 0;
15   #5 clk = 0;
16   #10 $finish;
17 end
18
19 endmodule
20
21
22
23
24
```

Log Share

The screenshot shows the EDA playground interface. On the left, there's a configuration sidebar with tabs for 'Languages & Libraries' (set to SystemVerilog/Verilog), 'Tools & Simulators' (set to Icarus Verilog 12.0), and 'Run Options'. The main area displays Verilog code for a testbench. The code defines a module 'tb' with a 'D_latch' instance. It sets initial values for 'D' and 'clk', and uses delays to change 'D' and 'clk' at specific times. It also includes a '#10 \$finish' statement. On the right, there's a waveform simulation window showing signals for 'c1k', 'D', 'Q', 'Q_b', 'c1k', 'D', 'Q', and 'Q_b'. The waveforms show that 'Q' and 'Q_b' follow the 'D' signal when 'clk' is high, and hold their last value when 'clk' goes low. A note at the bottom says: 'Note: To revert to EPWave opening in a new browser window, set that option on your profile page.'



2.D-Flip-Flop(Negative edge Triggered)

EDA playground

New Run Save Copy

2/4 KnowHow WEBINARS What is an SBOM and why should I care? FREE • 1 Hour FEB 25, 2026 REGISTER NOW

Doulos does not endorse training material from other suppliers on EDA Playground.

Languages & Libraries

Testbench + Design SystemVerilog/Verilog

UVM / OVM None

Other Libraries None OVL SVUnit

Enable TL-Verilog

Enable Easier UVM

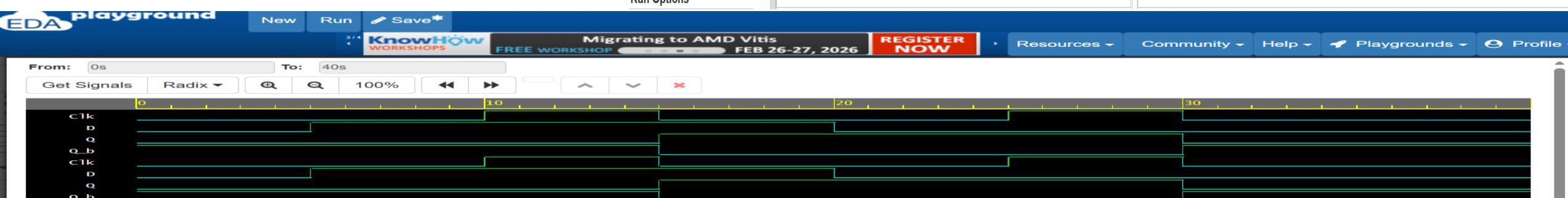
Enable VUnit

Tools & Simulators Icarus Verilog 12.0

Compile Options -Wall -g2012

Run Options

```
1 module tb;
2 reg D, Clk;
3 wire Q, Q_b;
4
5 D_FF uut(D, Clk, Q, Q_b);
6
7 initial begin
8   $dumpfile("dump.vcd");           // IMPORTANT
9   $dumpvars(0, tb);               // IMPORTANT
10
11   D = 0; Clk = 0;
12
13   #5 D = 1;
14   #5 Clk = 1;
15   #5 Clk = 0; // negedge → Q update
16
17   #5 D = 0;
18   #5 Clk = 1;
19   #5 Clk = 0; // negedge → Q update
20
21   #10 $finish;
22 end
23
24 endmodule
25
```



3. Positive Edge D Flip-Flop

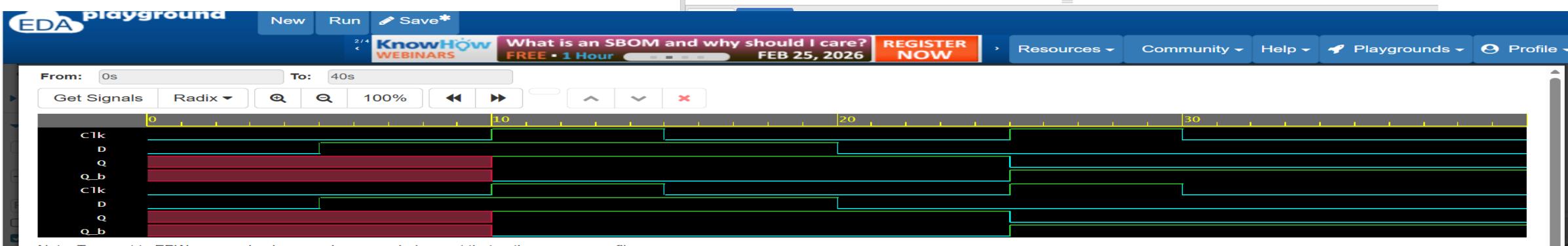
EXPLANATION:

In this experiment, a positive edge triggered D Flip-Flop was simulated. The output updates only at the rising edge of the clock. The waveform clearly shows the output changing when the clock goes from low to high.

The screenshot shows the EDA playground interface with two tabs of Verilog code. The left tab contains the testbench code (tb.v) and the right tab contains the D_FF_pos module code.

```
tb.v
module tb;
reg D, Clk;
wire Q, Q_b;
D_FF_pos uut(D, Clk, Q, Q_b);
initial begin
    $dumpfile("dump.vcd");
    $dumpvars(0, tb);
    D = 0; Clk = 0;
    #5 D = 1;
    #5 Clk = 1; // posedge + Q = 1
    #5 Clk = 0;
    #5 D = 0;
    #5 Clk = 1; // posedge + Q = 0
    #5 Clk = 0;
    #10 $finish;
end
endmodule
```

```
D_FF_pos
input D,
input Clk,
output reg Q,
output Q_b
);
assign Q_b = ~Q;
always @(posedge Clk)
begin
    Q <= D;
end
endmodule
```



Note: To revert to EPWave opening in a new browser window, set that option on your profile page.

4. Blocking Assignment

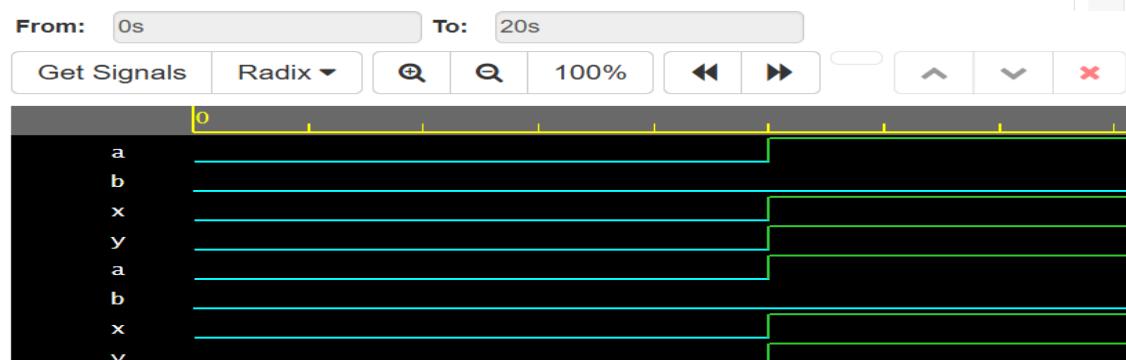
EXPLANATION:

Blocking assignment (=) was implemented and simulated. In this method, statements execute sequentially and update immediately. The waveform shows how values change step-by-step based on execution order.

```
1 module tb;
2
3 reg a, b;
4 wire x, y;
5
6 blocking_example uut(a, b, x, y);
7
8 initial begin
9   $dumpfile("dump.vcd");
10  $dumpvars(0, tb);
11
12  a = 0; b = 0;
13  #5 a = 1;
14  #5 a = 0;
15  #5 a = 1;
16  #5 $finish;
17 end
18
19 endmodule// Code your testbench here
20 // or browse Examples
21
```

Playground.

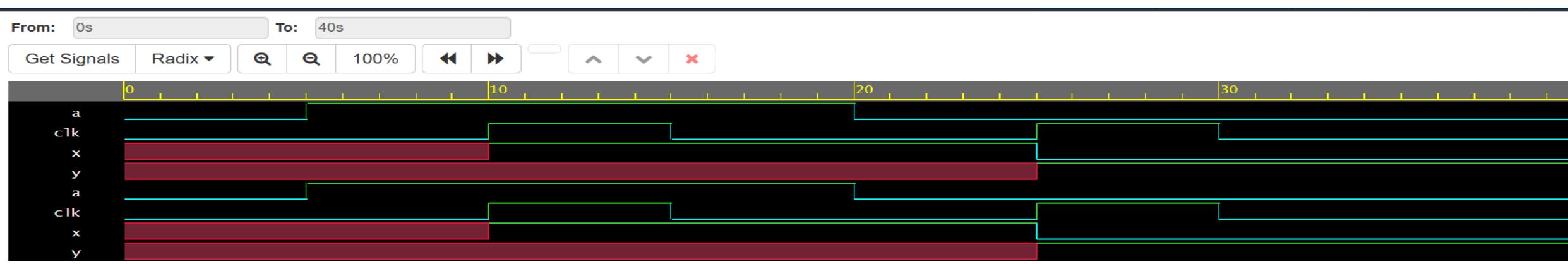
```
1 module blocking_example(
2   input a,
3   input b,
4   output reg x,
5   output reg y
6 );
7
8 always @(*)
9 begin
10   x = a; // Blocking assignment
11   y = x; // y ne updated x ni value turant male
12 end
13
14 endmodule// Code your design here
15
```



5. Non-Blocking Assignment

EXPLANATION:

Non-blocking assignment (`<=`) was implemented. In this case, all assignments update simultaneously at the clock edge. The simulation shows correct sequential behavior, making it suitable for flip-flop and sequential circuit design.



The screenshot shows a simulation interface with a waveform viewer at the bottom. The top part displays two Verilog code snippets. The left snippet is a testbench module (`tb`) that initializes variables `a`, `x`, and `y`, and performs non-blocking assignments. The right snippet is a design module (`nonblocking_example`) that contains an `always` block with non-blocking assignments for `x` and `y`.

```
module tb;
reg clk, a;
wire x, y;
nonblocking_example uut(clk, a, x, y);

initial begin
    $dumpfile("dump.vcd");
    $dumpvars(0, tb);
    clk = 0;
    a = 0;
    #5 a = 1;
    #5 clk = 1; // posedge → x=1, y=0 (old x)
    #5 clk = 0;

    #5 a = 0;
    #5 clk = 1; // posedge → x=0, y=1
    #5 clk = 0;

    #10 $finish;
end

endmodule// Code your testbench here
// or browse Examples
```

```
module nonblocking_example(
input clk,
input a,
output reg x,
output reg y
);

always @(posedge clk)
begin
    x <= a; // Non-blocking assignment
    y <= x; // y ne x ni old value male
end

endmodule// Code your design here
```

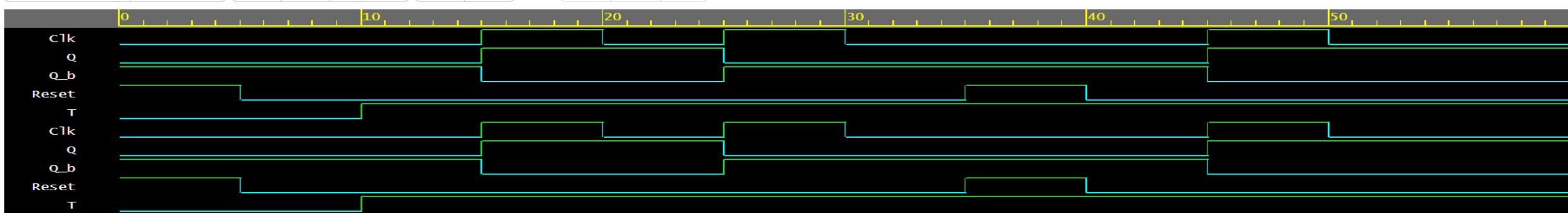
6.T flip-Flop (Asynchronous)

EXPLANATION:

An asynchronous T Flip-Flop with reset was designed. The output toggles when $T = 1$ and remains unchanged when $T = 0$. The waveform verifies proper toggle and reset operation.

```
1 module tb;
2
3 reg T, Clk, Reset;
4 wire Q, Q_b;
5
6 T_FF_async uut(T, Clk, Reset, Q, Q_b);
7
8 initial begin
9   $dumpfile("dump.vcd");
10  $dumpvars(0, tb);
11
12  Clk = 0;
13  Reset = 1;
14  T = 0;
15
16  #5 Reset = 0; // Release reset
17
18  #5 T = 1;
19  #5 Clk = 1; // Toggle - Q = 1
20  #5 Clk = 0;
21
22  #5 Clk = 1; // Toggle - Q = 0
23  #5 Clk = 0;
24
25  #5 Reset = 1; // Asynchronous reset - Q = 0
immediately
26  #5 Reset = 0;
27
28  #5 Clk = 1; // Toggle again
29  #5 Clk = 0;
30
31 #10 $finish;
```

```
1 module T_FF_async(
2   input T,
3   input Clk,
4   input Reset, // Asynchronous Reset
5   output reg Q,
6   output Q_b
7 );
8
9 assign Q_b = ~Q;
10
11 // Asynchronous Reset + Positive edge triggered
12 always @ (posedge Clk or posedge Reset)
13 begin
14   if (Reset)
15     Q <= 0;
16   else if (T)
17     Q <= ~Q; // Toggle
18 end
19
20 endmodule// Code your design here
21
```



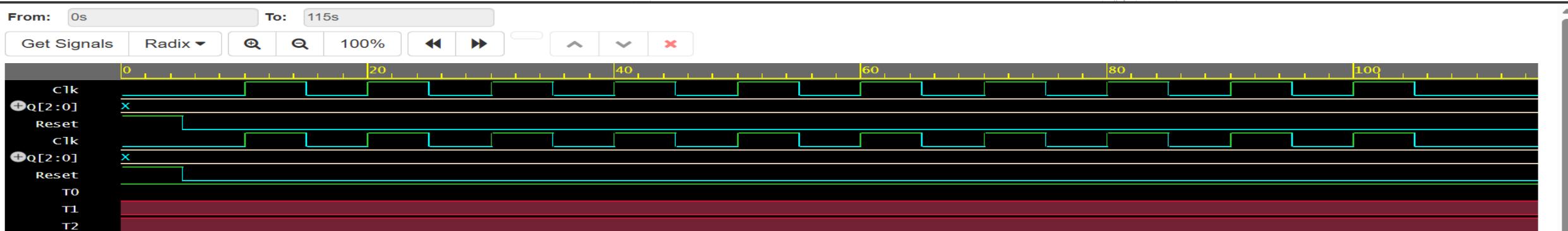
7.Synchronous Up Counter(T FF based)

EXPLANATION:

A synchronous up counter using T Flip-Flops was designed and simulated. All flip-flops operate using the same clock signal, and the counter increments sequentially. The waveform shows proper counting without ripple delay.

```
1 module tb;
2
3 reg Clk, Reset;
4 wire [2:0] Q;
5
6 sync_up_counter uut(Clk, Reset, Q);
7
8 initial begin
9     $dumpfile("dump.vcd");
10    $dumpvars(0, tb);
11
12    Clk = 0;
13    Reset = 1;
14
15    #5 Reset = 0;
16
17    repeat (10) begin
18        #5 Clk = 1;
19        #5 Clk = 0;
20    end
21
22    #10 $finish;
23 end
24
25 endmodule// Code your testbench here
26 // or browse Examples
27
```

```
1 module sync_up_counter(
2     input Clk,
3     input Reset,
4     output reg [2:0] Q
5 );
6
7 // T inputs logic
8 wire T0, T1, T2;
9
10 assign T0 = 1'b1;
11 assign T1 = Q[0];
12 assign T2 = Q[0] & Q[1];
13
14 // Synchronous T Flip-Flop operation
15 always @ (posedge Clk)
16 begin
17     if (Reset)
18         Q <= 3'b000;
19     else begin
20         if (T0) Q[0] <= ~Q[0];
21         if (T1) Q[1] <= ~Q[1];
22         if (T2) Q[2] <= ~Q[2];
23     end
24 end
25
26 endmodule// Code your design here
27
```



Difference between Synchronous and Asynchronous

- In synchronous systems, all operations are controlled by a common clock signal, so all outputs change at the same time. This makes the circuit faster and more reliable. In asynchronous systems, there is no common clock for all stages, and each stage is triggered by the output of the previous stage. Because of this, the outputs change one after another, which may cause delay and make the system slower.