FPGA LAB # 5

Group 13

Hassan Ahmed
Hamza Shabir
Neha Chaudhary

Submitted to: Sir Asim

5.1 Objectives:

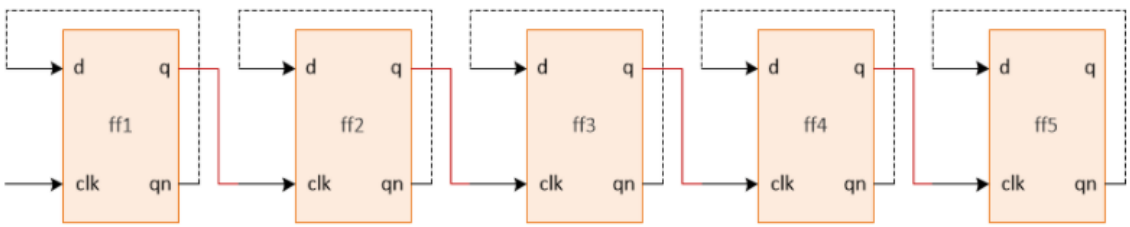
- Design a 4-bit ripple carry counter using D flip flops.
- Design a 4-bit ripple carry counter using T flip flops.
- Write Verilog code for both counters and verify using the test bench.

5.2 Background:

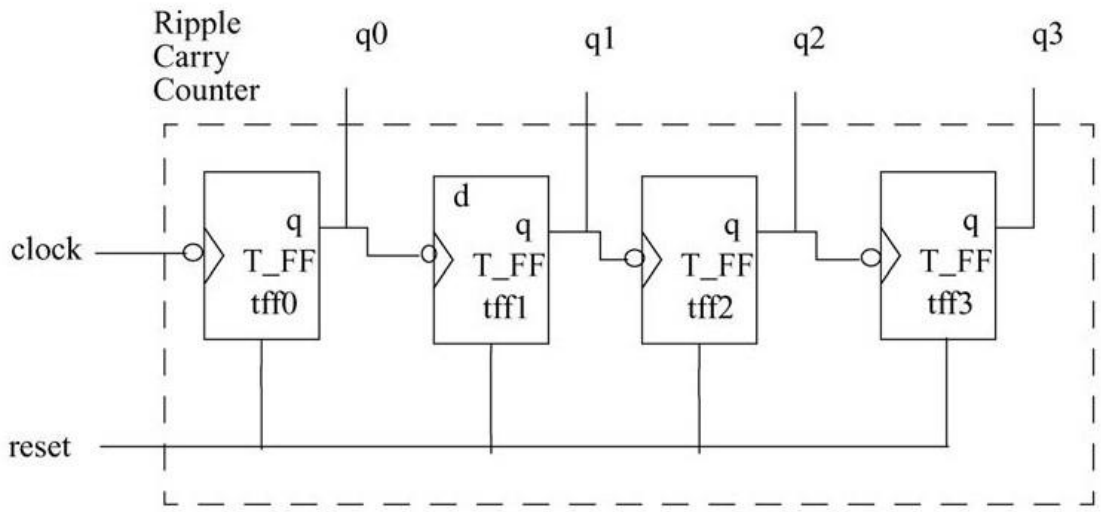
A four bit ripple carry counter can be implemented using both d and t flip flops with some slight changes to the code. The truth table for the four bit ripple carry counter is shown below:

Clock Input	Q3	Q2	Q1	Q0
1	0	0	0	1
2	0	0	1	0
3	0	1	0	0
4	1	0	0	0
5	0	0	0	1
6	0	0	1	0
7	0	1	0	0
8	1	0	0	0

When implementing the ripple carry counter we have to make sure that the d flip flop reset should be asynchronous and it works on the positive edge of the clock. For a 4 bit counter we will need 4 instances of d flip flops. The least significant one will use the system clock while its output will act as input for the next one and so on. The output of the counter will be taken from the not output of the d flip flop. The schematic diagram is shown below:



Now to implement a ripple counter using t flip flops we first implement t flip flop using d flip flop but we have to make a change in d flip flop which is that now the d flip flop should work at the negative edge of clock. The rest will remain the same, four instances of t flip flops will be used to implement a 4 bit ripple carry counter. The schematic diagram is shown below.



5.3 Procedure:

1. In the first step we implemented d flip flop using behavioural abstraction.
2. Similarly t flip flop was implemented using a single instance of d flip flop.
3. Next 4 bit ripple carry counter was implemented using 4 instances of d flip flop.
4. Similarly again the same counter was implemented using 4 instances of t flip flop.
5. Test bench was written to verify the codes.
6. Codes were verified by observing the simulations.

5.4Verilog Codes

Ripple Carry Counter using D Flip Flop:

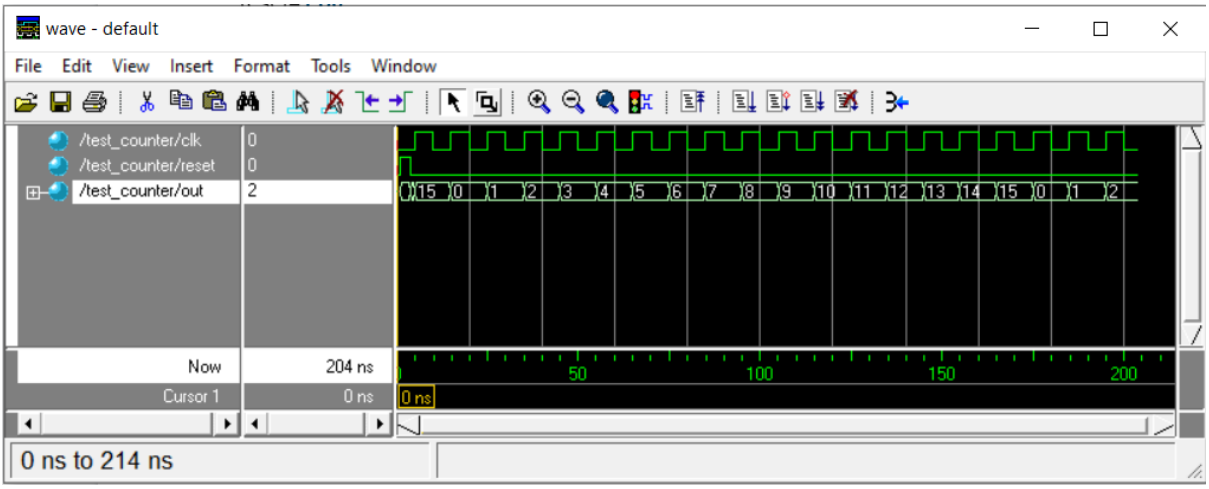
```
//D Flip Flop module

module dff(d, clk, reset, q, qn);
input d, clk, reset;
output reg q;
output qn;

always @(posedge clk or reset)
if (reset)
q <= 0;
else
q <= d;

assign qn=~q;
```

```
endmodule

//Ripple  Carry counter using D flip flops
module ripple_counter(out, clk, reset);
input clk, reset;
output[3:0] out;
wire q0, qn0, q1, qn1, q2, qn2, q3, qn3, q4, qn4;

dff d0(.d(qn0), .clk(clk), .reset(reset), .q(q0), .qn(qn0));
dff d1(.d(qn1), .clk(q0), .reset(reset), .q(q1), .qn(qn1));
dff d2(.d(qn2), .clk(q1), .reset(reset), .q(q2), .qn(qn2));
dff d3(.d(qn3), .clk(q2), .reset(reset), .q(q3), .qn(qn3));

assign out={qn3, qn2, qn1, qn0};
endmodule
```

Test Bench:

```
module test_counter();
reg clk, reset;
wire[3:0] out;
ripple_counter R1(.out(out), .clk(clk), .reset(reset));
initial
begin
clk=1'b0;
reset=1'b0;
#1 reset=1'b1;
#3 reset=1'b0;
#200 $stop;
 end
 always #5 clk=~clk;
 endmodule
```

Simulation Result:



Ripple Carry Counter using T Flip Flops:

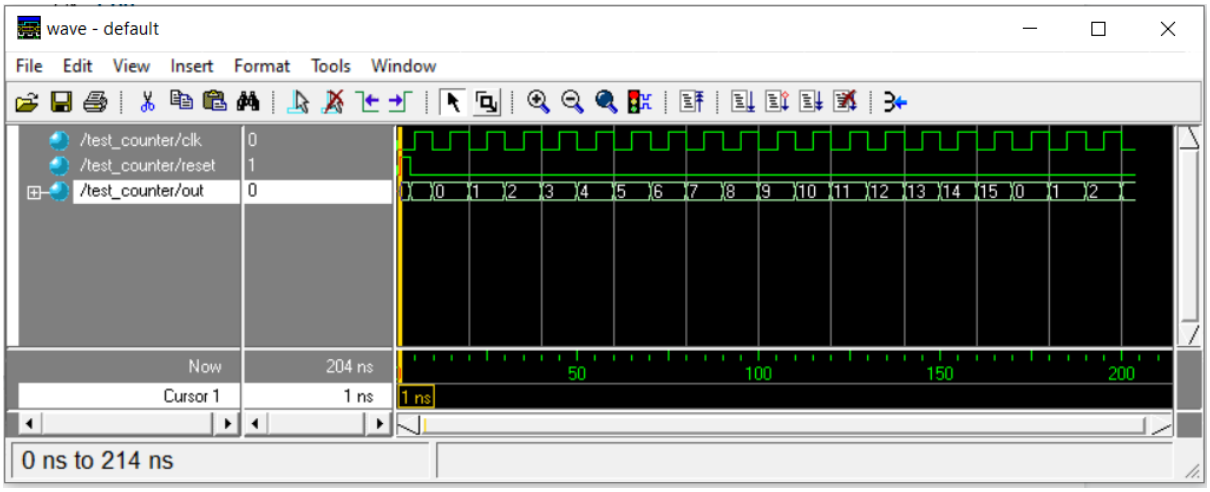
```
//D Flip Flop module

module dff(d, clk, reset, q, qn);
input d, clk, reset;
output reg q;
output qn;

always @(negedge clk or reset)
if (reset)
q <= 0;
else
q <= d;

assign qn=~q;
endmodule

//T Flip Flop module
module tff(clk, reset, q);
input clk, reset;
output q;
wire d;
dff d5(.d(d), .clk(clk), .reset(reset), .q(q), .qn(d));
endmodule

//Ripple Carry Counter using T Flip Flops
module ripple_counter_tff(out, clk, reset);
input clk, reset;
output[3:0] out;
```

```
tff t0(.clk(clk), .reset(reset), .q(out[0]));
tff t1(.clk(out[0]), .reset(reset), .q(out[1]));
tff t2(.clk(out[1]), .reset(reset), .q(out[2]));
tff t3(.clk(out[2]), .reset(reset), .q(out[3]));
endmodule
```

Test Bench:

```
module test_counter();
reg clk, reset;
wire[3:0] out;
ripple_counter_tff R1(.out(out), .clk(clk), .reset(reset));
initial
begin
clk=1'b0;
reset=1'b0;
#1 reset=1'b1;
#3 reset=1'b0;
#200 $stop;
 end
 always #5 clk=~clk;
 endmodule
```

Simulation Results:



5.5 Observations:

Ripple carry counters can be implemented using instances of d and t flip flops. Both the codes are primarily similar with the main difference that both are operating at different edges of clock. Making the reset synchronous with the clock will not reset all the flip flops at a single edge of the clock because other flip flops are operating on the basis of change in output of the previous one.

5.6 Conclusion:

To implement a synchronous reset we have mentioned reset in the sensitivity list. The reset we implemented is an active high reset. The counter can count till binary 1111 i.e 15 in decimal. In simulations the radix of the output is changed to unsigned to properly observe the output.