Digital Design Lab Report

Name	Dev Goel
Roll No.	B20CS090
Experiment Number	**VI**

Objective

(i) Write a program to implement a D f/f with synchronous and asynchronous reset.

(ii) Test the operation of blocking and non-blocking assignments using two D flip flops.

(iii) Implement the 8-bit serial addition (A+B) operation using shift register and a Full Adder. Load the two registers from input using parallel loading. The result should be stored in register A. Find the number of clocks cycles your design would take to implement the complete operation.

EXPERIMENT (i): Write a program to implement a D f/f with synchronous and asynchronous reset.

- **D f/f with synchronous reset:**

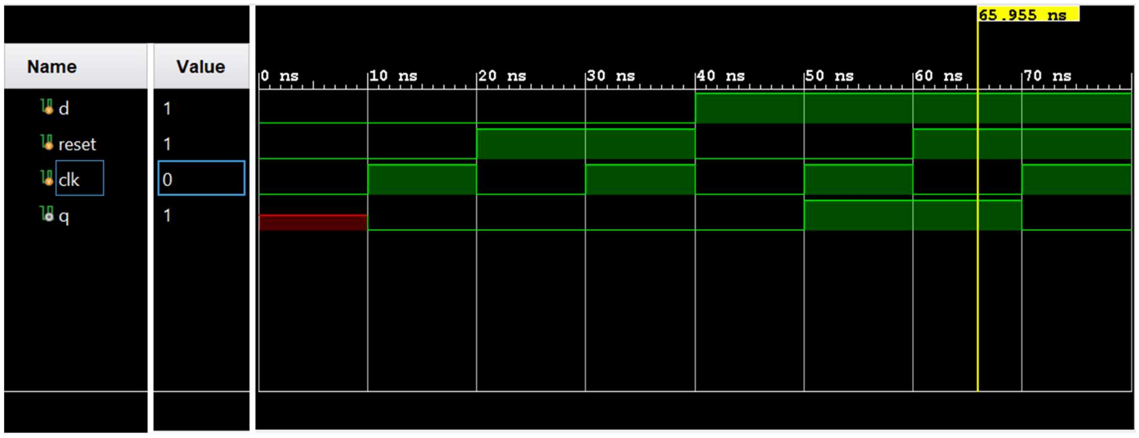
```
`timescale 1ns / 1ps
module d_flip_flop_syn(d, clk, reset, q);

input d, clk, reset;
output reg q;

always@(posedge clk)
   if(reset)
      q = 0;
```

```
    else
        q = d;
endmodule
```

Test Bench:

```
`timescale 1ns / 1ps
module test_d_flip_flop_syn;

reg d, reset, clk;
wire q;

d_flip_flop_syn dff(d, clk, reset, q);

always #10 clk=~clk;

initial
    begin
        d = 0; clk = 0; reset = 0;
        #20 reset = 1;
        #20 d = 1; reset = 0;
        #20 reset = 1;
    end
initial #80 $finish;
endmodule
```

Waveform:



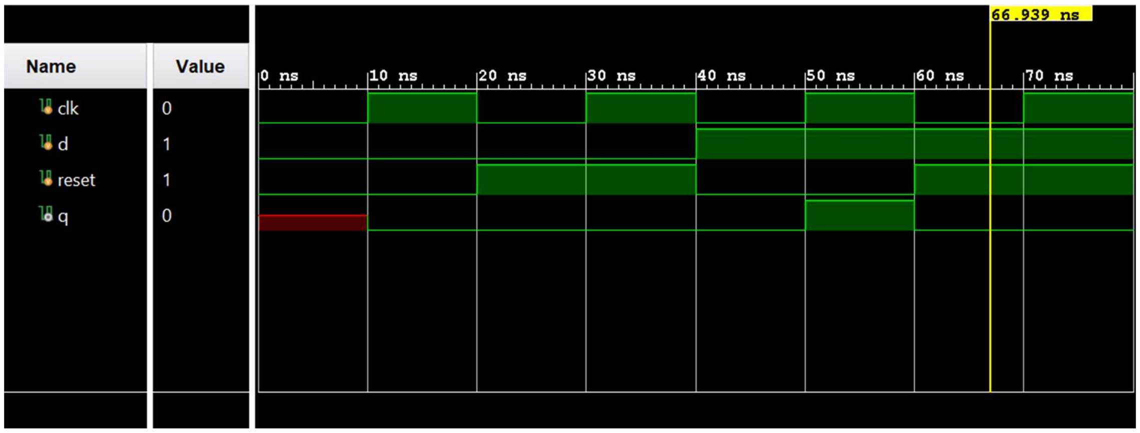
- **D f/f with asynchronous reset:**

```
`timescale 1ns / 1ps
module d_flip_flop_asyn(d, clk, reset, q);

input d, clk, reset;
output reg q;

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

Test Bench:

```
`timescale 1ns / 1ps
module test_d_flip_flop_syn;

reg d, reset, clk;
wire q;

d_flip_flop_syn dff(d, clk, reset, q);

always #10 clk=~clk;

initial
   begin
      d = 0; clk = 0; reset = 0;
      #20 reset = 1;
      #20 d = 1; reset = 0;
      #20 reset = 1;
   end
initial #80 $finish;
endmodule
```

Waveform:



EXPERIMENT (ii): Test the operation of blocking and non-blocking assignments using two D flip flops.

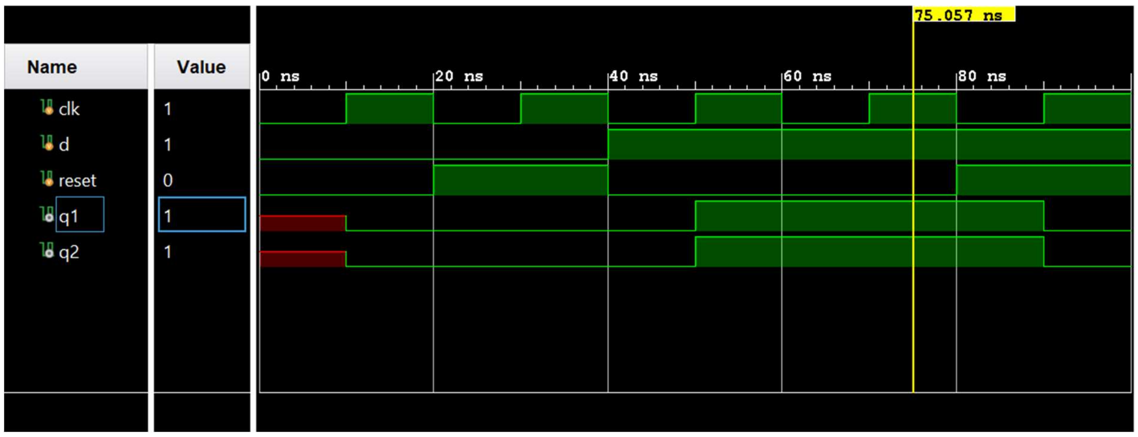
- **Operation of blocking assignment using two D flip flops:**

```
`timescale 1ns / 1ps
module d_ff_blocking(d, clk, reset, q1, q2);

input d, clk, reset;
output reg q1, q2;

always@(posedge clk)
   begin
      if(reset)
         begin
            q1 = 0;
            q2 = 0;
         end
      else
         begin
            q1 = d;
            q2 = q1;
            end
    end
endmodule
```

Test Bench:

```
`timescale 1ns / 1ps
module test_d_ff_blocking;

reg clk, d, reset;
wire q1, q2;

d_ff_blocking dffb(d, clk, reset, q1, q2);

always #10 clk=~clk;

initial
    begin
        d = 0; clk = 0; reset = 0;
        #20 reset = 1;
        #20 d = 1; reset = 0;
        #40 reset = 1;
    end
initial #100 $finish;
endmodule
```

Waveform:



Name	Value
clk	1
d	1
reset	0
q1	1
q2	1

- **Operation of non-blocking assignment using two D flip flops:**

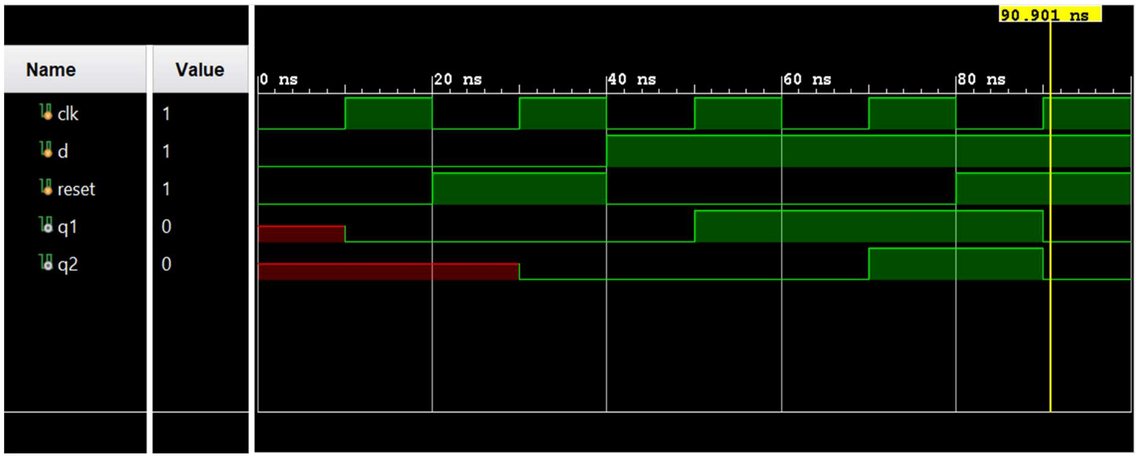
```
`timescale 1ns / 1ps
module d_ff_non_blocking(d, clk, reset, q1, q2);

input d, clk, reset;
output reg q1, q2;

always@(posedge clk)
   begin
      if(reset)
         begin
            q1 <= 0;
            q2 <= 0;
         end
      else
         begin
            q1 <= d;
            q2 <= q1;
         end
   end
endmodule
```

Test Bench:

```
`timescale 1ns / 1ps
module test_d_ff_non_blocking;

reg clk, d, reset;
wire q1, q2;

d_ff_non_blocking dffnb(d, clk, reset, q1, q2);

always #10 clk=~clk;
```

```
initial
   begin
      d = 0; clk = 0; reset = 0;
      #20 reset = 1;
      #20 d = 1; reset = 0;
      #40 reset = 1;
   end
initial #100 $finish;
endmodule
```

Waveform:



EXPERIMENT (iii): Implement the 8-bit serial addition (A+B) operation using shift register and a Full Adder. Load the two registers from input using parallel loading. The result should be stored in register A. Find the number of clocks cycles your design would take to implement the complete operation.

8-bit serial addition:

```
`timescale 1ns / 1ps
module SerialAdder(a, b, cout, sum, clk, reset);

input [7:0]a, b;
input clk, reset;
```

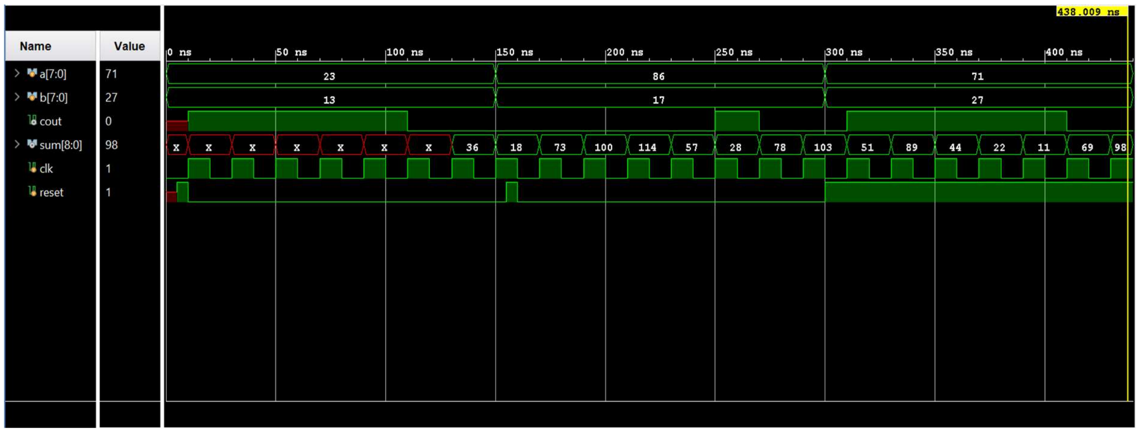
```
output reg cout;
output reg [8:0]sum;
reg [7:0]an, bn;
reg cin;

always@(posedge reset)
begin
   if(reset == 1)
      begin
         cin = 0;
         an[7:0] = a[7:0];
         bn[7:0] = b[7:0];
      end
end

always@(posedge clk)
   begin
      sum[7] = an[0] ^ bn[0] ^ cin;
      sum = sum >> 1;
      cout = (an[0] & bn[0]) | (cin & bn[0]) | (an[0] & cin);
      an = an>>1;
      bn = bn>>1;
      cin = cout;
   end

endmodule
```

Test Bench:

```
`timescale 1ns / 1ps
module Test_SerialAdder;

reg [7:0]a, b;
reg clk, reset;
wire [8:0]sum;
wire cout;
```

```
SerialAdder sa(a, b, cout, sum, clk, reset);

initial
   begin
      clk = 0;
   end

always #10 clk = !clk;

initial
   begin
      a=8'b00010111;b=8'b00001101;
      #5 reset=1'b1;
      #5 reset=1'b0;
      #140 a=8'b01010110;b=8'b00010001;
      #5 reset=1'b1;
      #5 reset=1'b0;
      #140 a=8'b01000111;b=8'b00011011;
      reset=1'b1;
      #140 $stop;
   end
endmodule
```

Waveform:


438.009 ns
Name
Value
a[7:0]
71
b[7:0]
27
cout
0
sum[8:0]
98
clk
1
reset
1
0 ns
50 ns
100 ns
150 ns
200 ns
250 ns
300 ns
350 ns
400 ns
23
86
71
13
17
27
X X X X X X X 36 18 73 100 114 57 28 78 103 51 89 44 22 11 69 98