DIGITAL DESIGN

LAB-6

1.D Flip Flop:

i) Asynchronous:

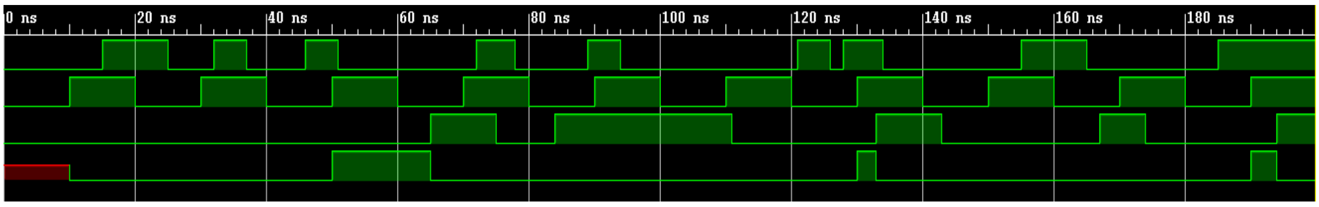
Logic Code:

```
module asynchronous_Dff(D,clk,reset,Q);
input D,clk,reset;
output reg Q;

always @(posedge clk,posedge reset)
begin
    if(reset)
        Q=0;
    else
        Q=D;
end
endmodule
```

Test Bench:

```
module test_asynchronous_Dff();
reg D,clk,reset;
wire Q;
asynchronous_Dff asDff(D,clk,reset,Q);

initial
begin
    D=0;clk=0;reset=0;
    #10 clk=1;#5 D=1;
    #5 clk=0; #5 D=0;
    #5 clk=1; #2 D=1; #5 D=0;
    #3 clk=0; #6 D=1;
    #4 clk=1; #1 D=0;
    #9 clk=0; #5 reset=1;
    #5 clk=1; #2 D=1; #3 reset=0; #3 D=0;
    #2 clk=0; #4 reset=1; #3 reset=1; #2 D=1;
    #1 clk=1; #4 D=0;
    #6 clk=0; #7 reset=1;
    #3 clk=1; #1 reset=0;
    #9 clk=0; #1 D=1; #5 D=0; #2 D=1;
    #2 clk=1; #3 reset=1; #1 D=0;
    #6 clk=0; #3 reset=0;
    #7 clk=1; #5 D=1;
    #5 clk=0; #5 D=0; #2 reset=1;
    #3 clk=1; #4 reset=0;
    #6 clk=0; #5 D=1;
    #5 clk=1; #4 reset=1;
end
initial #200 $finish;
endmodule
```

D
Clock
Reset
Q



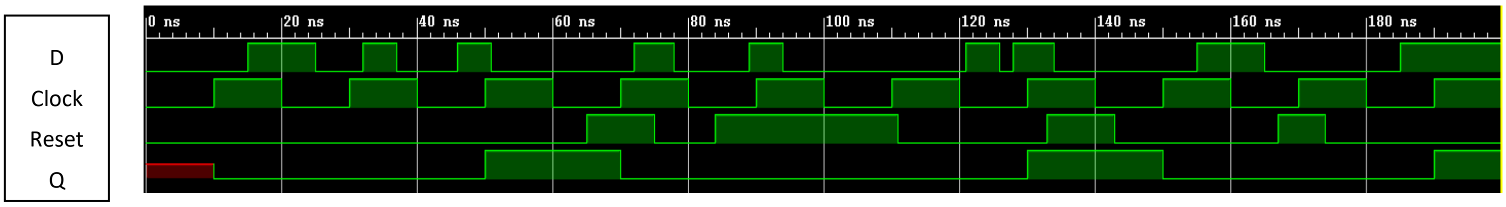
ii) Synchronous:

Logic Code:

```
module synchronous_Dff(D,clk,reset,Q);
input D,clk,reset;
output reg Q;

always @(posedge clk)
begin
    if(reset)
        Q=0;
    else
        Q=D;
end
endmodule
```

Test Bench:

```
module test_synchronous_Dff();
reg D,clk,reset;
wire Q;
synchronous_Dff sDff(D,clk,reset,Q);

initial
begin
    D=0;clk=0;reset=0;
    #10 clk=1;#5 D=1;
    #5 clk=0; #5 D=0;
    #5 clk=1; #2 D=1; #5 D=0;
    #3 clk=0; #6 D=1;
    #4 clk=1; #1 D=0;
    #9 clk=0; #5 reset=1;
    #5 clk=1; #2 D=1; #3 reset=0; #3 D=0;
    #2 clk=0; #4 reset=1; #3 reset=1; #2 D=1;
    #1 clk=1; #4 D=0;
    #6 clk=0; #7 reset=1;
    #3 clk=1; #1 reset=0;
    #9 clk=0; #1 D=1; #5 D=0; #2 D=1;
    #2 clk=1; #3 reset=1; #1 D=0;
    #6 clk=0; #3 reset=0;
    #7 clk=1; #5 D=1;
    #5 clk=0; #5 D=0; #2 reset=1;
    #3 clk=1; #4 reset=0;
    #6 clk=0; #5 D=1;
    #5 clk=1;
end
initial #200 $finish;
endmodule
```



2.D Flip Flops (assignment):

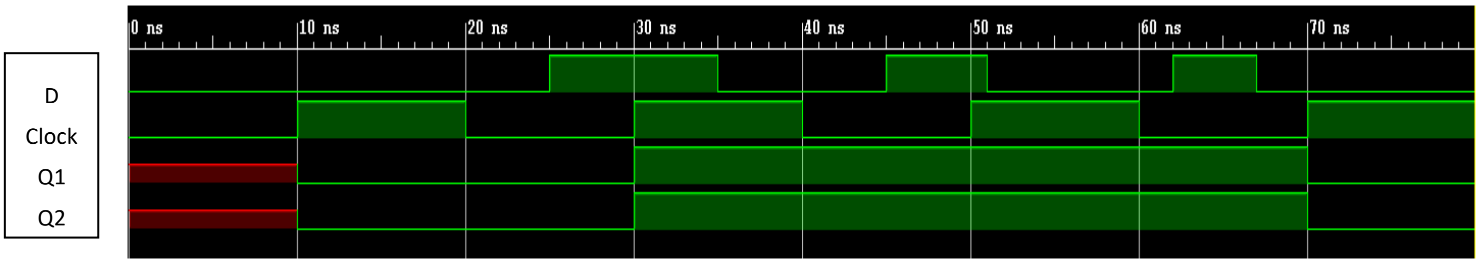
i) Blocking:

Logic Code:

```
module blocking_Dffs(D,clk,Q1,Q2);
input D,clk;
output reg Q1,Q2;

always @(posedge clk)
begin
    Q1=D;
    Q2=Q1;
end
endmodule
```

Test Bench:

```
module test_blocking_Dffs();
reg D,clk;
wire Q1,Q2;
blocking_Dffs bDffs(D,clk,Q1,Q2);

initial
begin
    clk=0; D=0;
    #10 clk=1;
    #10 clk=0; #5 D=1;
    #5 clk=1; #5 D=0;
    #5 clk=0; #5 D=1;
    #5 clk=1; #1 D=0;
    #9 clk=0; #2 D=1; #5 D=0;
    #3 clk=1;
end
initial #80 $finish;
endmodule
```



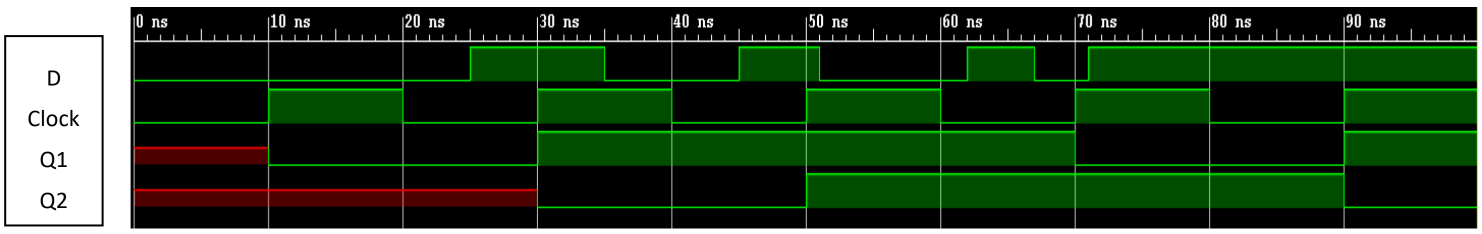
ii) Non-blocking:

Logic Code:

```
module non_blocking_Dffs(D,clk,Q1,Q2);
input D,clk;
output reg Q1,Q2;

always @(posedge clk)
begin
    Q1<=D;
    Q2<=Q1;
end
endmodule
```

Test Bench:

```
module test_non_blocking_Dffs();
reg D,clk;
wire Q1,Q2;
non_blocking_Dffs nbDffs(D,clk,Q1,Q2);

initial
begin
    clk=0; D=0;
    #10 clk=1;
    #10 clk=0; #5 D=1;
    #5 clk=1; #5 D=0;
    #5 clk=0; #5 D=1;
    #5 clk=1; #1 D=0;
    #9 clk=0; #2 D=1; #5 D=0;
    #3 clk=1; #1 D=1;
    #9 clk=0;
    #10 clk=1;
end
initial #100 $finish;
endmodule
```



3.Serial Adder:

Logic Code:

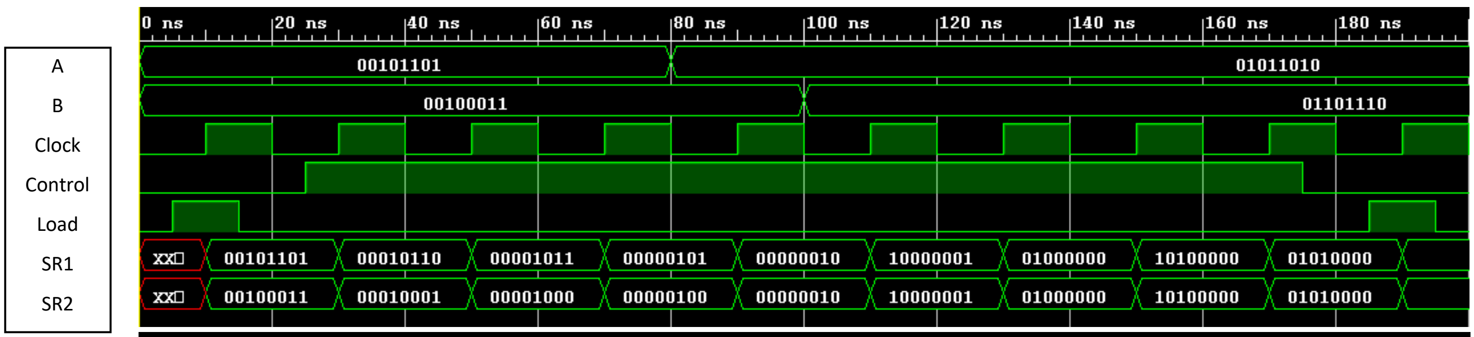
```
module serial_adder(A,B,clk,ctrl,load,SR1,SR2);
input [7:0]A,B;
input clk,ctrl,load;
output reg [7:0]SR1,SR2;
wire S,Carry;
reg C=0;
wire [7:0]D,E;

assign D[7]=load?A[7]:SR1[7];
assign D[6]=load?A[6]:SR1[6];
assign D[5]=load?A[5]:SR1[5];
assign D[4]=load?A[4]:SR1[4];
assign D[3]=load?A[3]:SR1[3];
assign D[2]=load?A[2]:SR1[2];
assign D[1]=load?A[1]:SR1[1];
assign D[0]=load?A[0]:SR1[0];

assign E[7]=load?B[7]:SR2[7];
assign E[6]=load?B[6]:SR2[6];
assign E[5]=load?B[5]:SR2[5];
assign E[4]=load?B[4]:SR2[4];
assign E[3]=load?B[3]:SR2[3];
assign E[2]=load?B[2]:SR2[2];
assign E[1]=load?B[1]:SR2[1];
assign E[0]=load?B[0]:SR2[0];

always @(posedge clk)
begin
    SR1[0]=D[0];
    SR1[1]=D[1];
    SR1[2]=D[2];
    SR1[3]=D[3];
    SR1[4]=D[4];
    SR1[5]=D[5];
    SR1[6]=D[6];
    SR1[7]=D[7];

    SR2[0]=E[0];
    SR2[1]=E[1];
    SR2[2]=E[2];
    SR2[3]=E[3];
    SR2[4]=E[4];
    SR2[5]=E[5];
    SR2[6]=E[6];
    SR2[7]=E[7];
end

assign S=SR1[0]^SR2[0]^C;
assign Carry=(SR1[0]&SR2[0])|(SR2[0]&C)|(C&SR1[0]);
```

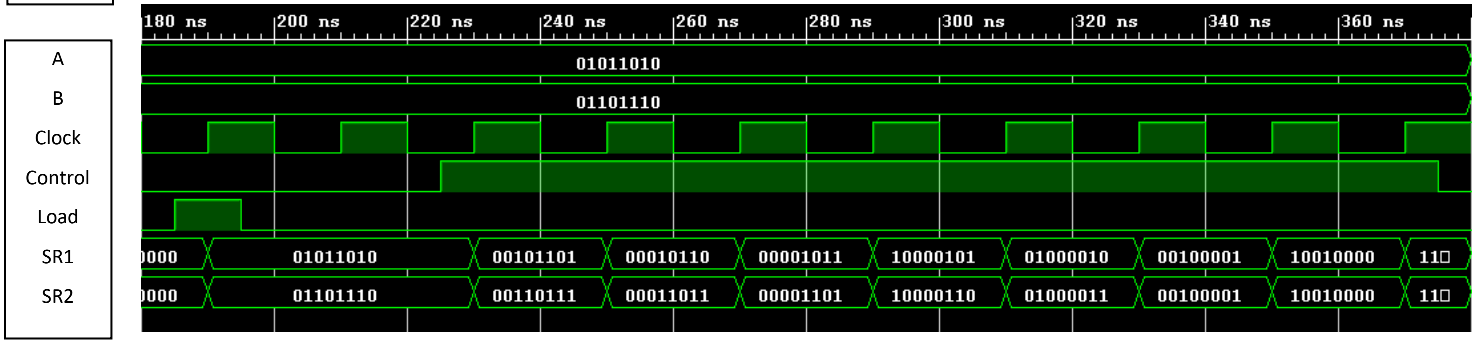
```
always @(posedge clk)
begin
    if(ctrl)
    begin
        if(Carry==1'bx)
            C=0;
        else
            C=Carry;
        SR1[0]=SR1[1];
        SR1[1]=SR1[2];
        SR1[2]=SR1[3];
        SR1[3]=SR1[4];
        SR1[4]=SR1[5];
        SR1[5]=SR1[6];
        SR1[6]=SR1[7];
        SR1[7]=S;
        SR2[0]=SR2[1];
        SR2[1]=SR2[2];
        SR2[2]=SR2[3];
        SR2[3]=SR2[4];
        SR2[4]=SR2[5];
        SR2[5]=SR2[6];
        SR2[6]=SR2[7];
        SR2[7]=S;
    end
    else
        C=0;
end
endmodule
```

Test Bench:

```
module test_serial_adder();
reg [7:0]A,B;
reg clk,ctrl,load;
wire [7:0] SR1,SR2;
serial_adder SA(A,B,clk,ctrl,load,SR1,SR2);
initial
begin
    load=0;clk=0;ctrl=0;A=8'd45;B=8'd35;#5 load=1;
    #5 clk=1; #5 load=0;
    #5 clk=0; #5 ctrl=1;
    #5 clk=1;
    #10 clk=0;
    #10 clk=1;
    #10 clk=0;
    #10 clk=1;
    #10 clk=0;A=8'd90;
    #10 clk=1;
    #10 clk=0;B=8'd110;
    #10 clk=1;
    #10 clk=0;
    #10 clk=1;
    #10 clk=0;
    #10 clk=1;
```

```
        #10 clk=0;
        #10 clk=1; #5 ctrl=0;
        #5 clk=0;#5load=1;
        #5 clk=1; #5load=0;
        #5 clk=0;
        #10 clk=1;
        #10 clk=0;#5 ctrl=1;
        #5 clk=1;#10 clk=0;#10 clk=1;
        #10 clk=0;#10 clk=1;#10 clk=0;#10 clk=1;
        #10 clk=0;#10 clk=1;#10 clk=0;#10 clk=1;
        #10 clk=0;#10 clk=1;#10 clk=0;#10 clk=1;
        #5 ctrl=0; #5clk=0;
    end
    initial #380 $finish;
    endmodule
```





It takes 9 cycles of clock to get the final ouput of 8 bit serial adder(1 for loading the numbers or data to the Shift Resister and 8 for addition operation).

Control input specifies when to start the addition operation.

Load is used for parallel load of Data to the SRs.