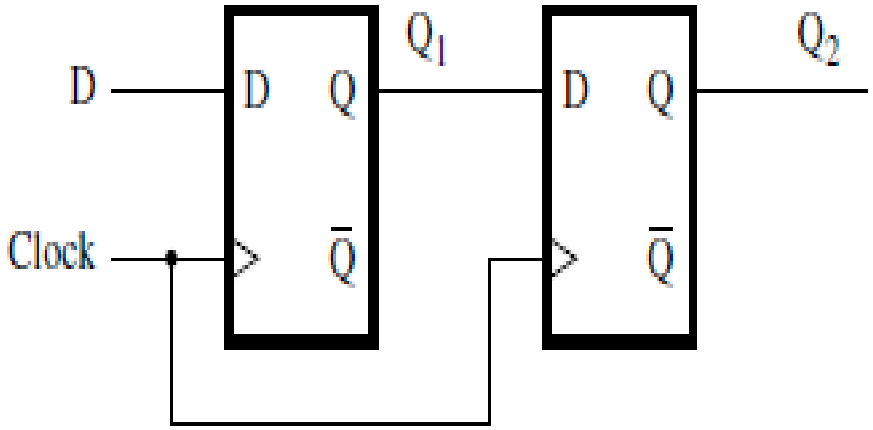
Using Verilog Constructs for Storage Elements

```
module flipflop (D, Clock, Q);
input D, Clock;
output reg Q;
always @(posedge Clock)
Q = D;
endmodule
```

```
`timescale 1ns/1ps

module flipflop_tb();
  reg D, Clock;
  wire Q;

  // Instantiate the DUT (Device Under Test)
  flipflop uut (
    .D(D),
    .Clock(Clock),
    .Q(Q)
  );
```

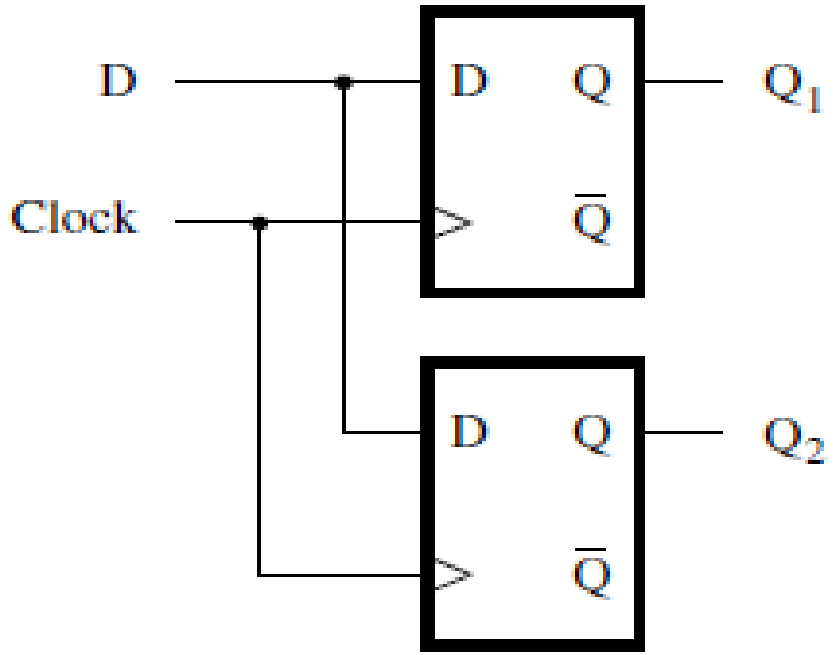
```
// Clock generation: Toggle every 5ns → 10ns period (100MHz)
 initial begin
   Clock = 0;
   forever #5 Clock = ~Clock;
 end

 // Apply stimulus
 initial begin
   // Monitor signals
   $monitor("Time=%0t | D=%b | Clock=%b | Q=%b", $time, D,
Clock, Q);

  // Initial values
   D = 0;
```




```
module example5_3 (D, Clock, Q1, Q2);
input D, Clock;
output reg Q1, Q2;
always @(posedge Clock)
begin
Q1 = D;
Q2 = Q1;
end
endmodule
```

```
module example5_4 (D, Clock, Q1, Q2);
input D, Clock;
output reg Q1, Q2;
always @(posedge Clock)
begin
Q1 < = D;
Q2 < = Q1;
end
endmodule
```

Incorrect code for two cascaded flip-flops



Blocking and Non-Blocking Assignments

f = x1 & x2;

This notation is called a *blocking* assignment.

A Verilog compiler evaluates the statements in an **always** block in the order in which they are written.

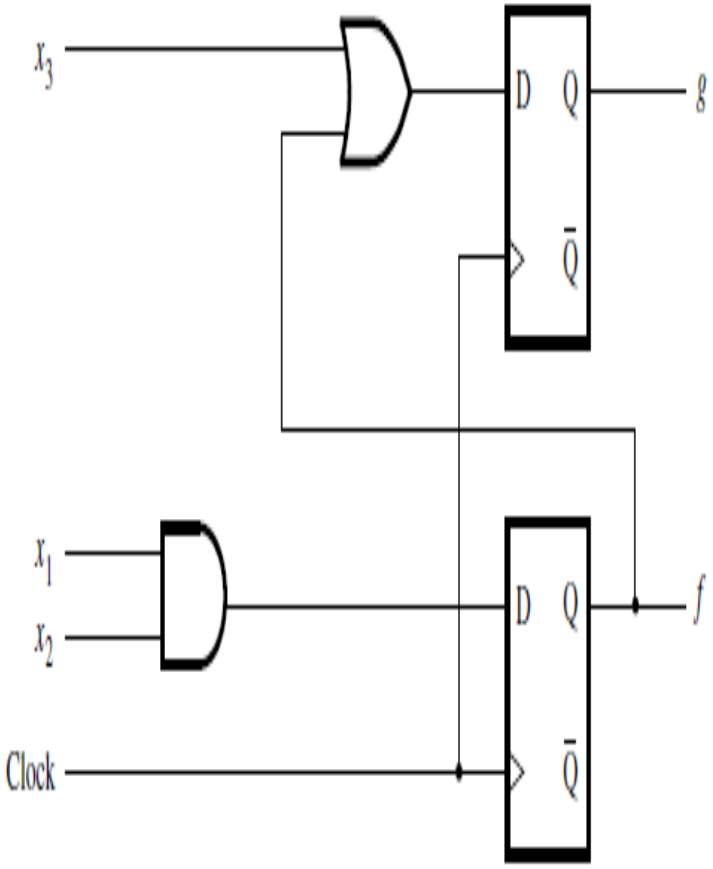
If a variable is given a value by a blocking assignment statement, then this **new value is used in evaluating all subsequent statements in the block.**

Verilog also provides a *non-blocking* assignment, denoted with <=.

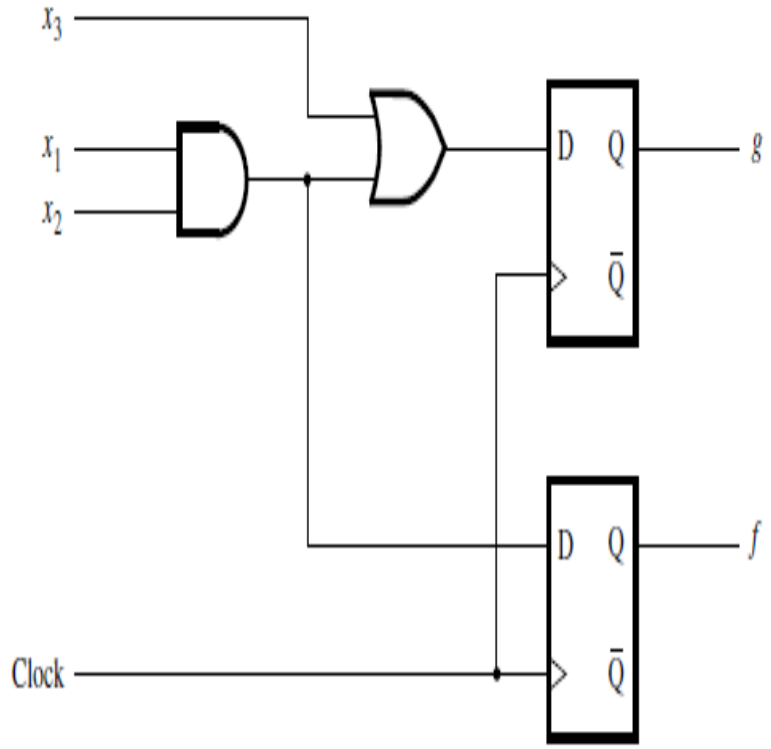
All non-blocking assignment statements in an **always** block are evaluated using the values that the variables have when the **always** block is entered.

Thus, a given variable has the same value for all statements in the block.

The meaning of non-blocking is that the result of each assignment is not seen until the end of the **always** block.




```
module example5_5 (x1, x2, x3,
Clock, f, g);
input x1, x2, x3, Clock;
output reg f, g;
always @(posedge Clock)
begin
f = x1 & x2;
g = f | x3;
end
endmodule
```




```
module example5_6 (x1, x2, x3, Clock, f, g);
input x1, x2, x3, Clock;
output reg f, g;
always @(posedge Clock)
begin
f <= x1 & x2;
g <= f | x3;
end
endmodule
```

ASYNCHRONOUS CLEAR

Following module defines a D flip-flop with an asynchronous active-low reset (clear) input.

When *Resetn*, the reset input, is equal to 0, the flip-flop's Q output is set to 0.

The sensitivity list specifies the negative edge of *Resetn* as an event trigger along with the positive edge of the clock.

We cannot omit the keyword **negedge** because the sensitivity list cannot have both edge-triggered and level sensitive signals.

```
module flipflop (D, Clock, Q);
input D, Clock;
output reg Q;
always @(posedge Clock)
Q <= D;
endmodule
```

DFF with no reset

```
module flipflop (D, Clock, Resetn, Q);
input D, Clock, Resetn;
output reg Q;
always @(negedge Resetn or posedge Clock)
if (!Resetn)
Q <= 0;
else
Q <= D;
endmodule
```

DFF with asynchronous active low reset

```
module flipflop (D, Clock, Resetn, Q);
input D, Clock, Resetn;
output reg Q;
always @(posedge Clock)
if (!Resetn)
Q <= 0;
else
Q <= D;
endmodule
```

DFF with synchronous active low reset

D flip-flop with asynchronous active-low reset.

Key points: Reset is asynchronous. Because the always block is sensitive to negedge Resetn.
As soon as Resetn goes low, output Q is immediately forced to 0 (even without waiting for clock).Clock is positive-edge triggered.
On every rising edge of clock, if reset is not active, Q takes the value of D.
This is called D flip-flop with asynchronous active-low reset.

Code for an *n*-bit register with asynchronous clear

```
module regn (D, Clock, Resetn, Q);
parameter n = 16;
input [n –1:0] D;
input Clock, Resetn;
output reg [n –1:0] Q;
always @(negedge Resetn, posedge Clock)
if (!Resetn)
Q <= 0;
else
Q <= D;
endmodule
```

Code for an *n*-bit register with asynchronous clear

This is an n-bit register with asynchronous active-low reset:If reset is active (Resetn=0) → output Q becomes 0 immediately.On each rising edge of the clock, if reset is not active, Q takes the value of input D.

👉 Essentially, it’s a D flip-flop register with parameterized width and asynchronous reset.

Code for a four-bit shift register(right shift)

```
module shift4 (w, Clock, Q);
input w, Clock;
output reg [3:0] Q;
always @(posedge Clock)
begin
Q[0] < = Q[1];
Q[1] < = Q[2];
Q[2] < = Q[3];
Q[3] < = w;
end
endmodule
```

```
for (k = 0; k < n –1; k = k+1)
Q[k] < = Q[k+1];
Q[n –1] <= w;
```

w → a 1-bit serial input.
Clock → the clock signal.

Q is declared as a **4-bit register** (Q[3:0]) that stores the shifted data.
always @(posedge Clock)
This block runs on every rising edge of the clock.

The shifting operation is synchronous with the clock.

The values are updated as follows:
Q[0] takes the previous value of Q[1]. Q[1] takes the previous value of Q[2]Q[2] takes the previous value of Q[3]Q[3] takes the new serial input w

This makes it a 4-bit Serial-In / Parallel-Out (SIPO) shift register.

```
module ring_count(q,clk,clr);
input clk,clr;
output [3:0]q;
reg [3:0]q;
always @(posedge clk)
if(clr==1)
q<=4'b1000;
else
begin
q[3]<=q[0];
q[2]<=q[3];
q[1]<=q[2];
q[0]<=q[1];
end
endmodule
```

Lab - 7

1.Write behavioral Verilog code for a negative edge triggered T FF with asynchronous active low reset.

2. Write behavioral Verilog code for a positive edge-triggered JK FF with synchronous active high reset

3. Design and simulate the following counters

a) 4-bit ring counter.

b) 5 bit Johnson counter.

Code for negative edge triggered T FF with asynchronous active low reset

```
module tff(T,Clk,Q,Reset);
input T,Reset,Clk;
output Q;
reg Q;
always@(negedge Clk or negedge Reset)
if(!Reset)
Q<=0;
else if(T==1)
Q<=~Q;
endmodule
```

Code for positive edge triggered JK FF with synchronous active high reset

```
module jkff(J,K,Clk,Reset,Q);
input J,K,Clk,Reset;
output Q;
reg Q;
always@(posedge Clk)
if(Reset)
Q<=0;
else
begin
case({J,K})
2'b00:Q<=Q;
2'b01:Q<=0;
2'b10:Q<=1;
2'b11:Q<=~Q;
endcase
end
endmodule
```

Ring Counter 2-bit counter and 2to4 decoder

```
module ring (clk,q);
input clk;
output [3:0]q;
wire [1:0]  count;
bitcounter2 one(clk,count[1:0]);
d2to4 two(en, count[1:0],q[3:0]);
endmodule
module dff11(clk,d,q);
input clk,d;
output q;
reg q;
always@(posedge clk)
begin
q<=d;
end
endmodule
```

```
module d2to4(en,w,y);
input en;
input [1:0]w;
output [3:0]y;
reg [3:0]y;
always@(en or w)
begin
y=0;
if(en==1)
case(w)
0:y[3]=1;
1:y[2]=1;
2:y[1]=1;
3:y[0]=1;
endcase
end
endmodule
```

```
module bitcounter2(clk,q);
input clk;
output[1:0]q;
dff11 ffo(clk,q[1]^q[0],q[1]);
dff11 ff1(clk,~q[0],q[0]);
endmodule
```

Johnson Counter

```
module johnson(clk,q);
input clk;
output [4:0]q;
dff1 ffo(clk,~q[4],q[0]);
dff1 ff1(clk,q[0],q[1]);
dff1 ff2(clk,q[1],q[2]);
dff1 ff3(clk,q[2],q[3]);
dff1 ff4(clk,q[3],q[4]);
endmodule
```

```
module dff1(clk,d,q);
input clk,d;
output q;
reg q;
always@(posedge clk)
begin
q=d;
end
endmodule
```

1. Write behavioral Verilog code for a negative edge triggered T FF with asynchronous active low reset.

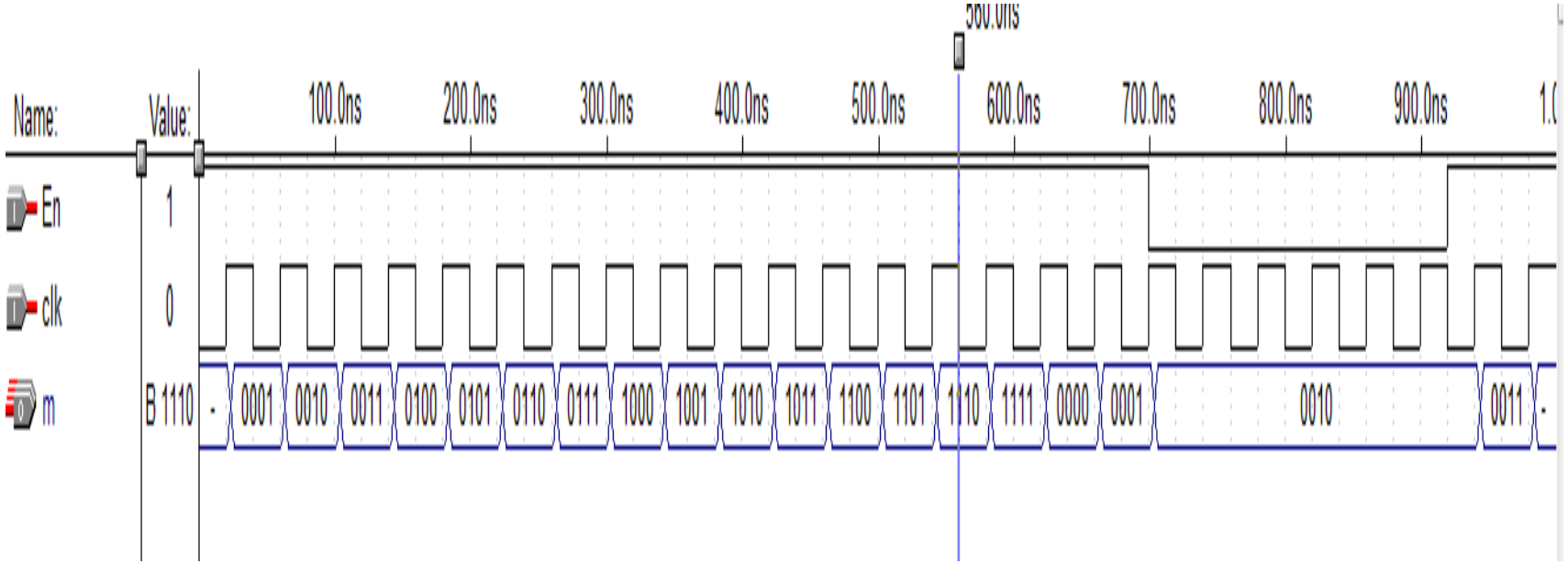
```
module l6q2(t, clock, reset,q);
input t, clock,reset;
output q;
reg q;
always@ (negedge clock or negedge reset)
begin
if(!reset)
q<=0;
else
case(t)
0:q<=q;
1:q<=~q;
endcase
end
endmodule
```

2. Write behavioral Verilog code for a positive edge-triggered JK FF with synchronous active high reset.

```
module l6q1(j,k,clock,reset,q);
input j,k,clock,reset;
output q;
reg q;
always@ (posedge clock)
begin
if(reset)
q<=0;
else
case({j,k})
0:q<=q;
1:q<=0;
2:q<=1;
3:q<=~q;
endcase        end                endmodule
```

Lab 7
1. 4 bit up counter

```
module sync_4bitup(m, clk, En);
input En;
input clk;
output [3:0]m;
tflipflop stage1(En, clk, m[0]);
tflipflop stage2(En & m[0], clk, m[1]);
tflipflop stage3(En & m[1]& m[0], clk, m[2]);
tflipflop stage4(En & m[2]& m[1]& m[0], clk, m[3]);
endmodule
module tflipflop(T, clock, Q);
input T, clock;
output Q;
reg Q;
always@ (posedge clock)
if(T)
Q <= ~Q;
endmodule
```

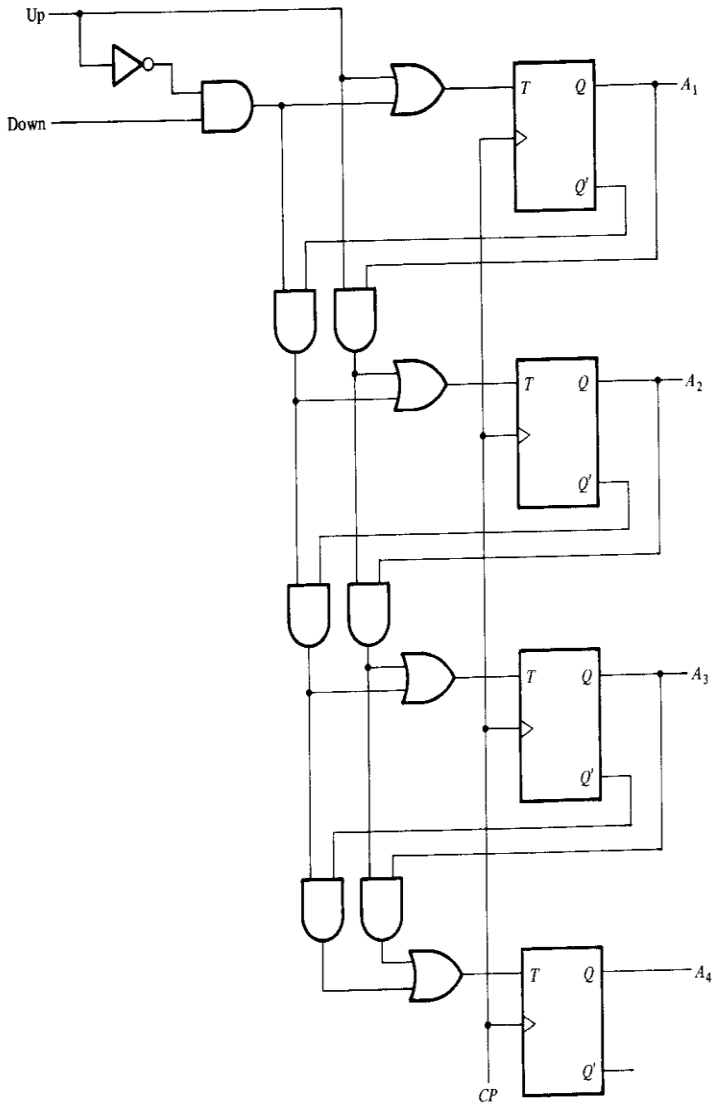

Name:
Value:
100.0ns
200.0ns
300.0ns
400.0ns
500.0ns
600.0ns
700.0ns
800.0ns
900.0ns
En
1
clk
0
m
B 1110
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111
0000
0001
0010
0011

Lab 7 (done similar with x=0 up and x=1 down in class

2. 3 bit up/down counter

UD = 1, up

UD =0, down




```
module sync_3bitupdown(clk,UD,Q);
input UD;
input clk;
output [2:0]Q;
wire control;
 tflipflop stage1(1, clk, Q[0]);
tflipflop stage2((UD & Q[0])|(~UD & ~Q[0]), clk, Q[1]);
tflipflop stage3((UD & Q[0] & Q[1])|(~UD & ~Q[0] & ~Q[1]), clk, Q[2]);
 endmodule
module tflipflop(T, clock, Q);
input T, clock;
output Q;
reg Q;
always@ (posedge clock)
if(T)
Q <= ~Q;
endmodule
```