CSE 460: VLSI Design

Lab Experiment 2 (Blocking and Non-blocing Statements in Verilog)

Procedural Statements

- We learnt about "If Else" procedural statement in lab lecture 1.
- This lecture we will learn about 2nd type procedural statement, the case statement.

- The bits in *expression* are called the *controlling expression*.
- *Controlling expression* are checked for a match with each alternative.
- The first successful match causes the associated statements to be evaluated.
- Default case evaluates only when no other alternative matches.

```
case (expression)
    alternative1: begin
                      statement;
                  end
    alternative2: begin
                      statement;
                  end
    [default:     begin
                      statement;
                  end]
endcase
```

Procedural Statements

- This is the code of 2 to 1 Mux using case statement.
- The mux can have two possible outputs because "*s*" is only 1 bit.
- Which is why the case statement has two alternatives.
- We could have included a default case because "*s*" can also have values of "*x*" and "*z*". But we will learn about them soon.
- We can also use "*1*" as alternative instead of "*1'b0*".
- If a statement in an alternative has multiple line it must be included in Begin-end block.

```
1  module mux2to1(w,s,f);
2
3   input [1:0]w;
4   input s;
5   output reg f;
6
7
8   always @(w or s)
9       case(s)
10          1'b0: f=w[0];
11          1'b1: f=w[1];
12      endcase
13
14  endmodule
```

BRAC UNIVERSITY
Inspiring Excellence

Procedural Statements

```
1  module mux4to1(w,s,f);
2
3   input [3:0]w;
4   input [1:0]s;
5   output reg f;
6
7   always @(w,s)
8       case(s)
9           0: f=w[0];
10          1: f=w[1];
11          2: f=w[2];
12          3: f=w[3];
13          default: f=1'bx;
14      endcase
15  endmodule
```

S	00	01	10	11
f	W[0]	W[1]	W[2]	W[3]

Procedural Statements

- In the "*case*" statement , controlling bits can also have value of "*x*" and "*z*".
- The values of "*x*" and "*z*" are also checked for exact match with the same values in the controlling expressions.
- The "*casex*" statement treats both "*x*" and "*z*" as don't cares.
- That means when they are present as input , code won't check for their alternatives.
- In the right there is a Verilog code of priority encoder with 4 bit input "*w*" and output "y".
- The first alternative "1xxx" specifies that if w[3] has the value of 1 , then the other inputs are treated as don't cares and so the output is set to "*y=3*"

```
1  module prioenc(w,y);
2
3   input [3:0]w;
4   output reg[1:0]y;
5
6   always @(w)
7       casex (w)
8             4'b1xxx: y=3;
9             4'b01xx: y=2;
10            4'b001x: y=1;
11            4'b0001: y=0;
12      endcase
13  endmodule
```

BRAC UNIVERSITY
Inspiring Excellence

Procedural Assignment Statements

- A value is assigned to a variable with a *procedural assignment statement*.
- There are two kinds of assignment statements.
  1. Blocking assignments
  2. Non-blocking assignments.

  - Blocking assignments are denoted by the "=" symbol.
  - Blocking means that first the assignment statement completes and updates it's left-hand side first.
  - This updated left-hand side value is then used for evaluation of subsequent statements.

```
S = X + Y;
p = S[0];
```

Procedural Assignment Statements

- At simulation time t_i the statements are evaluated in order.
- The first statement sets "*S*" to have the summation of current values of "*X*" and "*Y*" .
- Then the second statement sets "*p*" according to this current value of "*S*"

```
S = X + Y;
p = S[0];
```

Procedural Assignment Statements

- 2nd types of assignment statement is non-blocking assignments.
- Non-blocking assignments use the "<=" symbol.
- At simulation time t_i the statements still are evaluated in order but they both use the value of the variables that exist at the start of simulation time.
- The first statement assigns a new value to "*S*" based on the current value of "*X*" and "*Y*" .
- But "*S*" is not actually changed to this value until all statements in the always block have been evaluated.
- For this , the value of "*p*" at time t_i is based on the value of "S" at time t_{i-1}.

```
S <= X + Y;
p <= S[0];
```

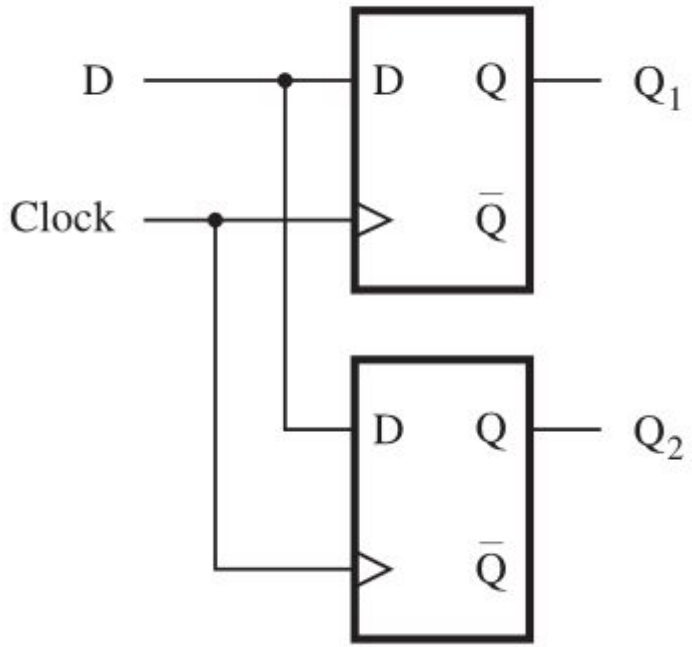
BRAC UNIVERSITY
Inspiring Excellence

Procedural Assignment Statements

```
module example7_3 (D, Clock, Q1, Q2);
    input D, Clock;
    output Q1, Q2;
    reg Q1, Q2;

    always @(posedge Clock)
    begin
        Q1 = D;
        Q2 = Q1;
    end

endmodule
```

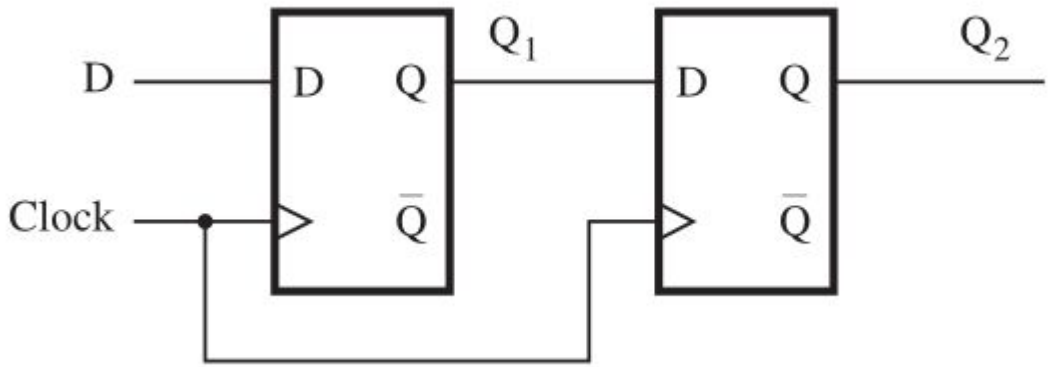


Procedural Assignment Statements

```
module example7_4 (D, Clock, Q1, Q2);
    input D, Clock;
    output Q1, Q2;
    reg Q1, Q2;

    always @(posedge Clock)
    begin
        Q1 <= D;
        Q2 <= Q1;
    end

endmodule
```



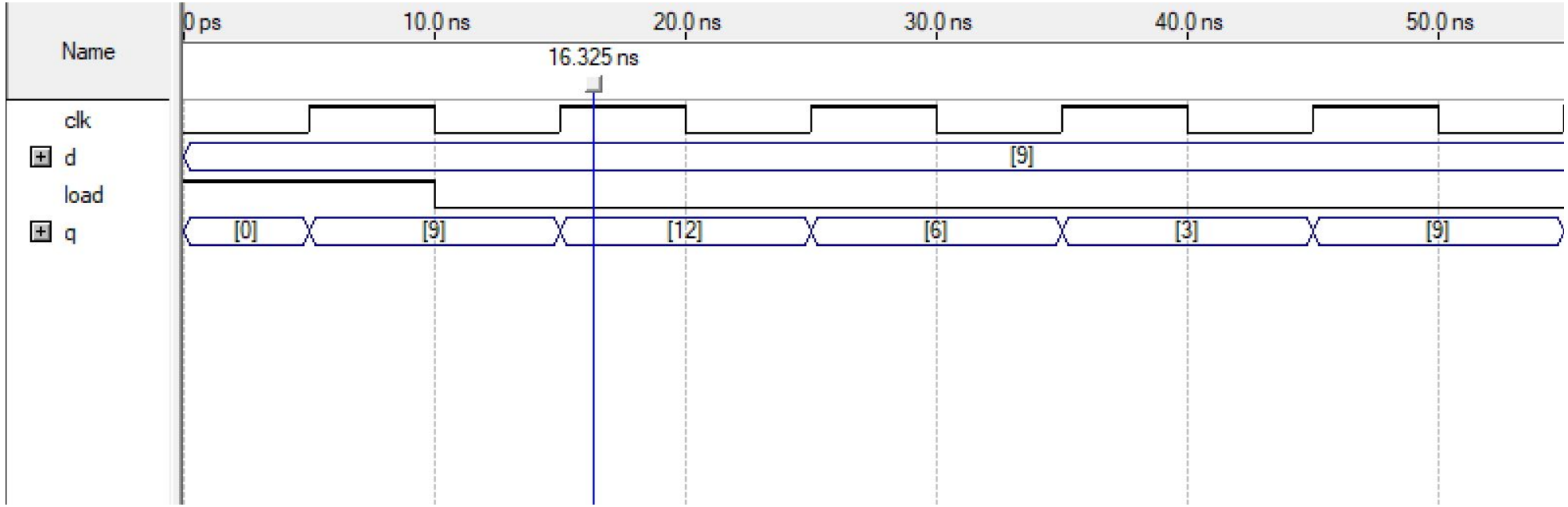
Procedural Assignment Statements

```
1  module shiftreg(d,load,clk,q);
2
3      input [3:0]d;
4      input load,clk;
5      output reg[3:0]q;
6
7  always @(posedge clk)
8      if (load)
9          q<=d;
10     else
11         begin
12             q[3]<=q[0];
13             q[2]<=q[3];
14             q[1]<=q[2];
15             q[0]<=q[1];
16         end
17 endmodule
```

	q[3]	q[2]	q[1]	q[0]
initial	1	0	0	1
1st cycle	1	1	0	0
2nd cycle	0	1	1	0
3rd cycle	0	0	1	1
4th cycle	1	0	0	1

BRAC UNIVERSITY
Inspiring Excellence

Procedural Assignment Statements
(Non-Blocking)



Procedural Assignment Statements

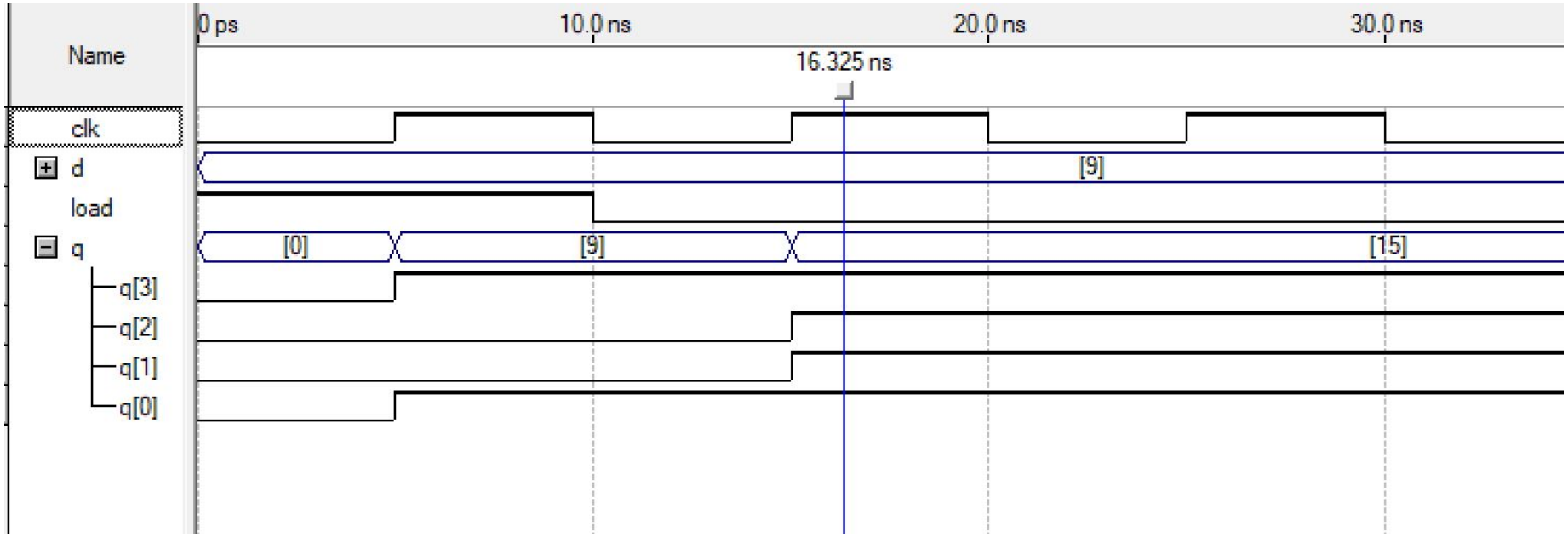
```
1  module shiftreg(d,load,clk,q);
2
3      input [3:0]d;
4      input load,clk;
5      output reg[3:0]q;
6
7   always @(posedge clk)
8      if (load)
9          q<=d;
10     else
11         begin
12             q[3]=q[0];
13             q[2]=q[3];
14             q[1]=q[2];
15             q[0]=q[1];
16         end
17  endmodule
```

	q[3]	q[2]	q[1]	q[0]
initial	1	0	0	1
1st statement	1	0	0	1
2nd statement	1	1	0	1
3rd statement	1	1	1	1
4th statement	1	1	1	1

All of these happened within one cycle!

BRAC UNIVERSITY
Inspiring Excellence

Procedural Assignment Statements
(Blocking)



Thank you!