Digital System Design with HDL (I)

Lecture 10

Dr. Ming Xu

Dept of Electrical & Electronic Engineering

XJTLU

In This Session

- Verilog for Sequential Circuits
  - Latches and Flip-Flops
  - Blocking and Non-Blocking Assignments
  - Counters and Registers

The Latch

A gated D latch

- Here the always block is sensitive to the **levels** of signals.

```
module D_latch (D, EN, Q);
    input D, EN;
    output reg Q;

    always @(D, EN)
        if (EN)
            Q = D;

endmodule
```


Flip-Flops

A D flip-flop

- Here the always block is sensitive to positive edges (**posedge**) of the signal.

```
module flipflop (D, Clock, Q);
    input D, Clock;
    output reg Q;

    always @(posedge Clock)
        Q = D;

endmodule
```

Blocking and Non-Blocking Assignments

- **= : Blocking assignments** – evaluate *in order*

```
begin
    Q1 = D;
    Q2 = Q1;  // new Q1 goes to Q2.
end
```

- **<= : No-blocking assignments** – evaluate *in parallel* with the variable values when entering **always** block

```
begin
    Q1<= D;
    Q2<= Q1;  // old Q1 goes to Q2
end
```

The order of statements doesn't matter

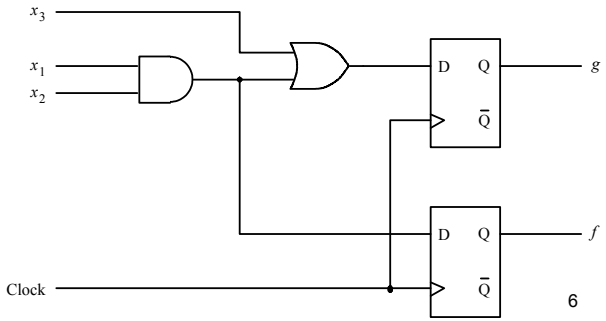
Blocking and Non-Blocking Assignments

```
module example7_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
```

- To reverse the statement order will make significant difference.
- better to combinational circuits.



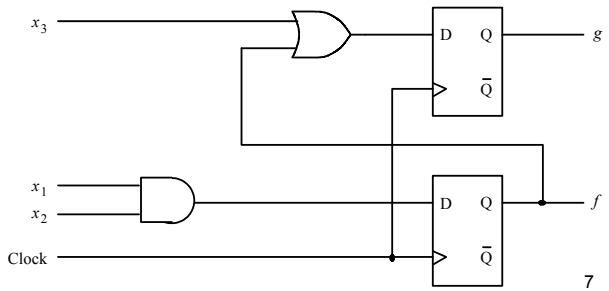
Blocking and Non-Blocking Assignments

```
module example7_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
```

- To reverse the statement order will make no difference.
- better to sequential circuits.



Flip-Flops

D flip-flop with asynchronous 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
```

- **negedge** must be used, because the sensitivity list cannot have both level- and edge-triggered signals.

Flip-Flops

D flip-flops with synchronous 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
```


Registers

An n-bit register

```
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
```


Shift Registers

A 4-bit shift register

- When load L = 1, the parallel input R is loaded.
- When L = 0, the data are shifted from MSB Q3 to LSB; Serial input w is shifted into Q3.

```
module shift4 (R, L, w, Clock, Q);
    input [3:0] R;
    input L, w, Clock;
    output reg [3:0] Q;

    always @(posedge Clock)
        if (L)
            Q <= R;
        else
        begin
            Q[0] <= Q[1];
            Q[1] <= Q[2];
            Q[2] <= Q[3];
            Q[3] <= w;
        end

endmodule
```


Shift Registers

An n-bit shift register

```
module shiftn (R, L, w, Clock, Q);
    parameter n = 16;
    input [n-1:0] R;
    input L, w, Clock;
    output reg [n-1:0] Q;
    integer k;

    always @(posedge Clock)
        if (L)
            Q <= R;
        else
        begin
            for (k = 0; k < n-1; k = k+1)
                Q[k] <= Q[k+1];
                Q[n-1] <= w;
        end

endmodule
```

Counters

A 4-bit up-counter

```
module upcount (Resetn, Clock, E, Q);
    input Resetn, Clock, E;
    output reg [3:0] Q;

    always @(negedge Resetn, posedge Clock)
        if (!Resetn)
            Q <= 0;
        else if (E)
            Q <= Q + 1;

endmodule
```


Counters

A down-counter with a parallel load

```
module downcount (R, Clock, E, L, Q);
    parameter n = 8;
    input [n-1:0] R;
    input Clock, L, E;
    output reg [n-1:0] Q;

    always @(posedge Clock)
        if (L)
            Q <= R;
        else if (E)
            Q <= Q - 1;

endmodule
```


Counters

An up-down counter

```
module updowncount (R, Clock, L, E, up_down, Q);
    parameter n = 8;
    input [n-1:0] R;
    input Clock, L, E, up_down;
    output reg [n-1:0] Q;
    integer direction;

    always @(posedge Clock)
    begin
        if (up_down)
            direction = 1;
        else
            direction = -1;
        if (L)
            Q <= R;
        else if (E)
            Q <= Q + direction;
    end

endmodule
```