Digital System Design with HDL (I)

Lecture 6

Dr. Ming Xu

Dept of Electrical & Electronic Engineering

XJTLU

In This Session

- Programming Statements
  - **if-else**
  - **case** statements
  - **casez** and **casex**
  - **for** loops
  - **while** loops
  - **repeat** and **forever** loops

if-else

- Syntax

  if (conditional_expression) statement1;

  else statement2;

  begin…end

- Behaviour
  - If the conditional_expression is true, then statement1 is executed;
  - otherwise statement2 is executed.
- Used inside an **always** block, the statements are procedural.

if-else

Example: a 2-to-1 multiplexer

```
module mux2to1 (w0, w1, s, f);
    input w0, w1, s;
    output f;
    reg f;

    always @(w0 or w1 or s)
      if (s= =0)
         f = w0;
      else
         f = w1;

endmodule
```

reg data type

Include all input signals

If… else if…else if

Example: a 4-to-1 multiplexer

```
module mux4to1 (W, S, f);
    input [0:3] W;
    input [1:0] S;
    output f;
    reg f;

    always @(W or S)
        if (S = = 0)
            f = W[0];
        else if (S = = 1)
            f = W[1];
        else if (S = = 2)
            f = W[2];
        else if (S = = 3)
            f = W[3];

endmodule
```

case Statements

- Syntax

  case(expression)

  alternative1: statement1;

  alternative2: statement1;

  …

  alternativej: statementj;

  [**default**: statementd;]

  endcase

- Behaviour
  - If there is a match, the statement is executed
  - If the alternatives do not cover all possibilities, **default** should be included.

case Statements

Example: a 4-to-1 multiplexer

```
module mux4to1 (W, S, f);
    input [0:3] W;
    input [1:0] S;
    output f;
    reg f;

    always @(W or S)
        case (S)
            0: f = W[0];
            1: f = W[1];
            2: f = W[2];
            3: f = W[3];
        endcase

endmodule
```

or binary numbers

casex and **casez**

- **case :** bit-by-bit comparison { 0,1,z,x }
- **casez** statement treats high-impedance (z) values as don't-care values.
- **casex** statement treats high-impedance (z) and unknown (x) values as don't care values.
- Only bit values other than *don't care* are used in the comparison.
- Only the first match will be considered**.**

casex and **casez**

casez (state)

// treats z as don't care during comparison :

// 3'b11z, 3'b1zz, ... match 3'b1??: fsm = 0 ;

3'b1??: fsm = 0 ; // if MSB is 1, matches 3'b1??

3'b01?: fsm = 1 ;

default: fsm = 2 ;

endcase

casex and **casez**

casex (state)

// treats both x and z as don't care

// during comparison : 3'b01z, 3'b01x, 3b'011

// ... match case 3'b01x

3'b01x: fsm = 0 ;

3'b0xx: fsm = 1 ;

default: **begin**

// default matches all other occurances

fsm = 1 ;

next_state = 3'b011 ;

end

endcase

casex and **casez**

Example: priority encoder

Truth table

	w_3	w_2	w_1	w_0	y_1	y_0	z
	0	0	0	0	d	d	0
i0 →	0	0	0	1	0	0	1
i1 →	0	0	1	x	0	1	1
i2 →	0	1	x	x	1	0	1
i3 →	1	x	x	x	1	1	1

```
module priority (W, Y, z);
    input [3:0] W;
    output [1:0] Y;
    output z;
    reg [1:0] Y;
    reg z;

    always @(W)
    begin
        z = 1;
        casex(W)
            4'b1xxx: Y = 3;
            4'b01xx: Y = 2;
            4'b001x: Y = 1;
            4'b0001: Y = 0;
            default: begin
                        z = 0;
                        Y = 2'bx;
                    end
        endcase
    end

endmodule
```

Loop Statements

- Loop statements provide a means of repeating blocks of procedural statements.
- There are four types of loop statements:
  - **for** loops
  - **repeat** loops
  - **while** loops
  - **forever** loops

The **for** Loop

Syntax:
for (initial_index; terminal_index; increment) statement;

- A control variable of **reg** (or **integer**) type is set to *initial_index*.
- After each iteration, the control variable is changed as defined in the *increment*.
- The iterations end after the control variable reaches *terminal_index*.

The **for** Loop

Example:
A bit-counting module

```
module bit_count (X, Count);
 parameter n = 4;
 parameter logn = 2;
 input [n-1:0] X;
 output reg [logn:0] Count;
 integer k;

 always @(X)
 begin
  Count = 0;
  for (k = 0; k < n; k = k+1)
   Count = Count + X[k];
 end

endmodule
```

The **while** Loop

Syntax:
while (expression) statement;

- The **while** instruction executes a given statement until the expression is false.
- If a **while** statement starts with a false value, then no statement will be executed.

```
module test;
parameter MSB = 8;
reg [MSB-1:0] Vector;
integer t;

initial
begin
 t = 0;
 while (t < MSB)
  begin
   //Initializes vector elelments
   Vector[t] = 1'b0;
   t = t + 1;
  end
end
endmodule
```

The **repeat** and **forever** Loops

Syntax:
repeat (expression) statement;

- The **repeat** loop executes a given statement a fixed number of times.
- The number of executions is set by the *expression*.

Example:

```
initial
begin
 repeat (10) a = a + ~b;
end
```

Syntax:
forever statement;

- The **forever** loop continuously repeats the statement that follows it.