Behavioral Modeling

Behavioral modeling

- describe design functionality in an algorithmic manner
- very high level of abstraction
- Structural procedures and procedural assignments
- Structural procedures are basic statements in behavioral modeling
- Procedural assignments appear only inside these statements

Structural Procedures

- Two statements
  - **`Initial`**
  - **`always`**
- **`initial`** → they execute only once
- **`always`** → they execute for ever
- The statements `always` and `initial` cannot be nested.

`initial` Statement

- All statements inside an `initial` statement constitute an initial block.
- An initial block starts at time 0, executes exactly once during a simulation, and then does not execute again.
- If there are multiple initial blocks, each block starts to execute concurrently at time 0.
- Each block finishes execution independently of other blocks.

`initial` Statement

```
module stimulus;
reg x,y, a,b, m;

initial
m = 1'b0;

initial
begin
#5 a = 1'b1; //multiple statements; need to be grouped
#25 b = 1'b0;
End

initial
begin
#10 x = 1'b0;
#25 y = 1'b1;
end
endmodule
```

`always` Statement

- All behavioral statements inside an always statement constitute an always block.
- The always statement starts at time 0 and executes the statements in the always block continuously in a looping fashion.
- This statement is used to model a block of activity that is repeated continuously in a digital circuit

`always` Statement

```
module generate (output reg y);

initial
y = 1'b0;

always
y = ~y;

endmodule
```

Blocking assignments

- Block assignments are executed in the order they appear
- executed one after another.
- first statement “blocks” the second until it is done
- The = operator is used to specify blocking assignments.

  – Example:

```
a = b;
b = a;
```

Both a & b = b

Non-blocking assignments

- Non-blocking assignments **<=** executed in parallel.
- an earlier statement does not block the later statement.

  – Example:

```
a <= b;
b <= a;
```

swap a & b

Event based control

- An event is the change in the value on a register or a net.
- Events can be utilized to trigger execution of a statement or a block of statements
- @ symbol is used to specify an event control
- Statements can be executed on changes in signal value or at a positive or negative transition of the signal value

Events

```
always @(signal1 or signal2 or ..) begin
..
end

always @(posedge clk) begin
..
end

always @(negedge clk) begin
..
end
```

Conditional Statements – data flow

Syntax:

```
conditional_expression ? expression_if_true :
                         expression_if_false
```

2-to-1 multiplexer:

```
assign output = s ? input1 : input0;
```

4-to-1 multiplexer:

```
assign output = s[1] ? (s[0] ? in3 : in2) : (s[0] ? in1 : in0)
```

Conditional Statements - Behavioral

```
//Type 1 conditional statement. No else statement.
//Statement executes or does not execute.
if (<expression>) true_statement ;


//Type 2 conditional statement. One else statement
//Either true_statement or false_statement is evaluated
if (<expression>)
     true_statement ;
else false_statement ;


//Type 3 conditional statement. Nested if-else-if.
//Choice of multiple statements. Only one is executed.
if (<expression1>) true_statement1 ;
else if (<expression2>) true_statement2 ;
else if (<expression3>) true_statement3 ;
else default_statement ;
```

Conditional Statements - Behavioral

- if – else
- Type 1
  - `if (enable) out = in;`

- Type 2
  -
```
if (num_queued < Max_Q)
  begin
  data_queue = data;
  number_queued = number_queued + 1;
   end
 else
  $display("Queue full")
```

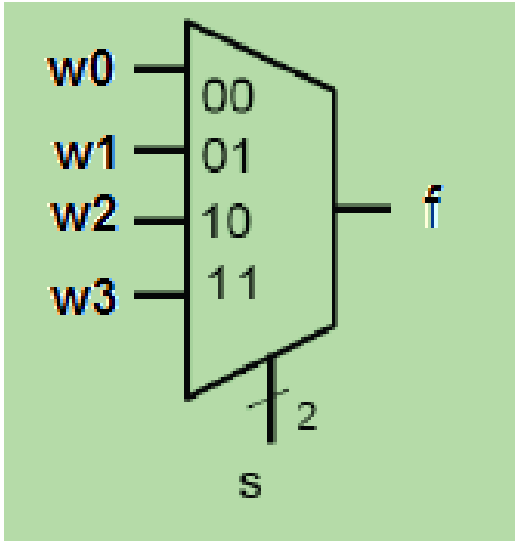
Conditional Statements

- Type 3
  - ```
    if (alu_control == 0)
            y = x + z;
    else if (alu_control == 1)
            y = x - z;
    else if (alu_control == 2)
            y = x * z;
    else
            $display("Invalid ALU control signal");
    ```


    ```

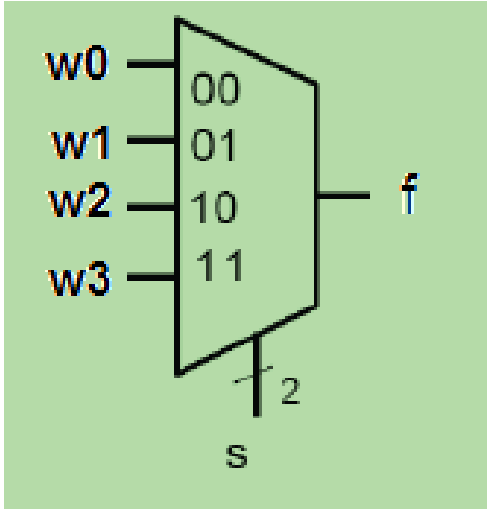
Conditional Statements - Examples

```
module mux4to1 (w0, w1, w2, w3, S, f);
  input w0, w1, w2, w3;
  input [1:0] S;
  output f;
  reg f;
  always @(w0 or w1 or w2 or w3 or S)
        if (S == 2'b00)
              f = w0;
        else if (S == 2'b01)
              f = w1;
        else if (S == 2'b10)
              f = w2;
        else if (S == 2'b11)
              f = w3;
endmodule
```



```
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
```



Building a 16-to-1 multiplexer combining 4-to-1 multiplexers:

```
module mux16to1 (W, S16, f);
   input [0:15] W;
   input [3:0] S16;
   output f;
   wire [0:3] M;

   mux4to1 Mux1 (W[0:3], S16[1:0], M[0]);
   mux4to1 Mux2 (W[4:7], S16[1:0], M[1]);
   mux4to1 Mux3 (W[8:11], S16[1:0], M[2]);
   mux4to1 Mux4 (W[12:15], S16[1:0], M[3]);
   mux4to1 Mux5 (M[0:3], S16[1:0], f);
endmodule
```

Multiway Branching

- Nested if-else-if becomes unwieldy if too many alternatives
- Solution is **`case`** statement

```
case (expression)
alternative1: statement1;
alternative2: statement2;
alternative3: statement3;
...
...
default: default_statement;
endcase
```

Multiway Branching

```
case (alu_control)
2'd0 : y = x + z;
2'd1 : y = x - z;
2'd2 : y = x * z;
default : $display("Invalid ALU control
signal");
endcase
```

```
if (alu_control == 0)
   y = x + z;
else if (alu_control == 1)
   y = x - z;
else if (alu_control == 2)
   y = x * z;
else
   $display("Invalid ALU control signal");
```

Examples

```
module mux4_to_1 (out, i0, i1, i2, i3, s1, s0);
// Port declarations from the I/O diagram
output out;
input i0, i1, i2, i3;
input s1, s0;
reg out;
always @(s1 or s0 or i0 or i1 or i2 or i3)
case ({s1, s0}) //Switch based on concatenation of
control signals
2'd0 : out = i0;
2'd1 : out = i1;
2'd2 : out = i2;
2'd3 : out = i3;
default: $display("Invalid control signals");
endcase
endmodule
```

Multiway Branching

- In case statement it is possible to use the logic values 0, 1, x, and z in the case alternatives.

- Verilog provides two variants of the case statement:
  - The **`casez`** statement treats all z values in the case alternatives as don't-cares.

  - The **`casex`** statement treats all z and x values as don't-cares.

`casex` - Examples

```
module priority_encoder (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
```

Loops

- **`while`** Loop
  - executes until the while expression is not true
  - loop is entered only when the while-expression is not true

```
//Increment count from 0 to 127. Exit at count 128.
integer count;
initial
begin
        count = 0;
        while (count < 128) //Executes till count is 127
        begin
        count = count + 1;
        end
end
```

Loops

- **`for`** Loop
  - An initial condition
  - A check to see if the terminating condition is true
  - A procedural assignment to change value of the control variable

```
for ( count=0; count < 128; count = count + 1)
...
```

```
integer i;
initial
begin
      for(i = 0; i < 32; i = i + 2)
      //initialize all even locations with 0
      state[i] = 0;
end
```