Behavioral Modeling

- Design functionality is described in an algorithmic manner
- The designer describes the behavior of the circuit
- This modeling represents the circuit at a very high level of abstraction
- Verilog is rich in behavioral constructs that provide the designer with a great amount of flexibility.
- **Structured Procedures**
  - Two structured procedure statements in Verilog
    - Always
    - Initial
- These are the two basic statements in behavioral modelling all other behavioural statements can appear only inside these structured procedure statements.
- **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

Behavioral Modeling – initial statments

```
module stimulus;
reg x,y, a,b, m;
initial
m = 1'b0; //single statement; does not need to be grouped
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
Initial
#50 $finish;
endmodule
```

Output:

```
time statement executed
0    m = 1'b0;
5    a = 1'b1;
10   x = 1'b0;
30   b = 1'b0;
35   y = 1'b1;
50   $finish;
```

Behavioral Modeling – initial statement

Combined Variable Declaration and Initialization

- Variables can be initialized when they are declared.

Initial Value Assignment

```
//The clock variable is defined first
reg clock;
//The value of clock is set to 0
initial clock = 0;
//Instead of the above method, clock variable can be initialized at the time of declaration. This is allowed only for variables declared at module level.
reg clock = 0;
```

Combined Port/Data Declaration and Initialization

```
module adder (sum, co, a, b, ci);
output reg [7:0] sum = 0; //Initialize 8 bit output sum
output reg co = 0; //Initialize 1 bit output co
input [7:0] a, b;
input ci;
--
--
endmodule
```

Behavioral Modeling – always statements

Always Statements

- 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.
- An example is a clock generator module that toggles the clock signal every half cycle.

  Example

```
module clock_gen (output reg clock);//Initialize clock at time zero
initial
clock = 1'b0;//Toggle clock every half-cycle (time period = 20)
always
#10 clock = ~clock;
initial
#1000 $finish;
endmodule
```

- always statements are a continuously repeated activity in a digital circuit starting from power on. The activity is stopped only by power off ($finish) or by an interrupt ($stop)

Behavioral Modeling – always statements

initial

```
begin
  … imperative statements …
end
```

- ✓ Runs when simulation starts
- ✓ Terminates when control reaches the end
- ✓ Good for providing stimulus

always

```
begin
  … imperative statements …
end
```

- ✓ Runs when simulation starts
- ✓ Restarts when control reaches the end
- ✓ Good for modeling/specifying hardware

Behavioral Modeling – Procedural Assignments

Procedural Assignments

- Procedural assignments update values of reg, integer, real, or time variables
- The value placed on a variable will remain unchanged until another procedural assignment updates the variable with a different value.

  syntax

  assignment ::= variable_lvalue = [delay_or_event_control] expression
- The left-hand side of a procedural assignment <lvalue> can be one of the following:
  - A reg, integer, real, or time register variable or a memory element
  - A bit select of these variables (e.g., addr[0])
  - A part select of these variables (e.g., addr[31:16])
  - A concatenation of any of the above
- The right-hand side can be any expression that evaluates to a value.

Behavioral Modeling – Procedural Assignment Types

Procedural Assignment Types

- Two types of procedural assignment statements:
  - blocking
  - nonblocking

Blocking Assignments

- The = operator is used to specify blocking assignments.
- Blocking assignment statements are executed in the order they are specified in a sequential block.

Example

```
reg x, y, z;
reg [15:0] reg_a, reg_b;
integer count;
initial
begin
x = 0; y = 1; z = 1; //Scalar assignments
count = 0; //Assignment to integer variables
reg_a = 16'b0; reg_b = reg_a; //initialize vectors
#15 reg_a[2] = 1'b1; //Bit select assignment with delay
#10 reg_b[15:13] = {x, y, z} //Assign result of concatenation
//to part select of a vector
count = count + 1; //Assignment to an integer (increment)
end
```

Behavioral Modeling – Procedural Assignment Types

Nonblocking Assignments

- A <= operator is used to specify nonblocking assignments.
- Nonblocking assignments allow scheduling of assignments without blocking execution of the statements that follow in a sequential block.

Example:

```
reg x, y, z;
reg [15:0] reg_a, reg_b;
integer count;
//All behavioral statements must be inside an initial or always
block
initial
begin
x = 0; y = 1; z = 1; //Scalar assignments
count = 0; //Assignment to integer variables
reg_a = 16'b0; reg_b = reg_a; //Initialize vectors
reg_a[2] <= #15 1'b1; //Bit select assignment with delay
reg_b[15:13] <= #10 {x, y, z}; //Assign result of concatenation to
part select of a vector
count <= count + 1; //Assignment to an integer (increment)
end
```

Example output sequence

- x = 0 through reg_b = reg_a are executed sequentially at time 0.
- Then the three nonblocking assignments are processed at the same simulation time.
  1. reg_a[2] = 0 is scheduled to execute after 15 units (i.e., time = 15)
  2. reg_b[15:13] = {x, y, z} is scheduled to execute after 10 time units (i.e., time = 10)
  3. count = count + 1 is scheduled to be executed without any delay (i.e., time = 0)

Behavioral Modeling – Timing Controls

- In Verilog, if there are no timing control statements, the simulation time does not advance.
- Timing controls provide a way to specify the simulation time at which procedural statements will execute
- three methods of timing control:
  - **delay-based timing control**

```
//define parameters
parameter latency = 20;
parameter delta = 2;
//define register variables
reg x, y, z, p, q;
initial
Begin
x = 0; // no delay control
#10 y = 1; // delay control with a number. Delay execution of  y = 1 by 10 units
end
```

  - **level-sensitive timing control**

```
always
wait (count_enable) #20 count = count + 1;
```

Behavioral Modeling – Timing Controls

- **Event-Based timing control**

Regular event control

Examples:

```
@(clock) q = d; //q = d is executed whenever signal clock changes value
@(posedge clock) q = d; //q = d is executed whenever signal clock does a positive transition ( 0 to 1,x or z, x to 1, z to 1 )
@(negedge clock) q = d; //q = d is executed whenever signal clock does a negative transition ( 1 to 0,x or z, x to 0, z to 0)
q = @(posedge clock) d; //d is evaluated immediately and assigned to q at the positive edge of clock
```

Named Event Control

Event OR Control

Examples:

```
event received_data; //Define an event called received_data
always @(posedge clock) //check at each positive clock edge
begin
if(last_data_packet) //If this is the last data packet
->received_data; //trigger the event received_data
end
always @(received_data) //Await triggering of event received_data When event is triggered, store all four packets of received data in data buffer
//use concatenation operator { }
data_buf = {data_pkt[0], data_pkt[1], data_pkt[2],
data_pkt[3]};
```

Behavioral Modeling – Conditional Statements

- Conditional statements are used for making decisions based upon certain conditions.
- These conditions are used to decide whether or not a statement should be executed.
- Keywords if and else are used for conditional statements.

Three Types

```
//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 ;
```

Behavioral Modeling – Conditional Statements

- Conditional statements are used for making decisions based upon certain conditions.
- These conditions are used to decide whether or not a statement should be executed.
- Keywords if and else are used for conditional statements.

Three Types

```
//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 ;
```

Examples

Conditional Statement Examples

```
//Type 1 statements
if(!lock) buffer = data;
if(enable) out = in;
//Type 2 statements
if (number_queued < MAX_Q_DEPTH)
begin
data_queue = data;
number_queued = number_queued + 1;
end
else
$display("Queue Full. Try again");
//Type 3 statements
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");
```

Behavioral Modeling – Multiway Branching

case Statement

The keywords case, endcase, and default are used in the case statement.

Syntax

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

casex, casez statements

There are two variations of the case statement. They are denoted by keywords, casex and casez.

- casez treats all z values in the case alternatives or the case expression as don't cares.

All bit positions with z can also represented by ? in that position.

- casex treats all x and z values in the case item or the case expression as don't cares.

Example

```
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
```

Behavioral Modeling – Loops

Four types

- While
- For
- Repeat
- Forever

While Loop

while loop executes until the while expression is not true.

```
initial
begin
count = 0;
while (count < 128) //Execute loop till count is 127.
//exit at count 128
begin
$display("Count = %d", count);
count = count + 1;
end
end
```

forever Loop

The keyword forever is used to express this loop. The loop does not contain any expression and executes forever until the $finish task is encountered.

```
//Use forever loop instead of always block
reg clock;
initial
begin
clock = 1'b0;
forever #10 clock = ~clock; //Clock with
period of 20 units
end
//Example 2: Synchronize two register values
at every positive edge of clock
reg clock;
reg x, y;
initial
forever @(posedge clock) x = y;
```

Behavioral Modeling – Loops

For Loop

The keyword for is used to specify this loop. The for loop contains three parts:

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

Example

```
integer count;
initial
for ( count=0; count < 128; count = count + 1)
$display("Count = %d", count);
```

Repeat Loop

- The repeat construct executes the loop a fixed number of times.
- A repeat construct cannot be used to loop on a general logical expression, a while loop is used for that purpose.
- A repeat construct must contain a number, which can be a constant, a variable or a signal value.

Example: Illustration 1 : increment and display count from 0 to 127

```
integer count;
initial
begin
count = 0;
repeat(128)
begin
$display("Count = %d", count);
count = count + 1;
end
end
```