Chapter 4: Behavioral Modeling

Prof. Ming-Bo Lin

Department of Electronic Engineering
National Taiwan University of Science and Technology
Syllabus

- Objectives
- Procedural constructs
- Procedural assignments
- Timing control
- Selection statements
- Iterative (loop) statements
Objectives

After completing this chapter, you will be able to:

- Describe the behavioral modeling structures
- Describe procedural constructs
- Understand the features of initial blocks
- Understand the features of always blocks
- Distinguish the differences between blocking and nonblocking assignments
- Understand the features of timing controls
- Understand the features of selection constructs
- Understand the features of loop constructs
Syllabus

- Objectives
- Procedural constructs
  - initial block
  - always block
- Procedural assignments
- Timing control
- Selection statements
- Iterative (loop) statements
An initial Block

- Executes exactly once during simulation
- Starts at simulation time 0

```verilog
reg x, y, z;
initial begin // complex statement
    x = 1'b0;  y = 1'b1;  z = 1'b0;
    #10     x = 1'b1;  y = 1'b1;  z = 1'b1;
end
initial x = 1'b0; // single statement
```
Syllabus

- Objectives
- Procedural constructs
  - initial block
  - always block
- Procedural assignments
- Timing control
- Selection statements
- Iterative (loop) statements
An always block

- Starts at simulation time 0
- Executes continuously during simulation

```verilog
reg clock;
initial clock = 1'b0;

always #5 clock = ~clock;
```
Syllabus

- Objectives
- Procedural constructs
- Procedural assignments
  - General syntax
  - Blocking assignments
  - Nonblocking assignments
  - Blocking vs. nonblocking assignments
- Timing control
- Selection statements
- Iterative (loop) statements
Procedural Assignments

- Syntax
  
  variable_lvalue = [timing_control] expression
  variable_lvalue <= [timing_control] expression
  [timing_control] variable_lvalue = expression
  [timing_control] variable_lvalue <= expression

- Two types
  - blocking
  - nonblocking
Procedural Assignments

- The bit widths
  - Keeping the least significant bits
  - Zero-extended in the most significant bits
Syllabus

- Objectives
- Procedural constructs
- Procedural assignments
  - General syntax
  - Blocking assignments
  - Nonblocking assignments
  - Blocking vs. nonblocking assignments
- Timing control
- Selection statements
- Iterative (loop) statements
Blocking Assignments

- Are executed in the specified order
- Use the “=“ operator

```verilog
// blocking assignments
initial begin
  x = #5 1'b0;    // at time 5
  y = #3 1'b1;    // at time 8
  z = #6 1'b0;    // at time 14
end
```
### Blocking Assignments

```verilog
output reg [3:0] sum;
output reg c_out;
reg [3:0] t;

always @(x, y, c_in) begin
  t = y ^ {4{c_in}};
  {c_out, sum} = x + t + c_in;
end
```

**Q:** What is wrong with: \( t = y \oplus c_{\text{in}} \)?

**Q:** Does a reg variable correspond to a memory element to be synthesized?
Syllabus

- Objectives
- Procedural constructs
- Procedural assignments
  - General syntax
  - Blocking assignments
  - Nonblocking assignments
  - Blocking vs. nonblocking assignments
- Timing control
- Selection statements
- Iterative (loop) statements
Nonblocking Assignments

- Use the $<=$ operator
- Used to model several concurrent data transfers

```verilog
reg x, y, z;
// nonblocking assignments
initial begin
    x <= #5 1'b0;  // at time 5
    y <= #3 1'b1;  // at time 3
    z <= #6 1'b0;  // at time 6
end
```
Nonblocking Assignments

```verilog
input  clk;
input  din;
output reg [3:0] qout;
// a 4-bit shift register
always @(posedge clk)
    qout <= {din, qout[3:1]};  // Right shift
```
Syllabus

- Objectives
- Procedural constructs
- Procedural assignments
  - General syntax
  - Blocking assignments
  - Nonblocking assignments
  - Blocking vs. nonblocking assignments
- Timing control
- Selection statements
- Iterative (loop) statements
Race Conditions

// using **blocking** assignment statements
always @(posedge clock) // has race condition
    x = y;
always @(posedge clock)
    y = x;

// using **nonblocking** assignment statements
always @(posedge clock) // has no race condition
    x <= y;
always @(posedge clock)
    y <= x;
Execution of Nonblocking Statements

- Read
- Evaluate and schedule
- Assign
Blocking vs. Nonblocking Assignments

// A 4-bit shift register
always @(posedge clk) begin
    qout[0] = sin;
    qout[1] = qout[0];
    qout[2] = qout[1];
    qout[3] = qout[2];
end
Blocking vs. Nonblocking Assignments

// A 4-bit shift register
always @(posedge clk) begin
  qout[0] <= sin;
  qout[1] <= qout[0];
  qout[2] <= qout[1];
  qout[3] <= qout[2];
end
Blocking vs. Nonblocking Assignments

- Suppose that count is 1 and finish is 0 before entering the always block

```verilog
always @(posedge clk) begin: block_a
  count = count - 1;
  if (count == 0) finish = 1;
end
```

Result: finish = 1.
(Different from that of gate-level.)

```verilog
always @(posedge clk) begin: block_b
  count <= count - 1;
  if (count == 0) finish <= 1;
end
```

Result: finish = 0.
(Same as that of gate-level.)
Coding Styles

- blocking operators (=)
  - combinational logic
- nonblocking operators (<=)
  - sequential logic
Syllabus

- Objectives
- Procedural constructs
- Procedural assignments
- Timing control
  - Delay timing control
  - Event timing control
- Selection statements
- Iterative (loop) statements
Delay Timing Control

- Delay timing control
  - Regular delay control
  - Intra assignment delay control
Regular Delay Control

- Regular delay control
  - Defers the execution of the entire statement

```verilog
define x, y;
define integer count;

#25 y <= ~x; // at time 25
#15 count <= count + 1; // at time 40
```
Intra-Assignment Delay Control

- **Intra-assignment delay control**
  - Defers the assignment to the left-hand-side variable

  ```verilog
ty = #25 ~x;               // assign to y at time 25
count = #15 count + 1;    // assign to count at time 40
  
y <= #25 ~x;             // assign to y at time 25
count <= #15 count + 1;  // assign to count at time 15
```
Syllabus

- Objectives
- Procedural constructs
- Procedural assignments
- Timing control
  - Delay timing control
  - Event timing control
- Selection statements
- Iterative (loop) statements
Event Timing Control

- Edge-triggered event control
  - Named event control
  - Event or control
- Level-sensitive event control

Q: What is an event?
Edge-Triggered Event Control

- Edge-triggered event control
  - @(posedge clock)
  - @(negedge clock)

```verilog
always @(posedge clock) begin
    reg1 <= #25 in_1;
    reg2 <= @(negedge clock) in_2 ^ in_3;
    reg3 <= in_1;
end
```
Named Event Control

- A named event

```verilog
event received_data; // declare
always @(posedge clock)
  if (last_byte) -> received_data; // trigger
always @(received_data) // recognize
  begin ... end
```
Event or Control

- Event or control
  - or
  - ,
  - * or (*) means any signals

```verilog
always @(posedge clock or negedge reset_n)
  if (!reset_n) q <= 1'b0;
  else              q <= d;
```
Level-Sensitive Event Control

- Level-sensitive event control

```verilog
always
  wait (count_enable) count = count - 1;

always
  wait (count_enable) #10 count = count - 1;
```
Syllabus

- Objectives
- Procedural constructs
- Procedural assignments
- Timing control
- Selection statements
  - if…else statement
  - case (casex/casez) statement
- Iterative (loop) statements
if...else Statement

- Syntax

  ```
  if (<expression>) true_statement;
  else false_statement;
  ```

  ```
  if (<expression1>) true_statement1;
  else if (<expression2>) true_statement2;
  else false_statement;
  ```
if...else Statement

// using if...else statement
always @(*) begin
    if (s1) begin
        if (s0)  out = i3; else out = i2; end
    else begin
        if (s0)  out = i1; else out = i0; end
end
A Simple 4-bit Counter

// the body of the 4-bit counter.
always @(negedge clock or posedge clear)
  if (clear)
    qout <= 4'd0;
  else
    qout <= (qout + 1) ; // qout = (qout + 1) % 16;
Syllabus

- Objectives
- Procedural constructs
- Procedural assignments
- Timing control
- Selection statements
  - if...else statement
  - case (casex/casez) statement
- Iterative (loop) statements
case (casex/casez) Statement

- Syntax

```verilog
case (case_expression)
    case_item1 {,case_item1}: procedural_statement1
    case_item2 {,case_item2}: procedural_statement2
    ...
    case_itemn {,case_itemn}: procedural_statementn
    [default: procedural_statement]
endcase
```

```
A 4-to-1 MUX Example

always @(I0 or I1 or I2 or I3 or S)
    case (S)
        2'b00: Y = I0;
        2'b01: Y = I1;
        2'b10: Y = I2;
        2'b11: Y = I3;
    endcase

Q: Model a 3-to-1 multiplexer.
casex and casez Statements

- casex and casez statements
  - Compare only non-x or z positions
- casez treats all z values as don’t cares
- casex treats all x and z values as don’t cares
casex and casez Statements

```
// count the trailing zeros in a nibble
always @(data)
  casex (data)
    4'bxxxx1: out = 0;
    4'bxx10: out = 1;
    4'bx100: out = 2;
    4'b1000: out = 3;
    4'b0000: out = 4;
    default:  out = 3'b111;
  endcase
```

Q: Is the default statement necessary?
Chapter 4: Behavioral Modeling

Syllabus

- Objectives
- Procedural constructs
- Procedural assignments
- Timing control
- Selection statements
- Iterative (loop) statements
  - while loop
  - for loop
  - repeat loop
  - forever loop
while Loop

- Syntax

```
while (condition_expr) statement;
```

```
while (count < 12) count <= count + 1;
while (count <= 100 && flag) begin
    // put statements wanted to be carried out here
end
```
while Loop

// count the zeros in a byte
integer i;
always @(data) begin
  out = 0; i = 0;
  while (i <= 7) begin // simple condition
    if (data[i] == 0) out = out + 1;
    i = i + 1;
  end
end

Q: Can the if … be replaced with out = out + ~data[i] ?
while Loop

// count the trailing zeros in a byte
integer i;
always @(data) begin
    out = 0; i = 0;
    while (data[i] == 0 && i <= 7) begin // complex condition
        out = out + 1;
        i = i + 1;
    end
end
Syllabus

- Objectives
- Procedural constructs
- Procedural assignments
- Timing control
- Selection statements
- Iterative (loop) statements
  - while loop
  - for loop
  - repeat loop
  - forever loop
for Loop

- Syntax

```vhdl
for (init_expr; condition_expr; update_expr) statement;

init_expr;
while (condition_expr) begin
    statement;
    update_expr;
end
```
for Loop

// count the zeros in a byte
integer i;
always @(data) begin
    out = 0;
    for (i = 0; i <= 7; i = i + 1) // simple condition
        if (data[i] == 0) out = out + 1;
end
for Loop

// count the trailing zeros in a byte
integer i;
always @(data) begin
  out = 0;
  for (i = 0; data[i] == 0 && i <= 7; i = i + 1)
    out = out + 1;
end
Syllabus

❖ Objectives
❖ Procedural constructs
❖ Procedural assignments
❖ Timing control
❖ Selection statements
❖ Iterative (loop) statements
  ▪ while loop
  ▪ for loop
  ▪ repeat loop
  ▪ forever loop
repeat Loop

- Syntax
  
  \[
  \text{repeat (counter\_expr) statement;}
  \]

  
  ```
  i = 0;
  repeat (32) begin
    state[i] = 0;
    i = i + 1;
  end
  repeat (cycles) begin
    @(posedge clock) buffer[i] <= data;
    i <= i + 1;
  end
  ```
Syllabus

- Objectives
- Procedural constructs
- Procedural assignments
- Timing control
- Selection statements
- Iterative (loop) statements
  - while loop
  - for loop
  - repeat loop
  - forever loop
forever Loop

- Syntax

```verilog
forever begin
  #10 clock <= 1;
  #5   clock <= 0;
end
```

```verilog
initial begin
  clock <= 0;
  forever begin
    #10 clock <= 1;
    #5   clock <= 0;
  end
end
```
The forever Loop Structure

reg clock, x, y;

initial
  forever @(posedge clock) x <= y;