Chapter 7: Advanced Modeling Techniques

Prof. Ming-Bo Lin

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

- Objectives
- Blocks
- Procedural continuous assignments
- Delay models and timing checks
Chapter 7: Advanced Modeling Techniques

Objectives

After completing this chapter, you will be able to:

- Describe the features of sequential blocks
- Describe the features of parallel blocks
- Describe the features of nested blocks
- Describe the features of procedural continuous assignments
- Describe how to model a module delay
- Describe the features of specify blocks
- Describe the features of timing checks
Syllabus

- Objectives
- Blocks
  - Sequential blocks
  - Parallel blocks
  - Special blocks
  - The disable statement
- Procedural continuous assignments
- Delay models and timing checks
Sequential Blocks

initial begin
    x = 1’b1;    // at time 0
    #12 y = 1’b1;    // at time 12
    #20 z = 1’b0;    // at time 32
end

initial begin
    x  = 1'b0;    // at time 0
    #20 w  = 1'b1;    // at time 20
    #12 y <= 1'b1;    // at time 32
    #10 z <= 1'b0;    // at time 42
    #25 x   = 1'b1; w = 1'b0;    // at time 67
end
Syllabus

- Objectives
- Blocks
  - Sequential blocks
  - Parallel blocks
  - Special blocks
  - The disable statement
- Procedural continuous assignments
- Delay models and timing checks
Parallel Blocks

\[
\text{initial fork}
\begin{align*}
    x &= 1'b0; \quad \text{at time 0} \\
    #12 \quad y &= 1'b1; \quad \text{at time 12} \\
    #20 \quad z &= 1'b1; \quad \text{at time 20}
\end{align*}
\]

join

\[
\text{initial fork}
\begin{align*}
    x &= 1'b0; \quad \text{at time 0} \\
    #12 \quad y &= 1'b1; \quad \text{at time 12} \\
    #20 \quad z &= 1'b1; \quad \text{at time 20}
\end{align*}
\]

join
Chapter 7: Advanced Modeling Techniques

Syllabus

❖ Objectives

❖ Blocks
  ▪ Sequential blocks
  ▪ Parallel blocks
  ▪ Special blocks
  ▪ The disable statement

❖ Procedural continuous assignments

❖ Delay models and timing checks
Types of Special Blocks

- Nested blocks
- Named blocks
Nested Blocks

```verilog
initial begin
  x = 1'b0;               // at time 0
  fork // parallel block -- enter at time 0
    #12 y <= 1'b1;       // at time 12
    #20 z <= 1'b0;       // at time 20
  join // leave at time 20
  #25 x = 1'b1;         // at time 45
end
```
Named Blocks

```verilog
initial begin: test // test is the block name
reg x, y, z; // local variables
    x = 1'b0;
    #12 y = 1'b1; // at time 12
    #10 z = 1'b1; // at time 22
end
```
Chapter 7: Advanced Modeling Techniques

Syllabus

- Objectives
- Blocks
  - Sequential blocks
  - Parallel blocks
  - Special blocks
  - The disable statement
- Procedural continuous assignments
- Delay models and timing checks
The disable statement

```verilog
initial begin: test
    while (i < 10) begin
        if (flag) disable test;
        i = i + 1;
    end
end
```
Syllabus

- Objectives
- Blocks
- Procedural continuous assignments
  - assign and deassign assignments
  - force and release assignments
- Delay models and timing checks
assign and deassign Constructs

- They
  - assign values to variables
- Their LHS
  - can be a variable or a concatenation of variables
- They override
  - the effect of regular procedural assignments
- They
  - are normally used for controlling periods of time
assign and deassign Constructs

// negative edge triggered D flip flop with asynchronous reset
module edge_dff(input clk, reset, d, output reg q, qbar);
always @(negedge clk) begin
    q <= d; qbar <= ~d;
end
always @(reset) // override the regular assignments
    if (reset) begin
        assign q = 1'b0;
        assign qbar = 1'b1;
    end else begin // release q and qbar
        deassign q;
        deassign qbar;
    end
endmodule
Syllabus

❖ Objectives
❖ Blocks
❖ Procedural continuous assignments
  ▪ assign and deassign assignments
  ▪ force and release assignments
❖ Delay models and timing checks
force and release Constructs

- They
  - assign values to nets or variables
- Their LHS
  - can be a variable or a net
- They override
  - the effect of regular procedural assignments
  - the effect of assign and deassign construct
- They
  - are normally used for controlled periods of time
Syllabus

- Objectives
- Blocks
- Procedural continuous assignments
- Delay models and timing checks
  - Delay models
  - Specify blocks
  - Timing checks
Types of Delay Models

- Distributed delays
- Lumped delays
- Module path (pin-to-pin) delays
Distributed Delay Model

module M (input x, y, z, output f);
wire a, b, c;

and #5 a1 (a, x, y);
not #2 n1 (c, x);
and #5 a2 (b, c, z);
or #3 o1 (f, a, b);
endmodule

module M (input x, y, z, output f);
wire a, b, c;

assign #5 a = x & y;
assign #2 c = ~x;
assign #5 b = c & z;
assign #3 f = a | b;
endmodule
Lumped Delay Model

module M (input x, y, z, output f);
wire a, b, c;
and a1 (a, x, y);
not n1 (c, x);
and a2 (b, c, z);
or #10 o1 (f, a, b);
endmodule

module M (input x, y, z, output f);
wire a, b, c;
assign a = x & y;
assign c = ~x;
assign b = c & z;
assign #10 f = a | b;
endmodule
Module Path Delay Model

Path $x$-$a$-$f$, delay = 8
Path $x$-$c$-$b$-$f$, delay = 10
Path $y$-$a$-$f$, delay = 8
Path $z$-$b$-$f$, delay = 8
Syllabus

- Objectives
- Blocks
- Procedural continuous assignments
- Delay models and timing checks
  - Delay models
  - Specify blocks
  - Timing checks
Specify Blocks

- Keywords
  - `specify` and `endspecify`
- Are used to
  - describe various paths across the module
  - assign delays to these paths
  - perform necessary timing checks
module M (input x, y, z, output f);
wire a, b, c;
// specify block with path delay statements
specify
  (x => f) = 10;
  (y => f) = 8;
  (z => f) = 8;
endspecify
// gate instantiations
and  a1 (a, x, y);
not   n1 (c, x);
and  a2 (b, c, z);
or    o1 (f, a, b);
endmodule

Path x-a-f, delay = 8
Path x-c-b-f, delay = 10
Path y-a-f, delay = 8
Path z-b-f, delay = 8
Path Declarations

- Single-path
- Edge-sensitive path
- State-dependent path
Chapter 7: Advanced Modeling Techniques

Single Path

- Single path connection methods
  - Parallel connection (source => destination)
  - Full connection (source *> destination)

(a) Parallel connection

(b) Full connection
Edge-Sensitive Path

- **posedge clock** => \( (\text{out} +: \text{in}) = (8, 6) \);
  - module path: from clock to out
  - rise time = 8 and fall delay = 6
  - data path: from in to out

- **negedge clock** => \( (\text{out} -: \text{in}) = (8, 6) \);
  - module path: from clock to out
  - rise time = 8 and fall delay = 6
  - data path: from in to out
Level-Sensitive Path

- clock => (out : in) = (8, 6);
  - At any change in clock, a module path extends from clock to out
State-Dependent Path

- **Syntax**

  ```verilog
  if (cond_expr) simple_path_declaration
  if (cond_expr) edge_sensitive_path_declaration
  ifnone simple_path_declaration
  ```

  ```verilog
  specify
  if (x) (x => f) = 10;
  if (~x) (y => f) = 8;
  endspecify
  ```

  ```verilog
  specify
  if (!reset && !clear) (positive clock => (out +: in) = (8, 6);
  endspecify
  ```
The specparam Statement

```verilog
specify
    // define parameters inside the specify block
    specparam d_to_q = (10, 12);
    specparam clk_to_q = (15, 18);
    (d => q) = d_to_q;
    (clk => q) = clk_to_q;
endspecify
```
An Example --- An NOR Gate

module my_nor (a, b, out);
    ...
    output out;
    nor nor1 (out, a, b);
    specify
        specparam trise = 1, tfall = 2
        specparam trise_n = 2, tfall_n = 3;
        if (a) (b => out) = (trise, tfall);
        if (b) (a => out) = (trise, tfall);
        if (~a)(b => out) = (trise_n, tfall_n);
        if (~b)(a => out) = (trise_n, tfall_n);
    endspecify
    ...

Syllabus

- Objectives
- Blocks
- Procedural continuous assignments
- Delay models and timing checks
  - Delay models
  - Specify blocks
  - Timing checks
Timing Checks

- Must be inside the `specify` blocks
- The most commonly used timing checks
  - `$setup`
  - `$hold`
  - `$setuphold`
  - `$width`
  - `$skew`
  - `$period`
  - `$recovery`
Timing Checks -- $setup Timing Check

$setup (data_event, reference_event, limit);
Violation when $t_{reference\_event} - t_{data\_event} < limit$

specify
   $setup\_event \text{ (data, posedge clock, 15);}$
endspecify
Timing Checks -- $hold Timing Check

$hold (reference_event, data_event, limit);
Violation when \( t_{data\_event} - t_{reference\_event} < limit \)

specify
  $hold (posedge clock, data, 8);
endspecify
Timing Checks -- $setuphold Timing Check

$setuphold (reference_event, data_event, setup_limit, hold_limit);
Violation when:

\[ t_{\text{reference\_event}} - t_{\text{data\_event}} < \text{setup\_limit} \]
\[ t_{\text{data\_event}} - t_{\text{reference\_event}} < \text{hold\_limit} \]

specify

\$setuphold (posedge clock, data, 10, 5);
endspecify
Timing Checks -- $width Timing Check

$width (reference_event, limit);
Violation when $t_{data\_event} - t_{reference\_event} < limit

specify
  $width (posedge reset, 6);
endspecify
Timing Checks -- $skew Timing Check

$skew (reference_event, data_event, limit);
Violation when $t_{data\_event} - t_{reference\_event} > limit$

specify

$skew$ (posedge clk1, posedge clk2, 5);
endspecify
Timing Checks -- $period Timing Check

$\text{period} (\text{reference\_event}, \text{limit});$

Violation when $t_{\text{data\_event}} - t_{\text{reference\_event}} < \text{limit}$

specify

\[ \text{$period$ (posedge clk, 15);} \]

endspecify
Timing Checks -- $\text{recovery}$ Timing Check

$a_{\text{sync}}$ input

$\text{clk}$

Violation when

\[ t_{\text{reference\_event}} \leq t_{\text{data\_event}} < t_{\text{reference\_event}} + \text{limit} \]

specify

$\text{recovery}$ (negedge $a_{\text{sync\_input}}$, posedge $\text{clk}$, 5);

endspecify