HARDWARE DESCRIPTION LANGUAGE FOR DIGITAL DESIGN

數位設計硬體描述語言

Advanced Modeling

Materials partly adapted from "Digital System Designs and Practices Using Verilog HDL and FPGAs," M.B. Lin.

NCKU EE
LPHP Lab

OUTLINE

- Verilog Basics
- Structural Modeling
- Dataflow Modeling
- Behavioral Modeling
- Tasks and Functions
- Hierarchical Structure Modeling
- Advanced Modeling

ADVANCED MODELING

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
```

PARALLEL BLOCKS

```
initial  fork
         x = 1'b0;    // at time 0
    #12 y = 1'b1;    // at time 12
    #20 z = 1'b1;    // at time 20
join
```

```
initial  fork
         x <= 1'b0;    // at time 0
    #12 y <= 1'b1;    // at time 12
    #20 z <= 1'b1;    // at time 20
join
```

TYPES OF SPECIAL BLOCKS

- Nested blocks
- Named blocks

NESTED BLOCKS

```
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

```
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
```

THE DISABLE STATEMENT

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

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

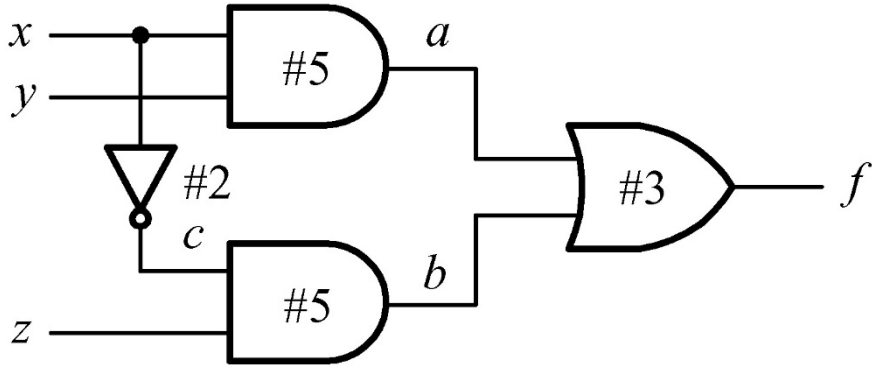
ASSIGN AND DEASSIGN CONSTRUCTS

```
// 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
```


TYPES OF DELAY MODELS

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

DISTRIBUTED DELAY MODEL



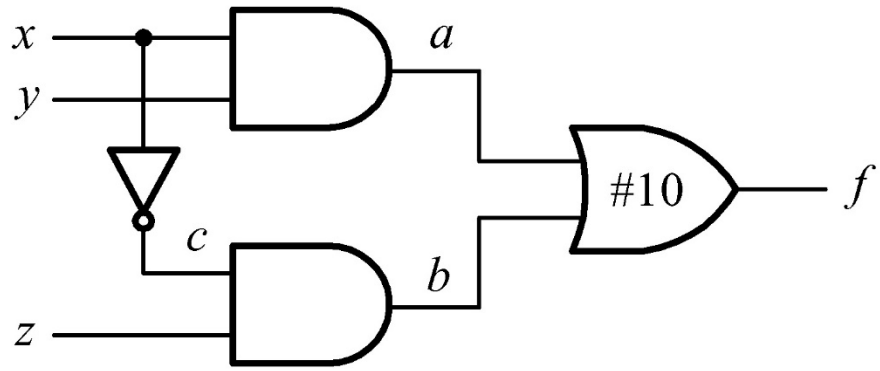

```
module M (input  x, y, z, output f);
wire a, b, c;

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



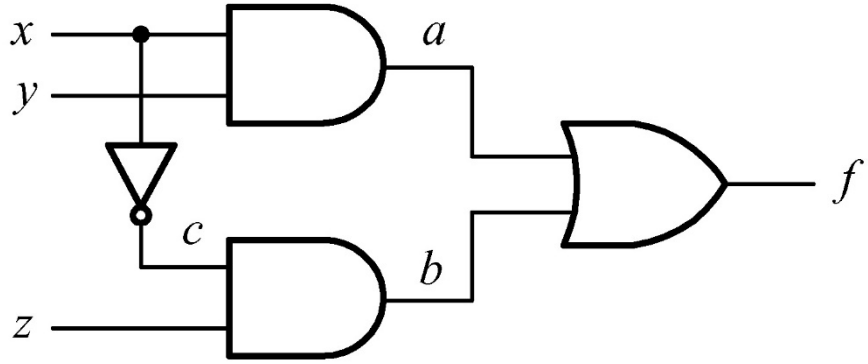

```
module M (input  x, y, z , output f);
wire a, b, c;

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



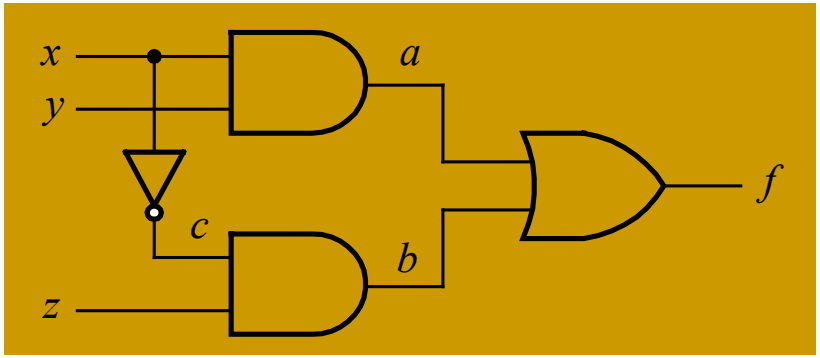

```
Path x-a-f,   delay = 8
Path x-c-b-f, delay = 10
Path y-a-f,   delay = 8
Path z-b-f,   delay = 8
```


SPECIFY BLOCKS

- Keywords
  - specify and endspecify
- Are used to
  - describe various paths across the module
  - assign delays to these paths
  - perform necessary timing checks

PATH DELAY MODELING – THE SPECIFY BLOCK

```
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
```



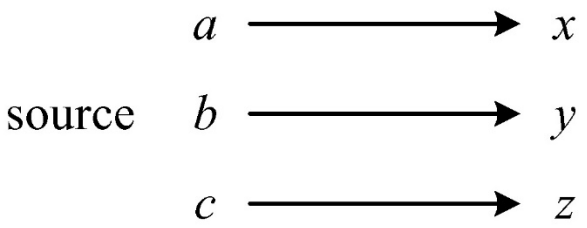
Path x-a-f, delay = 8
Path x-c-b-f, delay = 10
Path y-a-f, delay = 8
Path z-b-f, delay = 8

PATH DECLARATIONS

- Single-path
- Edge-sensitive path
- State-dependent path

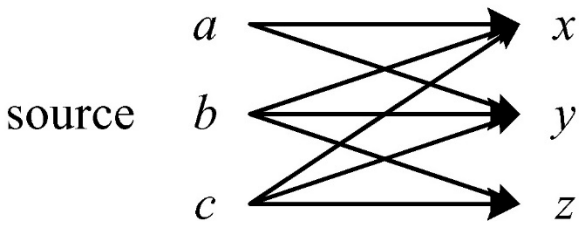
SINGLE PATH

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




```
(a => x) = 10;
(b => y) = 8;
(c => z) = 8;
```

(a) Parallel connection




```
(a, b, c *> x) = 8;
(a, b, c *> y) = 10;
(b, c *> z)    = 12;
```

(b) Full connection

EDGE-SENSITIVE PATH

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

  if (cond_expr) simple_path_declaration

  if (cond_expr) edge_sensitive_path_declaration

  ifnone simple_path_declaration

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

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

THE SPECPARAM STATEMENT

```
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
```

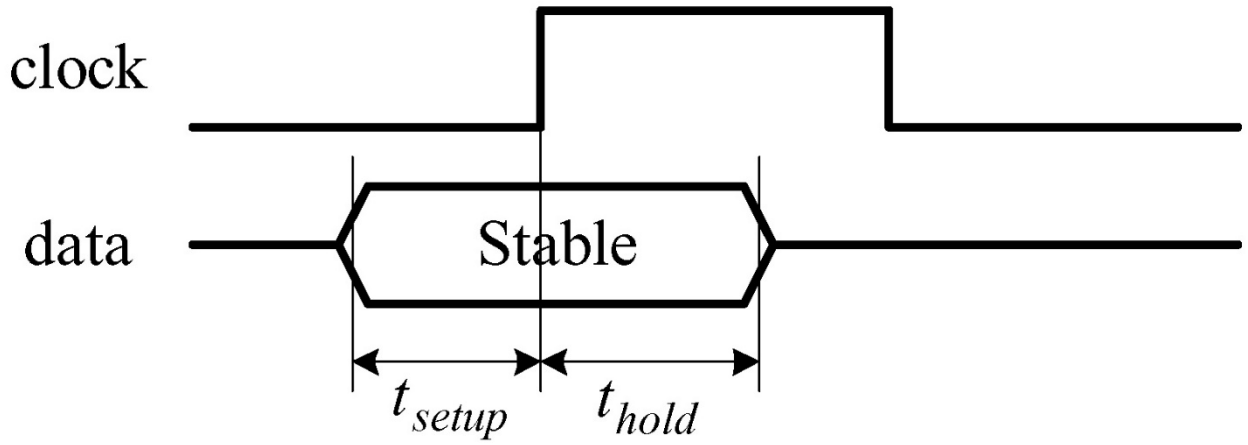
AN EXAMPLE --- AN NOR GATE

```
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
…
```

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

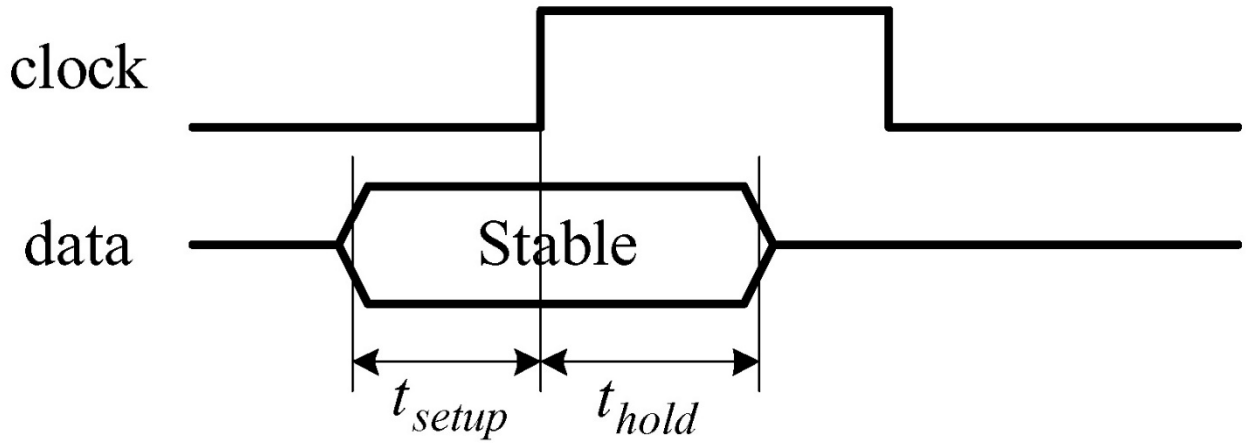


$setup (data_event, reference_event, limit);

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

```
specify
   $setup (data, posedge clock, 15);
endspecify
```

TIMING CHECKS -- $HOLD TIMING CHECK

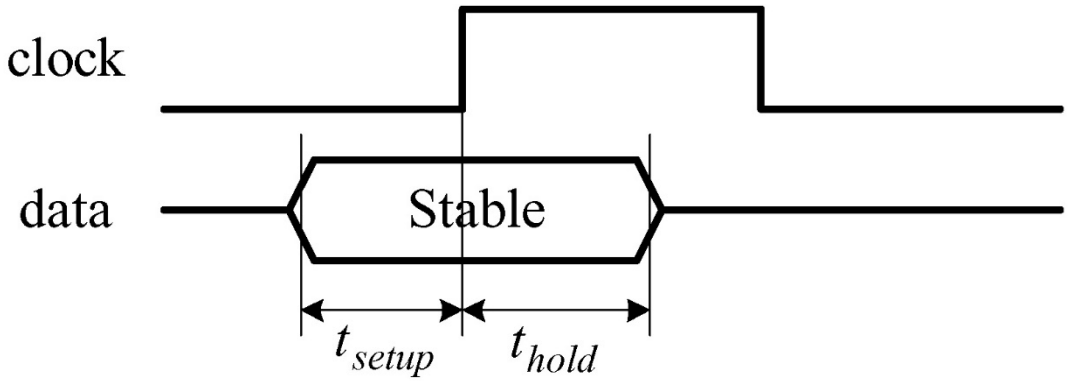


$hold (reference_event, data_event, limit);

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

```
specify
   $hold (posedge clock, data, 8);
endspecify
```

TIMING CHECKS -- $SETUPHOLD TIMING CHECK



$setuphold (reference_event, data_event, setup_limit, hold_limit);

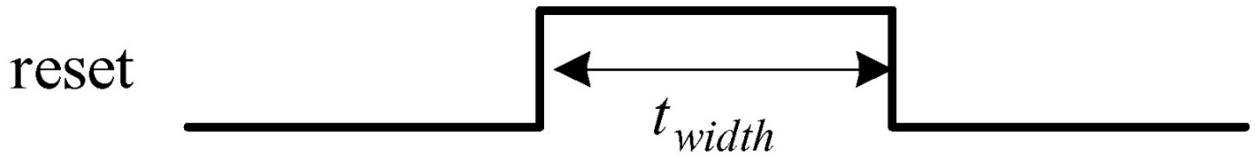
Violation when:

$$t_{\text{reference\_event}} - t_{\text{data\_event}} < \text{setup\_limit}$$

$$t_{\text{data\_event}} - t_{\text{reference\_event}} < \text{hold\_limit}$$

```
specify
   $setuphold (posedge clock, data, 10, 5);
endspecify
```

TIMING CHECKS -- $WIDTH TIMING CHECK

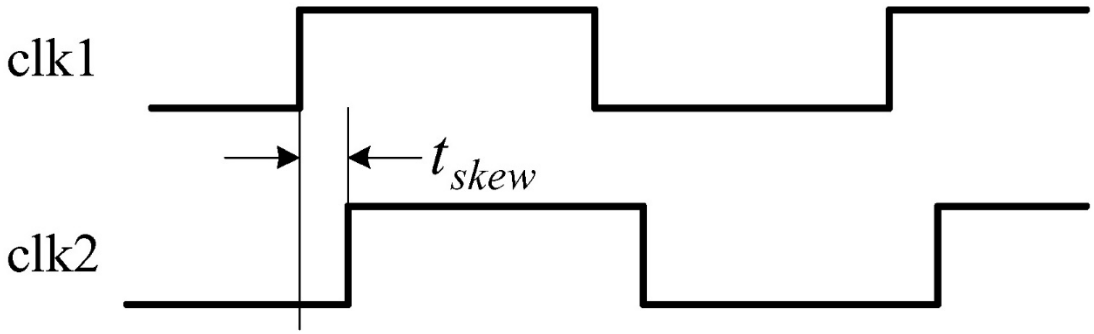


$width (reference_event, limit);

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

```
specify
   $width (posedge reset, 6);
endspecify
```

TIMING CHECKS -- $SKEW TIMING CHECK

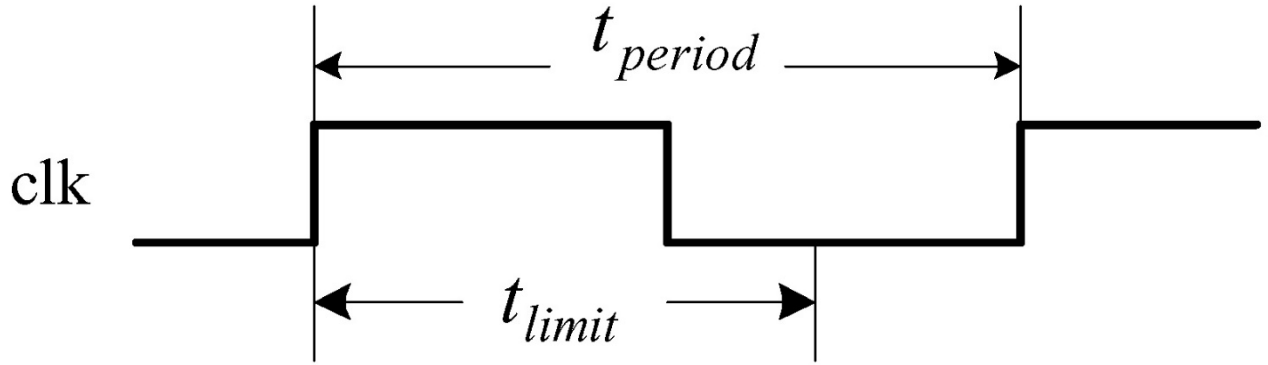


$skew (reference_event, data_event, limit);

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

```
specify
   $skew (posedge clk1, posedge clk2, 5);
endspecify
```

TIMING CHECKS -- $PERIOD TIMING CHECK

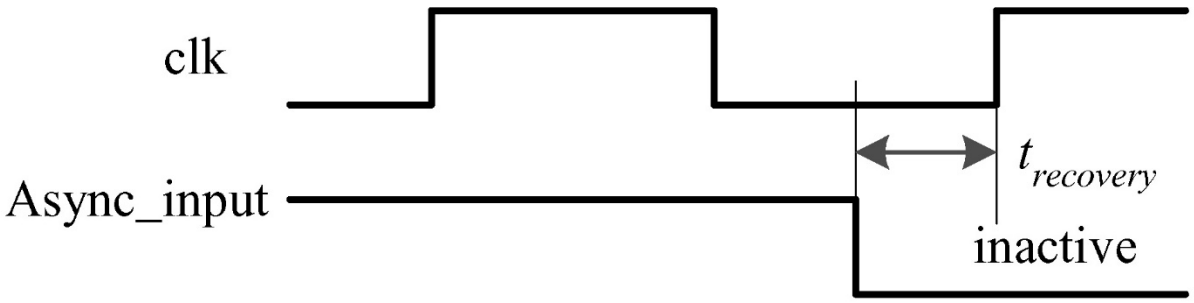


$period (reference_event, limit);

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

```
specify
   $period (posedge clk, 15);
endspecify
```

TIMING CHECKS -- $RECOVERY TIMING CHECK



(a) Recovery time

$recovery (reference_event, data_event, limit);

Violation when

$$t_{\text{reference\_event}} <= t_{\text{data\_event}} < t_{\text{reference\_event}} + \text{limit}$$

```
specify
   $recovery (negedge aysyn_input, posedge clk, 5);
endspecify
```