

Verilog HDL: Timing and Delays

Pravin Zode

- Modeling Delay
 - Lumped Delay
 - Distributed Delay
 - Pin to Pin Delay
- Path Delay Modeling
 - Specify block
- Timing Checks and Delay Back-Annotation

Introduction

- Delays are used to model real-world circuit behavior
- In simulation, delays help represent propagation time, setup/hold constraints, and pulse filtering
- Verilog provides delay modeling using explicit delay statements and specify blocks
- Two types of delay modeling:
 - Behavioral delay modeling (using `#delay` statements)
 - Structural delay modeling (using `specify` blocks for precise timing analysis)

Verification and Timing Simulation

Functional Verification

- Ensures that the designed circuit operates correctly as intended
- Verifies logic and functionality without considering delays

Importance of Timing Verification

- Real hardware has delays due to logic elements and signal paths
- Timing verification ensures circuits meet timing requirements
- Increasingly important as circuits become smaller and faster

Delay Models

- There are three types of delay models used in Verilog:
 - Distributed
 - Lumped
 - pin-to-pin (path) delays

Distributed Delay Model

- Delays are assigned inside the RTL code at individual logic elements
- Each gate or module has its own delay specification
- More realistic but complex to manage

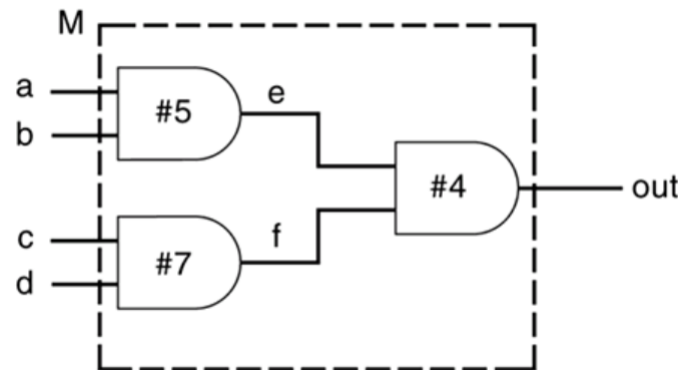
Example:

and #2 (y, a, b); // AND gate with 2-time unit delay

Distributed delays provide detailed delay modeling.

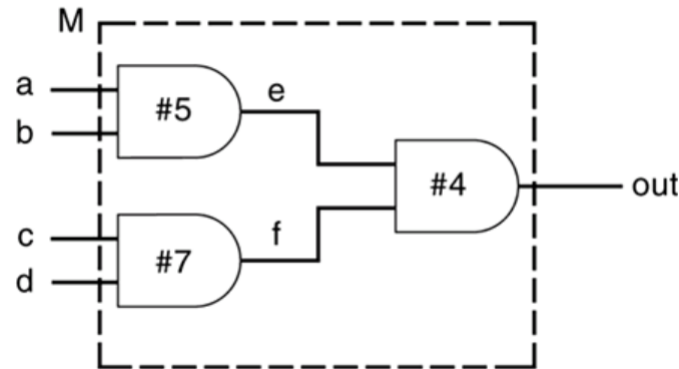
Delays in each element of the circuit are specified

Example: Distributed Delay Model



```
1  //Distributed delays in gate-level modules
2  module M (out, a, b, c, d);
3  output out;
4  input a, b, c, d;
5  wire e, f;
6  //Delay is distributed to each gate.
7  and #5 a1(e, a, b);
8  and #7 a2(f, c, d);
9  and #4 a3(out, e, f);
10 endmodule
```

Example: Distributed Delay Model



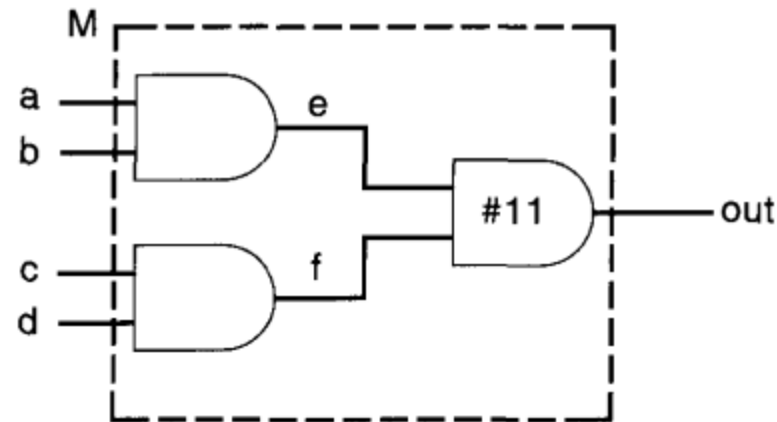
```
1  //Distributed delays in data flow definition of a module
2  module M (out, a, b, c, d);
3  output out;
4  input a, b, c, d;
5  wire e, f;
6  //Distributed delay in each expression
7  assign #5 e = a & b;
8  assign #7 f = c & d;
9  assign #4 out = e & f;
10 endmodule
```


Lumped Delay

- A single delay is assigned to an entire module
- Simplifies delay specification but is less accurate
- Treats all internal operations as instantaneous, with a final output delay.

```
1 module my_module (out, in);  
2     output out;  
3     input in;  
4     assign #5 out = ~in; // Lumped delay of 5-time units  
5 endmodule  
-
```

Example: Lumped Delay



```
1  //Lumped Delay Model
2  module M (out, a, b, c, d);
3  output out;
4  input a, b, c, d;
5  wire e, f ;
6  and a1(e, a, b);
7  and a2(f, c, d);
8  and #11 a3 (out, e, f ) ;//delay only on the output gate
9  endmodule
```

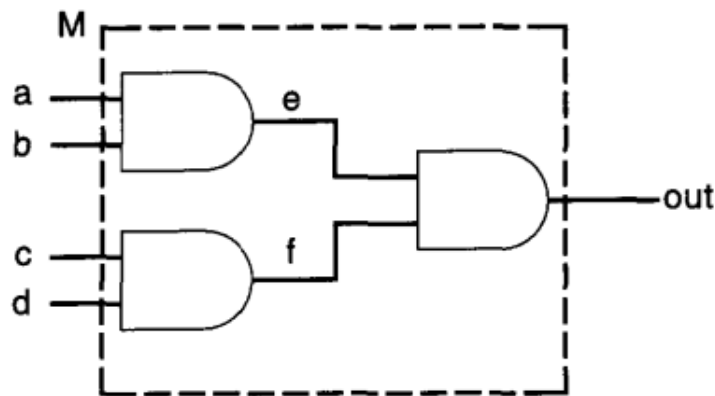
Lumped delays models are easy to model compared with distributed delays

Pin-to-Pin Delays

- Defines delays between specific input and output pins
- Used in gate-level modeling and timing analysis
- More accurate for complex circuits and used in specify blocks
- They are also called as path delays

```
1 specify
2   | (in1 => out) = 3; // 3-time unit delay from in1 to out
3   endspecify
```

Example: Pin-to-Pin Delays



path a-e-out, delay = 9
path b-e-out, delay = 9
path c-f-out, delay = 11
path d-f-out, delay = 11

```
1  //pin-to-pin delays
2  module M (out, a, b, c, d);
3  output out;
4  input a, b, c, d;
5  wire e, f;
6  //Specify block with path delay statements
7  specify
8    (a => out) = 9;
9    (b => out) = 9;
10   (c => out) = 11;
11   (d => out) = 11;
12 endspecify
13 //gate instantiations
14 and a1(e, a, b);
15 and a2(f, c, d);
16 and a3(out, e, f);
```

Pin-to-Pin (Path) Delays

- Directly available for standard parts from data books
- For custom modules, delays are obtained through circuit characterization using tools like SPICE
- More manageable than distributed delays for large circuits
- Requires knowledge of only I/O pins, not the internal design
- Independent of design style (gate-level, data flow, behavioral, or mixed)
- Also referred to as path delays in timing analysis.

Example: Pin-to-Pin Delays

```
1  `timescale 1ns/1ps // Define time unit and precision
2
3  module AND_Gate (input A, input B, output Y);
4
5      assign Y = A & B; // AND gate functionality
6
7      // Specify block for pin-to-pin delay
8      specify
9          (A => Y) = (2, 3); // Delay from A to Y: 2ns rise, 3ns fall
10         (B => Y) = (1, 2); // Delay from B to Y: 1ns rise, 2ns fall
11     endspecify
12
13 endmodule
```

- Pin-to-pin delays are added for inputs A and B to output Y:
- $(A \Rightarrow Y) = (2, 3) \rightarrow$ 2ns rise delay, 3ns fall delay
- $(B \Rightarrow Y) = (1, 2) \rightarrow$ 1ns rise delay, 2ns fall delay.

Module Path Delay

- A module path delay represents the delay between a source (input/inout) and a destination (output/inout) pin
- In Verilog, path delays are defined inside a specify block
- The specify block is enclosed between the **specify** and **endspecify** keywords

Specify Block

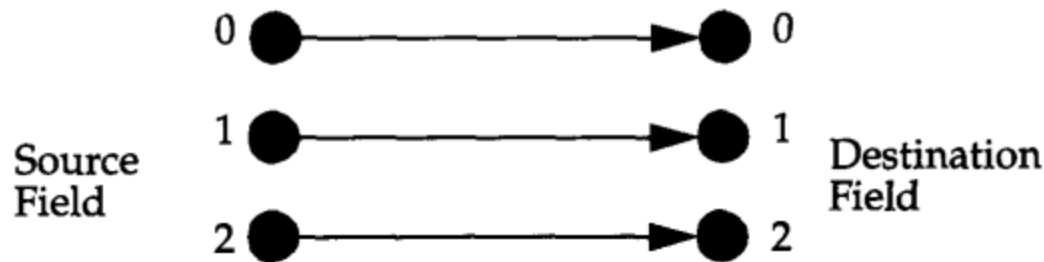
- Purpose of Specify Blocks:
 - Assign pin-to-pin timing delays in a module
 - Set up timing checks to ensure proper signal transitions
 - Define specparam constants to specify timing parameters.

```
specify  
| (input1 => output1) = delay_value;  
endspecify
```


Specify Block (Parallel Connection =>)

- Defines a bit-to-bit delay between source and destination
- Uses => symbol for bit-to-bit delays

```
( <sourcefield> => <destinationfield> ) = <delay-value>;
```



- Source and destination vectors must have the same width

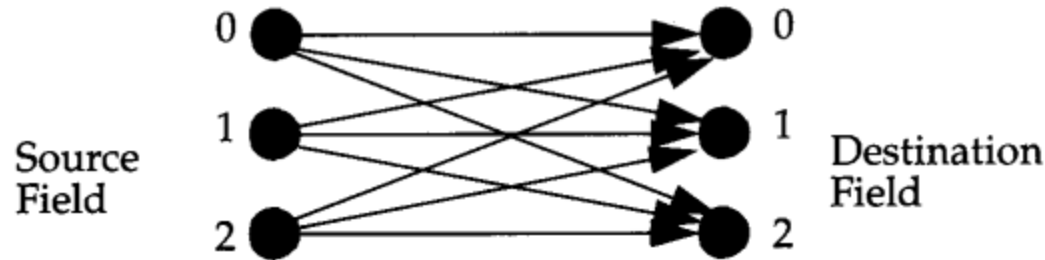
```
(a => out) = 9; // Single-bit connection  
(a[3:0] => out[3:0]) = 9; // 4-bit vector connection
```

- Illegal if bit widths don't match.

Specify Block (Full Connection *>)

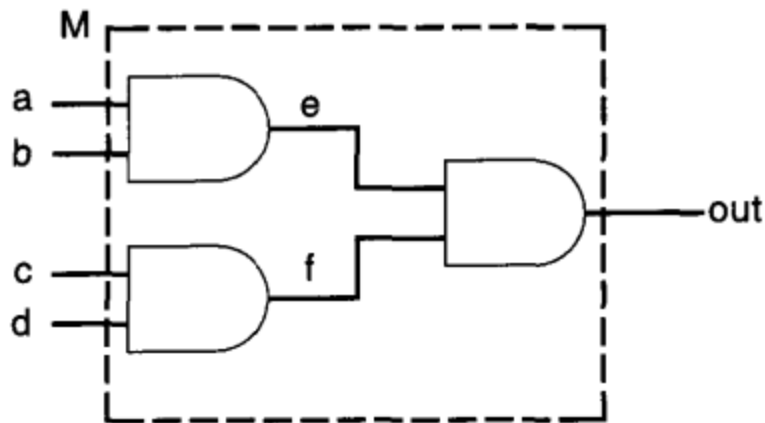
- Defines a full delay connection from each bit in source to every bit in destination
- Uses *> symbol for bit-to-bit delays

```
( <sourcefield> *> <destination-field> ) = <delay-value>;
```



- Works even if vector widths do not match
- More efficient than using multiple parallel connections

Specify Block (Full Connection *>)



*path a-e-out, delay = 9
path b-e-out, delay = 9
path c-f-out, delay = 11
path d-f-out, delay = 11*

```
1  //Full Connection
2  module M (out, a, b, c, d);
3  output out;
4  input a, b, c, d;
5  wire e, f;
6  //full connection
7  specify
8      (a,b*> out) = 9;
9      (c,d*> out) = 11;
10 endspecify
11 and a1(e,a, b);
12 //Full Connection
13 and a2(f, c, d);
14 and a3(out, e, f);
15 endmodule
```

specparam

- Special parameters declared inside a specify block using the keyword specparam
- Specparam is used to replace hardcoded delay values in pin-to-pin delay specifications
- Makes the design more flexible and readable
- specparam values define delays for different signal transitions (e.g., rise, fall, propagation delays).

```
1  specify
2      specparam t_rise = 5, t_fall = 6, t_prop = 9;
3      (a => out) = (t_rise, t_fall);
4      (b => out) = t_prop;
5  endspecify
```

Conditional Path Delays

- Conditional Path Delay are delays that change based on the states of input signals
- Expressed using the if conditional statement inside the specify block.
- else construct cannot be used.
- Logical, bitwise, reduction, concatenation, and conditional operators are supported
- Conditional path delays are also known as state dependent path delays(SDPD)

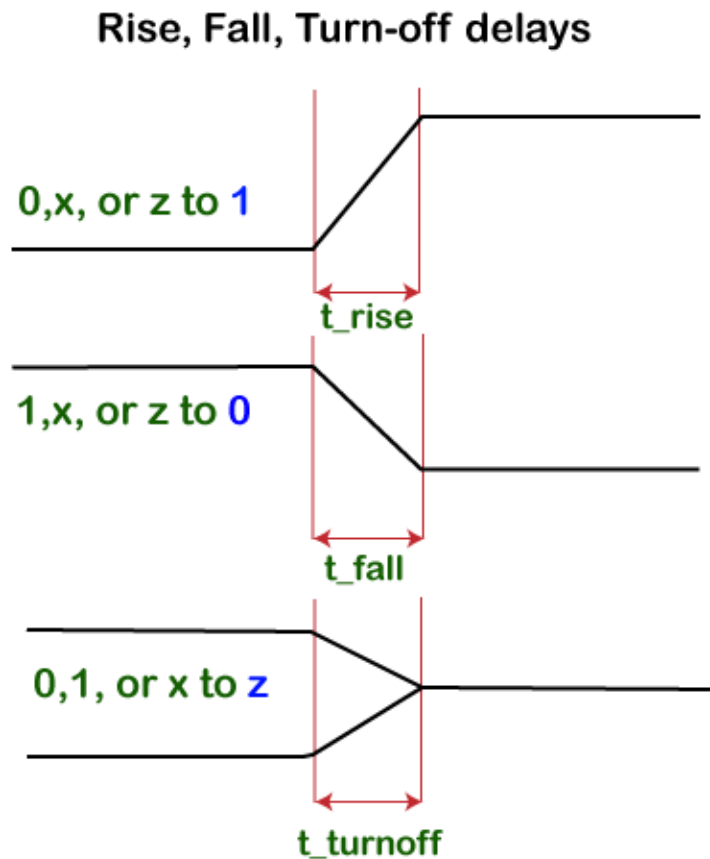
Example: Conditional Path Delays

```
1  specify
2      if (a) (a => out) = 9;    // Delay of 9 if 'a' is high
3      if (!a) (a => out) = 10; // Delay of 10 if 'a' is low
4
5      if (b & c) (b => out) = 9; // Delay of 9 if b AND c are high
6      if (!(b & c)) (b => out) = 13; // Delay of 13 otherwise
7
8      if ({c, d} == 2'b01) (c, d *> out) = 11;
9      if ({c, d} != 2'b01) (c, d *> out) = 13;
10 endspecify
```

Example: Conditional Path Delays

```
1  //Conditional Path Delays
2  module M (out, a, b, c, d);
3  output out;
4  input a, b, c, d;
5  wire e, f; //specify block with conditional pin-to-pin timing
6  specify
7  //different pin-to-pin timing based on state of signal a.
8  if (a) (a => out) = 9;
9  if (-a) (a => out) = 10;
10 //Conditionalexpression contains two signals b , c.
11 //If b & c is true, delay = 9,
12 //otherwise delay = 13.
13 if (b & c) (b => out) = 9;
14 if (!(b & c)) (b => out) = 13;
15 //Use concatenation operator
16 //Use Full connection
17 if ({c,d} == 2'b01) //Conditional Path Delays
18 |   (c,d*> out) = 11;
19 if ({c,d} != 2'b01) (c,d *> out) = 13;
20 endspecify
21 and a1(e, a, b);
22 and a2(f, c, d);
23 and a3(out, e, f);
24 endmodule
```

Rise, Fall, and Turn-off Delays



- The time taken for the output of a gate to change from some value to 1 is called a **rise delay**
- The time taken for the output of a gate to change from some value to 0 is called a **fall delay**
- The time taken for the output of a gate to change from some value to high impedance is called **turn-off delay**

Rise, Fall, and Turn-off Delays

- Specifies different delays based on signal transitions
- Can use one, two, three, six, or twelve delay values
- Other numbers of delay values are illegal

Delay Values Meaning:

- **1 Delay:** Used for all transitions
- **2 Delays:** Rise ($0 \rightarrow 1$, $0 \rightarrow Z$, $Z \rightarrow 1$) & Fall ($1 \rightarrow 0$, $1 \rightarrow Z$, $Z \rightarrow 0$)
- **3 Delays:** Rise, Fall, and Turn-off ($0 \rightarrow Z$, $1 \rightarrow Z$)
- **6 Delays:** Covers all major transitions ($0 \rightarrow 1$, $1 \rightarrow 0$, $0 \rightarrow Z$, $Z \rightarrow 1$, $1 \rightarrow Z$, $Z \rightarrow 0$)
- **12 Delays:** Covers all possible transitions including X states

Rise, Fall, and Turn-off Delays

```
//Specify one delay only. Used for all transitions  
specparam t_delay = 11;  
(clk => q) = t_delay;
```

```
//Specify two delays, rise and fall  
//Rise used for transitions 0->1, 0->z, z->1  
//Fall used for transitions 1->0, 1->z, z->0  
specparam t_rise = 9, t_fall = 13;
```

```
(clk => q) = (t_rise, t_fall);  
//Specify three delays, rise, fall, and turn-off  
//Rise used for transitions 0->1, z->1  
//Fall used for transitions 1->0, z->0  
//Turn-off used for transitions 0->z, 1->z  
specparam t_rise = 9, t_fall = 13, t_turnoff = 11;  
(clk => q) = (t_rise, t_fall, t_turnoff);
```

Rise, Fall, and Turn-off Delays

```
//specify six delays.
//Delays are specified in order
//for transitions 0->1, 1->0, 0->z, z->1, 1->z, z->0. Order
//must be followed strictly.
specparam t_01 = 9, t_10 = 13, t_0z = 11;
specparam t_z1 = 9, t_lz = 11, t_z0 = 13;
(clk => q) = (t_01, t_10, t_0z, t_z1, t_lz, t_z0);
//specify twelve delays.
//Delays are specified in order
//for transitions 0->1, 1->0, 0->z, z->1, 1-zz, z->0
// / 0->X, X->1, 1->X, X->0, X->z, z->X.
//Order must be followed strictly.
specparam t_01 = 9, t_10 = 13, t_0z = 11;
specparam t_z1 = 9, t_lz = 11, t_z0 = 13;
specparam t_0x = 4, t_x1 = 13, t_lx = 5;
specparam t_x0 = 9, t_xz = 11, t_zx = 7;
(clk => q) = (t_01, t_10, t_0z, t_z1, t_lz, t_z0,
t_0x, t_x1, t_lx, t_x0, t_xz, t_zx ) ;
```

Min, Max, and Typical Delays

- Delays can be expressed in min : typ : max form
- The **min value** is the minimum delay value that the gate is expected to have
- The **typ value** is the typical delay value that the gate is expected to have
- The **max value** is the maximum delay value that the gate is expected to have.
- Used to model process variations in manufacturing

```
specparam t_rise = 8:9:10, t_fall = 12:13:14, t_turnoff = 10:11:12;  
(clk => q) = (t_rise, t_fall, t_turnoff);
```

Handling X Transitions

- Transitions from a known state (0, 1, Z) to X should take the minimum possible time (because uncertainty propagates quickly)
- Transitions from X to a known state (0, 1, Z) should take the maximum possible time (because resolving uncertainty is slow).

Example: Handling X Transitions

```
//A timing specification for pin-to-pin transitions includes six delay values  
// Transitions: 0->1, 1->0, 0->z, z->0, 1->z, z->1
```

```
specparam t01 = 10, t10 = 14, t0z = 9, tz1 = 10, t1z = 12, tz0 = 15;
```

```
(clk => q) = (t01, t10, t0z, tz1, t1z, tz0);
```

- $t_{01} = 10 \rightarrow$ Delay for transition from 0 to 1
- $t_{10} = 14 \rightarrow$ Delay for transition from 1 to 0
- $t_{0z} = 9 \rightarrow$ Delay for transition from 0 to Z
- $t_{z1} = 10 \rightarrow$ Delay for transition from Z to 1
- $t_{1z} = 12 \rightarrow$ Delay for transition from 1 to Z
- $t_{z0} = 15 \rightarrow$ Delay for transition from Z to 0

Example: Handling X Transitions

```
//A timing specification for pin-to-pin transitions includes six delay values  
// Transitions: 0->1, 1->0, 0->z, z->0, 1->z, z->1
```

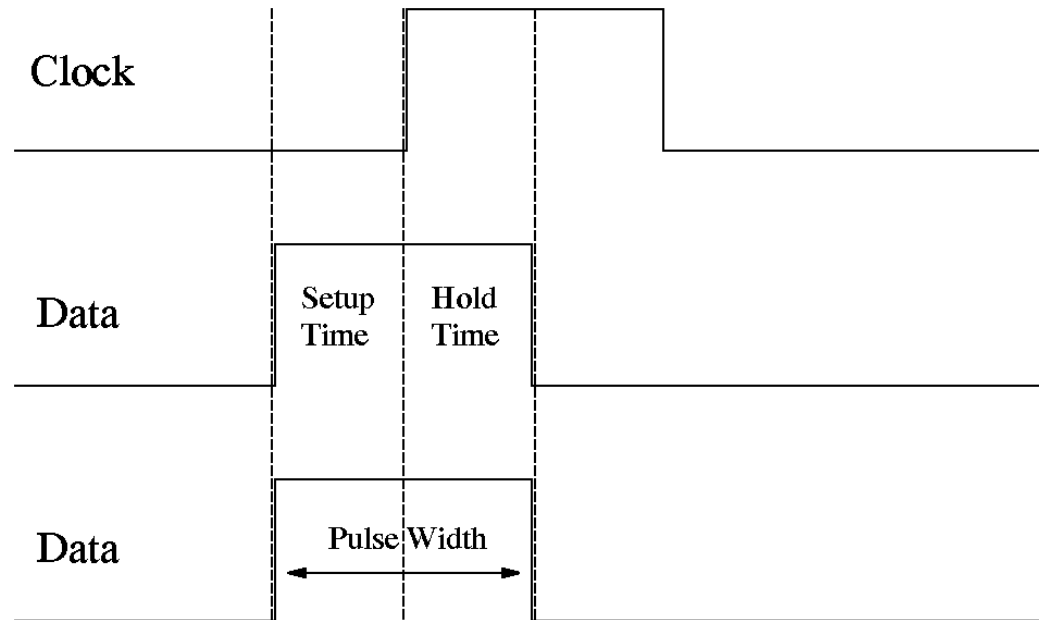
```
specparam t01 = 10, t10 = 14, t0z = 9, tz1 = 10, t1z = 12, tz0 = 15;
```

```
(clk => q) = (t01, t10, t0z, tz1, t1z, tz0);
```

Transition	Delay Calculation	Result
0 → X	min(t01, t0z)	min(10, 9) = 9
1 → X	min(t10, t1z)	min(14, 12) = 12
Z → X	min(tz0, t0z)	min(15, 9) = 9
X → 0	max(t10, tz0)	max(14, 15) = 15
X → 1	max(t01, tz1)	max(10, 10) = 10
X → Z	max(t1z, t0z)	max(12, 9) = 12

Setup and Hold Timing Checks (\$setup and \$hold)

- Setup Time (\$setup) is the minimum time the data must be stable before the active clock edge
- Violations occur if data changes too close to the clock edge



Setup Timing Checks (\$setup)

Syntax

`$setup(data_event, reference_event , limit);`

- `data_event` : Signal that is monitored for violations
- `Reference_event` : signal used as reference.
- `limit`: minimum time required between the two events.
- Violation if:

$T_{\text{reference_event}} - T_{\text{data_event}} < \text{limit}.$

```
// Violation reported if (Tposedge-clock - Tdata) < 3.  
✓ specify  
|   $setup(data, posedge clock, 3);  
endspecify
```

Example : \$setup

```
1  `timescale 1ns/1ps // Set the simulation time scale
2  module setup_check_tb;
3      reg clock;
4      reg data;
5      // Clock Generation: 10ns period (100MHz)
6      always #5 clock = ~clock;
7
8      initial begin
9          clock = 0;
10         data = 0;
11         // Normal case: Data changes before setup time requirement
12         #2 data = 1;
13         #4 data = 0;
14         // Violation case: Data changes too close to posedge clock
15         #8 data = 1;
16         #1 data = 0; // This violates the setup time
17         #20 $finish;
18     end
19 endmodule
20
21 module setup_check (input data, input clock);
22     specify
23         $setup(data, posedge clock, 3); // Setup time requirement: 3 time units
24     endspecify
25 endmodule
```

Hold Timing Checks (\$hold)

- Hold Time (\$hold) is the minimum time the data must remain stable after the active clock edge
- Violations occur if data changes too soon after the clock edge

Syntax :

`$hold(reference_event, data_event, limit);`

Reference_event : signal used as reference for monitoring

data_event: signal that is checked against the reference

limit: minimum time required between the two events.

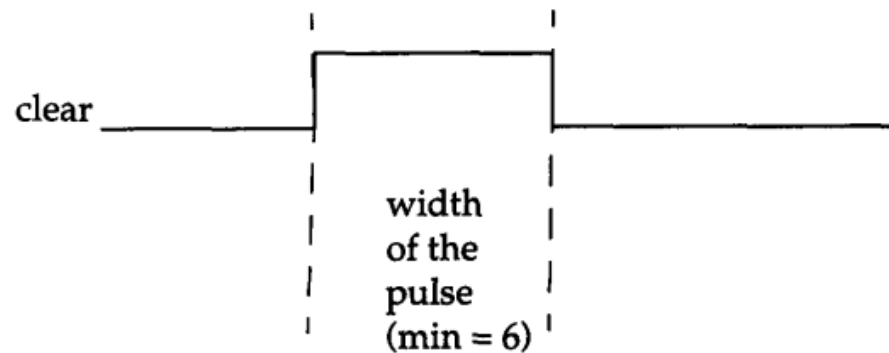
Violation if: $T_{data_event} - T_{reference_event} < time_limit$

Hold Timing Checks (\$hold)

```
1  `timescale 1ns/1ps // Set the simulation time scale
2  module hold_check_tb;
3      reg clock;
4      reg data;
5      // Clock Generation: 10ns period (100MHz)
6      always #5 clock = ~clock;
7
8      initial begin
9          clock = 0;
10         data = 0;
11         // Normal case: Data remains stable for at least 2 ns after posedge clock
12         #2 data = 1;
13         #6 data = 0; // Valid case, changes after 2 ns from clock edge
14         // Violation case: Data changes too soon after posedge clock
15         #8 data = 1;
16         #1 data = 0; // This violates the hold time
17         #20 $finish;
18     end
19 endmodule
20
21 module hold_check (input data, input clock);
22     specify
23         | $hold(posedge clock, data, 2); // Hold time requirement: 2 time units
24     endspecify
25 endmodule
```

Width Check (\$width)



- Ensures that a pulse maintains a minimum required width
- The system detects glitches or narrow pulses that may cause timing issues



```
✓ specify  
|   $width(posedge clock, 6); // Clock pulse width must be at least 6 time units  
endspecify
```

If $T(\text{negedge clock}) - T(\text{posedge clock}) < 6$, a width violation is reported

Width Check (\$width)



```
1  `timescale 1ns/1ps // Set the simulation time scale
2  module width_check_tb;
3      reg clock;
4      reg data;
5      // Clock Generation: 10ns period (100MHz)
6      always #5 clock = ~clock;
7
8      initial begin
9          clock = 0;
10         data = 0;
11         // Normal case: Pulse width of 'data' is at least 4 ns
12         #2 data = 1;
13         #5 data = 0; //  Valid case, pulse width = 5 ns (>= 4 ns)
14
15         // Violation case: Pulse width of 'data' is too short
16         #8 data = 1;
17         #2 data = 0; //  This violates the width constraint (only 2 ns)
18         #20 $finish;
19     end
20 endmodule
21
22 module width_check (input data, input clock);
23     specify
24         | $width(posedge data, 4); // Minimum pulse width of data must be 4 ns
25     endspecify
26 endmodule
```

Path Pulses (\$pathpulses)

- **\$pathpulses** is a Verilog timing check system task that detects glitches or pulses that propagate through a specific path but do not meet the minimum pulse width requirement.

```
1  module pathpulses_check (input clk, input data, output reg q);
2      always @(posedge clk) begin
3          q <= data;
4      end
5      specify
6          (data *> q) = (2, 2); // Setup and hold delays
7          $pathpulses(data, q, 3); // Minimum pulse width of 3 ns required
8      endspecify
9  endmodule
```

Path Pulses (\$pathpulses)

```
1  `timescale 1ns/1ps // Set the simulation time scale
2  module pathpulses_check_tb;
3      reg clock;
4      reg data;
5      wire q;
6      // Instantiate the DUT
7      pathpulses_check dut (.clk(clock), .data(data), .q(q));
8      // Clock Generation: 10ns period (100MHz)
9      always #5 clock = ~clock;
10
11     initial begin
12         clock = 0;
13         data = 0;
14         // Normal case: Data changes and remains stable
15         #2 data = 1;
16         #5 data = 0; //  Valid case, proper pulse width
17
18         // Violation case: Data glitches (very short pulse)
19         #8 data = 1;
20         #1 data = 0; //  Violation: Pulse is too short (only 1ns)
21         #20 $finish;
22     end
23 endmodule
```


Edge Sensitive Path

- Specifies timing constraints on specific edge transitions.
- Used in specify blocks for modeling timing behavior in ASIC/FPGA designs.

Syntax:

```
(posedge clk *> (out +: in)) = (min_delay, typ_delay, max_delay);
```

Edge Sensitive Path

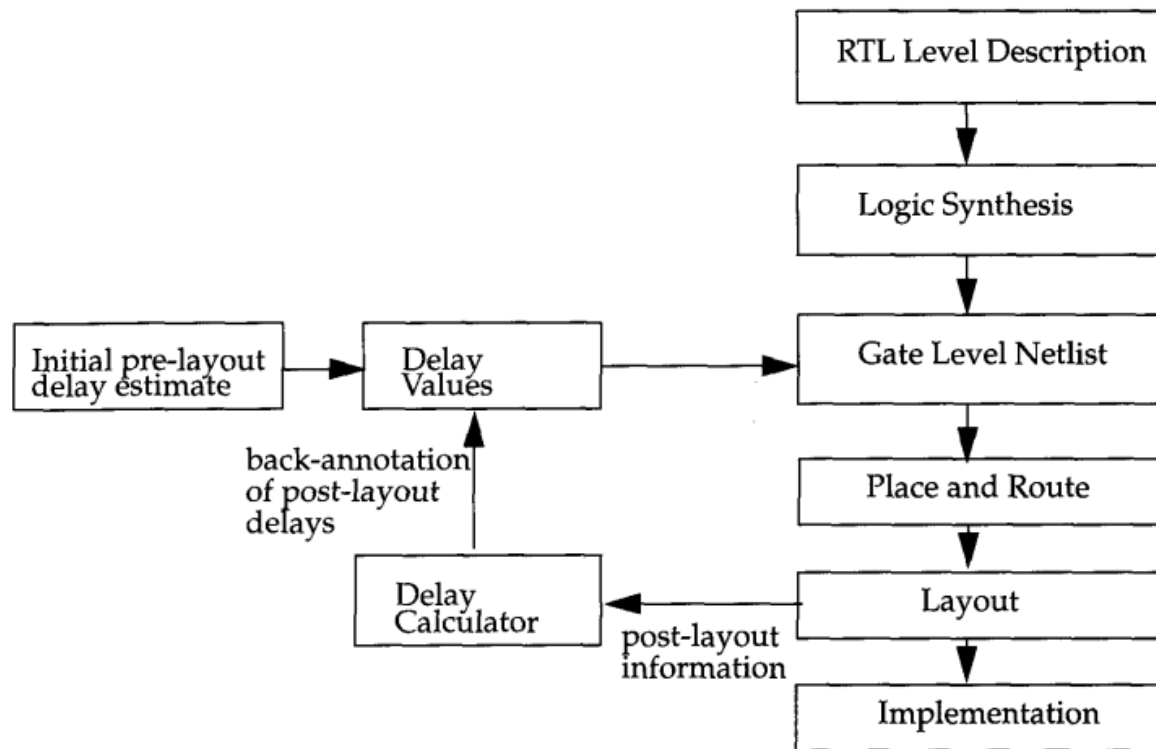
```
1  module edge_sensitive_path(input clk, input in, output reg out);
2  specify
3  |  (posedge clk *> (out += in)) = (2, 3, 4);
4  endspecify
5  |  always @(posedge clk)
6  |      out <= in;
7  endmodule
8
9  module tb_edge_sensitive_path;
10     reg clk, in;
11     wire out;
12     edge_sensitive_path uut (clk, in, out);
13     initial begin
14         clk = 0; in = 0;
15         #5 in = 1;
16         #10 clk = 1;
17         #10 clk = 0;
18         #5 in = 0;
19         #10 clk = 1;
20         #10 clk = 0;
21         #20 $finish;
22     end
23     always #5 clk = ~clk;
24 endmodule
```

Output:

```
Time: 10, clk: 1, in: 1, out: 1
Time: 30, clk: 1, in: 0, out: 0
```

Delay Back-Annotation

- Back-annotation is the process of applying real-world timing delays extracted from IC layout into the simulation for accurate timing verification



Delay Back-Annotation

Steps in Delay Back-Annotation:

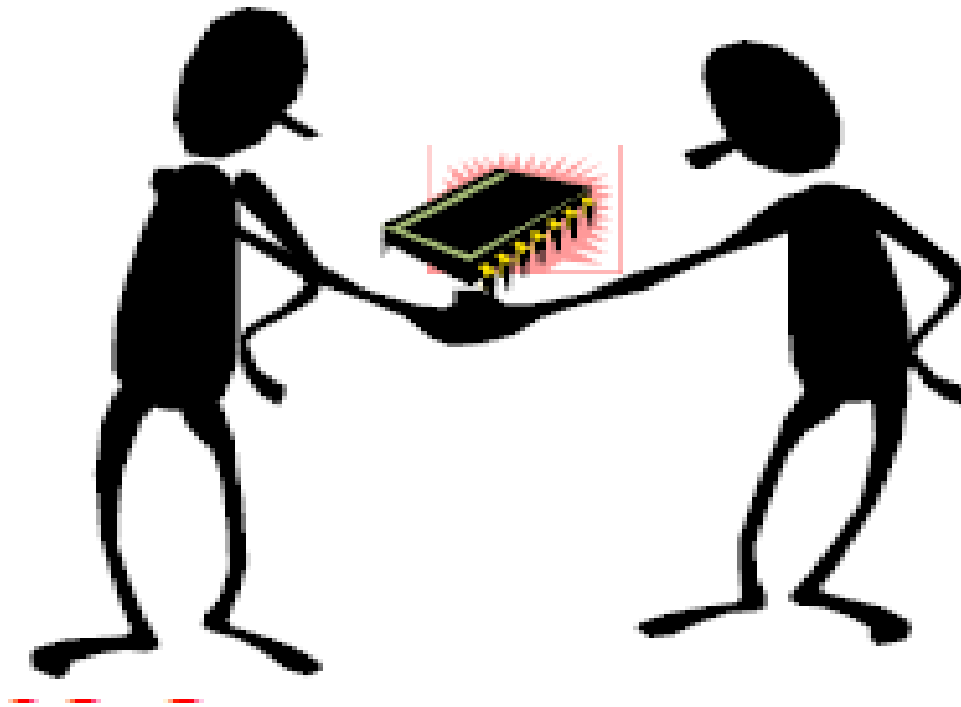
- **RTL Simulation:** The designer writes RTL code and tests functionality
- **Logic Synthesis:** The RTL is converted into a gate-level netlist
- **Pre-Layout Timing Estimation:** A delay calculator estimates delays before layout
- **Place & Route (Physical Design):** The gate-level netlist is converted into a layout
- **Post-Layout Timing Analysis:** Extracted resistance (R) and capacitance (C) values determine actual delays
- **Re-Simulation with Updated Delays:** Timing simulation is run again with back-annotated delays.
- **Optimization if Needed:** If timing violations occur, the design is optimized

Standard Delay Format (SDF)

- SDF files store timing delay information for gate-level simulation
- The format allows the annotation of delays into Verilog simulation
- Helps in post-layout verification by back-annotating delays into the simulation

Summary

- The specify block in Verilog is used for precise timing modeling
- timescale, \$time, and related functions help manage simulation time
- Delays can be lumped, distributed, or pin-to-pin based on circuit needs
- Path delays (`=>`, `*>`) and edge-sensitive paths ensure accurate timing control
- \$pathpulses prevents glitches by filtering out short pulses
- Proper delay modeling improves circuit reliability in high-speed designs.



Thank you !

Happy Learning