Verilog HDL:
Timing Parameters

Pravin Zode

Outline

- Timing Check and Notifier behavior
- Timing parameters
- Exercises on timing parameters

Timing Check & Notifier Behavior

- **Timing checks** ensure the correct sequence of signal transitions in digital circuits
- **Notifier behavior** is triggered when a timing violation occurs, alerting users for debugging.

Timing Check & Notifier Behavior

Importance of Timing Checks:

- Prevents setup and hold time violations
- Ensures correct synchronization between clock and data
- Detects timing faults early in the design process
- Helps avoid metastability issues in sequential circuits

Types of Timing Checks:

- Basic checks: $setup, $hold, $setuphold
- Recovery and removal checks: $recover, $removal, $recrem
- Skew-related checks: $skew, $fullskew, $timeskew
- Pulse width constraints: $width

Timing Parameters

- Define timing constraints to ensure proper circuit behavior

Key timing parameters:

- $setup, $hold, $setuphold – Ensure data stability before and after clock edge
- $recover, $width, $skew – Define recovery time, pulse width, and clock skew
- $fullskew, $removal, $recrem, $timeskew – Advanced timing constraints for accurate timing verification

Example : $setup & $hold

```
1   module timing_check_example(input clk, input d);
2   specify
3     $setup(d, posedge clk, 3);
4     $hold(posedge clk, d, 2);
5   endspecify
6   endmodule
7
8   module tb_timing_check_example;
9     reg clk, d;
10    timing_check_example uut (clk, d);
11    initial begin
12      clk = 0; d = 0;
13      #5 d = 1; #2 clk = 1;
14      #2 clk = 0;
15      #10 $finish;
16    end
17    always #5 clk = ~clk;
18  endmodule
```

Output:

```
Timing violation: Setup time not met at time 5
Timing violation: Hold time not met at time 7
```

$setuphold Timing Check

- Combines $setup and $hold into a single check
- Ensures data remains stable for the required time before and after the clock edge
- Syntax: $setuphold (reference, data, setup_time, hold_time, notifier);

Example : $setuphold

```
1   module setuphold_example(input clk, input d);
2     reg notifier;
3   specify
4     $setuphold(posedge clk, d, 3, 2, notifier);
5   endspecify
6   endmodule
7
8   module tb_setuphold_example;
9     reg clk, d;
10    setuphold_example uut (clk, d);
11    initial begin
12      clk = 0; d = 0;
13      #5 d = 1; #2 clk = 1;
14      #2 clk = 0;
15      #10 $finish;
16    end
17    always #5 clk = ~clk;
18  endmodule
```

Output:

```
Setup/Hold Violation Detected! Notifier triggered.
```

$recover, $width, $skew Timing Checks

- $recover: Ensures minimum recovery time between asynchronous reset and clock event
- $width: Specifies minimum pulse width constraints
- $skew: Defines maximum skew between related signals

Example : $recover

```
1   module timing_recover_example(input clk, input reset);
2     reg notifier;
3   specify
4     $recover(posedge reset, posedge clk, 5, notifier);
5   endspecify
6   endmodule
7
8   module tb_timing_recover_example;
9     reg clk, reset;
10    timing_recover_example uut (clk, reset);
11    initial begin
12      clk = 0; reset = 0;
13      #2 reset = 1; #3 clk = 1;
14      #5 clk = 0;
15      #10 $finish;
16    end
17    always #5 clk = ~clk;
18  endmodule
```

Output:

```
Recovery Time Violation Detected!
```

$fullskew Timing Check

- $fullskew: Defines the maximum allowed skew between two clock signals
- Ensures proper clock synchronization across multiple domains
- Syntax:

```
$fullskew(reference1, reference2, limit, notifier);
```

Example: $fullskew Timing Check

```
1   module fullskew_example(input clk1, input clk2);
2     reg notifier;
3   specify
4     $fullskew(posedge clk1, posedge clk2, 4, notifier);
5   endspecify
6   endmodule
7
8   module tb_fullskew_example;
9     reg clk1, clk2;
10    fullskew_example uut (clk1, clk2);
11    initial begin
12      clk1 = 0; clk2 = 0;
13      #2 clk1 = 1; #3 clk2 = 1;
14      #5 clk1 = 0; clk2 = 0;
15      #10 $finish;
16    end
17    always #5 clk1 = ~clk1;
18    always #6 clk2 = ~clk2;
19  endmodule
```

Output:

```
Clock Skew Violation Detected!
```

Example: $removal Timing Check

- $removal: Ensures that the asynchronous reset signal remains asserted for a minimum time after a clock edge
- Prevents metastability issues in sequential circuits.

Syntax:

```
$removal(reference, data, limit, notifier);
```

Example: $removal Timing Check

```
1   module removal_example(input clk, input reset);
2     reg notifier;
3   specify
4     $removal(posedge reset, posedge clk, 3, notifier);
5   endspecify
6   endmodule
7
8   module tb_removal_example;
9     reg clk, reset;
10    removal_example uut (clk, reset);
11    initial begin
12      clk = 0; reset = 0;
13      #2 reset = 1; #1 clk = 1;
14      #5 clk = 0;
15      #10 $finish;
16    end
17    always #5 clk = ~clk;
18  endmodule
```

Output:

```
Removal Time Violation Detected!
```

$recrem and $timeskew Timing Checks

- $recrem: Combines $recover and $removal into one constraint.
- $timeskew: Defines the maximum allowable time skew between two signals

$recrem and $timeskew Timing Checks

```
1   module recrem_timeskew_example(input clk, input rst, input sig);
2     reg notifier;
3   specify
4     $recrem(posedge rst, posedge clk, 5, 3, notifier);
5     $timeskew(posedge clk, posedge sig, 4, notifier);
6   endspecify
7   endmodule
8
9   module tb_recrem_timeskew_example;
10    reg clk, rst, sig;
11    recrem_timeskew_example uut (clk, rst, sig);
12    initial begin
13      clk = 0; rst = 0; sig = 0;
14      #2 rst = 1; #1 clk = 1;
15      #3 sig = 1;
16      #10 $finish;
17    end
18    always #5 clk = ~clk;
19  endmodule
```

Output:

```
Recovery/Removal Violation Detected!
Time Skew Violation Detected!
```

Summary

- Timing checks in Verilog ensure proper synchronization of signals in digital designs.
- Notifier behavior helps detect violations and debug timing issues effectively.
- Setup, hold, and recovery constraints prevent metastability and unreliable data transfer.
- Skew and pulse width constraints help manage clock variations and glitches.
- Using specify blocks allows precise modeling of delays and timing constraints.
- Practical verification with testbenches helps identify violations before hardware implementation.

Thank you !

Happy Learning