Chapter 13: Verification

Prof. Ming-Bo Lin

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

- Objectives
- Function verification
- Simulation
- Test bench design
- Dynamic timing analysis
- Static timing analysis
Objectives

After completing this chapter, you will be able to:

- Describe the importance and essential of verification
- Understand the essential of timing and functional verification
- Describe the essential issues of simulators
- Understand the essential principles of test bench designs
- Understand the principle of dynamic timing analysis
- Understand the principle of static timing analysis
- Understand issues of coverage analysis
- Describe the ISE design flow and related issues
Chapter 13: Verification

Syllabus

- Objectives
- Functional verification
  - Introduction
  - Functional verification
  - Simulation-based verification
- Simulation
- Test bench design
- Dynamic timing analysis
- Static timing analysis
Verification

- The goal of verification
  - To ensure 100% correct in functionality and timing
  - Spend 50 ~ 70% of time to verify a design

- Functional verification
  - Simulation
  - Formal proof

- Timing verification
  - Dynamic timing simulation (DTS)
  - Static timing analysis (STA)
Syllabus

- Objectives
- Verification
  - Introduction
  - Functional verification
  - Simulation-based verification
- Simulation
- Test bench design
- Dynamic timing analysis
- Static timing analysis
Functional Verification

- Simulated-based functional verification
  - A test bench
  - Input stimuli
  - Output analysis

- Formal verification
  - A protocol
  - An assertion
  - A property
  - A design rule
Models of Design under Test

- Black box model
- White box model
- Gray box model
Types of Assertion

- Static assertion
- Temporal assertion
Syllabus

- Objectives
- Verification
  - Introduction
  - Functional verification
  - Simulation-based verification
- Simulation
- Test bench design
- Dynamic timing analysis
- Static timing analysis
Simulation-Based Verification

1. Design specification
2. Functional test plan
3. Device under test (DUT)
4. Result checking
   - No: Meet the expected results
   - Yes: Done
Hierarchy of Functional Verification

- Designer level (or block-level)
- Unit level
- Core level
- Chip level
A Verification Test Set

- Verification test set includes at least
  - Compliance tests
  - Corner case tests
  - Random tests
  - Real code tests
  - Regression tests
  - Property check
Formal Verification

- Uses mathematical techniques
- Proves a design property
Syllabus

- Objectives
- Verification
- Simulation
  - Types of simulations and simulators
  - Architecture of HDL simulators
- Test bench design
- Dynamic timing analysis
- Static timing analysis
Types of Simulations

- Behavioral simulation
- Functional simulation
- Gate-level (logic) simulation
- Switch-level simulation
- Circuit-level (transistor-level) simulation
Variations of Simulations

- Software simulation
- Hardware acceleration
- Hardware emulation
Chapter 13: Verification

Syllabus

- Objectives
- Verification
- Simulation
  - Types of simulations and simulators
  - Architecture of HDL simulators
- Test bench design
- Dynamic timing analysis
- Static timing analysis
Chapter 13: Verification

An Architecture of HDL Simulators

Front end
- Parsing
- Elaboration
- Analysis
- Optimization
- Code generation

Compiler

RTL code (source)

User
- Simulation control
- Simulation engine

Results
Chapter 13: Verification

Verilog HDL Simulators

- **Interpreted simulators**
  - Cadence Verilog-XL simulator

- **Compiled code simulators**
  - Synopsys VCS simulator

- **Native code simulators**
  - Cadence Verilog-NC simulator
Event-Driven/Cycle-Based Simulators

- Event-driven simulators
  - Triggered by events
- Cycle-based simulators
  - On a cycle-by-cycle basis
Chapter 13: Verification

An Event-Driven Simulation

(a) A circuit example

(b) Timing wheel

(c) Scheduled events and the activity list

<table>
<thead>
<tr>
<th>Scheduled events</th>
<th>Activity list</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t = 0$</td>
<td>$a, c$</td>
</tr>
<tr>
<td>1</td>
<td>$b$</td>
</tr>
<tr>
<td>2</td>
<td>$f$</td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>$f = 1$</td>
</tr>
</tbody>
</table>
Syllabus

- Objectives
- Verification
- Simulation
- Test bench designs
  - Test bench design
  - Clock signal generation
  - Reset signal generation
  - Verification coverage
- Dynamic timing analysis
- Static timing analysis
Test Bench Design Principles

- Functions of a test bench
  - generates stimuli
  - checks responses in terms of test cases
  - employs reusable verification components

- Two types of test benches
  - deterministic
  - self-checking

- Options of choosing test vectors
  - Exhaustive test
  - Random test
  - Verification vector files
Test Bench Design Principles

- Two basic choices of stimulus generation
  - Deterministic versus random stimulus generation
  - Pregenerated test case versus on-the-fly test case generation

- Types of result checking
  - on-the-fly checking
  - end-of-test checking

- Result analysis
  - Waveform viewers
  - Log files
Types of Automated Response Checking

- **Golden vectors**
- **Reference model**
- **Transaction-based model**
Test Bench Designs --- A Trivial Example

// test bench design example 1: exhaustive test.
`timescale 1 ns / 100 ps

...  
nbit_adder_for UUT
    (.x(x), .y(y), .c_in(c_in), .sum(sum), .c_out(c_out));
reg [2*n-1:0] i;
initial for (i = 0; i <= 2**(2*n)-1; i = i + 1) begin
    x[n-1:0] = i[2*n-1:n]; y[n-1:0] = i[n-1:0]; c_in =1'b0;
    #20;
end
...
Test Bench Designs --- A Trivial Example

// test bench design example 2: Random test.
`timescale 1 ns / 100 ps

...  
nbit_adder_for UUT
       (.x(x), .y(y), .c_in(c_in), .sum(sum), .c_out(c_out));
integer i;
reg [n:0] test_sum;
initial for (i = 0; i <= 2*n ; i = i + 1) begin
    x = $random % 2**n;   y = $random % 2**n;
    c_in =1'b0;                    test_sum = x + y;
    #15;  if (test_sum != {c_out, sum})
           $display("Error iteration %h\n", i);
    #5; end
...
Test Bench Designs --- A Trivial Example

// test bench design example 3: Using Verification vector files.
`timescale 1 ns / 100 ps
parameter n = 4;
parameter m = 8;
...

// Unit Under Test port map
  nbit_adder_for UUT
    (.x(x), .y(y), .c_in(c_in), .sum(sum), .c_out(c_out));

integer i;
reg [n-1:0] x_array [m-1:0];
reg [n-1:0] y_array [m-1:0];
reg [n:0] expected_sum_array [m-1:0];
Test Bench Designs --- A Trivial Example

```
initial begin // reading verification vector files
    $readmemh("inputx.txt", x_array);
    $readmemh("inputy.txt", y_array);
    $readmemh("sum.txt", expected_sum_array);
endinitial

initial
    for (i = 0; i <= m - 1 ; i = i + 1) begin
        x = x_array[i];  y = y_array[i];
        c_in =1'b0;
        #15; if (expected_sum_array[i] != {c_out, sum})
            $display("Error iteration %h\n", i);
        #5;    end
initial  #200 $finish;
```
Syllabus

- Objectives
- Verification
- Simulation
- Test bench designs
  - Test bench design
  - Clock signal generation
  - Reset signal generation
  - Verification coverage
- Dynamic timing analysis
- Static timing analysis
Types of Clock Signals

- Types of clock signals
  - A general clock signal
  - Aligned derived clock signals
  - Clock multipliers
  - Asynchronous clock signals
A General Clock Signal

Examples

initial clk <= 1’b0;
always #10 clk <= ~clk;

initial begin
    clk <= 1’b0;
    forever #10 clk <= ~clk;
end

reg clk;
always begin
    #5 clk <= 1’b0;
    #5 clk <= 1’b1; end
A General Clock Signal

- **Truncation error**

  ```vhdl
  `timescale 1 ns / 1 ns
  reg clk;
  parameter clk_period = 25;
  
  always begin
    #(clk_period/2) clk <= 1ʼb0;
    #(clk_period/2) clk <= 1ʼb1;
  end
  ```

- **Rounding error**

  ```vhdl
  `timescale 1 ns / 1 ns
  reg clk;
  parameter clk_period = 25;
  
  always begin
    #(clk_period/2.0) clk <= 1ʼb0;
    #(clk_period/2.0) clk <= 1ʼb1;
  end
  ```

- **Proper precision**

  ```vhdl
  `timescale 1 ns / 100 ps
  reg clk;
  parameter clk_period = 25;
  
  always begin
    #(clk_period/2) clk <= 1ʼb0;
    #(clk_period/2) clk <= 1ʼb1;
  end
  ```
Aligned Derived Clock Signals

- **An improper approach**

  ```verilog
  always begin
    if (clk == 1'b1) clk2 <= ~ clk2;
  end
  ```

- **A proper approach**

  ```verilog
  // both clk1 and clk2 are derived from clk.
  always begin
    clk1 <= clk;
    if (clk == 1'b1) clk2 <= ~ clk2;
  end
  ```
Clock Multipliers

- An example

```verilog
initial begin
    clk1 <= 1'b0;
    clk4 <= 1'b0;
    forever begin
        repeat (4) begin
            #10 clk4 <= ~ clk4; end
        clk1 <= ~ clk1;
    end
end
```
Asynchronous Clock Signals

- “Asynchronous” means random

```
initial begin
    clk100 <= 1’b0;
    #2;
    forever begin
        #5 clk100 <= ~ clk100;
    end
end
```

```
initial begin
    clk33 <= 1’b0;
    #5;
    forever begin
        #15 clk33 <= ~ clk33;
    end
end
```
Syllabus

- Objectives
- Verification
- Simulation
- Test bench designs
  - Test bench design
  - Clock signal generation
  - Reset signal generation
  - Verification coverage
- Dynamic timing analysis
- Static timing analysis
Reset Signal Generations

- Race condition
  - Using nonblocking assignments

```verilog
always begin
    #5 clk = 1'b0;
    #5 clk = 1'b1;
end
initial begin // has race condition.
    reset = 1'b0;
    #20 reset = 1'b1;
    #40 reset = 1'b0;
end
```

```verilog
always begin
    #5 clk <= 1'b0;
    #5 clk <= 1'b1;
end
initial begin // no race condition.
    reset <= 1'b0;
    #20 reset <= 1'b1;
    #40 reset <= 1'b0;
end
```
Reset Signal Generations

- Increase of maintainability

```verilog
always begin
    #(clk_period/2) clk <= 1'b0;
    #(clk_period/2) clk <= 1'b1;
end
initial begin
    reset = 1'b0;
    wait (clk !== 1'bx);
    repeat (3) @(negedge clk) reset <= 1'b1;
    reset <= 1'b0;
end
```
Reset Signal Generations

❖ The use of a task

```verilog
always begin
    #(clk_period/2) clk <= 1'b0;
    #(clk_period/2) clk <= 1'b1;
end

// using a task
task hardware_reset;
begintask
    reset = 1'b0;
    wait (clk !== 1'bx);
    // set reset to 1 for two clock cycles
    repeat (3) @(negedge clk) reset <= 1'b1;
    reset <= 1'b0;
endtask
```
Syllabus

- Objectives
- Verification
- Simulation
- Test bench designs
  - Test bench design
  - Clock signal generation
  - Reset signal generation
  - Verification coverage
- Dynamic timing analysis
- Static timing analysis
Coverage Analysis

- Two major types
  - Structural coverage
  - Functional coverage

Q: What does 100% functional coverage mean?
- You have covered all the coverage points you included in the simulation
- By no means the job is done
Structural Coverage

- Statement coverage
- Branch or conditional coverage
- Toggle coverage
- Trigger coverage
- Expression coverage
- Path coverage
- Finite-state machine coverage
Chapter 13: Verification

Functional Coverage

- Functional coverage
  - Item coverage
  - Cross coverage
  - Transition coverage

<table>
<thead>
<tr>
<th>Module</th>
<th>Stmt count</th>
<th>Stmt hits</th>
<th>Stmt miss</th>
<th>Stmt %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Three-step Booth</td>
<td>35</td>
<td>35</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>Controller</td>
<td>21</td>
<td>21</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>Datapath</td>
<td>5</td>
<td>5</td>
<td>0</td>
<td>100%</td>
</tr>
</tbody>
</table>
Chapter 13: Verification

Syllabus

- Objectives
- Verification
- Simulation
- Test bench designs
- Dynamic timing analysis
  - SDF and delay back-annotation
  - ISE design flow
  - ISE simulation flow
- Static timing analysis
SDF and Generation Guidelines

- The standard delay format (SDF) file
- The SDF specifies
  - IOPATH delay
  - INTERCONNECT delay
  - Timing check (SETUP, HOLD, etc)
Generation of SDF Files

- **Pre-layout**
  - Gate delay information
  - `_map.sdf (contains gate delay only) in ISE design flow`

- **Post-layout**
  - Both gate and interconnect delay information
  - `_timesim.sdf in ISE design flow`
Delay Back-Annotation

Initial pre-layout delay estimation

Delay values

Back-annotation of pre-layout delays

Pre-layout information

Delay calculator

Back-annotation of post-layout delays

Post-layout information

Delay calculator

RTL description

Logic synthesis

Gate-level netlist

Placement

Pre-layout netlist

Routing

Post-layout netlist

\$sdf\_annotate (design\_file\_name \_map.sdf", design\_file\_name);

\$sdf\_annotate (design\_file\_name \_timesim.sdf", design\_file\_name);
Syllabus

- Objectives
- Verification
- Simulation
- Test bench designs
- Dynamic timing analysis
  - SDF and delay back-annotation
  - ISE design flow
  - ISE simulation flow
- Static timing analysis
The ISE Design Flow

- Design entry
- Synthesis to create a gate netlist
- Implementation
  - Translation
  - Map
  - Place and route
- Configure FPGA
The ISE Design Flow

1. Specification
2. Design entry
3. Synthesis
   - Timing constraints
   - Technology-independent optimization
   - Technology-dependent optimization
4. Translation
   - STA or DTA
5. Map
6. Place and route
7. Timing Analysis
   - Timing analysis (only logic delay)
   - Timing Analysis (logic delay & wire delay)
8. Generate programming file
9. Functional verification
10. Floorplanning
Chapter 13: Verification

Syllabus

- Objectives
- Verification
- Simulation
- Test bench designs
- Dynamic timing analysis
  - SDF and delay back-annotation
  - ISE design flow
  - ISE simulation flow
- Static timing analysis
A Simulation Flow --- An ISE-Based Flow

Files required for gate-level simulations:
- `glbl.v`
- `simprims` folder (compiled into `Xilinx_simprims` library)

Related files:
- `RTL_Source.v`
- `RTL_Source_tb.v`
- `*_translate.v`
- `*_map.sdf`
- `*_map.v`
- `*_timesim.sdf`
- `*_timesim.v`
- `RTL_Source_tb.v`
Syllabus

- Objectives
- Verification
- Simulation
- Test bench designs
- Dynamic timing analysis
- Static timing analysis
  - Introduction to timing analysis
  - Static timing analysis
  - Timing specifications
Timing Analysis

- Q: The output needs to be stable by $t = T$ for the correct functionality. But how to make sure of it?
- Two approaches
  - Dynamic timing simulation
  - Static timing analysis
Purposes of Timing Analysis

- **Timing verification**
  - if a design meets a given timing constraint?
  - Example: cycle-time constraint

- **Timing optimization**
  - Optimizes the critical portion of a design
  - Identifies critical paths
Syllabus

- Objectives
- Verification
- Simulation
- Test bench designs
- Dynamic timing analysis
- Static timing analysis
  - Introduction to timing analysis
  - Static timing analysis
  - Timing specifications
Why Static Timing Analysis?

- **Drawbacks of DTS**
  - has posed a bottleneck for large complex designs
  - relies on the quality and coverage of the test bench

- **Basic assumptions of STA**
  - No combinational feedback loops
  - All register feedback broken by the clock boundary
Static Timing Analysis

- In STA
  - Designs are broken into sets of signal paths
  - Each path has a start point and an endpoint

- Start points
  - Input ports
  - Clock pins of storage elements

- Endpoints
  - Output ports
  - Data input pins of storage elements
Four Types of Path Analysis

- Entry path (input-to-D path)
- Stage path (register-to-register path or clock-to-D path)
- Exit path (clock-to-output path)
- Pad-to-pad path (port-to-port path)
Path Groups

- Types of path groups
  - Path group
  - Default path group
Chapter 13: Verification

Syllabus

- Objectives
- Verification
- Simulation
- Test bench designs
- Dynamic timing analysis
- Static timing analysis
  - Introduction
  - Static timing analysis
  - Timing specifications
Timing Specifications

- Port-related constraints
  - Input delay (offset-in)
  - Output delay (offset-out)
  - Input-output (pad to pad)
  - Cycle time (period)
Setup Time and Hold Time Checks

- **Clock-related constraints**
  - Clock period
  - Setup time
  - Hold time
Timing Analysis

• A critical path
  • The path of longest propagation delay
  • A combinational logic path that has negative or smallest slack time

\[
\text{slack} = \text{required time} - \text{arrival time} \\
= \text{requirement} - \text{datapath (in ISE)}
\]
Timing Exceptions

- Two timing exceptions
  - False paths
  - Multi-cycle paths
False Paths

- A false path
  - A timing path does not propagate a signal
  - STA identifies as a failing timing path
Multi-Cycle Paths --- A Trivial Example

// a multiple cycle example
module multiple_cycle_example(clk, data_a, data_b, …);
…
// trivial multiple-cycle operations
always @(posedge clk) begin
    qout_a <= data_a * 5;
    @(posedge clk) qout_b <= data_b + 3;
    @(posedge clk) qout_c <= data_c - 7;
end

Q: Explain the operation of above code
Multi-Cycle Paths --- A Trivial Example

(a) Single-cycle timing relationship

(b) Two-cycle timing relationship

(c) Three-cycle timing relationship