Chapter 1
What Is Verification?

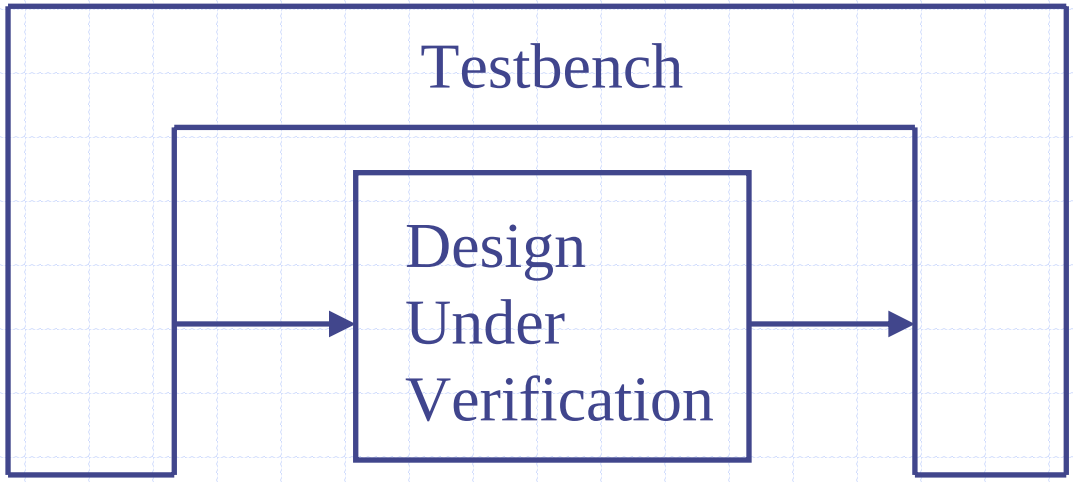
教育部顧問室
「超大型積體電路與系統設計」教育改進計畫
EDA聯盟編製

Verification

- A process used to demonstrate the functional correctness of design.
- Verification consumes about 70% of the design effort.
- The methodologies to reduce the verification time
  - Parallelism
  - Abstraction
  - Automation

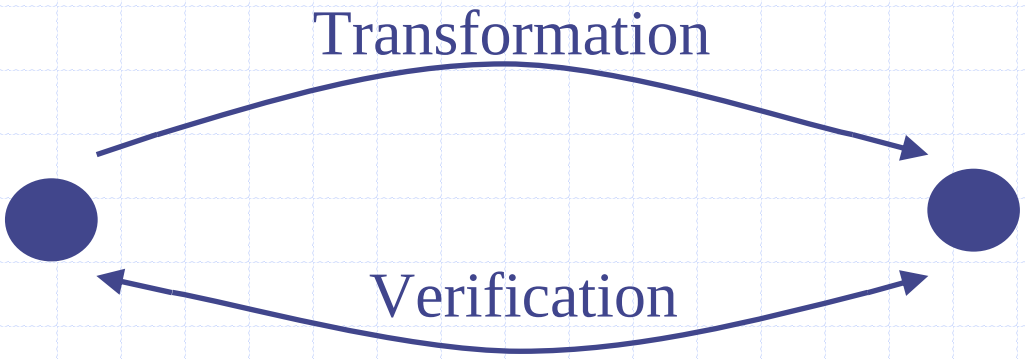
What is a Testbench?

- Create a pre-determined input sequence to a design, then optionally observe the response.



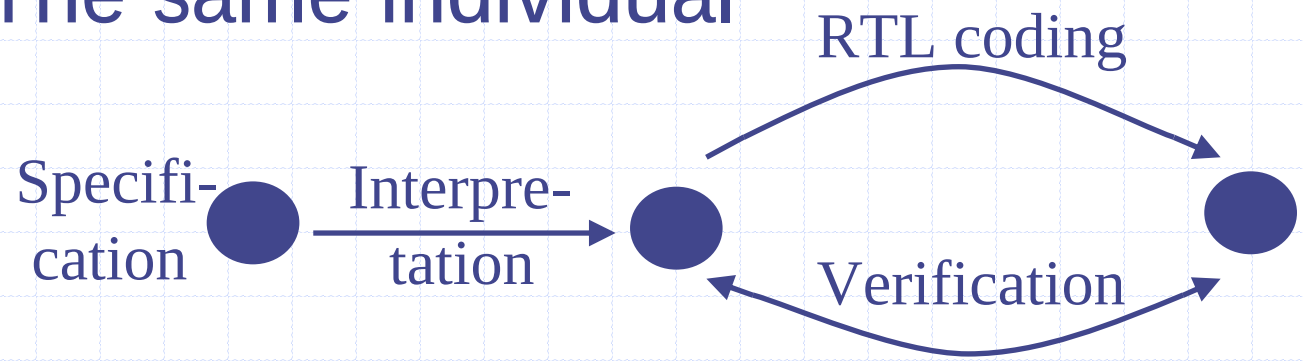
Reconvergence Model

- A conceptual representation of the verification process

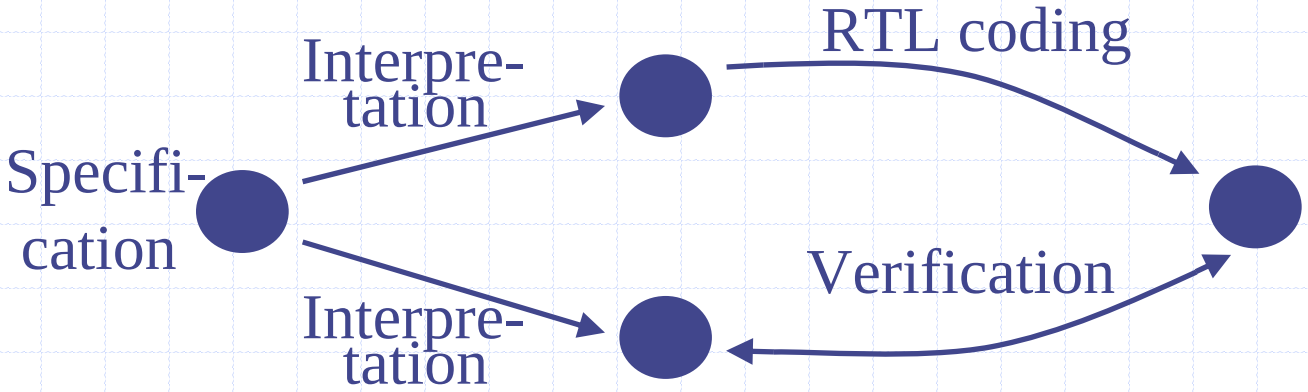


The Human Factor

- The same individual

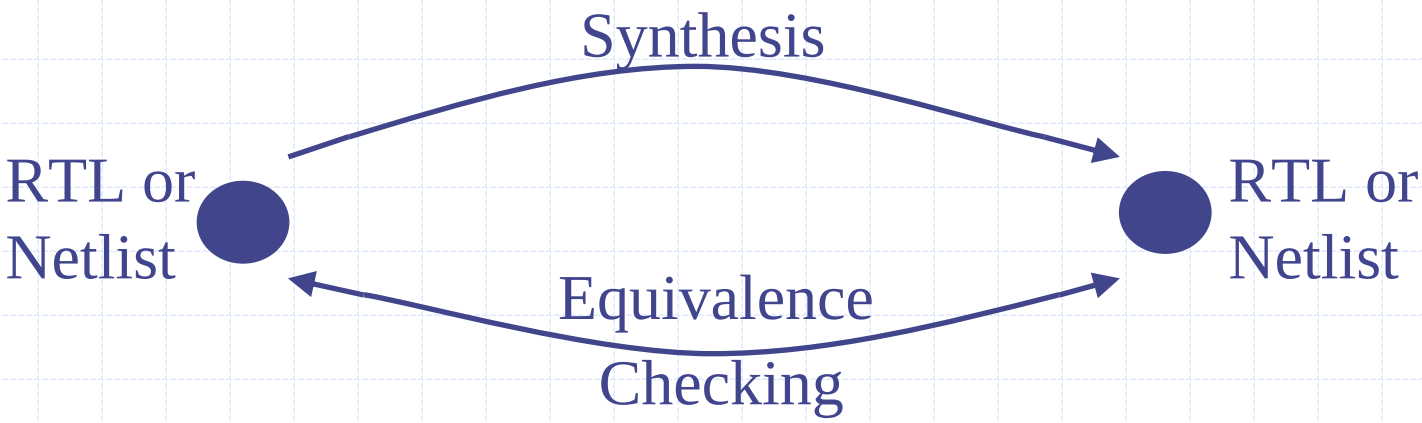


- Different individuals



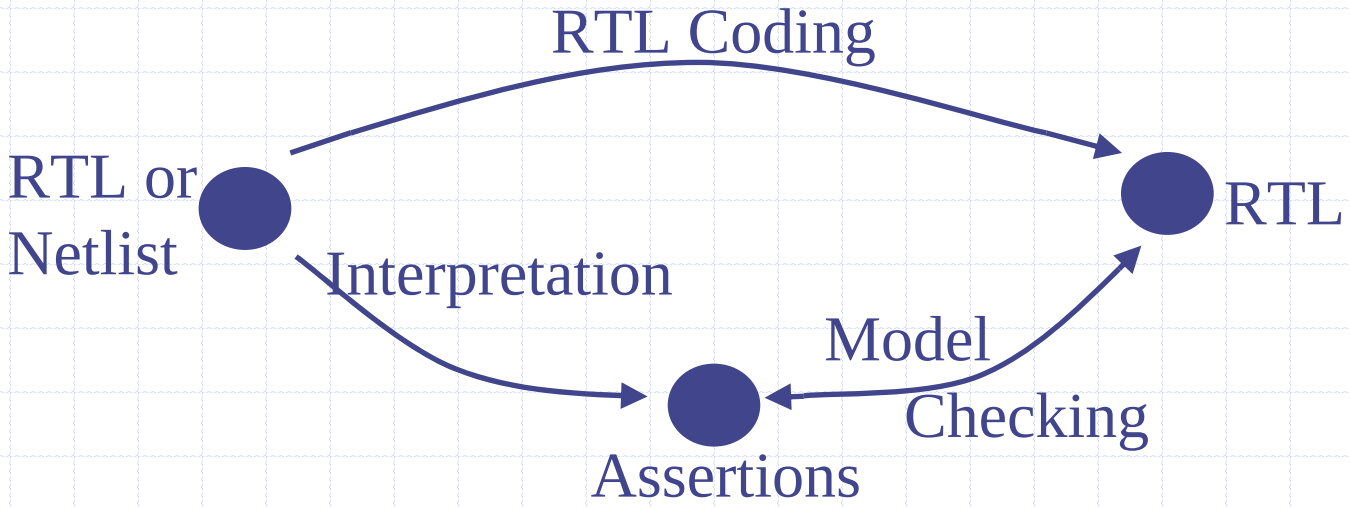
Equivalence Checking

- Compare two models
- Prove the origin and output are logically equivalent and the transformation preserved its functionality



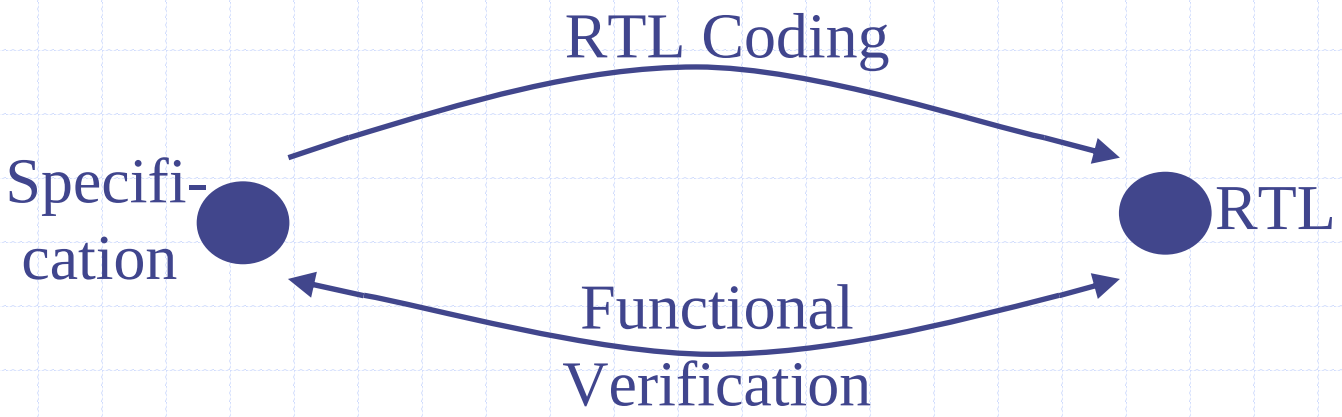
Model Checking

- Look for generic problems or violation of user-defined rules about the behavior or the design



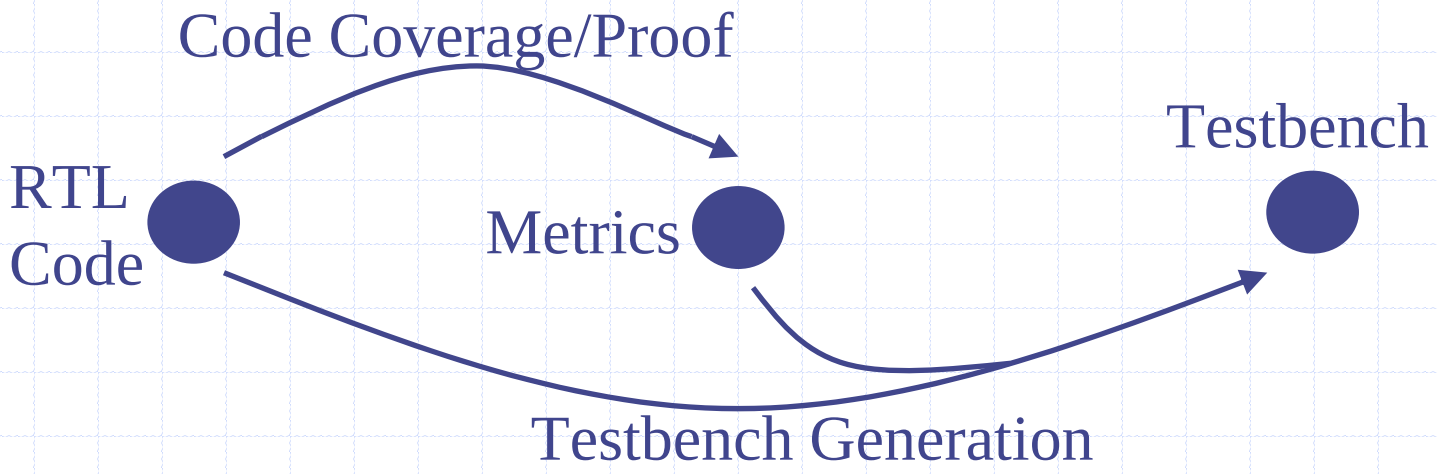
Functional Verification

- Ensure that a design implements intended functionality
- Show that a design meets the intent of its specification, but it cannot prove it



Testbench Generation

- Generate testbenches to either increase code coverage or to exercise the design to violate a property

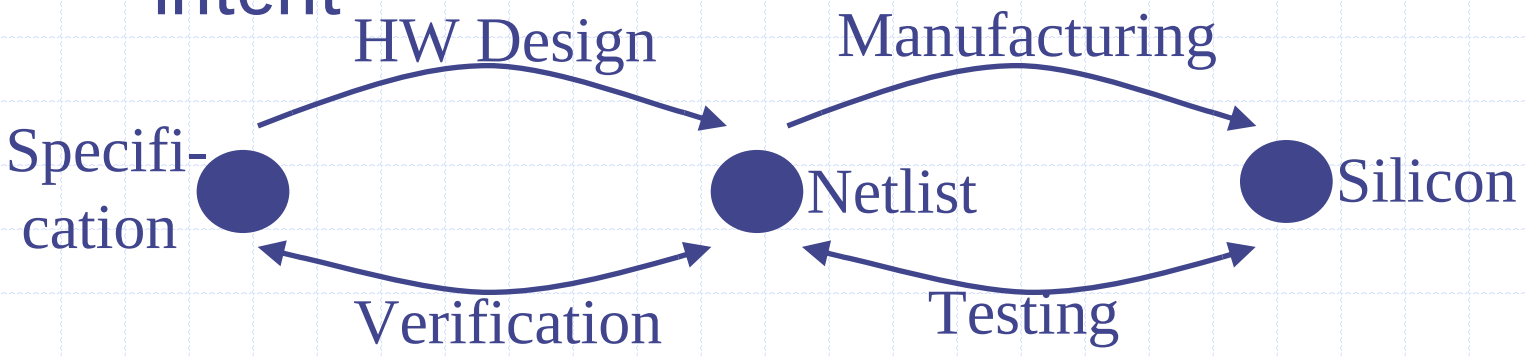


Functional Verification Approaches

- Black-box
  - Without any knowledge of the actual implementation of a design
- White-box
  - Has full visibility and controllability of the internal structure and implementation of the design being verified
- Grey-box
  - Controls and observes a design entirely through its top-level interfaces

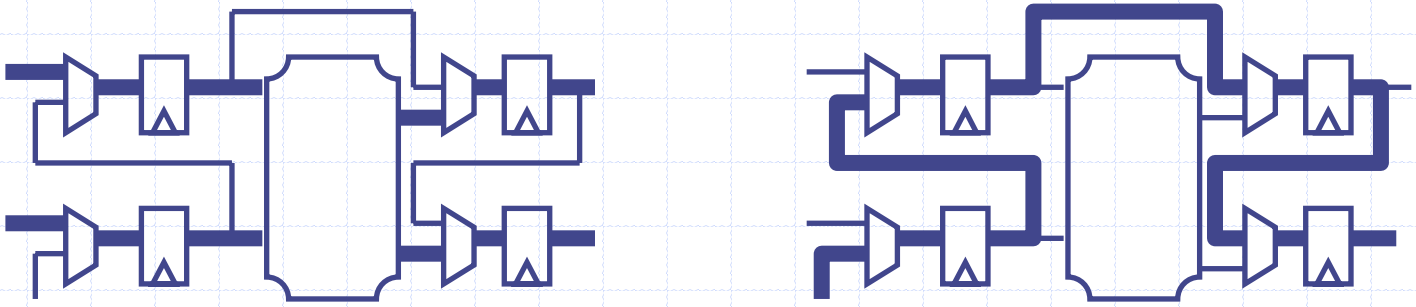
Testing vs Verification

- Testing
  - Verify that the design was manufactured correctly
- Verification
  - Ensure that a design meets its functional intent



Testing vs Verification

- Scan-based testing
  - All registers are hooked-up in a long serial chain.



- Design for verification
  - Addition design effort to simplify verification
  - Providing additional software-accessible registers to control and observe internal locations
  - Providing programmable multiplexors to isolate or by pass functional units

Verification and Design reuse

- Engineers do not trust that the other design is as good as reliable as one designed by themselves.
- Proper functional verification demonstrates trustworthiness of a design.

The cost of verification

- Is my design functionally correct?

	Errors	No Errors
Bad Design		Type Ⅱ (False Positive)
Good Design	Type Ⅰ (False Negative)	

- How much is enough?
- When will I be done?

Chapter 2
Verification Tools

教育部顧問室
「超大型積體電路與系統設計」教育改進計畫
EDA聯盟編製

Linting Tools

- Identify common mistakes programmer made, such as syntax errors
- Similar to spell checkers
- Only find problems that can be statically deduced by looking at the code structure, not problems in the algorithm or data flow.

Simulators

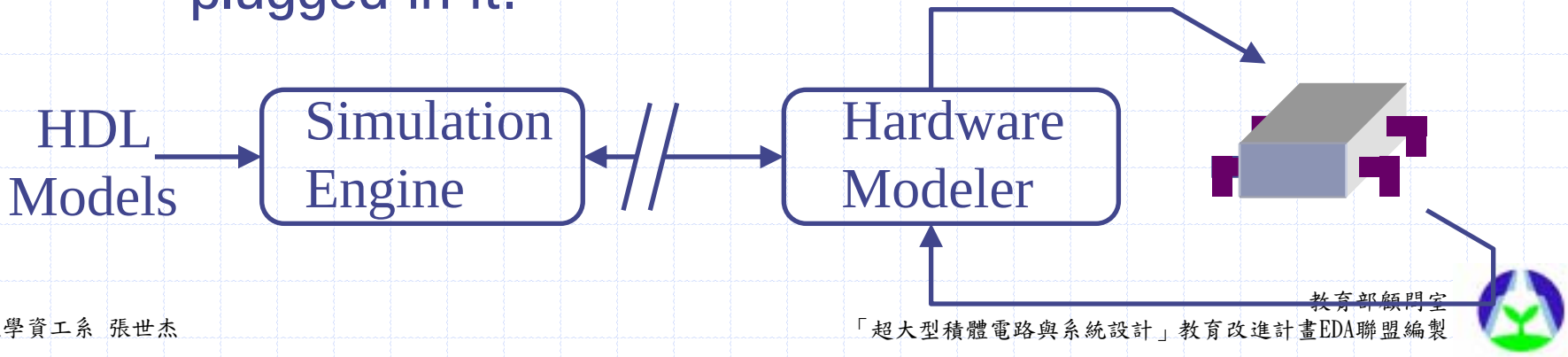
- An approximation of reality
- Not static tools (∵Simulation requires stimulus)
  - ⟷ Linting tools are static tools
- The simulation outputs are validated externally, against design intents.
- Co-simulators
  - Both simulators are running together, cooperating to simulate the entire design.

Simulators

- Event-driven simulation
  - Outputs change only when an input changes
  - Change in values, called ***events***, drive the simulation process
- Cycle-driven simulation
  - Has no timing information
  - Can only handle synchronous circuits

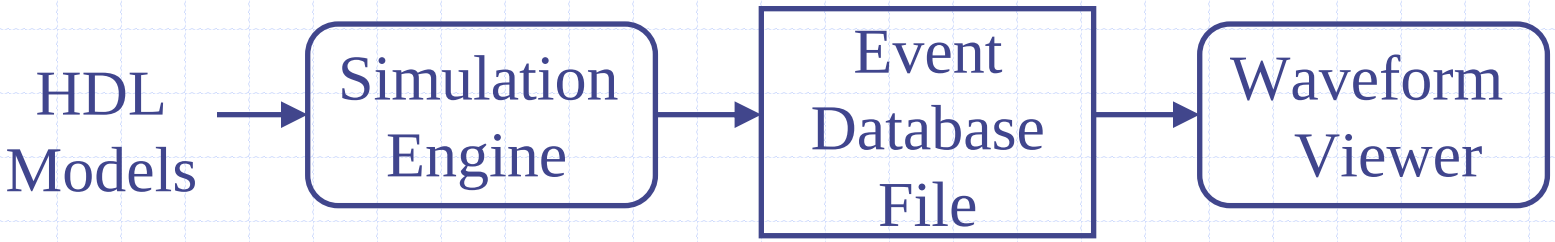
Third-Party Models

- It is cheaper to buy models than write them yourself.
- Your models is not as reliable as the one you buy.
- Hardware Modeler
  - A real physical chip that needs to be simulated is plugged in it.

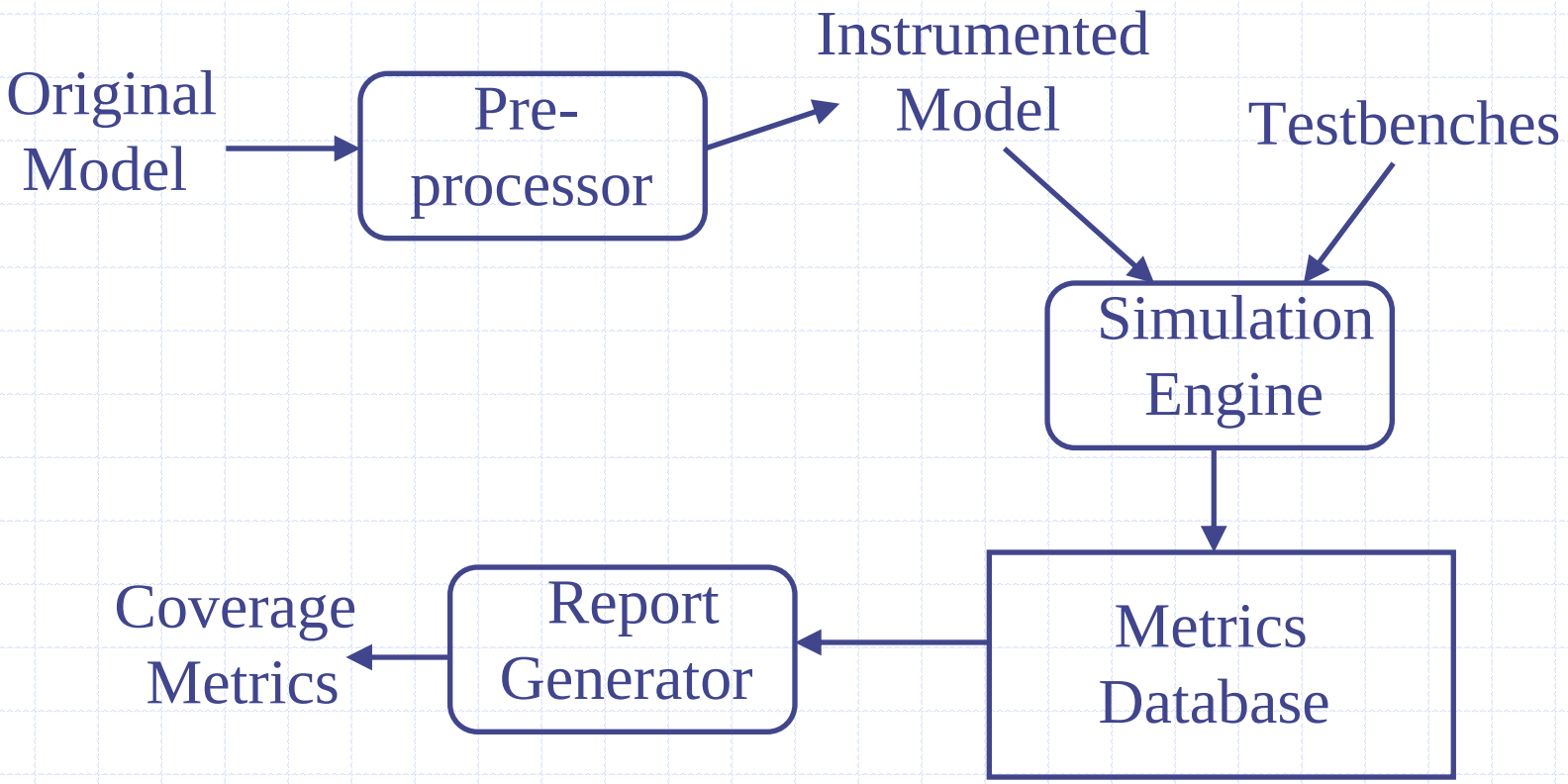


Waveform Viewers

- Display the changes in signal values over time
- Used to debug simulations
- Record trace information significantly reduce the performance of the simulator



Code coverage



Statement Coverage

- How much of the total lines of code were executed
- Ex:

```
if (parity == ODD || parity == EVEN) begin
        tx <= compute_parity(data,  parity);
        #(tx_time);
end
tx <= 1'b0;
#(tx_time);
if (stop_bits == 2) begin
        tx<=1'b0;
        #(tx_time);
end
```

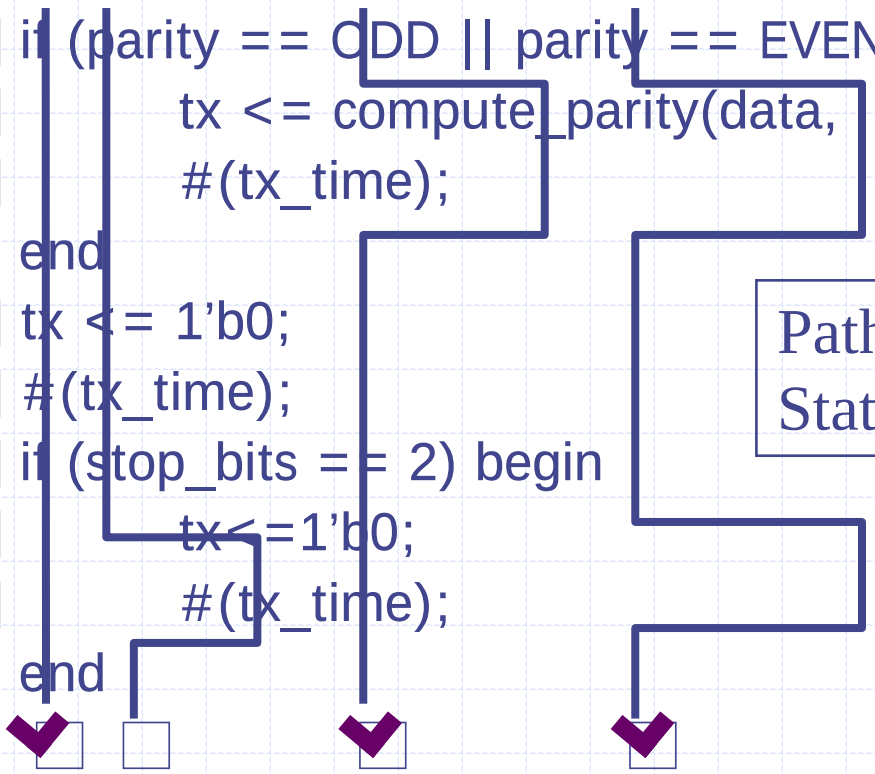
When parity!=ODD & Parity!=EVEN & Stop_bits=2

- Statement Coverage = 6/8 = 75%

Path Coverage

- All possible ways you can execute a sequence of statements
- Ex:

```
if (parity == ODD || parity == EVEN) begin
        tx <= compute_parity(data, parity);
        #(tx_time);
end
tx <= 1'b0;
#(tx_time);
if (stop_bits == 2) begin
        tx<=1'b0;
        #(tx_time);
end
```

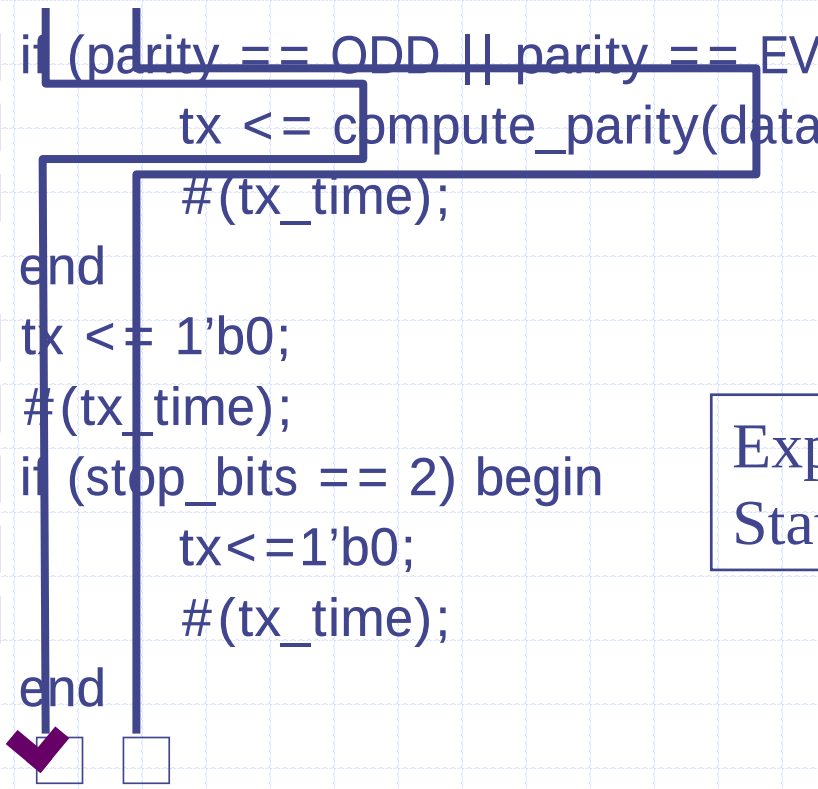


Path Coverage=75%
Statement Coverage=100%

Expression Coverage

- Measure the various ways paths through the code are executed
- Ex:

```
if (parity == ODD || parity == EVEN) begin
        tx <= compute_parity(data,  parity);
        #(tx_time);
end
tx <= 1'b0;
#(tx_time);
if (stop_bits == 2) begin
        tx<=1'b0;
        #(tx_time);
end
```



Expression Coverage=50%
Statement Coverage=100%

What Does 100 Percent Coverage Mean?

- Completeness does not imply correctness.
- Code coverage lets you know if you are not done.
- Some tools can help you reach 100% coverage.

Verification Language

- Verification languages can raise the level of abstraction.
- VHDL and Verilog are simulation languages, not verification languages.
- ***Specman*** from Verisity

  VERA from Synopsys

  Rave from Chronology

Revision Control

- Source Control Management Systems
  - Files must be centrally managed.
  - The history of a file is maintained.
- Configuration Management
  - Views need not be always composed of the latest version
  - Symbolic tags are attached to specific version of files
  - Ex: Submit, Bronze, Silver, Gold, To_Layout, To_Synthesis…etc
- Constantly updates their view to appropriate release.

Issue Tracking

- Issues:
  - Bugs
  - Ambiguities or incompleteness in the spec,
  - Architectural decisions and trade-offs
  - Errors
  - New relevant testcases
- What is an issue worth tracking?

Issue Tracking

- The Grapevine System
- The Post-It System
- The Procedural System
- Computerized System

Metrics

- Essential management tools.
- Best observed over time to see trends.
- Historical data should be used to create a baseline
- Can help assess the verification effort.

Metrics

- Code-Related Metrics
  - Code coverage
  - Number of lines of code
  - Ratio of lines of code
  - Source code changes
- Quality-Related Metrics
  - Number of known outstanding issues
  - Number of bugs found during its service life

Chapter 3
The Verification Plan

教育部顧問室
「超大型積體電路與系統設計」教育改進計畫
EDA聯盟編製

The Role of the Verification Plan

- Specifying the verification
  - Schedule
  - The verification plan is the specification document for the verification effort.
- Defining first-time success
  - Ensure all essential features are appropriately verified.

Levels of Verification

- Unit-Level Verification
- Reuseable Components Verification
- ASIC and FPGA Verification
- System-Level Verification
- Board-Level Verification

Verification Strategies

- Decide
  - Type of testcases (White-box or Black-box)
  - Level of abstraction (Cycle level or Device driver level)
- Verifying the response
  - How to check the response
- Random Verification
  - System-level verification
  - Create unexpected conditions or hit corner cases

From Specification to Features

- Identify features
- Label each features

- Features
  - Component-level features
  - System-level features

From Features to Testcases

- Prioritize
  - must-have
  - should-have
  - nice-to-have
- Group into testcases
  - Features should be grouped together and assigned to the same verification engineer.
- Design for verification
  - Modify the design to aid verification

From Testcases to Testbenches

- Group into testbenches
  - Each group of testcases is divided into testbenches.
- Verify testbenches