## Two's Complement Multiplication

$$
\begin{array}{ccccccccccc}
 & & & & & & -a_4 & a_3 & a_2 & a_1 & a_0 \\
 & & & & & \text{x} & -x_4 & x_3 & x_2 & x_1 & x_0 \\
\cline{6-11}
 & & & & & & -a_4x_0 & a_3x_0 & a_2x_0 & a_1x_0 & a_0x_0 \\
+ & & & & & -a_4x_1 & a_3x_1 & a_2x_1 & a_1x_1 & a_0x_1 & \\
 & & & & -a_4x_2 & a_3x_2 & a_2x_2 & a_1x_2 & a_0x_2 & & \\
 & & & -a_4x_3 & a_3x_3 & a_2x_3 & a_1x_3 & a_0x_3 & & & \\
 & \mathbf{0} & a_4x_4 & -a_3x_4 & -a_2x_4 & -a_1x_4 & -a_0x_4 & & & & \\
\hline
 & \mathbf{-p_9} & p_8 & p_7 & p_6 & p_5 & p_4 & p_3 & p_2 & p_1 & p_0 \\
 & \mathbf{2^9} & 2^8 & 2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0
\end{array}
$$

## SD number in accumulating partial products

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $\mathbf{-p_9}$ | $p_8$ | $p_7$ | $p_6$ | $p_5$ | $p_4$ | $p_3$ | $p_2$ | $p_1$ | $p_0$ |
| $\mathbf{2^9}$ | $2^8$ | $2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ |

$$\Updownarrow \equiv$$

## Two's complement number in result

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $\mathbf{p_9}$ | $p_8$ | $p_7$ | $p_6$ | $p_5$ | $p_4$ | $p_3$ | $p_2$ | $p_1$ | $p_0$ |
| $\mathbf{-2^9}$ | $2^8$ | $2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ |

## Baugh-Wooley Two's Complement Multiplier

$$
\begin{array}{ccccccccccc}
 & & & & & & -a_4 & a_3 & a_2 & a_1 & a_0 \\
 & & & & & \times & -x_4 & x_3 & x_2 & x_1 & x_0 \\
\hline
 & & & & & & -a_4x_0 & a_3x_0 & a_2x_0 & a_1x_0 & a_0x_0 \\
 & & & & & -a_4x_1 & a_3x_1 & a_2x_1 & a_1x_1 & a_0x_1 & \\
+ & & & & -a_4x_2 & a_3x_2 & a_2x_2 & a_1x_2 & a_0x_2 & & \\
 & & & -a_4x_3 & a_3x_3 & a_2x_3 & a_1x_3 & a_0x_3 & & & \\
 & & a_4x_4 & -a_3x_4 & -a_2x_4 & -a_1x_4 & -a_0x_4 & & & & \\
\hline
 & -\mathbf{p_9} & p_8 & p_7 & p_6 & p_5 & p_4 & p_3 & p_2 & p_1 & p_0 \\
\hline
 & \mathbf{p_9} & p_8 & p_7 & p_6 & p_5 & p_4 & p_3 & p_2 & p_1 & p_0
\end{array}
$$

## Baugh-Wooley Two's Complement Multiplier

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | $-a_4$ | $a_3$ | $a_2$ | $a_1$ | $a_0$ |
| | | | | | x | $-x_4$ | $x_3$ | $x_2$ | $x_1$ | $x_0$ |
| | | | | | | $a_4\overline{x_0}$ | $a_3x_0$ | $a_2x_0$ | $a_1x_0$ | $a_0x_0$ |
| + | | | | | $a_4\overline{x_1}$ | $a_3x_1$ | $a_2x_1$ | $a_1x_1$ | $a_0x_1$ | |
| | | | | $a_4\overline{x_2}$ | $a_3x_2$ | $a_2x_2$ | $a_1x_2$ | $a_0x_2$ | | |
| | | | $a_4\overline{x_3}$ | $a_3x_3$ | $a_2x_3$ | $a_1x_3$ | $a_0x_3$ | | | |
| | | $a_4x_4$ | $\overline{a_3}x_4$ | $\overline{a_2}x_4$ | $\overline{a_1}x_4$ | $\overline{a_0}x_4$ | | | | |
| | | $\overline{a_4}$ | $a_4 - a_4$ | $a_4 - a_4$ | $a_4 - a_4$ | $a_4$ | | | | |
| | $-1$ | $\overline{x_4}$ | $x_4 - x_4$ | $x_4 - x_4$ | $x_4 - x_4$ | $x_4$ | | | | |
| | $\mathbf{-p_9}$ | $p_8$ | $p_7$ | $p_6$ | $p_5$ | $p_4$ | $p_3$ | $p_2$ | $p_1$ | $p_0$ |

## Baugh-Wooley Multiplier

$$\begin{array}{ccccccccccc}
 & & & & & & -a_4 & a_3 & a_2 & a_1 & a_0 \\
 & & & & & \times & -x_4 & x_3 & x_2 & x_1 & x_0 \\
\hline
 & & & & & & a_4\overline{x_0} & a_3x_0 & a_2x_0 & a_1x_0 & a_0x_0 \\
 & & & & & a_4\overline{x_1} & a_3x_1 & a_2x_1 & a_1x_1 & a_0x_1 & \\
+ & & & & a_4\overline{x_2} & a_3x_2 & a_2x_2 & a_1x_2 & a_0x_2 & & \\
 & & & a_4\overline{x_3} & a_3x_3 & a_2x_3 & a_1x_3 & a_0x_3 & & & \\
 & & a_4x_4 & \overline{a_3}x_4 & \overline{a_2}x_4 & \overline{a_1}x_4 & \overline{a_0}x_4 & & & & \\
 & & \overline{a_4} & & & & a_4 & & & & \\
 & 1 & \overline{x_4} & & & & x_4 & & & & \\
\hline
 & \mathbf{p_9} & p_8 & p_7 & p_6 & p_5 & p_4 & p_3 & p_2 & p_1 & p_0
\end{array}$$

## Modified Baugh-Wooley Multiplier

$$
\begin{array}{ccccccccccc}
 & & & & & & -a_4 & a_3 & a_2 & a_1 & a_0 \\
 & & & & & \mathbf{x} & -x_4 & x_3 & x_2 & x_1 & x_0 \\
\hline
 & & & & & -1 & \overline{a_4x_0} & a_3x_0 & a_2x_0 & a_1x_0 & a_0x_0 \\
+ & & & & -1 & \overline{a_4x_1} & a_3x_1 & a_2x_1 & a_1x_1 & a_0x_1 & \\
 & & & -1 & \overline{a_4x_2} & a_3x_2 & a_2x_2 & a_1x_2 & a_0x_2 & & \\
 & & -1 & \overline{a_4x_3} & a_3x_3 & a_2x_3 & a_1x_3 & a_0x_3 & & & \\
 & & a_4x_4 & \overline{a_3x_4} & \overline{a_2x_4} & \overline{a_1x_4} & \overline{a_0x_4} & & & & \\
\hline
 & -\mathbf{p_9} & p_8 & p_7 & p_6 & p_5 & p_4 & p_3 & p_2 & p_1 & p_0
\end{array}
$$

## Modified Baugh-Wooley Multiplier

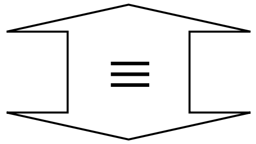
$$
\begin{array}{ccccccccccc}
 & & & & & & -a_4 & a_3 & a_2 & a_1 & a_0 \\
 & & & & & \times & -x_4 & x_3 & x_2 & x_1 & x_0 \\
\hline
 & & & & -1 & +1 & \overline{a_4x_0} & a_3x_0 & a_2x_0 & a_1x_0 & a_0x_0 \\
+ & & & & -1 & \overline{a_4x_1} & a_3x_1 & a_2x_1 & a_1x_1 & a_0x_1 & \\
 & & & -1 & \overline{a_4x_2} & a_3x_2 & a_2x_2 & a_1x_2 & a_0x_2 & & \\
 & & -1 & \overline{a_4x_3} & a_3x_3 & a_2x_3 & a_1x_3 & a_0x_3 & & & \\
 & & a_4x_4 & \overline{a_3x_4} & \overline{a_2x_4} & \overline{a_1x_4} & \overline{a_0x_4} & & & & \\
\hline
 & -\mathbf{p_9} & p_8 & p_7 & p_6 & p_5 & p_4 & p_3 & p_2 & p_1 & p_0
\end{array}
$$

# Modified Baugh-Wooley Multiplier

$$
\begin{array}{ccccccccccc}
 & & & & & & -a_4 & a_3 & a_2 & a_1 & a_0 \\
 & & & & & \mathbf{x} & -x_4 & x_3 & x_2 & x_1 & x_0 \\
\hline
 & & & & & +1 & \overline{a_4x_0} & a_3x_0 & a_2x_0 & a_1x_0 & a_0x_0 \\
+ & & & & & \overline{a_4x_1} & a_3x_1 & a_2x_1 & a_1x_1 & a_0x_1 & \\
 & & & & \overline{a_4x_2} & a_3x_2 & a_2x_2 & a_1x_2 & a_0x_2 & & \\
 & & & \overline{a_4x_3} & a_3x_3 & a_2x_3 & a_1x_3 & a_0x_3 & & & \\
 & -1 & a_4x_4 & \overline{a_3x_4} & \overline{a_2x_4} & \overline{a_1x_4} & \overline{a_0x_4} & & & & \\
\hline
 & \mathbf{-p_9} & p_8 & p_7 & p_6 & p_5 & p_4 & p_3 & p_2 & p_1 & p_0
\end{array}
$$

## SD number in accumulating partial products

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $-\mathbf{p_9}$ | $p_8$ | $p_7$ | $p_6$ | $p_5$ | $p_4$ | $p_3$ | $p_2$ | $p_1$ | $p_0$ |
| $\mathbf{2^9}$ | $2^8$ | $2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ |



## Two's complement number in result

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $\mathbf{p_9}$ | $p_8$ | $p_7$ | $p_6$ | $p_5$ | $p_4$ | $p_3$ | $p_2$ | $p_1$ | $p_0$ |
| $\mathbf{-2^9}$ | $2^8$ | $2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ |

## Sign-bit Conversion from SD to 2′s complement

| | | |
|---|---|---|
| $\begin{array}{rll} & -1 & a_4x_4 \\ + & c & \\ \hline & -(1-C) & p_8 \end{array}$ | $\begin{array}{rll} & 1 & a_4x_4 \\ + & -c & \\ \hline & (1-C) & p_8 \end{array}$ | $\begin{array}{rll} & 1 & a_4x_4 \\ + & c & \\ \hline & (1+C) & p_8 \end{array}$ |
| $\begin{array}{rll} & -1 & a_4x_4 \\ + & c & \\ \hline & -\mathbf{p_9} & p_8 \end{array}$ | $\begin{array}{rll} & 1 & a_4x_4 \\ + & -c & \\ \hline & \mathbf{p_9} & p_8 \end{array}$ | $\begin{array}{rll} & 1 & a_4x_4 \\ + & c & \\ \hline & \mathbf{p_9} & p_8 \end{array}$ |
| A: SD | B: 2's complement | C: 2's complement |
| $\mathbf{p_9 = (1-C)}$ | | $\mathbf{p_9 = (1+C)}$ |

## Modified Baugh-Wooley Multiplier: A

$$
\begin{array}{ccccccccccc}
 & & & & & & -a_4 & a_3 & a_2 & a_1 & a_0 \\
 & & & & & \mathbf{x} & -x_4 & x_3 & x_2 & x_1 & x_0 \\
\cline{6-11}
 & & & & & +1 & \overline{a_4x_0} & a_3x_0 & a_2x_0 & a_1x_0 & a_0x_0 \\
\mathbf{+} & & & & & \overline{a_4x_1} & a_3x_1 & a_2x_1 & a_1x_1 & a_0x_1 & \\
 & & & & \overline{a_4x_2} & a_3x_2 & a_2x_2 & a_1x_2 & a_0x_2 & & \\
 & & & \overline{a_4x_3} & a_3x_3 & a_2x_3 & a_1x_3 & a_0x_3 & & & \\
 & -1 & a_4x_4 & \overline{a_3x_4} & \overline{a_2x_4} & \overline{a_1x_4} & \overline{a_0x_4} & & & & \\
\hline
 & \mathbf{-p_9} & p_8 & p_7 & p_6 & p_5 & p_4 & p_3 & p_2 & p_1 & p_0
\end{array}
$$

## Example

**(-16) x (-16) = 256**

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | -1 | 0 | 0 | 0 | 0 |
| x | | | | | | -1 | 0 | 0 | 0 | 0 |
| | | | | | 1 | 1 | 0 | 0 | 0 | 0 |
| | | | | | 1 | 0 | 0 | 0 | 0 | |
| + | | | | 1 | 0 | 0 | 0 | 0 | | |
| | | | 1 | 0 | 0 | 0 | 0 | | | |
| | -1 | 1 | 1 | 1 | 1 | 1 | | | | |
| | 1 | carry | | | | | | | | |
| | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | $-p_9$ | $p_8$ | $p_7$ | $p_6$ | $p_5$ | $p_4$ | $p_3$ | $p_2$ | $p_1$ | $p_0$ |

## Modified Baugh-Wooley Multiplier: B,C

$$
\begin{array}{ccccccccccc}
 & & & & & & -a_4 & a_3 & a_2 & a_1 & a_0 \\
 & & & & & \mathbf{x} & -x_4 & x_3 & x_2 & x_1 & x_0 \\
\hline
 & & & & & 1 & \overline{a_4x_0} & a_3x_0 & a_2x_0 & a_1x_0 & a_0x_0 \\
+ & & & & & \overline{a_4x_1} & a_3x_1 & a_2x_1 & a_1x_1 & a_0x_1 & \\
 & & & & \overline{a_4x_2} & a_3x_2 & a_2x_2 & a_1x_2 & a_0x_2 & & \\
 & & & \overline{a_4x_3} & a_3x_3 & a_2x_3 & a_1x_3 & a_0x_3 & & & \\
 & 1 & a_4x_4 & \overline{a_3x_4} & \overline{a_2x_4} & \overline{a_1x_4} & \overline{a_0x_4} & & & & \\
\hline
 & \mathbf{p_9} & p_8 & p_7 & p_6 & p_5 & p_4 & p_3 & p_2 & p_1 & p_0
\end{array}
$$

## Example: B

**(-16) x (-16) = 256**

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | -1 | 0 | 0 | 0 | 0 |
| x | | | | | | -1 | 0 | 0 | 0 | 0 |
| | | | | | 1 | 1 | 0 | 0 | 0 | 0 |
| | | | | | 1 | 0 | 0 | 0 | 0 | |
| + | | | | 1 | 0 | 0 | 0 | 0 | | |
| | | | 1 | 0 | 0 | 0 | 0 | | | |
| | 1 | 1 | 1 | 1 | 1 | 1 | | | | |
| | -1 | carry | | | | | | | | |
| | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | **$p_9$** | $p_8$ | $p_7$ | $p_6$ | $p_5$ | $p_4$ | $p_3$ | $p_2$ | $p_1$ | $p_0$ |

## Example: C

**(-16) x (-16) = 256**

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | -1 | 0 | 0 | 0 | 0 |
| | | | | | x | -1 | 0 | 0 | 0 | 0 |
| | | | | | 1 | 1 | 0 | 0 | 0 | 0 |
| | | | | | 1 | 0 | 0 | 0 | 0 | |
| + | | | | 1 | 0 | 0 | 0 | 0 | | |
| | | | 1 | 0 | 0 | 0 | 0 | | | |
| | 1 | 1 | 1 | 1 | 1 | 1 | | | | |
| | 1 | **carry** | | | | | | | | |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | $\mathbf{p_9}$ | $p_8$ | $p_7$ | $p_6$ | $p_5$ | $p_4$ | $p_3$ | $p_2$ | $p_1$ | $p_0$ |

# Chapter 13: Verification

Prof. Soo-Ik Chae

## 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

## Verification

- The goal of verification is to ensure a module 100% correct in its functionality and timing.
- On average, design teams usually spend 50 ~ 70% of their time to verify their designs.
- Functional verification only considers if the logic function of the design meets the specifications.
  - simulation
  - formal proof
- Timing verification considers whether the design meets the timing constraints.
  - dynamic timing simulation
  - static timing analysis

## Functional Verification

- Simulated-based functional verification
  - The design is placed under a test bench.
  - Input stimuli are applied to the design.
  - The outputs from the design are compared with the reference outputs.
- Formal verification
  - A protocol, an assertion, a property, or a design rule are proved to hold for all possible cases in the design.

## Design Models

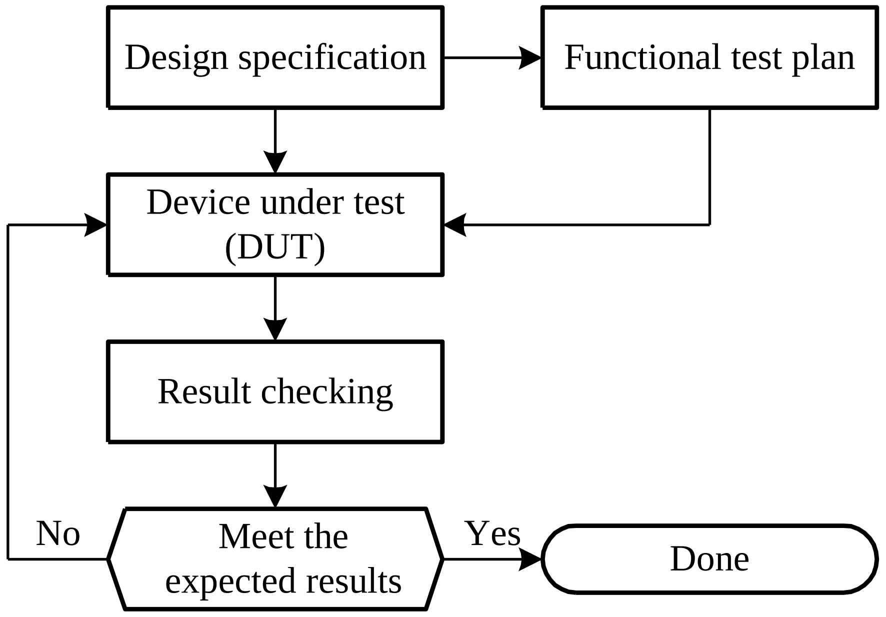
- Black box model
  - Only the external interfaces (namely, the input and output behavior of the design) are known.
  - The internal signals and constructs are unknown (namely, black).
  - Most simulation-based verifications begin with this model.
- White box model
  - Both the external interfaces and internal structures are known.
  - Most formal verification environments use this model.
- Gray box model
  - This model is a combination of both black box and white box.
  - Some of the internal signals in addition to the external interfaces are known.
  - Most simulation-based verification environments use this model.

## Assertion-Based Verification

- Types of assertions:
  - Static assertion: A static assertion is an atomic and simple check for the absence of an event.
  - Temporal assertion: Several events occur in sequence and many events have to occur before the final asserted event can be checked.

## Simulation-Based Verification

- Simulation-based verification
  - Test signals are applied to the DUT.
  - The results are stored and analyzed.
  - If the result checking and code coverage analysis meet the expected results, then we have done it.



## Hierarchy of Functional Verification

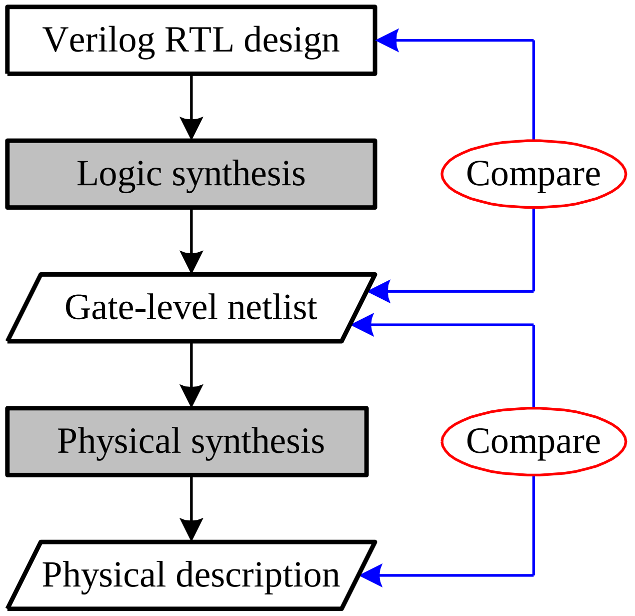
- Designer level (or block-level)
  - Verilog HDL or VHDL is used for both design and verification.
- Unit level
  - Randomized stimuli and autonomous checking are applied.
- Core level
  - A well-defined process coupled with well-documented specification are applied.
- Chip level
  - Ensuring that all units are properly connected and the design adheres to the interface protocols of all units.

## 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

- Formal verification
  - It uses mathematical techniques to prove an assertion or a property of the design.
  - It proves a design property by exploring all possible ways to manipulate the design.
  - It can prove the correctness of a design without doing simulation.



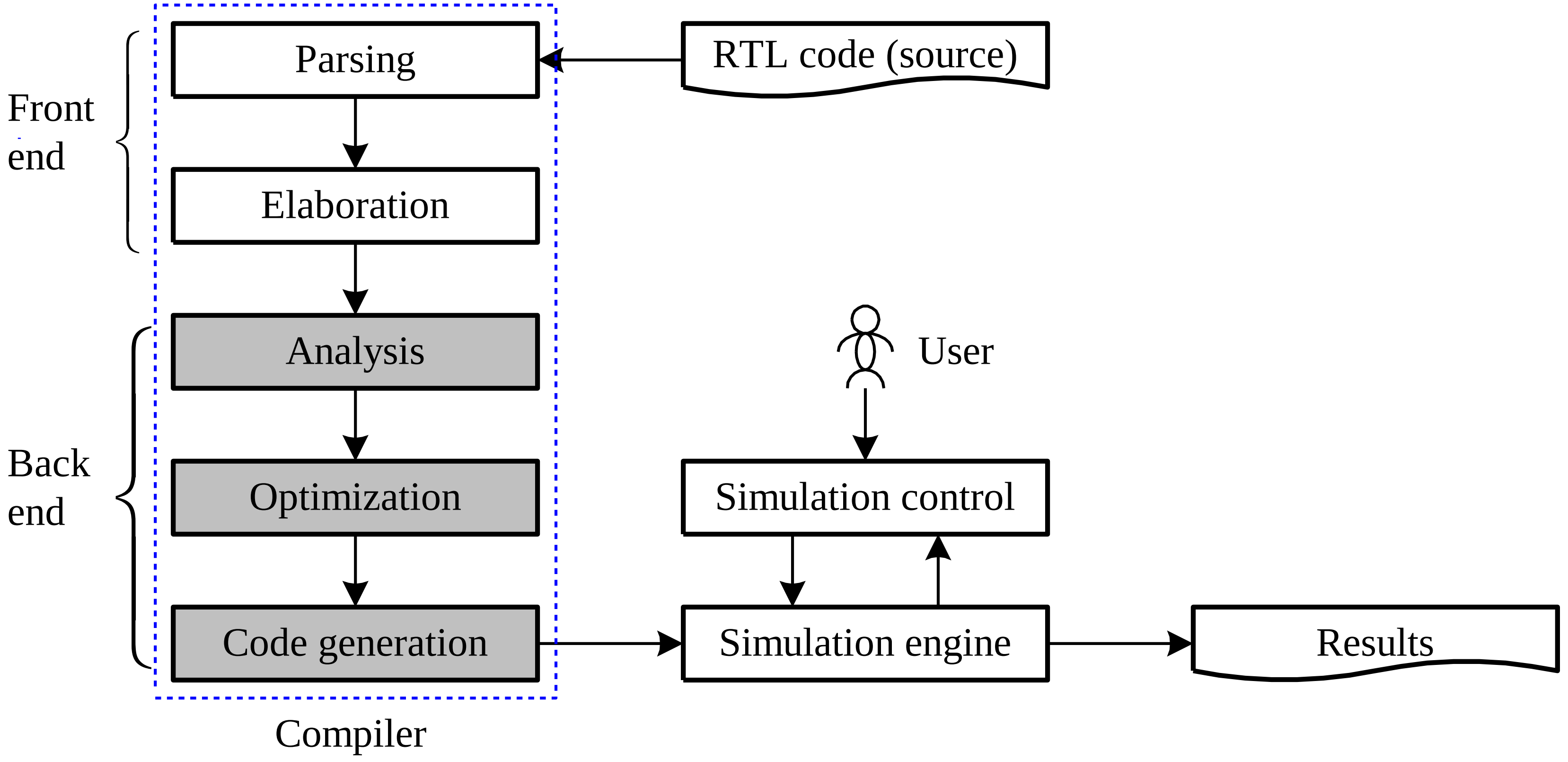
## Simulations

- Types of simulations
  - Behavioral simulation
  - Functional simulation
  - Gate-level (logic) simulation
  - Switch-level simulation
  - Circuit-level (transistor-level) simulation

## Variations of Simulations

- Software simulation
  - It is typically used to run Verilog HDL-based designs.
  - Software simulations consume large amount of time.
- Hardware acceleration
  - It is used to speed up existing simulation.
  - It can accelerate simulations by two to three orders of magnitude.
- Hardware emulation
  - It is used to verify the design in a real-world environment.
  - It is used to assert the design is stable enough.

## An Architecture of HDL Simulators



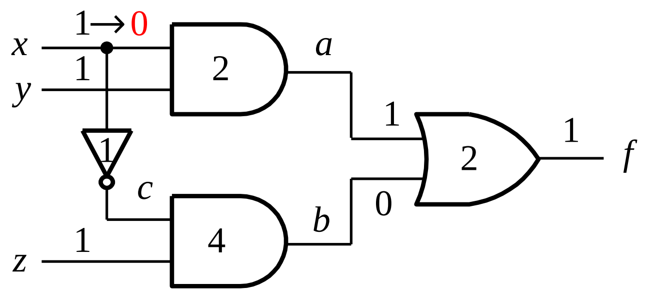
## Types of Software Simulators

- Interpreted simulators
  - They run the simulation interpretively.
  - For example, Cadence Verilog-XL simulator.
- Compiled code simulators
  - They convert the source code to an equivalent C code, then compile and run the C code.
  - For example, Synopsys VCS simulator.
- Native code simulators
  - They convert the source code directly to binary code for a specific machine platform.
  - For example, Cadence Verilog-NC simulator.

## Software Simulators

- Event-driven simulators process elements in the design only when signals at the inputs of these elements change.
  - They process all elements in the design, irrespective of changes in signals.
- Cycle-based simulators work on a cycle-by-cycle basis.
  - They collapse combinational logic into equations.
  - They are useful for synchronous designs where operations happen only at active clock edges.
  - Timing information between two clock edges is lost.
  - Most cycle-based simulators are integrated with an event-driven simulator.

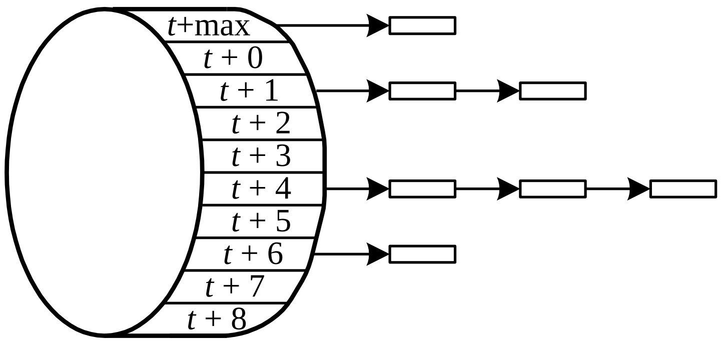
## An Event-Driven Simulation



(a) A circuit example

| | Scheduled events | Activity list |
|---|---|---|
| *t* = 0 | *x* = 0 | *a*, *c* |
| 1 | *c* = 1 | *b* |
| 2 | *a* = 0 | *f* |
| 3 | | |
| 4 | *f* = 0 | |
| 5 | *b* = 1 | *f* |
| 6 | | |
| 7 | *f* = 1 | |

(c) Scheduled events and the activity list



(b) Timing wheel

## Test Bench Design Principles

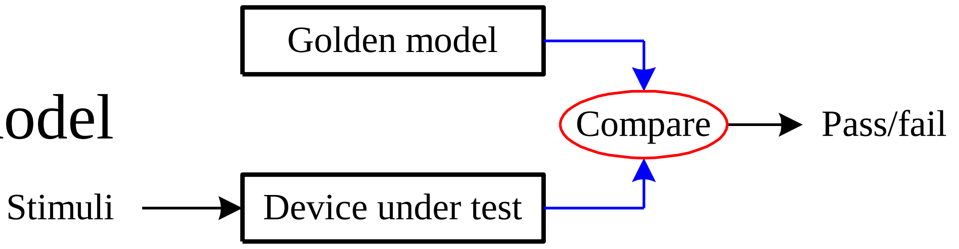
- The test bench should
  - generate stimuli.
  - check responses in terms of test cases,
  - employ reusable verification components.
- Two types of test benches: ?
  - deterministic:  verify basic functions in an early stage
  - self-checking: automate the tedious result checking process
- Options of choosing test vectors:
  - Exhaustive test
  - Random test
  - Verification vector files

## Test Bench Design Principles

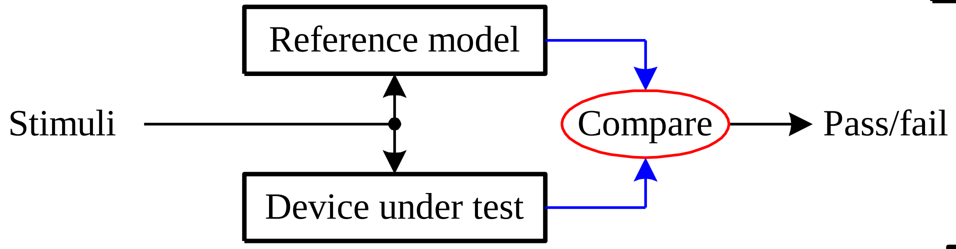
- Two basic choices of stimulus generation are:
  - Deterministic versus random stimulus generation
  - Pregenerated test case versus on-the-fly test case generation
- Types of result checking:
  - on-the-fly checking
  - end-of-test checking
- Result analysis:
  - Waveform viewers
  - Log files

## Test Bench Design Principles

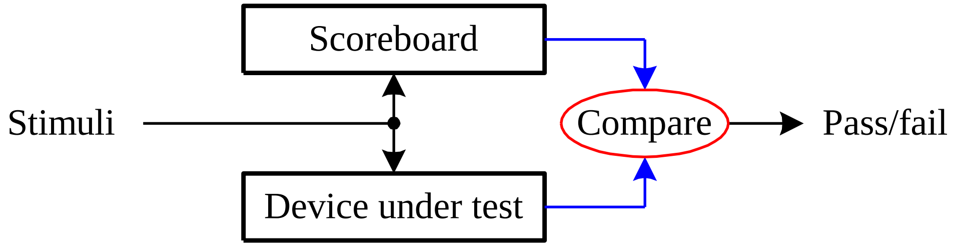
- Types of automated response checking:
  - Golden vectors: known outputs
  - Reference model
  - Transaction-based model



(a) Golden vectors



(b) Reference model



(c) Transaction-based model

## Test Bench Design Principles

- Guidelines
  - The time unit set in timescale must be matched with the actual propagation delay of gate-level circuitry.
  - Set reset signal properly, especially, the time interval of the reset signal must be large enough; otherwise, the initial operation of the gate-level simulation may not work properly.
- Coding style:
  - All response checking should be done automatically.

## Test Bench Designs --- A Trivial Example

```
// test bench design example 1: exhaustive test.
`timescale 1 ns / 100 ps
module nbit_adder_for_tb;
parameter n = 4;
reg  [n-1:0] x, y;
reg              c_in;
wire [n-1:0] sum;
wire c_out;
// Unit Under Test port map
     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
initial    #1280 $finish;
initial    $monitor($realtime,“ns %h %h %h %h", x, y, c_in, {c_out, sum});
endmodule
```

## Test Bench Designs --- A Trivial Example

```
// test bench design example 2: Random test.
`timescale 1 ns / 100 ps
module nbit_adder_for_tb1;
parameter n = 4;
reg  [n-1:0] x, y;
reg          c_in;
wire [n-1:0] sum;
wire c_out;
// 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: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
initial   #200 $finish;
initial   $monitor($realtime,"ns %h %h %h %h", x, y, c_in, {c_out, sum});
endmodule
```

## Test Bench Designs --- A Trivial Example

```
// test bench design example 3: Using Verification vector files.
`timescale 1 ns / 100 ps
module nbit_adder_for_tb2;
//Internal signals declarations:
parameter n = 4;
parameter m = 8;
reg   [n-1:0] x, y;
reg            c_in;
wire [n-1:0] sum;
wire c_out;
// 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);
end
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;
initial
   $monitor($realtime,“ns %h %h %h %h", x, y, c_in, {c_out, sum});
endmodule
```

| inputx.txt | inputy.txt | sum.txt |
|---|---|---|
| 4 | 1 | 05 |
| 9 | 3 | 0c |
| d | d | 1a |
| 5 | 2 | 07 |
| 1 | d | 0e |
| 6 | d | 13 |
| d | c | 19 |
| 9 | 6 | 0f |

## Coverage Analysis

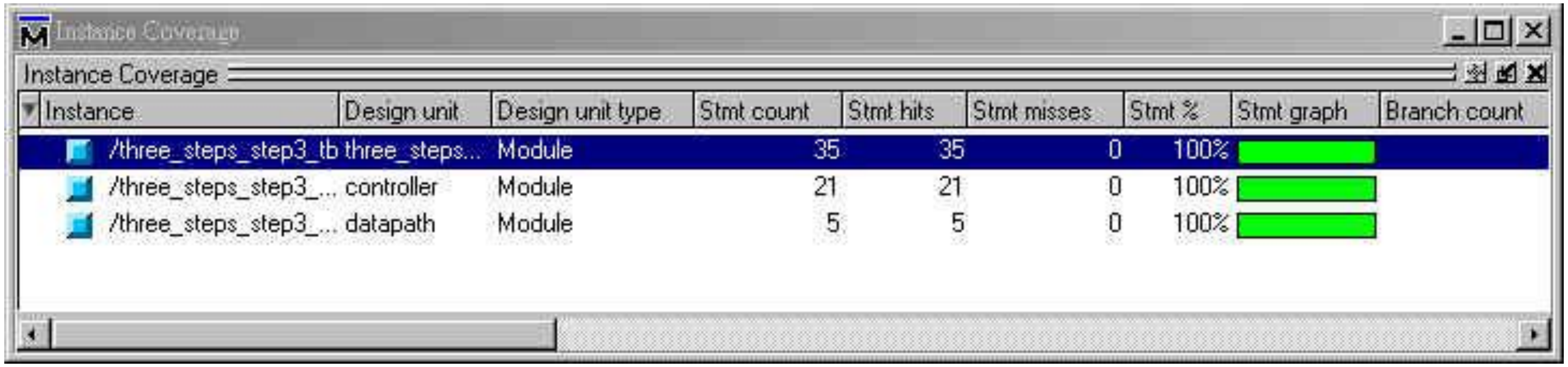
- Two major types verification coverage:
  - Structural coverage denotes the representation of the design to be covered.
  - Functional coverage means the semantics of the design implementation to be covered.
- What does 100% functional coverage mean?
  - You have covered all the coverage points you included in the simulation.
  - Functional coverage let you know if you are done.
  - A high coverage number is by no means an indication that the job is done.

## Structural (or code) Coverage

- Structural coverage:
  - statement coverage
  - branch or condition coverage: all branch sub-conditions
  - toggle coverage: signals
  - trigger coverage: signals in the sensitivity list of always block
  - expression coverage: similar to condition coverage, but covers signal assignments instead of branch decision
  - path coverage: paths
  - finite-state machine coverage: state coverage and transition coverage

## Functional Coverage

- Functional coverage
  - Item coverage
  - Cross coverage
  - Transition coverage
- Comments:
  - The quality of coverage analysis strongly depends on how well the test bench is.



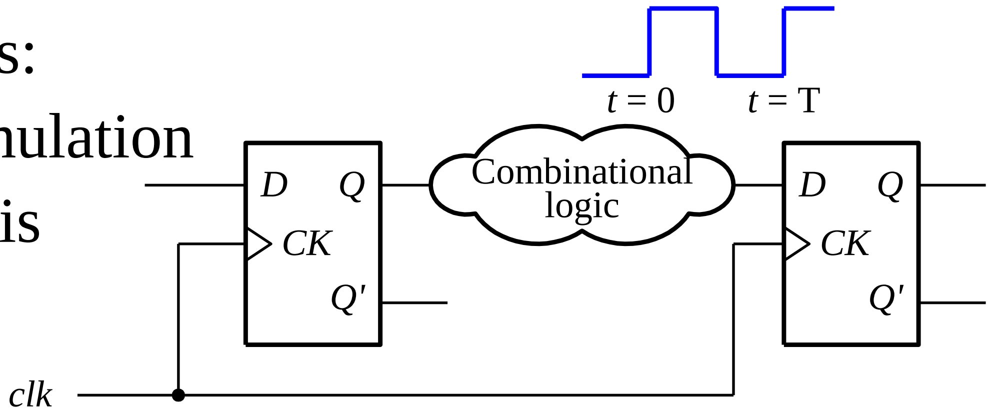
| Instance | Design unit | Design unit type | Stmt count | Stmt hits | Stmt misses | Stmt % | Stmt graph | Branch count |
|---|---|---|---|---|---|---|---|---|
| /three_steps_step3_tb | three_steps... | Module | 35 | 35 | 0 | 100% | 🟩 | |
| /three_steps_step3_... | controller | Module | 21 | 21 | 0 | 100% | 🟩 | |
| /three_steps_step3_... | datapath | Module | 5 | 5 | 0 | 100% | 🟩 | |

## Why Static Timing Analysis?

- Timing analysis is to estimate when the output of a given circuit gets stable.
- The purposes of timing analysis are as follows:
  - Timing verification
    - Verifies if a design meets a given timing constraint.
    - Example: cycle-time constraint.
  - Timing optimization
    - Needs to identify critical portion of a design for further optimization.
    - Identifies critical paths.

## Why Static Timing Analysis?

- The output needs to be stable by *t* = *T* for the correct functionality. But how to make sure of it?
- At least two approaches:
  - Dynamic timing simulation
  - Static timing analysis



- Why static timing analysis?
  - Using dynamic timing simulation has posed a bottleneck for large complex designs.
  - Dynamic simulation relies on the quality and coverage of the test bench used for verification.

## Static Timing Analysis

- Static timing analysis (STA): without having to simulate clock cycles.
  - No combinational feedback loops are allowed.
  - All register feedback paths are broken by the clock boundary.
  - The delay of each path is calculated.
  - All path delays are checked to see if timing constraints have been met.

## Static Timing Analysis

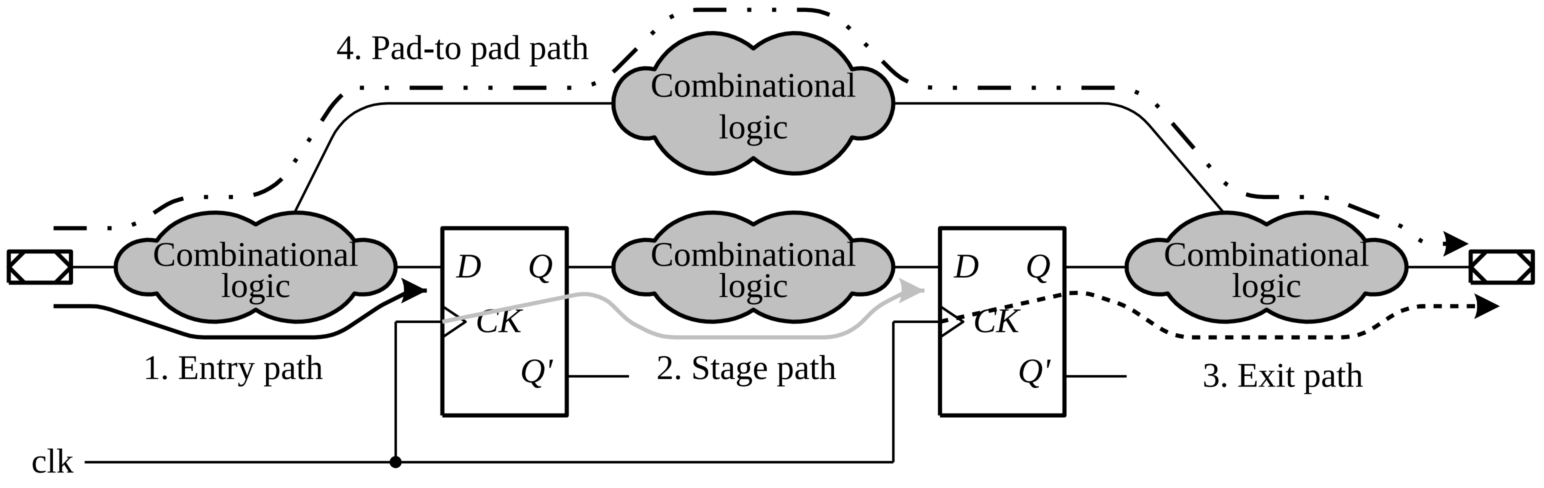
- Note that:
  - Comprehensive sets of test benches are still needed to verify the functionality of the source RTL.
  - STA is used to verify timing.
  - Formal verification technique is usually used to verify the functionality of the gate-level netlist against the source RTL.

## 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

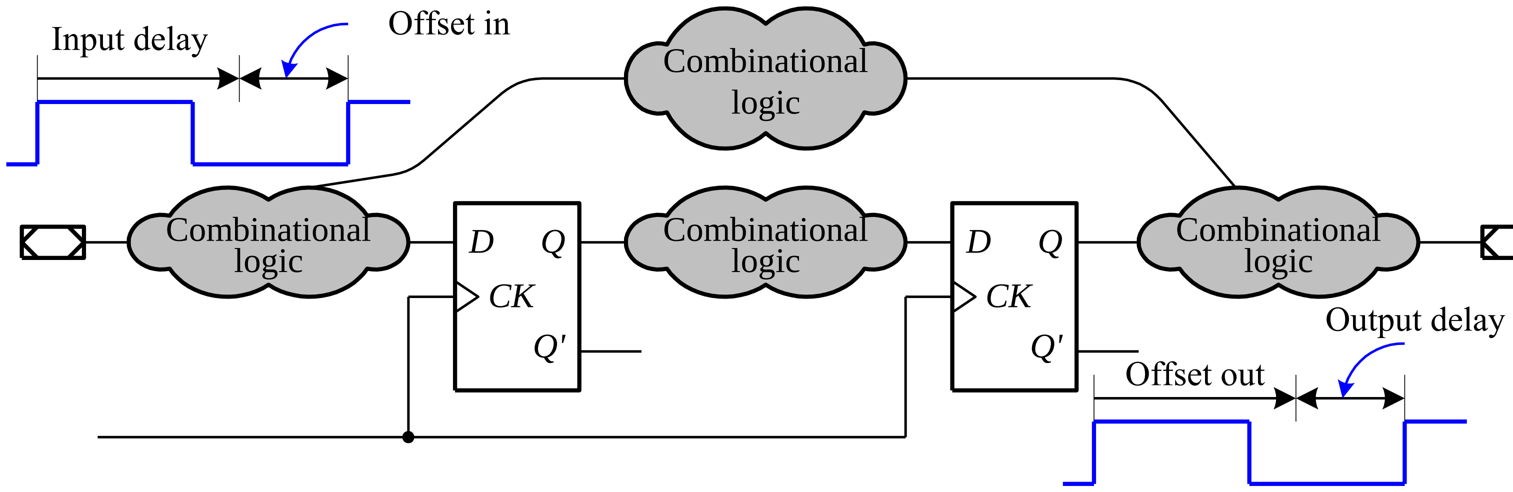
## What to Analyze in STA

- 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)



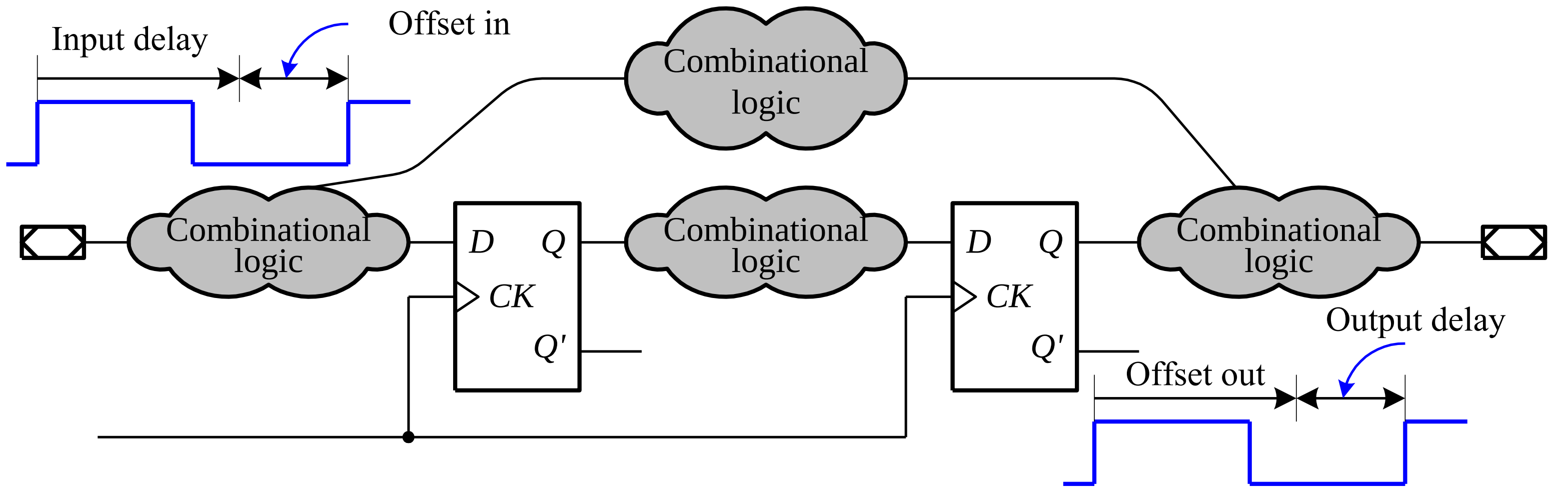
## Timing Specifications: port-related constraints

- Input delay (offset in) constraint applies to paths from input pads to the input of a storage element.
  - It specifies the arrival time of the input signal relative to the active edge of the clock.



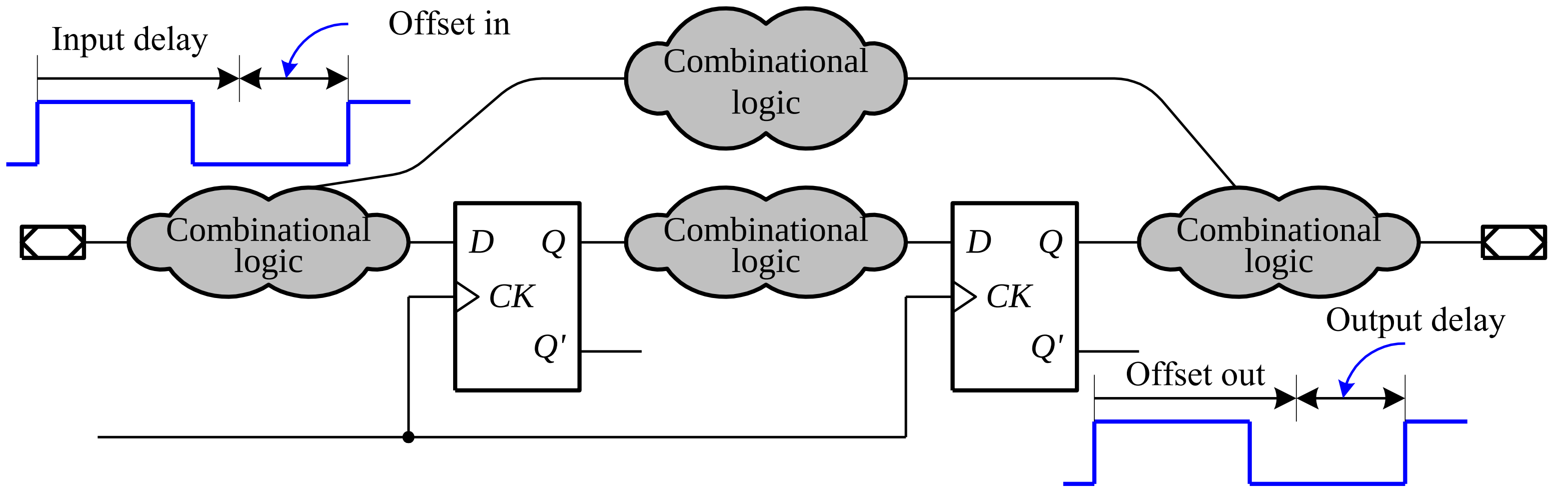
## Timing Specifications: port-related constraints

- Output delay (offset out) constraint applies to paths from the clock input of a storage element to output pads.
  - It specifies the latest time that a signal from the output of a register may reach to output pads.



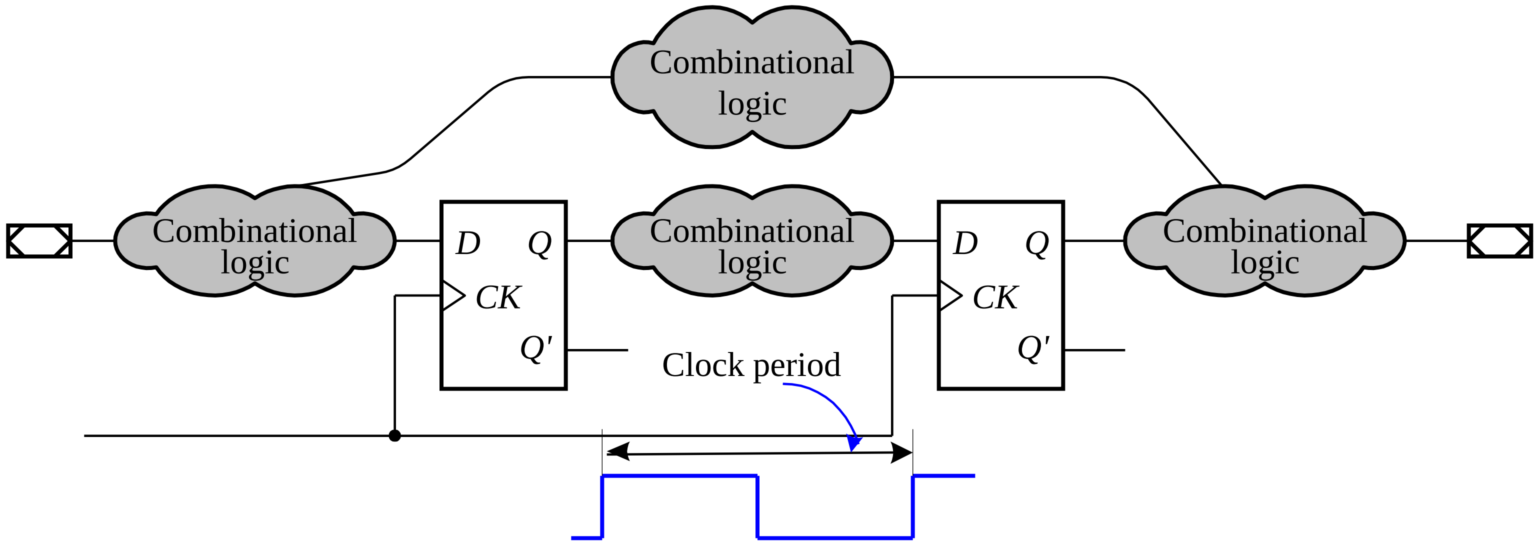
## Timing Specifications: port-related constraints

- Input-output (pad to pad) constraint applies to paths from input pads to output pads without passing through any register



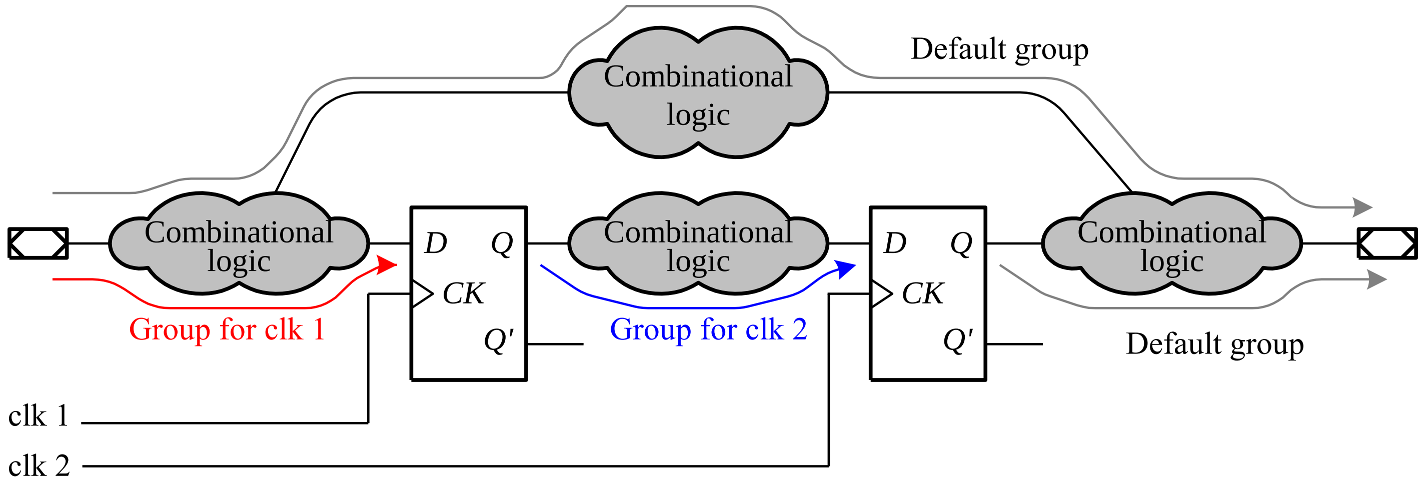
## Timing Specifications: clock-related constraints

- Cycle time (period) constraint applies to the paths between registers and specifies the maximum period of the clock of a synchronous circuit.
- clock jitter, clock-to-Q delay, slew rate, clock skew
- Set-up time, hold time



## Timing Specifications --- Path Groups

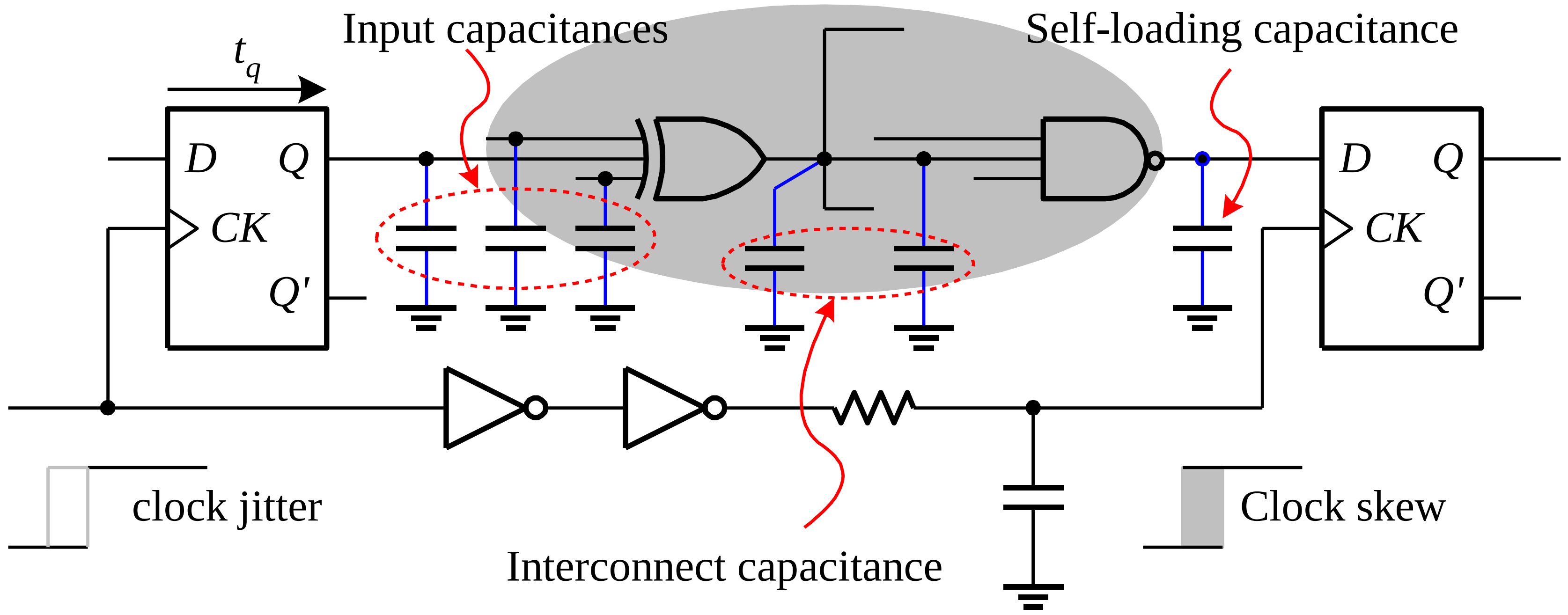
- The paths are grouped according to the clocks controlling their endpoints.
  - Each clock will be associated with a set of paths called a path group.
  - The default path group comprises all paths not associated with a clock.



# Factors Affecting Timing

Factors that affect timing are:

- Clock jitter
- Clock-to-Q delay:
- Input pin capacitance
- Slew rate
- Interconnect loading
- Fan-out loading
- Clock skew
- Temperature



## Setup Time and Hold Time Checks

- Setup time: The minimum time that data must stabilize before the active clock transition.
  - The maximum data path is used to determine whether setup constraint is met or not.
- Hold time: The minimum time that data must remain stable after the active clock transition.
  - The minimum data path is used to determine if hold time is met.

## Critical Paths

- A critical path is the path of longest propagation delay.
  - A critical path is a combinational logic path that has negative or smallest slack time, where slack time is defined as:

    slack = required time – arrival time

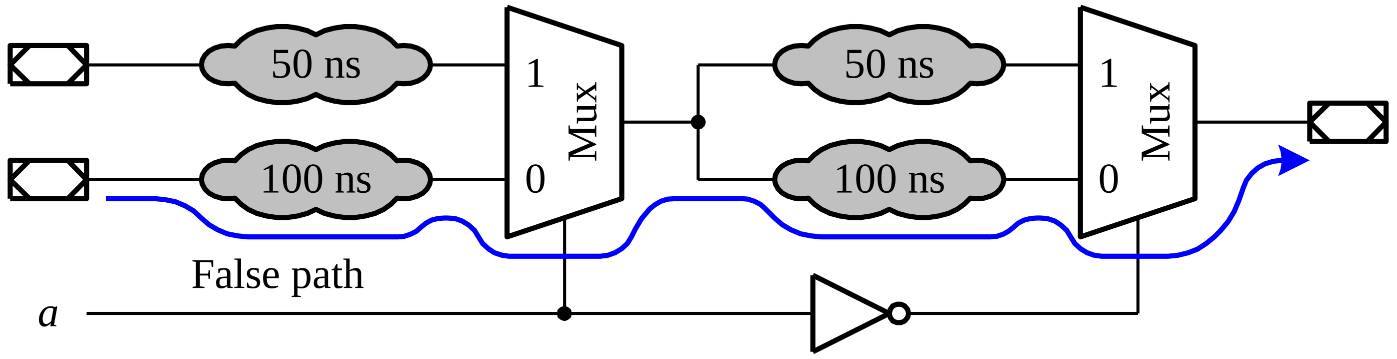
    = requirement – datapath (in ISE)
  - Critical paths limit the system performance.
  - Critical paths not only tell us the system cycle time, it also points out which part of the combinational logic must be changed to improve system performance.

## Timing Exceptions

- Timing analysis tools usually treat all paths in the design as single-cycle by default and perform STA accordingly.
- Two common timing exceptions:
  - False paths: A false path is identified as a timing path that does not propagate a signal.
  - Multi-cycle paths: In the model, data may take more than one clock cycle to reach its destination.

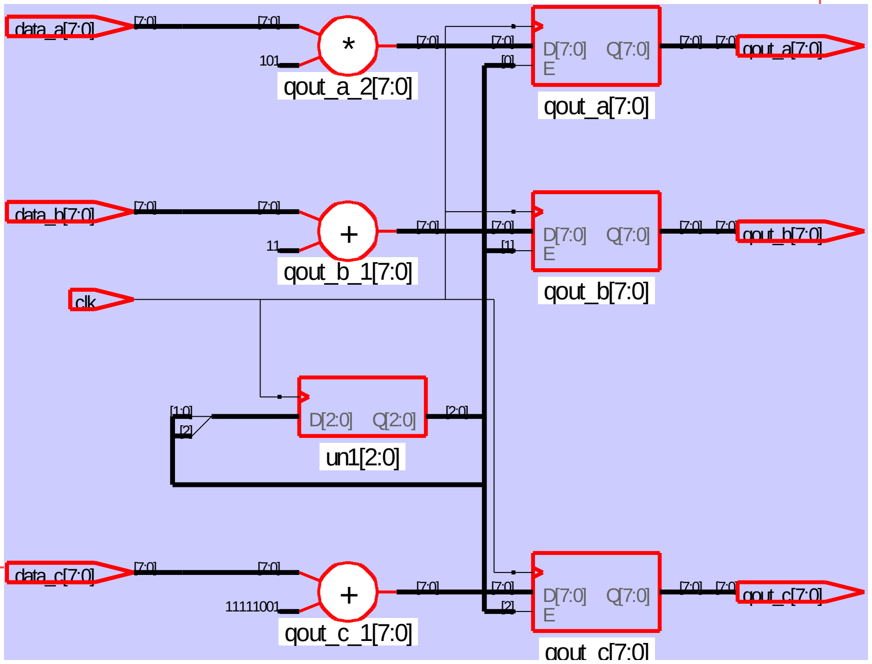
## False Paths

- False paths are paths that static timing analysis identifies as failing timing, but that the designer knows are not actually failing.
  - A false path is a timing path that does not propagate a signal.



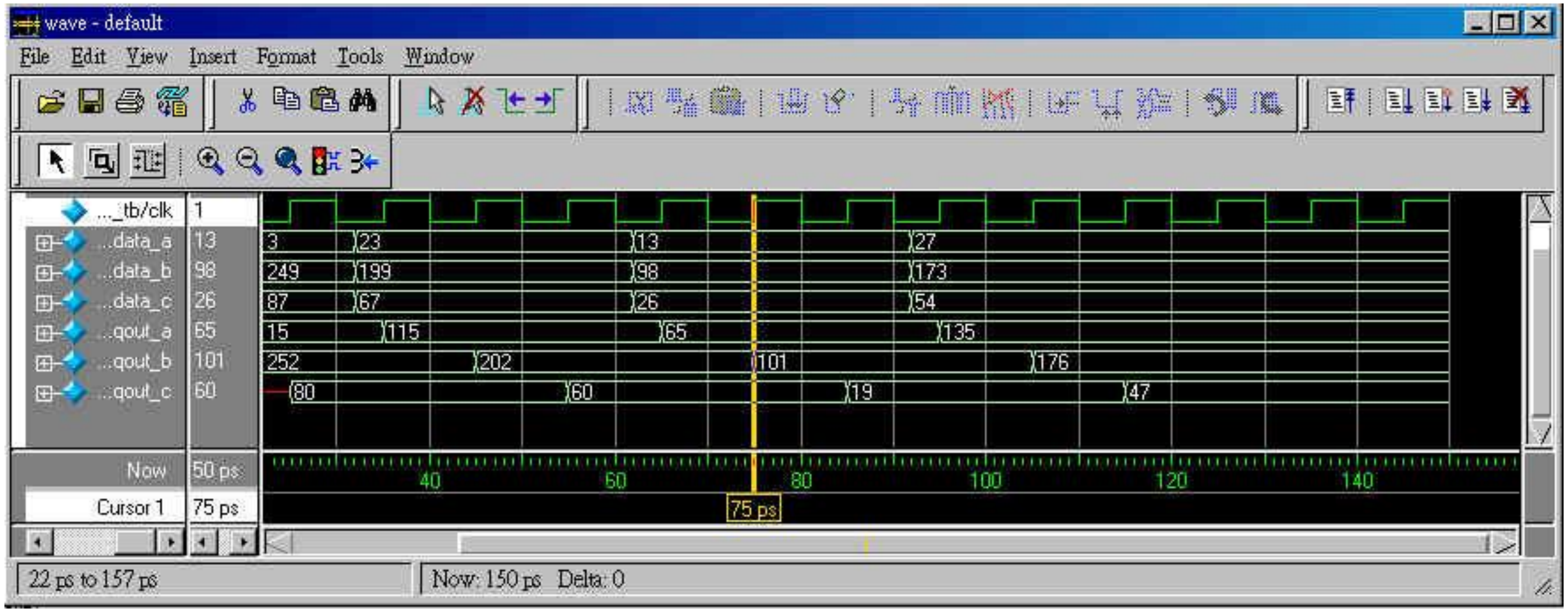
## Multi-Cycle Paths --- A Trivial Example

```
// a multiple cycle example
module multiple_cycle_example(clk, data_a, data_b, data_c, qout_a, qout_b, qout_c);
parameter N = 8;
input   clk;
input   [N-1:0] data_a, data_b, data_c;
output  reg [N-1:0] qout_a, qout_b, qout_c;
// tTrivial 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
endmodule
```

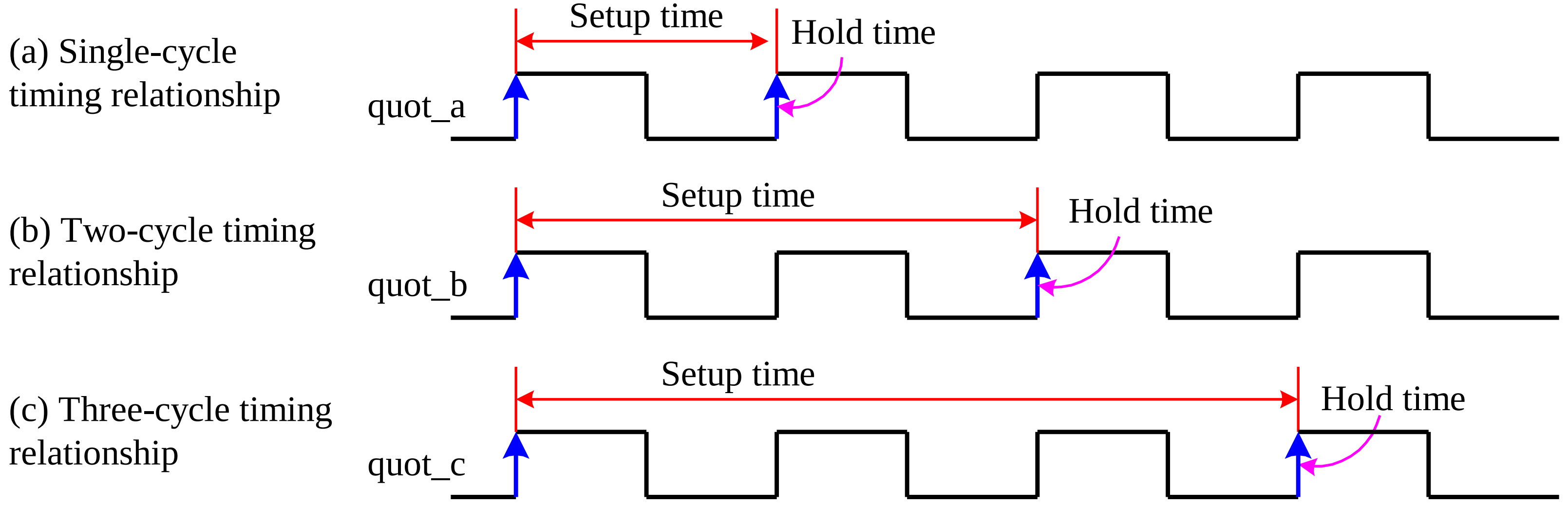


Q: Explain the operation of above code.

## Multi-Cycle Paths --- A Trivial Example


wave - default
File Edit View Insert Format Tools Window
..._tb/clk 1
...data_a 13 — 3, 23, 13, 27
...data_b 98 — 249, 199, 98, 173
...data_c 26 — 87, 67, 26, 54
...qout_a 65 — 15, 115, 65, 135
...qout_b 101 — 252, 202, 101, 176
...qout_c 60 — 80, 60, 19, 47
Now 150 ps
Cursor 1 75 ps
40 60 80 100 120 140
75 ps
22 ps to 157 ps
Now: 150 ps Delta: 0

## Multi-Cycle Paths --- Timing Relationship


(a) Single-cycle timing relationship
Setup time
Hold time
quot_a
(b) Two-cycle timing relationship
Setup time
Hold time
quot_b
(c) Three-cycle timing relationship
Setup time
Hold time
quot_c

Notice that:
STA tools treat all paths in the design as a single-cycle by default and perform the STA accordingly. So you need to tell them which paths are multi-cycle paths.

## An Example of Practical Verification Process

- Simulation
  - Functional (behavioral) simulation
  - Code coverage analysis
  - Assertion (property) checking
  - Gate-level simulation
  - Dynamic timing simulation (gate-level simulation + SDF back annotation)
- Static timing analysis
  - Critical paths
  - Timing violations
- Prototyping
  - FPGA prototyping
  - Cell-based prototyping