Reading Assignment

► This lecture: 3.1–3.4
► Next lecture: 3.5, 3.8
Logic Simulation

The Unknown Logic Value
We will focus on the simulation program for this chapter.

Figure 3.1  Simulation process

(Abramovici et al., 1990)

▶ We will focus on the simulation program for this chapter.
Purpose of Simulation

- Design verification: make sure the design performs its specified behavior.
  - For both functionality and timing.
  - Verification is done by comparing simulation results with expected results ("golden").
- Design analysis: evaluate alternative designs.
  - For trade-offs between cost/performance/etc.
  - AKA design space exploration
- Debugging
  - Provide accesses to signals not observable in the actual hardware.
- Prepare data for testing.
  - Generate golden results for guided-probe testing.
  - Assist stimuli (test) generation for physical faults.
Purpose of Simulation

- Design verification: make sure the design performs its specified behavior.
  - For both functionality and timing.
  - Verification is done by comparing simulation results with expected results ("golden").
- Design analysis: evaluate alternative designs.
  - For trade-offs between cost/performance/etc.
  - AKA design space exploration
- Debugging
  - Provide accesses to signals not observable in the actual hardware.
- Prepare data for testing.
  - Generate golden results for guided-probe testing.
  - Assist stimuli (test) generation for physical faults.
Purpose of Simulation

- Design verification: make sure the design performs its specified behavior.
  - For both functionality and timing.
  - Verification is done by comparing simulation results with expected results ("golden").
- Design analysis: evaluate alternative designs.
  - For trade-offs between cost/performance/etc.
  - AKA design space exploration
- Debugging
  - Provide accesses to signals not observable in the actual hardware.
- Prepare data for testing.
  - Generate golden results for guided-probe testing.
  - Assist stimuli (test) generation for physical faults.
Purpose of Simulation

- Design verification: make sure the design performs its specified behavior.
  - For both functionality and timing.
  - Verification is done by comparing simulation results with expected results ("golden").
- Design analysis: evaluate alternative designs.
  - For trade-offs between cost/performance/etc.
  - AKA design space exploration
- Debugging
  - Provide accesses to signals not observable in the actual hardware.
- Prepare data for testing.
  - Generate golden results for guided-probe testing.
  - Assist stimuli (test) generation for physical faults.
Stimuli Generation

- Generate test cases.
  - Each test case verifies a certain aspect of the behavior.
- For design verification and debugging, the objective is to find design errors.
  - It is hard to know what design errors may appear.
  - Experiences show errors are related to control of data flow rather than the data operations.
  - The usual strategy is to emphasize exercising the control.
  - e.g. cover all paths in FSM for hardware or cover all branches for software (whitebox testing).
- For testing of physical faults, the objective is to detect manufacturing defects.
  - Defects and faults can be modeled and enumerated.
  - Tests are generated to differentiate a system with known faults from a fault-free system.
  - Fault coverage can be used to evaluate the completeness of the test cases for specific fault models.
Stimuli Generation

- Generate test cases.
  - Each test case verifies a certain aspect of the behavior.
- For design verification and debugging, the objective is to find design errors.
  - It is hard to know what design errors may appear.
  - Experiences show errors are related to control of data flow rather than the data operations.
  - The usual strategy is to emphasize exercising the control.
  - e.g. cover all paths in FSM for hardware or cover all branches for software (whitebox testing).

- For testing of physical faults, the objective is to detect manufacturing defects.
  - Defects and faults can be modeled and enumerated.
  - Tests are generated to differentiate a system with known faults from a fault-free system.
  - Fault coverage can be used to evaluate the completeness of the test cases for specific fault models.
Stimuli Generation

- Generate test cases.
  - Each test case verifies a certain aspect of the behavior.
- For design verification and debugging, the objective is to find design errors.
  - It is hard to know what design errors may appear.
  - Experiences show errors are related to control of data flow rather than the data operations.
  - The usual strategy is to emphasize exercising the control.
  - e.g. cover all paths in FSM for hardware or cover all branches for software (whitebox testing).
- For testing of physical faults, the objective is to detect manufacturing defects.
  - Defects and faults can be modeled and enumerated.
  - Tests are generated to differentiate a system with known faults from a fault-free system.
  - Fault coverage can be used to evaluate the completeness of the test cases for specific fault models.
Correctness of Results

- A typical design flow starts with an informal description of the system.
- Simulatable (formal) specification of the system is created at a higher abstraction level and is checked against the informal model.
- Designs are then implemented at lower abstraction levels using models at higher abstraction levels as specifications.
  - Simulation results for specifications are used as golden to verify results for implementations.
Correctness of Results

▶ A typical design flow starts with an informal description of the system.
▶ Simulatable (formal) specification of the system is created at a higher abstraction level and is checked against the informal model.
▶ Designs are then implemented at lower abstraction levels using models at higher abstraction levels as specifications.
  ▶ Simulation results for specifications are used as golden to verify results for implementations.
Correctness of Results

- A typical design flow starts with an informal description of the system.
- Simulatable (formal) specification of the system is created at a higher abstraction level and is checked against the informal model.
- Designs are then implemented at lower abstraction levels using models at higher abstraction levels as specifications.
  - Simulation results for specifications are used as golden to verify results for implementations.
Limitations of Design Verification via Simulation

- Your design is as good as your tests.
  - Lack of formal procedure for stimuli generation.
  - Lack of formal estimation of coverage.

- For hardware design at logic level, logic simulation is primarily for power analysis.
  - Functional verification is mostly based on formal verification, especially equivalence checking as specifications are available.
  - Timing verification/analysis is mostly based on static timing analysis.

- For embedded system designs, simulations (virtual prototyping) are essential.
  - For all design tasks including debugging, analysis, and verification via the hardware/software codesign methodology.

- Simulation-based design verification are moving to higher abstraction levels where formal methods are not mature enough or when formal specifications are not available.
Limitations of Design Verification via Simulation

- Your design is as good as your tests.
  - Lack of formal procedure for stimuli generation.
  - Lack of formal estimation of coverage.

- For hardware design at logic level, logic simulation is primarily for power analysis.
  - Functional verification is mostly based on formal verification, especially equivalence checking as specifications are available.
  - Timing verification/analysis is mostly based on static timing analysis.

- For embedded system designs, simulations (virtual prototyping) are essential.
  - For all design tasks including debugging, analysis, and verification via the hardware/software codesign methodology.

- Simulation-based design verification are moving to higher abstraction levels where formal methods are not mature enough or when formal specifications are not available.
Limitations of Design Verification via Simulation

- Your design is as good as your tests.
  - Lack of formal procedure for stimuli generation.
  - Lack of formal estimation of coverage.
- For hardware design at logic level, logic simulation is primarily for power analysis.
  - Functional verification is mostly based on formal verification, especially equivalence checking as specifications are available.
  - Timing verification/analysis is mostly based on static timing analysis.
- For embedded system designs, simulations (virtual prototyping) are essential.
  - For all design tasks including debugging, analysis, and verification via the hardware/software codesign methodology.
- Simulation-based design verification are moving to higher abstraction levels where formal methods are not mature enough or when formal specifications are not available.
Limitations of Design Verification via Simulation

- Your design is as good as your tests.
  - Lack of formal procedure for stimuli generation.
  - Lack of formal estimation of coverage.

- For hardware design at logic level, logic simulation is primarily for power analysis.
  - Functional verification is mostly based on formal verification, especially equivalence checking as specifications are available.
  - Timing verification/analysis is mostly based on static timing analysis.

- For embedded system designs, simulations (virtual prototyping) are essential.
  - For all design tasks including debugging, analysis, and verification via the hardware/software codesign methodology.

- Simulation-based design verification are moving to higher abstraction levels where formal methods are not mature enough or when formal specifications are not available.
Types of Simulation

- **Compiled simulation**
  - Interpret a model as host operations.
  - Functional, i.e. static behavior with no notion of time.
  - For combinational circuits or those that can be treated so, e.g. for synchronous circuits when the timing is verified separately.

- **Event-driven simulation**
  - Simulate activities, e.g. the signals changing value at some arbitrary time in a circuit.
  - Consider notions of time, able to simulate real-time inputs.
  - For any kinds of circuits, but is much harder to implement than compiled simulator.

- **Activity factor**: ratio of active signals (those change value) to all signals in a logic circuit.
  - Very small: 1% $\sim$ 5%
  - Event-driven logic simulation may run faster than compiled logic simulation as it only visits active signals.
  - We will study both.
Types of Simulation

- Compiled simulation
  - Interpret a model as host operations.
  - Functional, i.e. static behavior with no notion of time.
  - For combinational circuits or those that can be treated so, e.g. for synchronous circuits when the timing is verified separately.

- Event-driven simulation
  - Simulate activities, e.g. the signals changing value at some arbitrary time in a circuit.
  - Consider notions of time, able to simulate real-time inputs.
  - For any kinds of circuits, but is much harder to implement than compiled simulator.
  - Activity factor: ratio of active signals (those change value) to all signals in a logic circuit.
    - Very small: 1% ~ 5%
    - Event-driven logic simulation may run faster than compiled logic simulation as it only visits active signals.
    - We will study both.
Types of Simulation

- **Compiled simulation**
  - Interpret a model as host operations.
  - Functional, i.e. static behavior with no notion of time.
  - For combinational circuits or those that can be treated so, e.g. for synchronous circuits when the timing is verified separately.

- **Event-driven simulation**
  - Simulate activities, e.g. the signals changing value at some arbitrary time in a circuit.
  - Consider notions of time, able to simulate real-time inputs.
  - For any kinds of circuits, but is much harder to implement than compiled simulator.

- **Activity factor**: ratio of active signals (those change value) to all signals in a logic circuit.
  - Very small: 1% ~ 5%
  - Event-driven logic simulation may run faster than compiled logic simulation as it only visits active signals.
  - We will study both.
Outline

Logic Simulation

The Unknown Logic Value
Response of a synchronous circuit to an input sequence depends on its initial state.

You may designate initial states in the FSM model.

However, the circuit can be at any state when power-up.

- Contents in storage and sequential elements are usually unpredictable.

The formal treatment is to include an initialization sequence.

- As inputs, applied before the normal operation begins.
- Bring the circuit into a known “reset” state from any state.

How can we simulate all possible behavior under such sequence?

- You cannot simulate for each possible initial state.
Response of a synchronous circuit to an input sequence depends on its initial state.

You may designate initial states in the FSM model.

However, the circuit can be at any state when power-up.

Contents in storage and sequential elements are usually unpredictable.

The formal treatment is to include an initialization sequence.

As inputs, applied before the normal operation begins.

Bring the circuit into a known “reset” state from any state.

How can we simulate all possible behavior under such sequence?

You cannot simulate for each possible initial state.
Response of a synchronous circuit to an input sequence depends on its initial state.

You may designate initial states in the FSM model.

However, the circuit can be at any state when power-up.
  - Contents in storage and sequential elements are usually unpredictable.

The formal treatment is to include an initialization sequence.
  - As inputs, applied before the normal operation begins.
  - Bring the circuit into a known “reset” state from any state.

How can we simulate all possible behavior under such sequence?
  - You cannot simulate for each possible initial state.
Response of a synchronous circuit to an input sequence depends on its initial state.

You may designate initial states in the FSM model. However, the circuit can be at any state when power-up.

- Contents in storage and sequential elements are usually unpredictable.

The formal treatment is to include an initialization sequence.

- As inputs, applied before the normal operation begins.
- Bring the circuit into a known “reset” state from any state.

How can we simulate all possible behavior under such sequence?

- You cannot simulate for each possible initial state.
Response of a synchronous circuit to an input sequence depends on its initial state.

You may designate initial states in the FSM model.

However, the circuit can be at any state when power-up.

Contents in storage and sequential elements are usually unpredictable.

The formal treatment is to include an initialization sequence.

As inputs, applied before the normal operation begins.

Bring the circuit into a known “reset” state from any state.

How can we simulate all possible behavior under such sequence?

You cannot simulate for each possible initial state.
The Unknown Logic Value

- Denoted by $u$
- Represent a value in the set \{0, 1\}.
  - Though you cannot tell whether it’s a 0 or 1.
- To derive rules for logic operations involving $u$, we can treat 0 as \{0\} and 1 as \{1\}, and see what are all possible results.
- For example,
  \[
  \begin{align*}
  \text{AND}(0, u) &= \text{AND}(\{0\}, \{0, 1\}) = \{\text{AND}(0, 0), \text{AND}(0, 1)\} = \{0, 0\} = \{0\} = 0 \\
  \text{NOT}(u) &= \text{NOT}(\{0, 1\}) = \{\text{NOT}(0), \text{NOT}(1)\} = \{1, 0\} = \{0, 1\} = u
  \end{align*}
  \]
The Unknown Logic Value

- Denoted by $u$
- Represent a value in the set $\{0, 1\}$.
  - Though you cannot tell whether it’s a 0 or 1.
- To derive rules for logic operations involving $u$, we can treat 0 as $\{0\}$ and 1 as $\{1\}$, and see what are all possible results.
- For example,
  
  $\text{AND}(0, u) = \text{AND}(\{0\}, \{0, 1\}) = \{\text{AND}(0, 0), \text{AND}(0, 1)\} = \{0, 0\} = \{0\} = 0$
  
  $\text{NOT}(u) = \text{NOT}(\{0, 1\}) = \{\text{NOT}(0), \text{NOT}(1)\} = \{1, 0\} = \{0, 1\} = u$
The Unknown Logic Value

- Denoted by \( u \)
- Represent a value in the set \( \{0, 1\} \).
  - Though you cannot tell whether it’s a 0 or 1.
- To derive rules for logic operations involving \( u \), we can treat 0 as \( \{0\} \) and 1 as \( \{1\} \), and see what are all possible results.

  - For example,
    \[
    \begin{align*}
    \text{AND}(0, u) &= \text{AND}(\{0\}, \{0, 1\}) = \{\text{AND}(0, 0), \text{AND}(0, 1)\} = \{0, 0\} = \{0\} = 0 \\
    \text{NOT}(u) &= \text{NOT}(\{0, 1\}) = \{\text{NOT}(0), \text{NOT}(1)\} = \{1, 0\} = \{0, 1\} = u
    \end{align*}
    \]
The Unknown Logic Value

- Denoted by \( u \)
- Represent a value in the set \( \{0, 1\} \).
  - Though you cannot tell whether it’s a 0 or 1.
- To derive rules for logic operations involving \( u \), we can treat 0 as \( \{0\} \) and 1 as \( \{1\} \), and see what are all possible results.
- For example,
  \[
  AND(0, u) = AND(\{0\}, \{0, 1\}) = \{AND(0, 0), AND(0, 1)\} = \{0, 0\} = \{0\} = 0
  \]
  \[
  NOT(u) = NOT(\{0, 1\}) = \{NOT(0), NOT(1)\} = \{1, 0\} = \{0, 1\} = u
  \]
Therefore, all Boolean functions can be extended to handle three-valued logic.
Therefore, all Boolean functions can be extended to handle three-valued logic.
However, three-valued logic is pessimistic. In the sense that certain signal in real circuit will only take a definite value of 0 or 1, though simulation indicates $u$. So it is possible an initialization sequence working well in practice cannot be verified.
However, three-valued logic is pessimistic.

- In the sense that certain signal in real circuit will only take a definite value of 0 or 1, though simulation indicates $u$.

- So it is possible an initialization sequence working well in practice cannot be verified.

Figure 3.4 Pessimistic result in 3-valued simulation

(Abramovici et al., 1990)
What If ... 

We may attempt to address the issue with a new unknown value $u$.

However ...

Figure 3.4  Pessimistic result in 3-valued simulation

(Abramovici et al., 1990)
What If ...

▶ We may attempt to address the issue with a new unknown value $u$.

▶ However ...

**Figure 3.4** Pessimistic result in 3-valued simulation

(Abramovici et al., 1990)

- We may attempt to address the issue with a new unknown value $\bar{u}$.
- However ...
Wrong Result!

That’s because the two DFFs may store different bits.

Can you verify initialization sequence without using \( u \)?

- Hint: use Fig. 2.9 and SAT. However, it could be too costly in verification time.

- We trade pessimism for fast verifications.

Figure 3.5 Incorrect result from using \( u \) and \( \overline{u} \)

(Abramovici et al., 1990)
Wrong Result!

That’s because the two DFFs may store different bits.

Can you verify initialization sequence without using $u$?

- Hint: use Fig. 2.9 and SAT. However, it could be too costly in verification time.

We trade pessimism for fast verifications.

Figure 3.5 Incorrect result from using $u$ and $\bar{u}$

(Abramovici et al., 1990)
Wrong Result!

That's because the two DFFs may store different bits.

Can you verify initialization sequence without using $u$?

- Hint: use Fig. 2.9 and SAT. However, it could be too costly in verification time.

We trade pessimism for fast verifications.

Figure 3.5 Incorrect result from using $u$ and $\overline{u}$

(Abramovici et al., 1990)
Simulation programs generate results based on stimuli and internal model.

Three-valued logic is used to address the problem of unknown initial state but is pessimistic.