Reading Assignment

- This lecture: 2.1, 2.2
- Next lecture: 2.4
Naval Gupte: ngupte1@hawk.iit.edu
Office hours/Location: Tue. 1:30 PM – 4:30 PM, SH 306A
Outline

Basic Concepts

Functional Modeling at Logic Level

Binary Decision Diagram
Models

- Model: a formal (mathematical) description of a system as components and their interactions.
  - Boolean Algebra: literals and operations
  - Finite State Machine (FSM): states and transitions
- Models are widely used for state of the art software and hardware designs.
  - As system specification to ensure correctness.
  - As technical jargon to facilitate communications between designers.
  - As internal representations to make CAD tools possible.
Models

- Model: a formal (mathematical) description of a system as components and their interactions.
  - Boolean Algebra: literals and operations
  - Finite State Machine (FSM): states and transitions
- Models are widely used for state of the art software and hardware designs.
  - As system specification to ensure correctness.
  - As technical jargon to facilitate communications between designers.
  - As internal representations to make CAD tools possible.
Models as Internal Representations

- Tasks related to digital systems testings depend on the choice of models.
  - Simulation
  - Fault modeling and simulation
  - Test generation.

- Pay attention to how various models map to data structures and their complexities.
Tasks related to digital systems testings depend on the choice of models.

- Simulation
- Fault modeling and simulation
- Test generation.

Pay attention to how various models map to data structures and their complexities.
A system can be viewed from outside as a black box that generates outputs from inputs.

A functional model describes how outputs are computed from inputs.

- In reality, it always takes time to compute outputs from inputs, which is ignore by a functional model.
- A behavioral model (as defined in this book) adds timing to a functional model.
Functional Modeling

- A system can be viewed from outside as a black box that generates outputs from inputs.
- A functional model describes how outputs are computed from inputs.
  - In reality, it always takes time to compute outputs from inputs, which is ignored by a functional model.
  - A behavioral model (as defined in this book) adds timing to a functional model.
Structural Modeling

- A structural model describes the system black box as a collection of interconnected smaller boxes (components, elements).
- Structural models are usually hierarchical.
  - Components are modeled as interconnections of lower-level components.
  - Types are usually introduced to group components with the same functionality.
  - Bottom-level boxes are primitive elements with known functional/behavioral models.
- Functional and structural modelings are usually intermixed in practice.
  - A structural model always carries information regarding the functionality of its components.
  - A functional model usually contain some structural information.
Structural Modeling

- A structural model describes the system black box as a collection of interconnected smaller boxes (components, elements).

- Structural models are usually hierarchical.
  - Components are modeled as interconnections of lower-level components.
  - Types are usually introduced to group components with the same functionality.
  - Bottom-level boxes are primitive elements with known functional/behavioral models.

- Functional and structural modelings are usually intermixed in practice.
  - A structural model always carries information regarding the functionality of its components.
  - A functional model usually contain some structural information.
Structural Modeling

- A structural model describes the system black box as a collection of interconnected smaller boxes (components, elements).
- Structural models are usually hierarchical.
  - Components are modeled as interconnections of lower-level components.
  - Types are usually introduced to group components with the same functionality.
  - Bottom-level boxes are primitive elements with known functional/behavioral models.
- Functional and structural modelings are usually intermixed in practice.
  - A structural model always carries information regarding the functionality of its components.
  - A functional model usually contain some structural information.
External and Internal Models

- External models are used by the designers.
  - e.g. schematic diagrams or textual inputs
- Internal models are used by CAD tools.
  - Data structure and associated algorithms.
- Compilers translate from external models to internal models.
  - The choice of the external model affects the complexity of the compiler.
  - For this course, we would usually assume that the internal model of a system is available, or that the external model is simple enough to be easily converted into an internal model.
External and Internal Models

- External models are used by the designers.
  - e.g. schematic diagrams or textual inputs
- Internal models are used by CAD tools.
  - Data structure and associated algorithms.
- Compilers translate from external models to internal models.
  - The choice of the external model affects the complexity of the compiler.
  - For this course, we would usually assume that the internal model of a system is available, or that the external model is simple enough to be easily converted into an internal model.
External and Internal Models

- External models are used by the designers.
  - e.g. schematic diagrams or textual inputs
- Internal models are used by CAD tools.
  - Data structure and associated algorithms.
- Compilers translate from external models to internal models.
  - The choice of the external model affects the complexity of the compiler.
  - For this course, we would usually assume that the internal model of a system is available, or that the external model is simple enough to be easily converted into an internal model.
Outline

Basic Concepts

Functional Modeling at Logic Level

Binary Decision Diagram
Combinational Circuit and Truth Table

- Combinational circuits are usually specified by designers in Boolean algebra.
- A combinational circuit with \( n \) input bits and \( m \) output bits can be represented by \( m \ n \)-input Boolean functions.
  - So we can represent a combinational circuit as a Boolean function.
- A Boolean function can be represented by its truth table.
Combinational Circuit and Truth Table

- Combinational circuits are usually specified by designers in Boolean algebra.
- A combinational circuit with $n$ input bits and $m$ output bits can be represented by $m$ $n$-input Boolean functions.
  - So we can represent a combinational circuit as a Boolean function.
- A Boolean function can be represented by its truth table.
Combinational Circuit and Truth Table

- Combinational circuits are usually specified by designers in Boolean algebra.
- A combinational circuit with $n$ input bits and $m$ output bits can be represented by $m$ $n$-input Boolean functions.
  - So we can represent a combinational circuit as a Boolean function.
- A Boolean function can be represented by its truth table.
Truth Table Implementations

- As an array of size $2^n$.
  - Each index (among 0, 1, . . . , $2^n − 1$) corresponds to a set of inputs, usually as the bits in the binary form of the index, following certain order.
  - Each element of the array is a bit indicating the output under the inputs corresponding to the index.

- We may extend each element to $m$ bits to represent the whole combinational logic.
  - Time complexity to compute an output: $O(n + m)$ (assume inputs/outputs are in the format of bits)
  - Space complexity: $O(m2^n)$ – a concern if $n$ is not small.

- When $n$ and $m$ are small, one may implement a combination circuit in any memory (ROM, RAM, flash, etc.).
  - Functions could be changed on the fly – reconfigurable.
  - As in all FPGA implementations.
Truth Table Implementations

- As an array of size $2^n$.
  - Each index (among 0, 1, ..., $2^n - 1$) corresponds to a set of inputs, usually as the bits in the binary form of the index, following certain order.
  - Each element of the array is a bit indicating the output under the inputs corresponding to the index.

- We may extend each element to $m$ bits to represent the whole combinational logic.
  - Time complexity to compute an output: $O(n + m)$ (assume inputs/outputs are in the format of bits)
  - Space complexity: $O(m2^n)$ – a concern if $n$ is not small.

- When $n$ and $m$ are small, one may implement a combination circuit in any memory (ROM, RAM, flash, etc.).
  - Functions could be changed on the fly – reconfigurable.
  - As in all FPGA implementations.
Truth Table Implementations

- As an array of size $2^n$.
  - Each index (among 0, 1, . . ., $2^n - 1$) corresponds to a set of inputs, usually as the bits in the binary form of the index, following certain order.
  - Each element of the array is a bit indicating the output under the inputs corresponding to the index.
- We may extend each element to $m$ bits to represent the whole combinational logic.
  - Time complexity to compute an output: $O(n + m)$ (assume inputs/outputs are in the format of bits)
  - Space complexity: $O(m2^n)$ – a concern if $n$ is not small.
- When $n$ and $m$ are small, one may implement a combination circuit in any memory (ROM, RAM, flash, etc.).
  - Functions could be changed on the fly – reconfigurable.
  - As in all FPGA implementations.
Synchronous Sequential Circuit and FSM

- Sequential circuit utilizes storage elements to
  - Reduce system cost by reusing combination parts.
  - Enable computations that need access to previous results (history).

- Synchronous sequential circuits are usually specified as finite state machines (FSM).
  - Sets: states $Q$, input symbols $I$, set of output symbols $O$.
  - Functions: next-state $N : Q \times I \rightarrow Q$, output $Z : Q \times I \rightarrow O$.
  - Encodings: states, input symbols, and output symbols are represented using a fixed number of bits. $N$ and $Z$ are combinational circuits.

- Synchronous sequential circuits can be represented as their FSMs.
Synchronous Sequential Circuit and FSM

- Sequential circuit utilizes storage elements to
  - Reduce system cost by reusing combination parts.
  - Enable computations that need access to previous results (history).
- Synchronous sequential circuits are usually specified as finite state machines (FSM).
  - Sets: states $Q$, input symbols $I$, set of output symbols $O$.
  - Functions: next-state $N : Q \times I \rightarrow Q$, output $Z : Q \times I \rightarrow O$.
  - Encodings: states, input symbols, and output symbols are represented using a fixed number of bits. $N$ and $Z$ are combinational circuits.
- Synchronous sequential circuits can be represented as their FSMs.
Sequential circuit utilizes storage elements to
- Reduce system cost by reusing combination parts.
- Enable computations that need access to previous results (history).

Synchronous sequential circuits are usually specified as finite state machines (FSM).
- Sets: states $Q$, input symbols $I$, set of output symbols $O$.
- Functions: next-state $N: Q \times I \rightarrow Q$, output $Z: Q \times I \rightarrow O$.
- Encodings: states, input symbols, and output symbols are represented using a fixed number of bits. $N$ and $Z$ are combinational circuits.

Synchronous sequential circuits can be represented as their FSMs.
State Table

- Represent a FSM by specifying $N$ and $Z$.
- A special kind of truth tables.
  - $n + q$ inputs and $m + q$ outputs, where $q$ is the number of state bits.
  - Implemented as an array of $2^{n+q}$ elements, each has $m + q$ bits.
- Complexities are similar to truth table.
State Table

- Represent a FSM by specifying $N$ and $Z$.
- A special kind of truth tables.
  - $n + q$ inputs and $m + q$ outputs, where $q$ is the number of state bits.
  - Implemented as an array of $2^{n+q}$ elements, each has $m + q$ bits.
- Complexities are similar to truth table.
State Table

- Represent a FSM by specifying $N$ and $Z$.
- A special kind of truth tables.
  - $n + q$ inputs and $m + q$ outputs, where $q$ is the number of state bits.
  - Implemented as an array of $2^{n+q}$ elements, each has $m + q$ bits.
- Complexities are similar to truth table.
Combinational Modeling of Synchronous Sequential Circuit

(Fig. 2.5 and 2.9, Abramovici et al., 1990)

- Allow to treat a synchronous sequential circuit for a bounded time frame as a combinational circuit.
- The pseudo flip-flops model the combinational relations between FF inputs $Y$ and FF outputs $y$.
  - For D flip-flops (DFF), $y = Y$. 

ECE 586 – Fault Detection in Digital Circuits  Spring 2015  15/22
Allow to treat a synchronous sequential circuit for a bounded time frame as a combinational circuit.

The pseudo flip-flops model the combinational relations between FF inputs Y and FF outputs y.

For D flip-flops (DFF), y = Y.
 Allow to treat a synchronous sequential circuit for a bounded time frame as a combinational circuit.

- The pseudo flip-flops model the combinational relations between FF inputs Y and FF outputs y.
  - For D flip-flops (DFF), \( y = Y \).

(Fig. 2.5 and 2.9, Abramovici et al., 1990)
Discussions

- Both combinational and synchronous sequential circuits can be represented as truth tables.
  - Implementing truth tables is straight-forward.
    - One array per circuit.
  - Memory usage is prohibitive for most circuits.
    - 8GB memory for a circuit with 16 inputs, 16 outputs, 16 state bits.
- Essentially we need better ways to represent Boolean functions.
Both combinational and synchronous sequential circuits can be represented as truth tables.

Implementing truth tables is straight-forward.

- One array per circuit.

Memory usage is prohibitive for most circuits.

- 8GB memory for a circuit with 16 inputs, 16 outputs, 16 state bits.

Essentially we need better ways to represent Boolean functions.
Discussions

- Both combinational and synchronous sequential circuits can be represented as truth tables.
- Implementing truth tables is straight-forward.
  - One array per circuit.
- Memory usage is prohibitive for most circuits.
  - 8GB memory for a circuit with 16 inputs, 16 outputs, 16 state bits.
- Essentially we need better ways to represent Boolean functions.
Both combinational and synchronous sequential circuits can be represented as truth tables.

Implementing truth tables is straight-forward.
  ▶ One array per circuit.

Memory usage is prohibitive for most circuits.
  ▶ 8GB memory for a circuit with 16 inputs, 16 outputs, 16 state bits.

Essentially we need better ways to represent Boolean functions.
Outline

Basic Concepts

Functional Modeling at Logic Level

Binary Decision Diagram
Any Boolean function with \( n \) inputs can be implemented as a \( 2^n \)-to-1 mux with all data inputs tied to 0 or 1.

A \( 2^n \)-to-1 mux can be implemented as \( 2^n - 1 \) 2-to-1 mux’s.

- The circuit diagram is simplified as a binary tree.
- Each node is a mux controlled by the input inside it, with data coming from either branch.
Any Boolean function with $n$ inputs can be implemented as a $2^n$-to-1 mux with all data inputs tied to 0 or 1.

A $2^n$-to-1 mux can be implemented as $2^n - 1$ 2-to-1 mux’s.

- The circuit diagram is simplified as a binary tree.
- Each node is a mux controlled by the input inside it, with data coming from either branch.
Nodes could be removed or merged.

- The overall structure is a binary decision diagram.
  - It is not a binary tree any more.
  - But the number of nodes are reduced.
Nodes could be removed or merged.

The overall structure is a binary decision diagram.
  
  - It is not a binary tree any more.
  - But the number of nodes are reduced.
A BDD is stored utilizing the binary tree data structure.

- Unlike usual binary trees, subtrees may be shared in a BDD.
- For multiple Boolean functions, multiple BDDs should be constructed.
  - Unlike truth tables, we cannot extend the leaves of BDDs from a simple bit to multiple bits, as that may prevent sharing.
  - Subtrees may be shared among multiple BDDs to save storage.
- Time complexity to compute an output: $O(nm)$
  - Computing each of the $m$ output bits requires a traversal of the BDD from root to leaf, taking at most $O(n)$ time.
- Space complexity: $O(N)$ for $N$ nodes
  - $N$ could be as large as $O(m2^n)$ in the worst case, but is usually much smaller.
BDD Implementations

- A BDD is stored utilizing the binary tree data structure.
  - Unlike usual binary trees, subtrees may be shared in a BDD.
- For multiple Boolean functions, multiple BDDs should be constructed.
  - Unlike truth tables, we cannot extend the leaves of BDDs from a simple bit to multiple bits, as that may prevent sharing.
  - Subtrees may be shared among multiple BDDs to save storage.
- Time complexity to compute an output: $O(nm)$
  - Computing each of the $m$ output bits requires a traversal of the BDD from root to leaf, taking at most $O(n)$ time.
- Space complexity: $O(N)$ for $N$ nodes
  - $N$ could be as large as $O(m2^n)$ in the worst case, but is usually much smaller.
BDD Implementations

- A BDD is stored utilizing the binary tree data structure.
  - Unlike usual binary trees, subtrees may be shared in a BDD.
- For multiple Boolean functions, multiple BDDs should be constructed.
  - Unlike truth tables, we cannot extend the leaves of BDDs from a simple bit to multiple bits, as that may prevent sharing.
  - Subtrees may be shared among multiple BDDs to save storage.
- Time complexity to compute an output: $O(nm)$
  - Computing each of the $m$ output bits requires a traversal of the BDD from root to leaf, taking at most $O(n)$ time.
- Space complexity: $O(N)$ for $N$ nodes
  - $N$ could be as large as $O(m2^n)$ in the worst case, but is usually much smaller.
A BDD is stored utilizing the binary tree data structure.
  ▶ Unlike usual binary trees, subtrees may be shared in a BDD.
▶ For multiple Boolean functions, multiple BDDs should be constructed.
  ▶ Unlike truth tables, we cannot extend the leaves of BDDs from a simple bit to multiple bits, as that may prevent sharing.
  ▶ Subtrees may be shared among multiple BDDs to save storage.
▶ Time complexity to compute an output: $O(nm)$
  ▶ Computing each of the $m$ output bits requires a traversal of the BDD from root to leaf, taking at most $O(n)$ time.
▶ Space complexity: $O(N)$ for $N$ nodes
  ▶ $N$ could be as large as $O(m2^n)$ in the worst case, but is usually much smaller.
Discussions

- BDDs may be further simplified by allowing negations along the edges.
  - This is especially helpful to reduce the number of nodes in XOR-rich circuits, e.g. a parity generator.
- Number of nodes in a BDD depends on the ordering of inputs.
- Software packages are available to handle BDDs.
  - Decide a good ordering to reduce number of nodes.
  - Share subtrees whenever possible.
  - Compose BDDs using Boolean operations (AND, OR, etc).
- However, in the worse case, a BDD may still incur prohibitive memory usage as a truth table.
BDDs may be further simplified by allowing negations along the edges.

This is especially helpful to reduce the number of nodes in XOR-rich circuits, e.g. a parity generator.

Number of nodes in a BDD depends on the ordering of inputs.

Software packages are available to handle BDDs.

Decide a good ordering to reduce number of nodes.
Share subtrees whenever possible.
Compose BDDs using Boolean operations (AND, OR, etc).

However, in the worse case, a BDD may still incur prohibitive memory usage as a truth table.
Discussions

- BDDs may be further simplified by allowing negations along the edges.
  - This is especially helpful to reduce the number of nodes in XOR-rich circuits, e.g. a parity generator.
- Number of nodes in a BDD depends on the ordering of inputs.
- Software packages are available to handle BDDs.
  - Decide a good ordering to reduce number of nodes.
  - Share subtrees whenever possible.
  - Compose BDDs using Boolean operations (AND, OR, etc).
- However, in the worse case, a BDD may still incur prohibitive memory usage as a truth table.
Discussions

- BDDs may be further simplified by allowing negations along the edges.
  - This is especially helpful to reduce the number of nodes in XOR-rich circuits, e.g. a parity generator.

- Number of nodes in a BDD depends on the ordering of inputs.

- Software packages are available to handle BDDs.
  - Decide a good ordering to reduce number of nodes.
  - Share subtrees whenever possible.
  - Compose BDDs using Boolean operations (AND, OR, etc).

- However, in the worse case, a BDD may still incur prohibitive memory usage as a truth table.
Both combinational and synchronous sequential circuits can be represented as truth tables.

With the same worst-case space complexity as truth tables, BDDs usually require much less storage and are widely used in practice.