Reading Assignment

- This lecture: 2.4
- Next lecture: 2.4
Outline

And-Inverter Graph

External Representation of Structural Models

Structural Properties
Motivation

- Both truth table and BDD have a worst-case space complexity of $O(2^n)$.
  - Is there a model that can represent all $n$-input Boolean functions using less than $2^n$ bits?
  - No.
    - With less than $2^n$ bits (1 bits, 2 bits, ..., $2^n - 1$ bits), one can represent at most $2^{2^n} - 2$ different symbols.
    - But there are $2^{2^n}$ different $n$-input Boolean functions.
- However, we don’t need to represent all $n$-input Boolean functions.
  - We just need to represent the Boolean functions that can be specified by designers.
Motivation

- Both truth table and BDD have a worst-case space complexity of $O(2^n)$.
  - Is there a model that can represent all $n$-input Boolean functions using less than $2^n$ bits?
  - No.
  - With less than $2^n$ bits (1 bits, 2 bits, ..., $2^n - 1$ bits), one can represent at most $2^{2^n} - 2$ different symbols.
  - But there are $2^{2^n}$ different $n$-input Boolean functions.
- However, we don’t need to represent all $n$-input Boolean functions.
  - We just need to represent the Boolean functions that can be specified by designers.
Both truth table and BDD have a worst-case space complexity of $O(2^n)$.

- Is there a model that can represent all $n$-input Boolean functions using less than $2^n$ bits?

- No.

  - With less than $2^n$ bits (1 bits, 2 bits, ..., $2^n - 1$ bits), one can represent at most $2^{2^n} - 2$ different symbols.
  - But there are $2^{2^n}$ different $n$-input Boolean functions.

- However, we don’t need to represent all $n$-input Boolean functions.

  - We just need to represent the Boolean functions that can be specified by designers.
Graph-Based Representation

- Assume Boolean functions are specified as Boolean expressions.
  - One can use a graph to represent them.
    - Nodes are inputs, Boolean operations, or outputs.
    - Directed edges are intermediate results.
  - If each operation has at most two operands, then a data structure similar to BDD can be used for implementation.
- Let $N$ be the length of the expressions.
  - Space complexity: $O(N)$.
  - Time complexity: $O(N)$ (as we will show later)
- However, removing redundancy is difficult.
  - Take long time to optimize.
  - May use more memory than BDD.
Assume Boolean functions are specified as Boolean expressions.

One can use a graph to represent them.

- Nodes are inputs, Boolean operations, or outputs.
- Directed edges are intermediate results.

If each operation has at most two operands, then a data structure similar to BDD can be used for implementation.

Let $N$ be the length of the expressions.

- Space complexity: $O(N)$.
- Time complexity: $O(N)$ (as we will show later)

However, removing redundancy is difficult.

- Take long time to optimize.
- May use more memory than BDD.
Graph-Based Representation

- Assume Boolean functions are specified as Boolean expressions.
- One can use a graph to represent them.
  - Nodes are inputs, Boolean operations, or outputs.
  - Directed edges are intermediate results.
- If each operation has at most two operands, then a data structure similar to BDD can be used for implementation.
  - Let $N$ be the length of the expressions.
    - Space complexity: $O(N)$.
    - Time complexity: $O(N)$ (as we will show later)
- However, removing redundancy is difficult.
  - Take long time to optimize.
  - May use more memory than BDD.
Assume Boolean functions are specified as Boolean expressions.

One can use a graph to represent them.

- Nodes are inputs, Boolean operations, or outputs.
- Directed edges are intermediate results.

If each operation has at most two operands, then a data structure similar to BDD can be used for implementation.

Let $N$ be the length of the expressions.

- Space complexity: $O(N)$.
- Time complexity: $O(N)$ (as we will show later)

However, removing redundancy is difficult.

- Take long time to optimize.
- May use more memory than BDD.
Assume Boolean functions are specified as Boolean expressions.

One can use a graph to represent them.

- Nodes are inputs, Boolean operations, or outputs.
- Directed edges are intermediate results.

If each operation has at most two operands, then a data structure similar to BDD can be used for implementation.

Let $N$ be the length of the expressions.

- Space complexity: $O(N)$.
- Time complexity: $O(N)$ (as we will show later)

However, removing redundancy is difficult.

- Take long time to optimize.
- May use more memory than BDD.
And-Inverter Graph (AIG)

- All Boolean operations can be decomposed as a number of and/not operations.
  - AIG: a special graph-based representation
    - Operation nodes are all 2-input and gate.
    - Every edge has the option to be negated (not gate).
  - Complexities remain the same as the generic graph-based representation.
- Redundancy can be removed efficiently.
  - AIGs are constructed in a way such that many equivalent sub-expressions can be shared.
  - Logic minimizations are decomposed into a series of local changes.
  - More scalable than BDD memory-wise.
And-Inverter Graph (AIG)

- All Boolean operations can be decomposed as a number of and/not operations.
- AIG: a special graph-based representation
  - Operation nodes are all 2-input and gate.
  - Every edge has the option to be negated (not gate).
- Complexities remain the same as the generic graph-based representation.
- Redundancy can be removed efficiently.
  - AIGs are constructed in a way such that many equivalent sub-expressions can be shared.
  - Logic minimizations are decomposed into a series of local changes.
  - More scalable than BDD memory-wise.
And-Inverter Graph (AIG)

- All Boolean operations can be decomposed as a number of and/not operations.
- AIG: a special graph-based representation
  - Operation nodes are all 2-input and gate.
  - Every edge has the option to be negated (not gate).
- Complexities remain the same as the generic graph-based representation.
- Redundancy can be removed efficiently.
  - AIGs are constructed in a way such that many equivalent sub-expressions can be shared.
  - Logic minimizations are decomposed into a series of local changes.
  - More scalable than BDD memory-wise.
And-Inverter Graph (AIG)

- All Boolean operations can be decomposed as a number of and/not operations.
- AIG: a special graph-based representation
  - Operation nodes are all 2-input and gate.
  - Every edge has the option to be negated (not gate).
- Complexities remain the same as the generic graph-based representation.
- Redundancy can be removed efficiently.
  - AIGs are constructed in a way such that many equivalent sub-expressions can be shared.
  - Logic minimizations are decomposed into a series of local changes.
  - More scalable than BDD memory-wise.
Discussions

- **Purpose of functional models**
  - Simulation: compute outputs from inputs.
  - Synthesis: generate implementations automatically.
  - Verification: decide whether two designs are equivalent.

- Simulation and verification are efficient for BDDs.
  - Scalability is a big concern for large circuits because of space complexity.

- Graph-based representations are scalable in memory usage.
  - Good for synthesis as they are closely related to structural models.
  - However, optimization is difficult.

- AIGs allow efficient optimizations.
  - Lead to efficient simulation and synthesis tools.
  - Improve the scalability of SAT solvers.
Discussions

- **Purpose of functional models**
  - Simulation: compute outputs from inputs.
  - Synthesis: generate implementations automatically.
  - Verification: decide whether two designs are equivalent.

- **Simulation and verification are efficient for BDDs.**
  - Scalability is a big concern for large circuits because of space complexity.

- Graph-based representations are scalable in memory usage.
  - Good for synthesis as they are closely related to structural models.
  - However, optimization is difficult.

- AIGs allow efficient optimizations.
  - Lead to efficient simulation and synthesis tools.
  - Improve the scalability of SAT solvers.
Discussions

- **Purpose of functional models**
  - Simulation: compute outputs from inputs.
  - Synthesis: generate implementations automatically.
  - Verification: decide whether two designs are equivalent.

- **Simulation and verification are efficient for BDDs.**
  - Scalability is a big concern for large circuits because of space complexity.

- **Graph-based representations are scalable in memory usage.**
  - Good for synthesis as they are closely related to structural models.
  - However, optimization is difficult.

- **AIGs allow efficient optimizations.**
  - Lead to efficient simulation and synthesis tools.
  - Improve the scalability of SAT solvers.
Discussions

- **Purpose of functional models**
  - Simulation: compute outputs from inputs.
  - Synthesis: generate implementations automatically.
  - Verification: decide whether two designs are equivalent.

- **Simulation and verification are efficient for BDDs.**
  - Scalability is a big concern for large circuits because of space complexity.

- **Graph-based representations are scalable in memory usage.**
  - Good for synthesis as they are closely related to structural models.
  - However, optimization is difficult.

- **AIGs allow efficient optimizations.**
  - Lead to efficient simulation and synthesis tools.
  - Improve the scalability of SAT solvers.
Outline

And-Inverter Graph

External Representation of Structural Models

Structural Properties
A minimal structural model should include

- Inputs, outputs, and components (gates).
- The way to interconnect them with signals (nets).
A minimal structural model should include
- Inputs, outputs, and components (gates).
- The way to interconnect them with signals (nets).

(Abramovici et al., 1990)
Hierarchical Design

Terminals of a component may have different functionalities.

It is helpful to state the bindings to predefined names.

Hierarchical design allows to define new component types.

Predefined components are organized into a library.

We say components used in a circuit are instances of generic components defined in a library.

(U1 JKFF Q=ERR, QB=NOTERR, J=ERRDET, K=A, C=CLOCK1)

(Fig. 2.16, Abramovici et al., 1990)
Hierarchical Design

- Terminals of a component may have different functionalities.
  - It is helpful to state the bindings to predefined names.
- Hierarchical design allows to define new component types.
  - Predefined components are organized into a library.
  - We say components used in a circuit are instances of generic components defined in a library.

\[ U1 \text{ JKFF } Q=\text{ERR}, \quad QB=\text{NOTERR}, \quad J=\text{ERRDET}, \quad K=A, \quad C=\text{CLOCK1} \]

(Fig. 2.16, Abramovici et al., 1990)
Textual formats are usually preferred over graphical ones as external representations for large designs.

You may be more familiar with more sophisticated representations like structural VHDL or structural Verilog.

- Though it is much more difficult to write a compiler.

Timing (delay) may be added to such models.
Textual formats are usually preferred over graphical ones as external representations for large designs.

You may be more familiar with more sophisticated representations like structural VHDL or structural Verilog.

- Though it is much more difficult to write a compiler.

- Timing (delay) may be added to such models.
Textual formats are usually preferred over graphical ones as external representations for large designs.

You may be more familiar with more sophisticated representations like structural VHDL or structural Verilog.

- Though it is much more difficult to write a compiler.

Timing (delay) may be added to such models.
Outline

And-Inverter Graph

External Representation of Structural Models

Structural Properties
Graph-Based Representations

- Similar to graph-based representations for functional models.
- If every net connects only two components,
  - Nodes represent components instead of Boolean expressions.
  - Directed edges represent (physical) nets, though signals on them still correspond to intermediate results.

Concerns
- What if the signal on a net has more than one destinations (i.e. having fanout)?
- We cannot restrict the number of incoming edges to each node so similar data structure like AIGs, BDDs, and binary trees cannot be used.
Graph-Based Representations

- Similar to graph-based representations for functional models.
- If every net connects only two components,
  - Nodes represent components instead of Boolean expressions.
  - Directed edges represent (physical) nets, though signals on them still correspond to intermediate results.

Concerns

- What if the signal on a net has more than one destinations (i.e. having fanout)?
- We cannot restrict the number of incoming edges to each node so similar data structure like AIGs, BDDs, and binary trees cannot be used.
Similar to graph-based representations for functional models.

If every net connects only two components,
- Nodes represent components instead of Boolean expressions.
- Directed edges represent (physical) nets, though signals on them still correspond to intermediate results.

Concerns
- What if the signal on a net has more than one destinations (i.e. having fanout)?
- We cannot restrict the number of incoming edges to each node so similar data structure like AIGs, BDDs, and binary trees cannot be used.
Fanout-Free Circuits

- Fanout-free circuits are represented by trees.
- Obviously, most circuits are not fanout-free.

Figure 2.17  Fanout-free circuit and its graph (tree) representation

(Abramovici et al., 1990)
Fanout-Free Circuits

Fanout-free circuits are represented by trees.

Obviously, most circuits are not fanout-free.

Figure 2.17  Fanout-free circuit and its graph (tree) representation

(Abramovici et al., 1990)
Bipartite Graph Model

Nodes are introduced to represent nets.

(Abramovici et al., 1990)
Bipartite Graph Model Explained

Two groups of nodes
- Elements: inputs, outputs, and components.
- Signals: nets

Edges are between nodes from different groups.
- Either from element to signal,
- or from signal to element.

Each signal has one incoming edge.

(Abramovici et al., 1990)
Bipartite Graph Model Explained

- Two groups of nodes
  - Elements: inputs, outputs, and components.
  - Signals: nets
- Edges are between nodes from different groups.
  - Either from element to signal,
  - or from signal to element.
- Each signal has one incoming edge.

(Abramovici et al., 1990)
Bipartite Graph Model Explained

- Two groups of nodes
  - Elements: inputs, outputs, and components.
  - Signals: nets
- Edges are between nodes from different groups.
  - Either from element to signal,
  - or from signal to element.
- Each signal has one incoming edge.

(Abramovici et al., 1990)
Logic Levels

▶ For a combinational circuit, a (logic) level can be assigned to each element.
  ▶ Such that signals only propagate from an element with a lower level to an element with a higher level.
  ▶ This also applies to the combinational part of a sequential circuit.

▶ Essentially, levels model causality among events associated with signal propagation.
  ▶ Very useful when the ordering of events matter, e.g. for simulation and for timing analysis.

▶ We may compute the level of an element as its longest “distance” from the primary inputs (PIs).
  ▶ Count the distance as the number of nets a signal need to pass from a PI to the element.

▶ We will discuss how to actually compute the levels, i.e. the algorithm itself, after we learn the data structure for the bipartite graph model.
Logic Levels

- For a combinational circuit, a (logic) level can be assigned to each element.
  - Such that signals only propagate from an element with a lower level to an element with a higher level.
  - This also applies to the combinational part of a sequential circuit.
- Essentially, levels model causality among events associated with signal propagation.
  - Very useful when the ordering of events matter, e.g. for simulation and for timing analysis.
- We may compute the level of an element as its longest “distance” from the primary inputs (PIs).
  - Count the distance as the number of nets a signal need to pass from a PI to the element.
- We will discuss how to actually compute the levels, i.e. the algorithm itself, after we learn the data structure for the bipartite graph model.
Logic Levels

- For a combinational circuit, a (logic) level can be assigned to each element.
  - Such that signals only propagate from an element with a lower level to an element with a higher level.
  - This also applies to the combinational part of a sequential circuit.
- Essentially, levels model causality among events associated with signal propagation.
  - Very useful when the ordering of events matter, e.g. for simulation and for timing analysis.
- We may compute the level of an element as its longest “distance” from the primary inputs (PIs).
  - Count the distance as the number of nets a signal need to pass from a PI to the element.
- We will discuss how to actually compute the levels, i.e. the algorithm itself, after we learn the data structure for the bipartite graph model.
Logic Levels

- For a combinational circuit, a (logic) level can be assigned to each element.
  - Such that signals only propagate from an element with a lower level to an element with a higher level.
  - This also applies to the combinational part of a sequential circuit.
- Essentially, levels model causality among events associated with signal propagation.
  - Very useful when the ordering of events matter, e.g. for simulation and for timing analysis.
- We may compute the level of an element as its longest “distance” from the primary inputs (PIs).
  - Count the distance as the number of nets a signal need to pass from a PI to the element.
- We will discuss how to actually compute the levels, i.e. the algorithm itself, after we learn the data structure for the bipartite graph model.
Example Logic Levels

(Fig. 20, Abramovici et al., 1990)
Summary

- AIG is a scalable functional representation.
- Structure models specify components and interconnects, and can include hierarchy into the system.