Outline

1. Multilevel minimization

2. Timing
Section outline

1. Multilevel minimization
   Examples
   Ad-hoc method
   Kernel extraction
Multi-level example

- So far, we have talked about two-level logic
  - Fewest number of levels that is fully general
- However, there are some circuits that can more efficiently be implemented using three or more levels
- Even basic functions, like NANDS and NORS, are commonly composed of multiple gates if the fan-in is high
Two-level form

\[ X(A, B, C, D, E, F, G) = ADF + AEF + BDF + BEF + CDF + CEF + G \]
Two-level form

- 6 x 3-input AND gates
- 1 x 7-input OR gate (may not exist!)
- 25 wires (19 literals plus 6 internal wires)
Multi-level form

Factor:

\[ X(A, B, C, D, E, F, G) = ADF + AEF + BDF + BEF + CDF + CEF + G \]
\[ = (A + B + C)(D + E)F + G \]
Multi-level form

- 1 x 3-input OR gate, 2 x 2-input OR gates
- 1 x 3-input AND gate
- 10 wires (7 literals plus 3 internal wires)
Multi-level implementation of NAND8

Convert middle ANDs to NORs with inverted inputs
Multi-level implementation of NAND8

Convert middle ANDs to NORs with inverted inputs
Multi-level implementation of NAND8
Multi-level implementation of NAND8

Convert middle ANDs to NORs with inverted inputs.
Multi-level implementation of NAND8

Convert middle ANDs to NORs with inverted inputs
Multi-level logic

\[ F = A \left( B + C \cdot D \right) + B \]

Original
AND–OR Network

Introduction and
Conservation of Bubbles

Redrawn in terms
of conventional
NAND Gates
Multi-level logic

Same beginning network after introduction of bubbles

Final network, redrawn in NOR-only form
NAND/NOR conversion for uneven paths

Original circuit

Add double bubbles at inputs

Distribute bubbles
some mismatches

Insert inverters to fix mismatches
Section outline

1. Multilevel minimization
   Examples
   Ad-hoc method
   Kernel extraction
Multi-level minimization objectives

- Factor out common sub-logic (reduce fan-in, increase gate levels), subject to timing constraints
- Map factored form onto library of gates
- Minimize number of literals (correlates with number of wires)

\[ X = (AB + \overline{B} C)(C + D(E + A\overline{C})) + (D + E)(FG) \]
Multi-level minimization

Can interactively apply the following operations

- Decomposition
- Extraction
- Factoring
- Substitution
- Collapsing
Recall that two-level minimization is difficult
- Quine-McCluskey is too slow for large problems
- Resort to potentially sub-optimal heuristic
  - Espresso

Multi-level minimization is much harder
- No known efficient algorithm gives optimal solution
- MIS uses heuristics to usually produce good solutions
Decomposition

Take a single Boolean expression and replace it with a collection of new expressions

\[ L = ABC + ABD + \overline{A} \overline{C} \overline{D} + \overline{B} \overline{C} \overline{D} \quad 12 \text{ literals} \]

\( L \) rewritten as

\[ L = MN + \overline{M} \overline{N} \quad 8 \text{ literals} \]

\[ M = AB \]

\[ N = C + D \]
Decomposition

\[ L = ABC + ABD + \overline{A} \overline{C} \overline{D} + B \overline{C} \overline{D} \]
\[ = AB(C + D) + (\overline{A} + \overline{B})\overline{C} \overline{D} \]
\[ = AB(C + D) + (AB)(C + D) \]
\[ M = AB \]
\[ N = C + D \]
\[ L = MN + \overline{M} \overline{N} \]
Decomposition

Before Decomposition

After Decomposition
Extraction

- Extraction is decomposition applied to multiple functions.
- The best decomposition for a single function may not be the best if there are multiple outputs.
- Ideally, extracted sub-function can be re-used in producing many outputs.
Extraction

\[ L = (A + B)CD + E \quad \text{11 literals} \]
\[ M = (A + B)\overline{E} \]
\[ N = CDE \]

Can be re-written as

\[ L = XY + E \quad \text{11 literals} \]
\[ M = X\overline{E} \]
\[ N = YE \]
\[ X = A + B \]
\[ Y = CD \]
Extraction

**Before Extraction**
Gate count has improved, literals is unchanged

**After Extraction**

Gate count has improved, literals is unchanged
Extraction example

Given equations of a Boolean Network

\[ L = AF + BF + AG + CG + ADE + BDE + CDE \] 33 literals
\[ M = AF + BF + ACE + BCE \]
\[ N = ADE + CDE \]

Find the best kernel of these functions. If the kernel is \((A + B)\), then

\[ L = DEM + FP + AG + CG \] 22 literals
\[ M = CEP + FP \]
\[ N = ADE + CDE \]
\[ P = A + B \]
Cube extraction

Can restrict divisors to single cubes

\[ L = ABC + ABD + EG \]  \hspace{2cm} \text{16 literals}
\[ M = ABFG \]
\[ N = BD + EF \]

We can obtain the best cube \( AB \) to divide the functions

\[ L = PC + PD + EG \]  \hspace{2cm} \text{15 literals}
\[ M = PFG \]
\[ N = BD + EF \]
\[ P = AB \]
Expression in two-level form re-expressed in multi-level form without introducing intermediate functions

\[ L = AC + AD + BC + BD + E \]  
9 literals

Can be rewritten as

\[ L = (A + B)(C + D) + E \]  
5 literals
Factoring

Before Factoring

After Factoring
Substitution

Express $L$ in terms of $M$

$L = A + BC$

$M = A + B$

$L$ rewritten in terms of $M$

$L = M(A + C)$

$M = A + B$
Collapsing

Reverse of substitution. Sometimes eliminates levels to meet timing constraints.

\[ L = M(A + C) \]
\[ M = A + B \]

5 literals

only 2 additional

\[ L = (A + B)(A + C) \]
\[ = AA + AC + AB + BC \]
\[ = A + BC \]
Redundancy

- Minimizing gate count and literals isn’t always good for performance or power consumption
- Using redundant sub-circuits can result in improvements
- The wiring required to re-use sub-functions can result in larger delays than redundant sub-functions
Boolean division

\[ L = PQ + R \]

- **P** divisor
- **Q** quotient
- **R** remainder

\[ X = AC + AD + BC + BD + E \]
\[ Y = A + B \]

\( X \) divided by \( Y \) expressed as

\[ X = Y(C + D) + E \]

\( P \) and \( Q \) are symmetrical
Finding divisors that lead greatest number of common subexpressions is difficult

\[ L = AD + BCD + E \]
\[ M = A + B \]

\( M \) does not divide \( L \) under algebraic division rules

\( M \) does divide \( L \) under Boolean rules (very large number of these!)

\[ L / M = (A + C)D \]
Boolean division

\[ L = AD + BCD + E \]
\[ M = A + B \]
\[ L = (A + B)(AD + CD) + E \]
\[ = (A + B)(A + C)D + E \]
Boolean division

This follows from writing $L$ in $MQ + R$ form

$L = M((A + C)D) + E$

$= (A + B)(A + C)D + E$

$= (AA + AC + AB + BC)D + E$

$= (A + BC)D + E$

$= AD + BCD + E$
Section outline

1. Multilevel minimization
   Examples
   Ad-hoc method
   Kernel extraction
Multi-level minimization definitions

- A variable is a symbol representing a single coordinate in a Boolean space, e.g., $A$ or $B$
- Literal is a variable or its negation, e.g., $A$ or $\overline{A}$
- A cube, $C$, is a set of literals, e.g., $A \ B \ \overline{C}$ or $A \ \overline{B}$
- Consider a sum-of-products expression
  $$L = AF + BF + AG + CG + ADE + BDE + CDE$$
- The primary divisors of an expression form a set of expressions
  $$D(L) = \{L/C|C \text{ is a cube}\}$$
Multi-level minimization definitions

- Equation is cube-free if no cubes divide it evenly
- \( AB + C \) is cube-free
- \( AB + AC \) is not cube-free
  - Divided evenly by \( A \)
Multi-level minimization definitions

- The kernels of an expression are sets of expressions
  \[ K(L) = \{ G | G \in D(L) \text{ and } G \text{ is cube-free} \} \]
- In other words, the kernels are divisors of the function for cube quotients that can not be evenly divided (remainder-free) by other cubes
- The cube \( C \) used to divide \( L \) to obtain the kernel \( K(L) \) is called the co-kernel
\[ L = AF + BF + AG + CG + ADE + BDE + CDE \]

<table>
<thead>
<tr>
<th>Cube</th>
<th>Primary divisor</th>
<th>Kernel (cube free)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>(F + G + DE)</td>
<td>(k_1)</td>
</tr>
<tr>
<td>B</td>
<td>(F + DE)</td>
<td>(k_2)</td>
</tr>
<tr>
<td>C</td>
<td>(G + DE)</td>
<td>(k_3)</td>
</tr>
<tr>
<td>D</td>
<td>(AE + BE + CE)</td>
<td>no</td>
</tr>
<tr>
<td>E</td>
<td>(AD + BD + CD)</td>
<td>no</td>
</tr>
<tr>
<td>F</td>
<td>(A + B)</td>
<td>(k_4)</td>
</tr>
<tr>
<td>G</td>
<td>(A + C)</td>
<td>(k_5)</td>
</tr>
<tr>
<td>DE</td>
<td>(A + B + C)</td>
<td>(k_6)</td>
</tr>
</tbody>
</table>
Divisor selection

Kernels are often good divisors, extract and substitute

$$L = AF + BF + AG + CG + ADE + BDE + CDE$$  \hspace{1cm} 17 \text{ literals}$$

Divide by \( k_6 = (A + B + C) \)

$$L = AF + BF + AG + CG + DEk_6$$

Remaining kernels: \( k_1 = (F + G), k_4 = (A + B), k_5 = (A + C) \)

Pick one

$$L = Fk_4 + AG + CG + DEk_6$$
Divisor selection

\[ L = Fk_4 + AG + CG + DEk_6 \]

Remaining kernel: \( k_5 = (A + C) \)

\[ L = Fk_4 + AG + CG + DEk_6 \]
\[ L = Fk_4 + Gk_5 + DEk_6 \]

End result

\[ L = F(A + B) + G(A + C) + DE(A + B + C) \quad 11 \text{ literals} \]
During multi-level logic synthesis, representation broken into sub-functions

Each sub-function (e.g., SOP form) needs to be minimized

\[ L = AF + BF + AG + CG + ADE + BDE + CDE \]

Similar to using Karnaugh Maps, Quine McCluskey, or Espresso to simply expressions

\[ L = \bar{A} B + AC + BC \]
\[ = \bar{A} B + AC \]
Node simplification

F(A, B, C)
Node simplification

- Recall that two-level simplification can be improved using **Don’t Care** information
- Need to find **Don’t Care** automatically from Boolean network
- Take advantage of satisfiability **Don’t Cares**
  - Input can never occur
- Take advantage of observability **Don’t Cares**
  - Output is ignored for certain inputs
Outline

1. Multilevel minimization

2. Timing
Section outline

2. Timing
  Delay models
  Thresholds, noise, and debouncing
  Hazards
Delay models

Delay models

A → B

A

B

time
Delay models

A → B

Ideal

A

B

time

Ideal

46 Robert Dick Advanced Digital Logic Design
Delay models

A \rightarrow B

\begin{align*}
A & \\
B & \\
time & \\
\end{align*}
Delay models

**Ideal**

**Transport**

**Inertial**

**Real**
Delay models

A \rightarrow B

A

B

time
Delay models

Ideal

Transport

Inertial

Real

\[ A \rightarrow B \]

\[ \text{Inertial} \]

\[ \text{time} \]
Delay models

![Diagram showing two signals A and B with a delay model](image)

- **A**
- **B**

**time**
Delay models

A → B

A

B

Real

time
Delay models

A --→ B

A

B

time

Ideal

Transport

Inertial

Real

Multilevel minimization
Timing

Delay models
Thresholds, noise, and debouncing
Hazards

Robert Dick
Advanced Digital Logic Design
Delay models

- Ideal – No delay, outputs respond instantly to inputs
- Transport – Output shifted by some fixed duration
  - All input changes reflected at output
Delay models

- Inertial – Transitions that are not stable for more than some threshold period of time are absorbed by the gate
  - Provides a coarse approximation to real digital logic gate behavior
- Real – Outputs quickly respond to input changes
  - However, output changes have limited slew-rates
RC delay

- What happens when a transistor drives another transistor?
- Recall how CMOS transistors work
IO thresholds
IO thresholds
IO thresholds
IO thresholds
IO thresholds
IO thresholds
IO thresholds
IO thresholds
IO thresholds

![Graph of IO thresholds](image-url)
Section outline

2. Timing
   Delay models
   Thresholds, noise, and debouncing
   Hazards
**IO thresholds**

- *Digital* is a simplifying concept in a (macroscopically) *analog* world
- To treat signals as digital, need to define a dividing line between false and true
- What actually happens to the next stage when the voltage is at that line?
  - NMOS $V_{GS} > V_T$
  - PMOS $V_{GS} < -V_T$
Digital is a simplifying concept in a (macroscopically) analog world.
To treat signals as digital, need to define a dividing line between false and true.
What actually happens to the next stage when the voltage is at that line?
- NMOS $V_{GS} > V_T$
- PMOS $V_{GS} < -V_T$
Molten transistor
Molten transistor

A=0
B=1

A=1
B=0

A=?
B=?
Molten transistor
Molten transistor

A=1
B=0

A=0
B=1

A=?
B=?
Molten transistor
Molten transistor
IO thresholds

- Define valid ranges of logic high and logic low signals
- Undefined signals considered invalid
IO thresholds

- Better – However, still have a problem
- What happens if the output of a gate is near that threshold?
- Slight variation between gates might result in crossing this threshold for next gate
- How to compensate?
- Use separate input and output thresholds
IO thresholds

- Better – However, still have a problem
- What happens if the output of a gate is near that threshold?
- Slight variation between gates might result in crossing this threshold for next gate
- How to compensate?
- Use separate input and output thresholds
Separate IO thresholds

- Now, we can safely treat the system as digital
- However, digital systems talk with analog systems
- It is necessary to deal with noisy signals
- Real slew isn’t ideal
Schmitt triggers

A → B

A

57 Robert Dick
Advanced Digital Logic Design
Schmitt triggers

A → B

A

A
Schmitt triggers

A → B

A

B

57 Robert Dick

Advanced Digital Logic Design
Schmitt triggers

A → B

transition
Schmitt triggers

![Schmitt trigger circuit diagram]

- **A** transitions to **B**
- *transition* point highlighted with red arrows and dashed lines

---

**Advanced Digital Logic Design**

Robert Dick
Reason for gradual transition

- A logic stage is an RC network
- Whenever a transition occurs, capacitance is driven through resistance
- Consider the implementation of a CMOS inverter
CMOS NOT is RC network
CMOS NOT is RC network
CMOS NOT is RC network
CMOS NOT is RC network
CMOS NOT is RC network
CMOS NOT is RC network
What is driven in NAND2?
Debouncing

- Mechanical switches bounce!
- What happens if multiple pulses?
  - Multiple state transitions
- Need to clean up signal
Debouncing
Improving device delay

- Decrease driven resistance and load
  - Use smaller transistors
  - Use shorter wire
  - Use wider wire (can increase load)
Improving device delay

- Increase drive by increasing width of driving transistors
  - Causes problems
  - Larger transistors provide larger loads to their inputs
  - In general, keep transistors small unless they absolutely need to drive large loads
Driving large loads

- Sometimes large loads need to be driven
- Long wires
- Output pads
- What happens if we go from a minimum-size inverter to a huge inverter?
  - Huge delay driving huge inverter’s gate
Driving large loads

- Sometimes large loads need to be driven
- Long wires
- Output pads
- What happens if we go from a minimum-size inverter to a huge inverter?
  - Huge delay driving huge inverter’s gate
Tapered buffer chain

- Instead, gradually increase buffers in chain
- Optimal number of stages: \( \ln \left( \frac{C_{BIG}}{C_{SMALL}} \right) \)
- Stage width exponentially increases in \( \alpha \)

\[
W_1 = W_{MIN} \cdot \alpha^0 \\
W_2 = W_{MIN} \cdot \alpha^1 \\
W_3 = W_{MIN} \cdot \alpha^2 \\
\ldots
\]
Useful delay: Debouncing

- Mechanical switches bounce
  - Noisy transition
- Use RC delay network to decrease transition speed
- Convert multiple noisy to single smooth transition
- Use Schmitt trigger to clean signal
Delay estimation in multi-level circuits

- Can get delay for a gate by knowing its drive (resistance) and the load it drives (RC)
  - Gate libraries will have this information
- Still need to get network delay
- Conduct topological sort of network to find earliest start times (EST)
- Always visit a node’s parent before visiting it
- EST is maximum any parent’s EST plus it’s delay
Topological sort

5 → 2 → 6 → 2 → 2 → 3

EST = max(9, 11) = 11
Topological sort

EST = 0

5

6

2

2

3

2

2

max(9, 11) 13

129
Topological sort

EST = 0

5

2

5

6

2

2

2

3
Topological sort

EST = 0

5

2

5

6

2

2

3

7

\[
\text{max}(9, 11) = 13
\]
Topological sort

EST = 0

5 6 2 7 3

max(9, 11) 13 129
Topological sort

EST = 0
Topological sort

EST = 0

5

2

6

7

3

2

5

9

129

max(9, 11) 15

13

9

Robert Dick
Advanced Digital Logic Design
Topological sort

![Graph showing topological sort](image)
Topological sort

EST = 0

\[
\begin{align*}
\text{2} & \rightarrow \text{7} \\
\text{3} & \rightarrow \phantom{\text{2}} \\
\text{5} & \rightarrow \phantom{\text{2}} \\
\text{6} & \rightarrow \phantom{\text{2}} \\
\text{2} & \rightarrow \phantom{\text{2}} \\
\text{13} & \rightarrow \phantom{\text{2}} \\
\end{align*}
\]

\[
\begin{align*}
\text{max}(9, 11) & \rightarrow \phantom{\text{2}} \\
\end{align*}
\]
Topological sort

EST = 0

max(9, 11) 13
Topological sort

EST = 0

max(9, 11) 13

15
Topological sort

- Node 5
  - EST = 0

- Node 2
  - Inputs: 5, 6
  - Output: 7

- Node 6
  - Input: 2

- Node 7
  - Input: 2

- Node 3
  - Inputs: 9
  - Output: 12

- Node 2
  - Inputs: 7, max(9, 11)
  - Output: 13

- Node 12

- Node 15

- Node 15

- Node 12

- Node 15

- Node 12

- Node 15

- Node 12

- Node 15

- Node 12
Slack

- Can use reverse topological sort from end nodes (given some target delay) to get latest start time (LST)
- Subtract delays and take minimum over children instead of adding delays and taking maximum over parents
- Subtract EST from LST to get node slack
- Gates with lowest slack are on critical path
- Make this path faster...
- ...or save area (at the expense of speed) on non-critical paths
Useful delay: Rising edge pulse shaping

Problem: Pulse width poorly controlled
Useful delay: Pulse rising/falling edge pulse shaping

Problem: Pulse width poorly controlled
Section outline

2. Timing
   Delay models
   Thresholds, noise, and debouncing
   Hazards
Dynamic hazards

- Potential for two or more spurious transitions before intended transition
- Results from uneven path delays in some multi-level circuits
Dynamic hazards
Eliminating dynamic hazards

- Some approaches allow preservation of multi-level structure
  - Quite complicated to apply
- Simpler solution – Convert to two-level implementation
Static hazards

- Still have static hazards
- Potential for transient change of output to incorrect value

\[
\begin{array}{cc}
1 & 1 \\
0 & \text{Static 1-hazard} \\
1 & \text{Static 0-hazard} \\
0 & 0
\end{array}
\]
Problems with glitches

- These transitions result in incorrect output values at some times.
- Also result in uselessly charging and discharging wire and gate capacitances through wire, gate, and channel resistances.
  - Increase power consumption.
Glitches increase power consumption

A 1

V_{DD} V_{DD}

V_{SS} V_{SS}

BA

V_{DD} V_{DD}

V_{SS} V_{SS}

1

Robert Dick

Advanced Digital Logic Design
Detecting hazards

- The observable effect of a hazard is a glitch
  - A circuit that might exhibit a glitch has a hazard
- Whether or not a hazard is observed as a glitch depends on relative gate delays
- Relative gate delays change depending on a number of factors—Conditions during fabrication, temperature, age, etc.
- Best to use abstract reasoning to determine whether hazards might be observed in practice, under some conditions
# Eliminating static hazards

Input change within product term

\[ F = \overline{A} \, D + A \, \overline{C} \]
Eliminating static hazards

\[ F = \overline{A} \overline{D} + A \overline{C} \]
Eliminating static hazards

\[ ABCD = 1101 \]

\[ ABCD = 0101 \ (\bar{A} \text{ is still } 0) \]

\[ ABCD = 0101 \ (\bar{A} \text{ is } 1) \]
Eliminating static hazards

- Add redundant primes covering all 1-1 transitions in SOP
- Add redundant primes covering all 0-0 transitions in POS
- Clearly primes can be used, consider contradiction stemming from assumption that non-prime is necessary to cover a transition
Where do static hazards really come from?

- **Static-0**: $A \overline{A}$
- **Static-1**: $A + \overline{A}$

Assume SOP form has no product terms containing a variable in complemented and uncomplemented forms
  - Reasonable assumption, if true, drop product term
Where do static hazards really come from?

- Assume POS form has no sum terms containing a variable in complemented and uncomplemented forms
  - Reasonable assumption, if true, drop sum term
- Assume only one input switches at a time
- Conclusion: SOP has no 0-hazards and POS has no 1-hazards
  - In other words, if you are doing two-level design, you need not analyze the other form for hazards
Living with hazards

Sometimes hazards can be tolerated

- Combinational logic whose outputs aren’t observed at all times
- Synchronous systems
- Systems without tight power consumption limits
Summary

- Brief review of cascaded carry lookahead adder
- Common ALU operations
- Overview of memory types
- Timing behavior