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
  - Even basic functions, like NANDs and NORS, are commonly composed of multiple gates if the fan-in is high

Multi-level example

- 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 implementation of NAND8

- 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)
Advanced Digital Logic Design

Decomposition

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

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)
- \[ X = (AB + \overline{C})(C + D(E + \overline{AC})) + (D + E)(FG) \]

Multi-level minimization

Can interactively apply the following operations
- Decomposition
- Extraction
- Factoring
- Substitution
- Collapsing

Decomposition

Take a single Boolean expression and replace it with a collection of new expressions
- \[ L = ABC + ABD + \overline{A} \overline{C} D + B \overline{C} \overline{D} \]
- 12 literals

\[ L \text{ rewritten as } L = MN + \overline{M} \overline{N} \]
- 8 literals

\[ M = AB \]

\[ N = C + D \]

\[ L = MN + \overline{M} \overline{N} \]

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

\[ L = (A + B)CD + E \]
\[ M = (A + B)E \]
\[ N = CDE \]

Can be re-written as
\[ L = XY + E \]
\[ M = XE \]
\[ N = YE \]
\[ X = A + B \]
\[ Y = CD \]

Cube extraction

Can restrict divisors to single cubes
\[ L = ABC + ABD + EG \]
\[ M = ABFG \]
\[ N = BD + EF \]

We can obtain the best cube \( AB \) to divide the functions
\[ L = PC + PD + EG \]
\[ M = PFG \]
\[ N = BD + EF \]
\[ P = AB \]

Factoring

Expression in two-level form re-expressed in multi-level form without introducing intermediate functions
\[ L = AC + AD + BC + BD + E \]

Can be rewritten as
\[ L = (A + B)(C + D) + E \]

Substitution

Express \( L \) in terms of \( M \)
\[ L = A + BC \]
\[ M = A + B \]

\( L \) rewritten in terms of \( M \)
\[ L = M(A + C) \]
\[ M = A + B \]
Reverse of substitution. Sometimes eliminates levels to meet timing constraints.

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

5 literals

only 2 additional

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

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.

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!)

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

This follows from writing \( L \) in \( MQ + R \) form.

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

\[ L = M( (A + C)D + E ) \]
\[ = (A + B)(A + C)D + E \]
\[ = (AA + AC + AB + BC)D + E \]
\[ = (A + BC)D + E \]
\[ = AD + BCD + E \]

Multi-level minimization definitions

- A variable is a symbol representing a single coordinate in a Boolean space, e.g., \( A \) or \( A \).
- Literal is a variable or its negation, e.g., \( A \) or \( A \).
- A cube, \( C \), is a set of literals, e.g., \( AB \) or \( A \).
- 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.
  \[ D(L) = \{ L/C | C \text{ is a cube} \} \]
Node simplification

- 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

Kernels are often good divisors, extract and substitute
\[ L = AF + BF + AG + CG + ADE + BDE + CDE \]

\[ L = AF + BF + AG + CG + DEk_6 \]

Remaining kernels: \( k_6 = (F + G) \), \( k_4 = (A + B) \), \( k_5 = (A + C) \)

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

F(A, B, C)

Delay models

- 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

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

\[
\begin{array}{|c|c|}
\hline
\text{Cube} & \text{Primary divisor} \\
\hline
A & \{F + G + DE\} \\
B & \{(F + DE)\} \\
C & \{(G + DE)\} \\
D & \{(AE + BE + CE)\} \\
E & \{(AD + BD + CD)\} \\
F & \{(A + B)\} \\
G & \{(A + C)\} \\
DE & \{(A + B + C)\} \\
\hline
\end{array}
\]
Delay models

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

\[ \text{Ideal} \quad \text{Transport} \]

Timing

- 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

\[ \text{Inertial} \quad \text{Real} \]

Delay models

- RC delay
  - What happens when a transistor drives another transistor?
  - Recall how CMOS transistors work

\[ \text{RC delay} \]

IO thresholds

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

\[ \text{IO thresholds} \]

Molten transistor

- 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 \( V_{GS} < -V_T \)

\[ \text{Molten transistor} \]

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

\[ \text{IO thresholds} \]
Multilevel minimization

Timing

Delay models

Thresholds, noise, and debouncing

Hazards

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

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

What is driven in NAND2?

Debouncing

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

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

Tapered buffer chain

- Instead, gradually increase buffers in chain
- Optimal number of stages: \( \ln \left( \frac{C_{\text{BIG}}}{C_{\text{SMALL}}} \right) \)
- Stage width exponentially increases in \( \alpha \)
- \( W_1 = W_{\text{MIN}} \cdot \alpha^0 \)
- \( W_2 = W_{\text{MIN}} \cdot \alpha^1 \)
- \( W_3 = W_{\text{MIN}} \cdot \alpha^2 \)
  . . .

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

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

- 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
Useful delay: Pulse rising/falling edge pulse shaping

**Dynamic hazards**

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

```
    0 1
    1 0
```

**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

```
    1 1
    0 0
```

**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

**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
Multilevel minimization
Timing
Delay models
Thresholds, noise, and debouncing
Hazards

Eliminating static hazards

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

- 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
- 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