CSE140: Components and Design Techniques for Digital Systems

Logic minimization algorithm summary
Definition of terms for two-level simplification

• ON-set:
  – the set of all 1s in the result of a logic function (i.e. all “boxes” of a Kmap where the value is 1)

• OFF-set:
  – the set of all 0s in the result of a logic function (i.e. all “boxes” of a Kmap where the value is 0)

• SOP: Sum of Products

• Canonical SOP = minterms expansion

• Minimal SOP = resulting from 2-level minimization (i.e. Kmaps) by covering 1s

• Similar definitions for POS

Sources: TSR, Katz, Boriello & Vahid
Definition of terms for two-level simplification

• Implicant
  – single element of ON-set or DC-set or any group of these elements that can be combined to form a subcube

• Prime implicant
  – implicant that can't be combined with another to form a larger subcube

• Essential prime implicant
  – prime implicant is essential if it alone covers an element of ON-set
  – will participate in ALL possible covers of the ON-set
  – DC-set used to form prime implicants but not to make implicant essential

• Cover:
  – a subset of implicants that covers all 1s in the Kmap

• Objective:
  – grow implicant into prime implicants (minimize literals per term)
  – cover the ON-set with as few prime implicants as possible (minimize number of product terms)
Examples to illustrate terms
Examples to illustrate terms

6 prime implicants: A'B'D, BC', AC, A'C'D, AB, B'CD
essential
minimum cover: AC + BC' + A'B'D

5 prime implicants: BD, ABC', ACD, A'BC, A'C'D
essential
minimum cover: 4 essential implicants
Algorithm for two-level simplification

- **Algorithm**: minimum sum-of-products expression from a Karnaugh map
  - **Step 1**: choose an element of the ON-set
  - **Step 2**: find "maximal" groupings of 1s and Xs adjacent to that element
    - consider top/bottom row, left/right column, and corner adjacencies
    - this forms prime implicants (number of elements always a power of 2)
  - Repeat Steps 1 and 2 to find all prime implicants
  - **Step 3**: revisit the 1s in the K-map
    - if covered by single prime implicant, it is essential, and participates in final cover
    - 1s covered by essential prime implicant do not need to be revisited
  - **Step 4**: if there remain 1s not covered by essential prime implicants
    - select the smallest number of prime implicants that cover the remaining 1s
Algorithm for two-level simplification (example)

2 primes around A'BC'D'

3 primes around AB'C'D'

2 primes around ABC'D

minimum cover (3 primes)

Sources: TSR, Katz, Boriello & Vahid
Essential primes

Which are the essential prime implicants?
A. CD’
B. BD
C. AC’D
D. All of the above
E. None of the above

For more practice: think about essential prime implicates!
CSE140: Components and Design Techniques for Digital Systems

Muxes and demuxes
Multiplexer (Example)

- Four possible display items
  - Temperature (T), Average miles-per-gallon (A), Instantaneous mpg (I), and Miles remaining (M) -- each is 8-bits wide
  - Choose which to display using two inputs x and y
  - Use 8-bit 4x1 mux

Sources: TSR, Katz, Boriello & Vahid
Multiplexer (Example)

- You are not sure whether an input of the MUX would have a 1 or a 0
- You should build the MUX with a component that is good at passing both 0s and 1s
Transmission Gate: Mux/Tristate building block

- nMOS are on when gate=1
  - good at passing 0s from source to drain
  - pass 1’s poorly from source to drain
- pMOS are on when gate=0
  - good at passing 1s from source to drain
  - pass 0’s poorly from source to drain
- Transmission gate is a better switch
  - passes both 0 and 1 well
- When $EN = 1$, the switch is ON:
  - $EN = 0$ and $A$ is connected to $B$
- When $EN = 0$, the switch is OFF:
  - $A$ is not connected to $B$
- *The pass transistor acts as a tristate buffer*
Floating: Z, Tristate Buffer and Tristate Busses

- Floating, high impedance, open, high Z
  - Disconnected
- Floating nodes are used in tristate busses
  - many different drivers, but only one is active at once

**Tristate Buffer**

<table>
<thead>
<tr>
<th>$E$</th>
<th>$A$</th>
<th>$Y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Z</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Z</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Note: do not confuse this with the inverter symbol!
2:1 Multiplexer or Mux

- Selects between one of $N$ inputs to connect to output
- $\log_2 N$-bit select input – control input
- **Example: 2:1 Mux**

### Logic gates
- Tristates
- Pass gates

### Truth table

<table>
<thead>
<tr>
<th>$S$</th>
<th>$D_1$</th>
<th>$D_0$</th>
<th>$Y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- $Y = D_0 S + D_1 S$

### Example circuit
Multiplexers/selectors

• 2:1 mux: \( Z = A'I_0 + AI_1 \)
• 4:1 mux: \( Z = A'B'I_0 + A'BI_1 + AB'I_2 + ABI_3 \)
• 8:1 mux: \( Z = A'B'C'I_0 + A'B'CI_1 + A'BC'I_2 + A'BCI_3 + AB'C'I_4 + AB'CI_5 + ABC'I_6 + ABCI_7 \)

• In general: \( \sum_{k=0}^{2^n-1} m_k I_k \)
  – shorthand form for a \( 2^n:1 \) Mux

For example
You can implement a 2-variables logic function using a 2:1 multiplexer:
- Use one variable for the selection input
- Connect the MUX inputs to either
  - The second variable (in true or complemented form)
  - GND
  - VDD
- You can also use larger MUXs (see next slide)
This multiplexer implements the same functionality for $Y$ as the truth table:

A. Yes

B. No

<table>
<thead>
<tr>
<th>$A$</th>
<th>$B$</th>
<th>$Y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

$Y = AB$
Mux as general-purpose logic

- Example: \( Z(A,B,C) = AC + BC' + A'B'C \)
Function $Z(A,B,C)$ implemented by 2:1 Muxes above is:

A. $A'B'C' + ABC + BC'$
B. $(A'+AC)B + B'C'$
C. $A'B' + B'C + BC'$
D. $A'+AC + BC'$
E. None of the above
### Mux example: Logical function unit

<table>
<thead>
<tr>
<th>C0</th>
<th>C1</th>
<th>C2</th>
<th>Function</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>always 1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>A + B</td>
<td>logical OR</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>(A • B)'</td>
<td>logical NAND</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>A xor B</td>
<td>logical xor</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>A xnor B</td>
<td>logical xnor</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>A • B</td>
<td>logical AND</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>(A + B)'</td>
<td>logical NOR</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>always 0</td>
</tr>
</tbody>
</table>

**Diagram:**

8:1 MUX

Sources: TSR, Katz, Boriello & Vahid
BREAK!
Demux or Decoder

- \( N \) inputs, \( 2^N \) outputs
- One-hot outputs: only one output HIGH at a time when enable signal is 1 (EN=1)

For the moment, we call it decoder or demuxer with no distinction. We’ll see the (slight) difference later
**Decoder: logic equations & implementation**

- **Decoders/demultiplexers**
  - control inputs (called “selects” (S)) represent binary index of output to which the input is connected
  - data input usually called “enable” or G in equations

**1:2 Decoder:**

\[ O_0 = G \cdot S' \]
\[ O_1 = G \cdot S \]

**2:4 Decoder:**

\[ O_0 = G \cdot S_1' \cdot S_0' \]
\[ O_1 = G \cdot S_1' \cdot S_0 \]
\[ O_2 = G \cdot S_1 \cdot S_0' \]
\[ O_3 = G \cdot S_1 \cdot S_0 \]

**3:8 Decoder:**

\[ O_0 = G \cdot S_2' \cdot S_1' \cdot S_0' \]
\[ O_1 = G \cdot S_2' \cdot S_1' \cdot S_0 \]
\[ O_2 = G \cdot S_2' \cdot S_1 \cdot S_0' \]
\[ O_3 = G \cdot S_2' \cdot S_1 \cdot S_0 \]
\[ O_4 = G \cdot S_2 \cdot S_1' \cdot S_0' \]
\[ O_5 = G \cdot S_2 \cdot S_1' \cdot S_0 \]
\[ O_6 = G \cdot S_2 \cdot S_1 \cdot S_0' \]
\[ O_7 = G \cdot S_2 \cdot S_1 \cdot S_0 \]
Logic Using Decoders

- OR minterms

Example: \( F = \sum m(3,4,7) \)
Example of demux as general-purpose logic

F1 = A'BC'D + A'B'CD + ABCD
F2 = ABC'D' + ABC
F3 = (A' + B' + C' + D')

How many 4:16 demux are required to implement these three functions?
Another example

• $F(A,B,C) = \Pi M(0,2,4)$
Implement a 6-2^6 decoder with 3-2^3 decoders.

You can use smaller decoders to build a larger one.
Demultiplexer and Decoder

- **Demultiplexer**: has a data input and it reports it on the output line specified by the selection
- **Decoder**: takes an input address and switches to 1 the corresponding output line
- Note: the behavior of a decoder with “enable” would be the same of a demultiplexer, but we give different names to the inputs
Multi-level logic

- $x = \text{A} \cdot \text{D} \cdot \text{F} + \text{A} \cdot \text{E} \cdot \text{F} + \text{B} \cdot \text{D} \cdot \text{F} + \text{B} \cdot \text{E} \cdot \text{F} + \text{C} \cdot \text{D} \cdot \text{F} + \text{C} \cdot \text{E} \cdot \text{F} + \text{G}$
  - reduced sum-of-products form – already simplified
  - 6 x 3-input AND gates + 1 x 7-input OR gate (that may not even exist!)
  - 25 wires (19 literals plus 6 internal wires)
- $x = (\text{A} + \text{B} + \text{C}) \cdot (\text{D} + \text{E}) \cdot \text{F} + \text{G}$
  - factored form – not written as two-level S-o-P
  - 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 logic

• No global definition of “optimal” multilevel circuit
  – Optimality depends on user-defined goals
  – Synthesize an implementation that meets design goals

• Synthesis requires CAD-tool help
  – No simple hand methods like K-maps
  – CAD tools manipulate Boolean expressions
  – Factoring, decomposition, etc

• Advantages over 2-level logic
  – Smaller circuits
  – Reduced fan-in
  – Less wires

• Disadvantages w.r.t 2-level logic
  – More difficult design
  – Less powerful optimizing tools
  – Dynamic hazards

Sources: TSR, Katz, Boriello & Vahid