Course Structure

Part 1
ASIC Design Overview

Part 2
Digital CMOS Circuits

Part 3
CAD Algorithms

Prereq
Computer Architecture
Part 3: CAD Algorithms

Topic 12
Synthesis Algorithms

RTL to Logic Synthesis

Technology Independent Synthesis

\[ x = a'b'c + a'bc' \]
\[ y = b'c' + ab' + ac \]

Technology Dependent Synthesis

\[ x = a'b \]
\[ y = b'c' + ac \]

Topic 13
Physical Design Automation

Placement

Global Routing

Detailed Routing
Step 1: Single Assignment Form

For each output, create exactly one assignment that is a function only of the inputs

```vhdl
// 1b inputs: y, z, a, q
// 2b inputs: f
// 3b inputs: g, c, b, e

wire x = y && z;
wire [2:0] b = (a) ? {2'b0,x} : c;
wire [2:0] d, h;
always @(*) begin
  d = b + e;
  if ( q ) d = 3'b101;
  if ( f )
    h = 3'b0;
  else
    h = g << 1;
end
```

```vhdl
wire x = y && z;
wire [2:0] b = (a) ? {2'b0,x} : c;
wire [2:0] d
  = (q) ? 3'b101
    : ((a) ? {2'b0,x} : c)
        + e;
wire [2:0] h = (f) ? 3'b0 : ( g << 1 );
```
Step 2: Bit Blast Outputs

Generate separate assignment for each bit, removes arithmetic operators leaving only boolean operators (assume ternary operator is short hand for equivalent boolean operator)

```verbatim
wire x = y && z;

wire[2:0] b
  = (a) ? {2'b0,x} : c;

wire[2:0] h
  = (f) ? 3'b0 : ( g << 1 );

1 // 1b inputs: y, z, a, q
2 // 2b inputs: f
3 // 3b inputs: g, c, b, e
4
5 wire x = y & z;
6
7 wire [2:0] b
8   = (a) ? {2'b0,x} : c;
9
10 wire [2:0] h
11   = (f) ? 3'b0
12     : ( g << 1 );
```
wire [15:0] a = b + c;

Ripple-Carry

Carry Lookahead

Parallel-Prefix Tree-Based

Adapted from [Weste’11]
Example from Synopsys DesignWare

RTL Datapath Synthesis directly transforms arithmetic operators into technology independent optimized gate-level netlists

### Table 1-1 Pin Description

<table>
<thead>
<tr>
<th>Pin Name</th>
<th>Width</th>
<th>Direction</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>width bit(s)</td>
<td>Input</td>
<td>Input data</td>
</tr>
<tr>
<td>B</td>
<td>width bit(s)</td>
<td>Input</td>
<td>Input data</td>
</tr>
<tr>
<td>CI</td>
<td>1 bit</td>
<td>Input</td>
<td>Carry-in</td>
</tr>
<tr>
<td>SUM</td>
<td>width bit(s)</td>
<td>Output</td>
<td>Sum of (A + B + CI)</td>
</tr>
<tr>
<td>CO</td>
<td>1 bit</td>
<td>Output</td>
<td>Carry-out</td>
</tr>
</tbody>
</table>

### Table 1-2 Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>width</td>
<td>≥1</td>
<td>Word length of A, B, and SUM</td>
</tr>
</tbody>
</table>

### Table 1-3 Synthesis Implementations

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Function</th>
<th>License Feature Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>rpl</td>
<td>Ripple-carry synthesis model</td>
<td>none</td>
</tr>
<tr>
<td>cla</td>
<td>Carry-look-ahead synthesis model</td>
<td>none</td>
</tr>
<tr>
<td>pparch</td>
<td>Delay-optimized flexible parallel-prefix</td>
<td>DesignWare</td>
</tr>
</tbody>
</table>

* During synthesis, Design Compiler will select the appropriate architecture for your constraints. However, you may force Design Compiler to use one of the architectures described in this table. For more details, please refer to the DesignWare Building Block IP User Guide.

Adapted from [Synopsys’11]
Technology-Independent Synthesis

- **Two-level boolean minimization** – based on assumption that reducing the number of product terms in an equation and reducing the size of each product term will result in a smaller/faster implementation.

- **Optimizing finite-state machines** – look for equivalent FSMs (i.e., FSMs that produce the same outputs give the same sequence of inputs) that have fewer states.

- **FSM state encodings** – minimize implementation area (= size of state storage + size of logic to implement next state and output functions).

Note that none of these optimizations are completely isolated from the target technology, but experience has shown that it’s advantageous to reduce the size of the problem as much as possible before starting the technology-dependent optimizations.
Karnaugh Map Method Review

1. Choose an element of ON-set not already covered by an implicant

2. Find “maximal” groups of 1’s and X’s adjacent to that element. Remember to consider top/bottom row, left/right column, and corner adjacencies. This forms prime implicants.

Repeat steps 1 and 2 to find all prime implicants

3. Revise the 1’s elements in the K-map. If covered by single prime implicant, it is essential, and participates in the final cover. The 1’s it covers do not need to be revisited.

4. If there remain 1’s not covered by essential prime implicants, then select the smallest number of prime implicants that cover the remaining 1’s
Karnaugh Map Method Example

Primes around A B C' D

Primes around A B' C' D'

Essential Primes with Min Cover

Adapted from [Zhou'02]
Quine-McCluskey (QM) Method

- Quine-McCluskey method is an exact algorithm which finds a minimum-cost sum-of-products implementation of a boolean function.

- Four main steps:
  1. Generate prime implicants
  2. Construct prime implicant table
  3. Reduce prime implicant table
     - Remove essential prime implicants
     - Row dominance
     - Column dominance
     - Iterate at this step until no further reductions
  4. Solve prime implicant table
QM Example #1 – Step 1

Start by expressing your Boolean function using 0-terms (product terms with no don’t care care entries). For compactness the table for example 4-input, 1-output function $F(w,x,y,z)$ shown to the right includes only entries where the output of the function is 1 and we’ve labeled each entry with it’s decimal equivalent.

<table>
<thead>
<tr>
<th>$W$</th>
<th>$X$</th>
<th>$Y$</th>
<th>$Z$</th>
<th>label</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>9</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>14</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>15</td>
</tr>
</tbody>
</table>

Look for pairs of 0-terms that differ in only one bit position and merge them in a 1-term (i.e., a term that has exactly one ‘–’ entry). Next 1-terms are examined in pairs to see if the can be merged into 2-terms, etc. Mark k-terms that get merged into (k+1) terms so we can discard them later.

1-terms:  
0, 8  -000 [A]  
5, 7  01-1 [B]  
7,15  -111 [C]  
6, 9  100-  
8,10  10-0  
9,11  10-1  
10,11  101-  
10,14  1-10  
11,15  1-11  
14,15  111-  

2-terms:  
8, 9,10,11  10-- [D]  
10,11,14,15  1-1- [E]  

3-terms: none!

Label unmerged terms: 
these terms are prime!

Example due to Srini Devadas

Adapted from [Terman'02]
QM Example #1 – Step 2, Step 3a

An “X” in the prime term table in row R and column C signifies that the O-term corresponding to row R is contained by the prime corresponding to column C.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>X</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>0101</td>
<td>.</td>
<td>X</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>0111</td>
<td>.</td>
<td>X</td>
<td>X</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>1000</td>
<td>X</td>
<td>.</td>
<td>.</td>
<td>X</td>
<td>.</td>
</tr>
<tr>
<td>1001</td>
<td>.</td>
<td>.</td>
<td>X</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>1010</td>
<td>.</td>
<td>.</td>
<td>X</td>
<td>X</td>
<td>.</td>
</tr>
<tr>
<td>1011</td>
<td>.</td>
<td>.</td>
<td>X</td>
<td>X</td>
<td>.</td>
</tr>
<tr>
<td>1110</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>X</td>
<td>.</td>
</tr>
<tr>
<td>1111</td>
<td>.</td>
<td>.</td>
<td>X</td>
<td>.</td>
<td>X</td>
</tr>
</tbody>
</table>

Goal: select the minimum set of primes (columns) such that there is at least one “X” in every row. This is the classical minimum covering problem.

Each row with a single X signifies an essential prime term since any prime implementation will have to include that prime term because the corresponding O-term is not contained in any other prime.

In this example the essential primes “cover” all the O-terms.

Adapted from [Terman’02]
Column Dominance

- 5 prime implicants, each covers 2 ON-set minterms
- A’C’D’ and ACD are essential prime implicants, must be in final cover
- Pick min subset of remaining 3 prime implicants which covers ON-set

Karnaugh map with set of prime implicants:
illustrating "column dominance"

Adapted from [Nowick'12]
Column Dominance

- Cross out columns A’C’D’ and ACD since they are essential.

- Each row intersected by one of the essential prime columns is also crossed out because that minterm is already covered.

- BC’D covers minterm 5 and 13, but A’BC’ only covers minterm 5 and ABD only covers minterm 13.

- BC’D *column dominates* A’BC’ and ABD, dominated prime implicants can be crossed out.

- Only column BC’D remains.

### Karnaugh Map

<table>
<thead>
<tr>
<th></th>
<th>A’C’D’</th>
<th>A’BC’</th>
<th>BC’D</th>
<th>ABD</th>
<th>ACD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>13</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

[(Nowick’12)]
Row Dominance

- 4 prime implicants, no essential prime implicants
- Pick min subset of the 4 prime implicants to cover the 5 ON-set minterms

![Karnaugh map with set of prime implicants: illustrating “row dominance”](image)

Adapted from [Nowick’12]
Row Dominance

- Row 3 is contained in 3 columns: A'B', A'D, and A'C
- Row 2 is covered by two of these three columns, so any prime implicant which contains row 2 also contains row 3
- Row 7 is covered by two of these three columns, so any prime implicant which contains row 7 also contains row 3
- Thus we can ignore row 3, it will always be covered as long as cover row 2 or row 7
- Cross out row 3, similarly row 1 dominates row 5 so cross out row 1

<table>
<thead>
<tr>
<th></th>
<th>$A'B'$</th>
<th>$C'D$</th>
<th>$A'D$</th>
<th>$A'C$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>3</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>5</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>7</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

Adapted from [Nowick'12]
QM Example #2 – Step 1

<table>
<thead>
<tr>
<th>Column I</th>
<th>Column II</th>
<th>Column III</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0000</td>
<td>(0,2) 00-0</td>
<td>(0,2,8,10) -0-0</td>
</tr>
<tr>
<td>2 0010</td>
<td>(0,8) -000</td>
<td>(0,8,2,10) -0-0</td>
</tr>
<tr>
<td>8 1000</td>
<td>(2,6) 0-10</td>
<td>(2,6,10,14) –10</td>
</tr>
<tr>
<td>5 0101</td>
<td>(2,10) -010</td>
<td>(2,10,6,14) –10</td>
</tr>
<tr>
<td>6 0110</td>
<td>(8,10) 10-0</td>
<td>(8,10,12,14) 1–0</td>
</tr>
<tr>
<td>10 1010</td>
<td>(8,12) 1-00</td>
<td>(8,12,10,14) 1–0</td>
</tr>
<tr>
<td>12 1100</td>
<td>(5,7) 01-1</td>
<td>(5,7,13,15) -1-1</td>
</tr>
<tr>
<td>7 0111</td>
<td>(5,13) -101</td>
<td>(5,13,7,15) -1-1</td>
</tr>
<tr>
<td>13 1101</td>
<td>(6,7) 011-</td>
<td>(6,7,14,15) -11-</td>
</tr>
<tr>
<td>14 1110</td>
<td>(6,14) -110</td>
<td>(6,14,7,15) -11-</td>
</tr>
<tr>
<td>15 1111</td>
<td>(10,14) 1-10</td>
<td>(12,13,14,15) 11–</td>
</tr>
<tr>
<td></td>
<td>(12,13) 110-</td>
<td>(12,14,13,15) 11–</td>
</tr>
<tr>
<td></td>
<td>(12,14) 11-0</td>
<td></td>
</tr>
</tbody>
</table>

Column III contains a number of duplicate entries, e.g. (0,2,8,10) and (0,8,2,10). Duplicate entries appear because a product in Column III can be formed in several ways. For example, (0,2,8,10) is formed by combining products (0,2) and (8,10) from Column II, and (0,8,2,10) (the same product) is formed by combining products (0,8) and (2,10).

Duplicate entries should be crossed out. The remaining unchecked products cannot be combined with other products. These are the prime implicants: (0,2,8,10), (2,6,10,14), (5,7,13,15), (6,7,14,15), (8,10,12,14) and

- 1111

Eliminate redundant entries in table.

Adapted from [Nowick’12]
QM Example #2 – Step 2, Step 3a

<table>
<thead>
<tr>
<th></th>
<th>(B'D') ((\ast))</th>
<th>(CD')</th>
<th>(BD) ((\ast))</th>
<th>(BC)</th>
<th>(AD')</th>
<th>(AB)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>((0,2,8,10))</td>
<td>((2,6,10,14))</td>
<td>((5,7,13,15))</td>
<td>((6,7,14,15))</td>
<td>((8,10,12,14))</td>
<td>((12,13,14,15))</td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

* indicates an essential prime implicant

○ indicates a distinguished row, i.e. a row covered by only 1 prime implicant

Adapted from [Nowick’12]
### QM Example #2 – Step 3b, 3c, 3a

#### 3.b Row Dominance – row 14 dominates both row 6 and 12; remove row 14 since if some product covers row 6, row 14 is guaranteed to be covered

#### 3.c Column Dominance – column CD’ dominates column BC (actually they co-dominate each other); remove column BC since it is redundant with column CD’

#### 3.a Remove Essential Prime Implicants – second iteration of step 3, both remaining prime implicants are essential

\[
F = B'D' + BD + CD' + AD'
\]

---

#### Table: Prime Implicant Table

<table>
<thead>
<tr>
<th></th>
<th>(CD')</th>
<th>(BC)</th>
<th>(AD')</th>
<th>(AB)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(2,6,10,14)</td>
<td>(6,7,14,15)</td>
<td>(8,10,12,14)</td>
<td>(12,13,14,15)</td>
</tr>
<tr>
<td>6</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>14</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

#### Table: Prime Implicant Table (continued)

<table>
<thead>
<tr>
<th></th>
<th>(CD')</th>
<th>(BC)</th>
<th>(AD')</th>
<th>(AB)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(2,6,10,14)</td>
<td>(6,7,14,15)</td>
<td>(8,10,12,14)</td>
<td>(12,13,14,15)</td>
</tr>
<tr>
<td>6</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

---

#### Table: Prime Implicant Table (continued)

<table>
<thead>
<tr>
<th></th>
<th>(CD'(**))</th>
<th>(AD'(**))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(o)6</td>
<td>X</td>
<td>(8,10,12,14)</td>
</tr>
<tr>
<td>(o)12</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

---

Adapted from [Nowick’12]
**QM Example #3 – Step 2**

\[ F(A, B, C, D) = \sum m(0, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13) \]

<table>
<thead>
<tr>
<th></th>
<th>(A' D')</th>
<th>(B' D')</th>
<th>(C' D')</th>
<th>(A' C)</th>
<th>(B' C)</th>
<th>(A' B)</th>
<th>(BC')</th>
<th>(AB')</th>
<th>(AC')</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Step 1: Generate Prime Implicants.
Use the method described in Example #1.

Step 2: Construct Prime Implicant Table.

Step 3: Reduce Prime Implicant Table.

Iteration #1.
(i) Remove Primary Essential Prime Implicants
There are no primary essential prime implicants: each row is covered by at least two products.

Adapted from [Nowick'12]
QM Example #3 – Step 3a, 3b, 3c, 3a

3a. No essential prime implicants

3b. Row dominance: 2 > 3, 4 > 5, 6 > 7, 8 > 9, 10 > 11, 12 > 13

<table>
<thead>
<tr>
<th></th>
<th>A'D'</th>
<th>B'D'</th>
<th>C'D'</th>
<th>A'C</th>
<th>B'C</th>
<th>A'B</th>
<th>BC'</th>
<th>AB'</th>
<th>AC'</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

3.c Column Dominance – column A’D’, B’D’, and C’D’ dominate each other; remove any two of them

3.a Remove Essential Prime Implicants – second iteration of step 3, remove A’D’

** indicates a secondary essential prime implicant
○ indicates a distinguished row

Adapted from [Nowick’12]
QM Example #3 – Step 4

<table>
<thead>
<tr>
<th></th>
<th>$A'C$</th>
<th>$B'C$</th>
<th>$A'B$</th>
<th>$BC'$</th>
<th>$AB'$</th>
<th>$AC'$</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>11</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

No essential prime implicants; no row dominance; no column dominance

Cyclic covering problem – can be solved using a branch-and-bound search technique (see notes for details)

Adapted from [Nowick’12]
Heuristic Espresso Method

- Quine-McCluskey
  - Number of prime implicants grows rapidly with number of inputs
  - Finding a minimum cover is NP-complete (i.e., computationally expensive)

- Espresso
  - Don’t generate all prime implicants (i.e., QM step 1)
  - Carefully select a subset of primes that still covers ON-set
  - Heuristically explore space of covers
  - Similar in spirit (but more structured) to finding primes in K-map

Adapted from [Zhou’02]
Redundancy in Boolean Space

- Every point in boolean space is an assignment of values to variables
- Redundancy involves inclusion or covering in boolean space
- Irredundant cover has no redundancy

\[ g = AB'C, \quad h = AB' \]
\[ f = AC, \quad g = B'C', \quad h = AB' \]

Can exhaustively check if each prime is redundant with any other prime

Adapted from [Zhou’02]
Irredundant Covers vs. Minimal Covers

- Irredundant cover is not necessarily a minimal cover
- Can use reduce, expand, remove redundancy operations to explore space of irredundant covers

Original Irredundant Cover
\[ F = B'C + A'C + AB'C' \]

Reduce
\[ F = AB'C + A'C + AB'C' \]

Expand
\[ F = AB' + A'C + AB'C' \]

New Irredundant Cover
\[ F = AB' + A'C \]

Adapted from [Zhou'02]
Espresso Algorithm

1. EXPAND implicants (choice of which implicants requires heuristic)
2. REMOVE-REDUNDANCY
3. REDUCE implicants (choice of which implicants requires heuristic)
4. EXPAND implicants (choice of which implicants requires heuristic)
5. REMOVE-REDUNDANCY
6. Goto step 3

Adapted from [Zhou'02]
RTL to Logic Synthesis

- Technology-Independent Synthesis
- Technology-Dependent Synthesis

Espresso Example

Initial Set of Primes found by Steps 1 and 2 of the Espresso Method

4 primes, irredundant cover, but not a minimal cover!

Result of REDUCE: Shrink primes while still covering the ON-set

Choice of order in which to perform shrink is important

Adapted from [Zhou'02]
Espresso Example

Second EXPAND generates a different set of prime implicants

REMOVE-REDUNDANCY generates only three prime implicants!

Adapted from [Zhou'02]
Two-Level Logic vs. Multi-Level Logic

2-Level:

\[ f_1 = AB + AC + AD \]
\[ f_2 = \overline{AB} + \overline{AC} + \overline{AE} \]

- 6 product terms which cannot be shared.
- 24 transistors in static CMOS

Multi-level:

Note that \( B + C \) is a common term in \( f_1 \) and \( f_2 \)

\[ K = B + C \]
\[ f_1 = AK + AD \]
\[ f_2 = \overline{AK} + \overline{AE} \]

- 3 Levels
- 20 transistors in static CMOS
- not counting inverters

Adapted from [Devadas'06]
# Two-Level Logic vs. Multi-Level Logic

## Two-Level Logic
- At most two gates between primary input and primary output
- Real life circuits: programmable logic arrays
- Exact optimization methods: well-developed, feasible
- Heuristic methods also possible

## Multi-Level Logic
- Any number of gates between primary input and primary output
- Most circuits in real life are multi-level
- Smaller, less power, and (in many cases) faster
- Exact optimization methods: few, high complexity, impractical
- Heuristic methods pretty much required
Part 3: CAD Algorithms

**Topic 12**
Synthesis Algorithms

- $x = a'bc + a'bc'$
- $y = b'c' + ab' + ac$

**Topic 13**
Physical Design Automation

- Placement
- Global Routing
- Detailed Routing
Technology Mapping

Once minimized logic equations, next step is to map each equation to the gates in the target standard cell library

Classic approach uses DAG covering (K. Keutzer)
  - “Normal Form”: use basic gates (e.g., 2-input NAND gates, inverters)
  - Represent logic equations as input netlist in normal form (subjective DAG)
  - Represent each library gate in normal form (primitive DAG)
  - GOAL: Find a min cost covering of subjective DAG by primitive DAGs

Sound algorithmic approach, but is NP-hard optimization problem

Adapted from [Devadas’06]
Tree Heuristic Transformation

If subject and primitive DAGs are trees, efficient algorithm can find optimum cover in linear time via dynamic programming, so use tree covering heuristic approach by partitioning graph into subtrees.

Original Graph

Partitioned Graph

Adapted from [Devadas'06]
Problem statement: find an “optimal” mapping of this circuit:

Into this library:

Adapted from [Terman'02,Devadas'06]
Primitive DAGs for Standard-Cell Library Gates

<table>
<thead>
<tr>
<th>Element/Area Cost</th>
<th>Tree Representation (normal form)</th>
</tr>
</thead>
<tbody>
<tr>
<td>INVERTER 2</td>
<td><img src="image" alt="Inverter DAG" /></td>
</tr>
<tr>
<td>NAND2 3</td>
<td><img src="image" alt="NAND2 DAG" /></td>
</tr>
<tr>
<td>NAND3 4</td>
<td><img src="image" alt="NAND3 DAG" /></td>
</tr>
<tr>
<td>NAND4 5</td>
<td><img src="image" alt="NAND4 DAG" /></td>
</tr>
<tr>
<td>AOI21 4</td>
<td><img src="image" alt="AOI21 DAG" /></td>
</tr>
<tr>
<td>AOI22 5</td>
<td><img src="image" alt="AOI22 DAG" /></td>
</tr>
</tbody>
</table>

Adapted from [Terman’02, Devadas’06]
Possible Covers

Hmmm. Seems promising but is there a systematic and efficient way to arrive at the optimal answer?

- Area cost 31
- Area cost 19

Adapted from [Terman'02, Devadas'06]
Use Dynamic Programming!

Principle of optimality: Optimal cover for a tree consists of a best match at the root of the tree plus the optimal cover for the sub-trees starting at each input of the match.

Complexity: To determine the optimal cover for a tree we only need to consider a best-cost match at the root of the tree (constant time in the number of matched cells), plus the optimal cover for the subtrees starting at each input to the match (constant time in the fanin of each match) \( \rightarrow O(N) \)

Adapted from [Terman'02, Devadas'06]
Optimal Tree Covering Example

**Step 1.**

**Step 2.**

**Step 3.**

**Step 4.**

Cover with ND2 or ND3?

1 NAND2 3
+ subtree 5
Area cost 8

1 NAND3 = 4

Cover with INV or AO21?

1 AO21 4
+ subtree 1 3
+ subtree 2 2
Area cost 13

Adapted from [Terman'02, Devadas'06]
Optimal Tree Covering Example

Step 5. Cover with ND2 or ND3?

Step 6. Cover with INV or AOI21?

Step 7. Cover with ND2 or ND3 or ND4?

Step 7a. Cover with ND2?

Adapted from [Terman'02,Devadas'06]
Optimal Tree Covering Example

This matches our earlier intuitive cover, but accomplished systematically.

Refinements: timing optimization incorporating load-dependent delays, optimization for low power.

Adapted from [Terman’02, Devadas’06]
Acknowledgments


