Simplification

- Finding a minimal sum of products or product of sums realization
- Exploit don't care information in the process
- Algebraic simplification
  - Not an algorithmic/systematic procedure
  - How do you know when the minimum realization has been found?
- Computer-aided design tools
  - Precise solutions require very long computation times, especially for functions with many inputs (> 10)
  - Heuristic methods employed – "educated guesses" to reduce amount of computation and yield good if not best solutions
- Hand methods still relevant
  - To understand automatic tools and their strengths and weaknesses
  - Ability to check results (on small examples)

Boolean cubes

- Visual technique for indentifying when the unifying theorem can be applied
- n input variables = n-dimensional "cube"

Mapping truth tables onto cubes

- Uniting theorem combines two "faces" of a cube into a larger "face"

Example:

```
A  B  F
0  0  1
0  1  0
1  0  1
1  1  0
```

A varies within face, B does not

this face represents the literal B

The Uniting Theorem

- Key tool to simplification: A (B' + B) = A
- Essence of simplification of two-level logic

Find two element subsets of the ON-set where only one variable changes its value – this single varying variable can be eliminated and a single product term used to represent both elements

```
F = A'B - AB' = (A+B)B' = B
```

Three variable example

- Binary full-adder carry-out logic

```
A  B  Cin  Cout
0  0  0  0
0  1  0  0
1  0  1  0
1  1  1  1
```

the on-set is completely covered by the combination (OR) of the subcubes of lower dimensionality - note that "111" is covered three times

Cout = BCin+AB+ACin
### Higher dimensional cubes
- Sub-cubes of higher dimension than 2
  - \( F(A,B,C) = \Sigma m(4,5,6,7) \)
  - \( A \) is asserted (true) and unchanged
  - \( B \) and \( C \) vary
  - A on-set forms a square, i.e., a cube of dimension 2
  - Represents an expression in one variable i.e., 3 dimensions – 2 dimensions
  - This subcube represents the literal \( A \)

### Karnaugh maps (cont’d)
- Numbering scheme based on Gray-code
  - e.g., 00, 01, 11, 10
  - Only a single bit changes in code for adjacent map cells

### m-dimensional cubes
- In a 3-cube (three variables):
  - 0-cube, i.e., a single node, yields a term in 3 literals
  - 1-cube, i.e., a line of two nodes, yields a term in 2 literals
  - 2-cube, i.e., a plane of four nodes, yields a term in 1 literal
  - 3-cube, i.e., a cube of eight nodes, yields a constant term “1”
  - In general,
    - m-subcube within an n-cube (m < n) yields a term with \( n – m \) literals

### Adjacencies in Karnaugh maps
- Wrap from first to last column
- Wrap top row to bottom row

### Karnaugh maps examples
- \( f(A,B,C) = \Sigma m(0,4,6,7) \)
  - Obtain the complement of the function by covering \( 0s \) with subcubes
  - \( AC' \cdot B' \)
  - \( AB' \cdot AC' + BC' \)
  - \( AB + AC' + B'C' \)
More Karnaugh map examples

\[ F(A, B, C) = \sum m(0, 4, 5, 7) \]

\[ F'(A, B, C) = \sum m(1, 2, 3, 6) \]

\[ F' \] simply replace 1’s with 0’s and vice versa.

\[ G(A, B, C) = A \]

Karnaugh map: 4-variable example

\[ F(A, B, C, D) = \sum m(0, 2, 3, 5, 6, 7, 8, 10, 11, 14, 15) \]

Karnaugh maps: don’t cares

\[ f(A, B, C, D) = \sum m(1, 3, 5, 7, 9) + d(6, 12, 13) \]

- \[ f = A'D + BC'D \] without don’t cares
- \[ f = A'D + C'D \] with don’t cares

Example: two-bit comparator

K-map for LT

K-map for EQ

K-map for GT

LT = \[ A'B'D + A'C + B'CD \]

EQ = \[ A'B'C'D + A'BC'D + ABCD + AB'CD' = (A \text{xnor} C) \cdot (B \text{xnor} D) \]

GT = \[ B'CD' + A'C' + ABD' \]

LT and GT are similar (flip A/C and B/D)
Example: two-bit comparator

```plaintext
two alternative implementations of EQ with and without XOR

XNOR is implemented with at least 3 simple gates
```

Example: 2x2-bit multiplier

```plaintext
4-variable K-map for each of the 4 output functions
```

Example: BCD increment by 1

```plaintext
4-variable K-map for each of the 4 output functions
```

Example: BCD increment by 1

```plaintext
Definition of terms
- **Implicant**
  - Single element of ON-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
- **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

Two-level simplification algorithm

Finite State Machine Optimization

Example

Algorithmic Approach

State Minimization Example
Method of Successive Partitions

Minimized FSM

Minimized FSM

Incompletely Specified FSMs

More Complex State Minimization