ECE 474A/57A
Computer-Aided Logic Design

Logic Optimization:
Espresso Representations and Basic Operations

Function Definition

- Assume you have a logic function with n-input variables and m-output variables

Let B = {0, 1} //input alphabet is 1 or 0
Y = {0, 1, 2} // output alphabet is 1, 0, or 2 (don't care)

- Logic function is simply a mapping of input combinations to output values

$F : B^n \rightarrow Y^m$ where $x = [x_1, \ldots, x_n] \in B^n$ is the input
and $y = [y_1, \ldots, y_m]$ is the output of F

Function Mapping

Example 1: $F_1(a, b) = a'$; $F_2(a, b) = a + b$

n = 2
m = 2

When m ≥ 2 we have a multiple output function

a	b	F_1	F_2
0	0	1	0
0	1	1	1
1	0	0	1
1	1	0	1

$F : B^n \rightarrow Y^m$

For each input combination we map to an output combination

Example 2: $F_o(a, b, c) = b' + ac$

n = 3
m = 1

When m =1 we have a single output function

a	b	c	F_o
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

On, Off, and DC Sets

- For a given function we can define the **on-set** $x_i^{ON} \subseteq B^n$ as the set of input values x such that F(x) = 1

F^{ON} = { [0, 0, 0], [0, 0, 1], [1, 0, 0], [1, 0, 1], [1, 1, 1] }

- For a given function we can define the **off-set** $x_i^{OFF} \subseteq B^n$ as the set of input values x such that F(x) = 0

F^{OFF} = { [0, 1, 0], [0, 1, 1], [1, 1, 0] }

- For a given function we can define the **don't care-set** $x_i^{DC} \subseteq B^n$ as the set of input values x such that F(x) = 2

F^{DC} = { } //empty

a	b	c	F_o
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

On, Off, and DC Sets

Example 3: Given the following truth table, identify F^{ON}, F^{OFF}, and F^{DC}

F^{ON} = { [0, 0, 0], [0, 0, 1], [1, 0, 0], [1, 0, 1], [1, 1, 0] }

F^{OFF} = { [0, 1, 0], [0, 1, 1] }

F^{DC} = { [1, 1, 1] }

a	b	c	F
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	2

Completely Specified Functions

- A completely specified function (c.s.f) is a function where all values of the input map to a 1 or 0 (i.e. no don't care conditions)

Ex 1 and 2 are completely specified functions

a	b	F1	F2
0	0	1	0
0	1	1	1
1	0	0	1
1	1	0	1

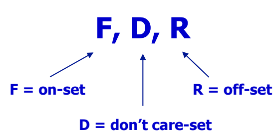
a	b	c	F_o
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

Ex 3 is not a completely specified function

a	b	c	F
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	2

On, Off, and DC Sets in Espresso

- In Espresso you will see the triple F, D, R
  - Typically F and D are provided and R is derived



How to get R?

R = (F ∪ D)'

Logic Functions Operations & Definitions

Complement of a c.s.f. (completely specified function) F' is defined as $F'^{ON} = F^{OFF}$ and $F'^{OFF} = F^{ON}$
(i.e. switch on and off sets)

x1	x2	F	F'
0	0	0	1
0	1	1	0
1	0	0	0
1	1	0	1

Intersect (or product) of two c.s.f. F and G, denoted as F·G or F ∩ G, is the c.s.f. H where $H^{ON} = F^{ON} \cap G^{ON}$
(i.e. must be in both)

x1	x2	F	G	H
0	0	0	1	0
0	1	1	1	1
1	0	1	0	0
1	1	0	0	0

Difference between two c.s.f. F and G, denoted as F – G, is the c.s.f. H where $H^{ON} = F^{ON} \cap G'^{ON}$
(i.e. it's in F but not in G)

x1	x2	F	G	G'	H
0	0	0	0	1	0
0	1	1	1	0	0
1	0	1	0	1	1
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

Logic Functions Operations & Definitions

Union (or sum) of two c.s.f. F and G, denoted F + G or F ∪ G, is the c.s.f. where $H^{ON} = F^{ON} \cup G^{ON}$
(i.e. it's in either F or G)

x1	x2	F	G	H
0	0	1	0	1
0	1	0	0	0
1	0	0	1	1
1	1	0	1	1

A **tautology** is a c.s.f. whose off-set is empty, written F ≡ 1 (i.e. function always evaluates to 1)

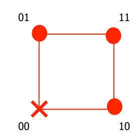
F ∪ D ∪ R is a tautology
F, D, R are mutually disjoint (no elements in common)

Cubes and Covers

- Function can also be represented as a cube in a Boolean n-space
  - Each vertex represents a value of the input and used to specify which components of F it belongs to (ON, OFF, or DC)

Example 4: F1(a, b) = Σm(1, 2, 3)

a	b	F
0	0	0
0	1	1
1	0	1
1	1	1



● vertices in on-set
○ vertices in don't care-set
✕ vertices in off-set

** Adjacent vertices differ by 1 term*

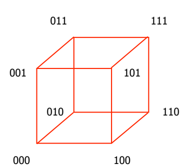
Cubes and Covers

Example 5: F1(x1, x2, x3) = Σm(0, 1, 4, 5, 6) + Σ d(7)

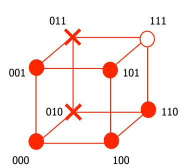
F2(x1, x2, x3) = Σm(0, 2, 3, 6, 7) + Σ d(5)

x1	x2	x3	F1	F2
0	0	0	1	1
0	0	1	1	0
0	1	0	0	1
0	1	1	0	1
1	0	0	1	0
1	0	1	1	2
1	1	0	1	1
1	1	1	2	1

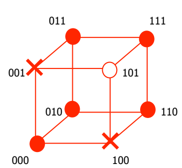
● vertices in on-set
○ vertices in don't care-set
✕ vertices in off-set



Generic 3-space representation



F1 representation

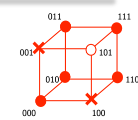


F2 representation

Compact Cubical Form

- We want to provide this information in a more compact form

x1	x2	x3	F1	F2
0	0	0	1	1
0	0	1	1	0
0	1	0	0	1
0	1	1	0	1
1	0	0	1	0
1	0	1	1	2
1	1	0	1	1
1	1	1	2	1



F2 representation

Let p be a product term associated with a S.O.P expression with n inputs and m outputs

Then a cube p is specified by a row vector $c = [\, c_1, \ldots, c_n, c_{n+1}, c_{n+m} \,]$ where

c_i = 0 if x_i appears complemented in p for i=1, …, n
1 if x_i appear NOT complemented in p for i=1, …, n
2 if x_i does not appear in p for i=1, …, n
3 if p is NOT present in the algebraic representation of f_{i-n} for i=n+1, …, n+m
4 if p is present in the algebraic representation of f_{i-n} for i=n+1, …, n+m

Compact Cubical Form

Example 6: F1 = B′ + AC′

F2 = B + AC

c_i = 0 if x_i appears complemented in p
1 if x_i appear NOT complemented in p
2 if x_i does not appear in p
3 if p is NOT present in the algebraic representation of f_{i-n}
4 if p is present in the algebraic representation of f_{i-n}

How do we represent B′ in compact cubical form?

c = [2 0 2 4 3]

Is B′ a product term in F2? No, c_5 = 3
Is B′ a product term in F1? Yes, c_4 = 4
} These are the m outputs

How does C appear in F? C does not appear, c_3 = 2
How does B appear in F? B is complemented, c_2 = 0
How does A appear in F? A does not appear, c_1 = 2
} These are the n inputs

Compact Cubical Form

Example 6: F1 = B′ + AC′

F2 = B + AC

c_i = 0 if x_i appears complemented in p
1 if x_i appear NOT complemented in p
2 if x_i does not appear in p
3 if p is NOT present in the algebraic representation of f_{i-n}
4 if p is present in the algebraic representation of f_{i-n}

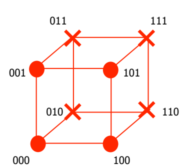
Compact cubical form represents vertices of a cube corresponding to a product term, to represent the remainder of F1 and F2

F = [2 0 2 4 3] // B′
[1 2 0 4 3] // AC′
[2 1 2 3 4] // B
[1 2 1 3 4] // AC

This matrix representation is the input used by Espresso

This set of cubes represents a cover

Boolean n-space vs. Compact Cubical Form



Graphical representation of B′

B′ as vertices in a Boolean n-space =
{ [0, 0, 0], [0, 0, 1], [1, 0, 0], [1, 0, 1] }

B′ in compact cubical form = [2 0 2 4 3]

All vertices with a 0 in second coordinate represented by this one cube

Operations on Cubes

Intersect (or product) of two cubes, written as c ∩ d or cd, is the cube e given by the following table

Basic idea: want an input cube whose input part corresponds to vertices common to c and d, resulting cube output part also represents when c_i and d_i are both present

I(c) = (c_i rows, d_i columns)

∩	0	1	2
0	0	Φ	0
1	Φ	1	1
2	0	1	2

O(c) = (c_i rows, d_i columns)

∩	3	4
3	3	3
4	3	4

c_i = [2 0 2 3 4]

d_i = [2 1 2 3 4]

$c_i \cap d_i = e_i$ = [2 Φ 2 3 4]

When there is an index = Φ, or if the output part is all 3's, the cube is empty and we can get rid of it

Operations on Cubes

Intersection of two sets of cubes is the set obtained by performing pair wise intersection of all cubes in the two sets

I(c) = ∩ (c_i rows, d_i columns)

∩	0	1	2
0	0	Φ	0
1	Φ	1	1
2	0	1	2

O(c) = ∩ (c_i rows, d_i columns)

∩	3	4
3	3	3
4	3	4

Example 6:

2 0 2 4 3	∩	1 1 1 4 4
1 1 0 4 3		0 0 0 4 3
0 2 2 3 4		2 0 2 3 4

[2 0 2 4 3] ∩ [1 1 1 4 4] = [1 Φ 1 4 3]
[2 0 2 4 3] ∩ [0 0 0 4 3] = [0 0 0 4 3]
[2 0 2 4 3] ∩ [2 0 2 3 4] = [2 0 2 3 3]
[1 1 0 4 3] ∩ [1 1 1 4 4] = [1 1 Φ 4 3]
[1 1 0 4 3] ∩ [0 0 0 4 3] = [Φ Φ 0 4 3]
[1 1 0 4 3] ∩ [2 0 2 3 4] = [1 Φ 0 3 3]
[0 2 2 3 4] ∩ [1 1 1 4 4] = [Φ 1 1 3 4]
[0 2 2 3 4] ∩ [0 0 0 4 3] = [0 0 0 3 3]
[0 2 2 3 4] ∩ [2 0 2 3 4] = [0 0 2 3 4]

Operations on Cubes

Intersection of two sets of cubes is the set obtained by performing pair wise intersection of all cubes in the two sets

I(c) = ∩ (c_i rows, d_i columns)

∩	0	1	2
0	0	Φ	0
1	Φ	1	1
2	0	1	2

O(c) = ∩ (c_i rows, d_i columns)

∩	3	4
3	3	3
4	3	4

Example 6:

2 0 2 4 3	∩	1 1 1 4 4
1 1 0 4 3		0 0 0 4 3
0 2 2 3 4		2 0 2 3 4

~~[1 Φ 1 4 3]~~ has a Φ entry
[0 0 0 4 3]
~~[2 0 2 3 3]~~ all 3's in output
~~[1 1 Φ 4 3]~~ has a Φ entry
~~[Φ Φ 0 4 3]~~ has a Φ entry
~~[1 Φ 0 3 3]~~ has a Φ entry
~~[Φ 1 1 3 4]~~ has a Φ entry
~~[0 0 0 3 3]~~ all 3's in output
[0 0 2 3 4]

Solution = [0 0 0 4 3]
[0 0 2 3 4]

When there is an index = Φ, or if the output part is all 3's, the cube is empty and we can get rid of it

Operations on Cubes

Union (or sum) of two cubes, written as c ∪ d or c+d, is the set of verticies covered by the input part of either c or d

Basic idea: combine all cubes

c_i = [2 0 2 4 3]

d_i = [1 2 0 4 3]

$c_i \cup d_i$ = [2 0 2 4 3]
[1 2 0 4 3]

Operations on Cubes

What about **Complement**?

Use DeMorgan's Law to determine what happens to a product term when complemented

(ab)' = a' + b'

F(a, b) = a [1 2]

F'(a, b) = a' [0 2]

If the cube has a single term we can simply switch the 1 & 0 entry

F(a, b, c) = ab [1 1 2]

F'(a, b, c) = (ab)'
= a' + b' [0 2 2]
[2 0 2]

switch every 1 & 0 entry, but we must also make sure each row has only one 1/0 entry

Operations on Cubes

What about **Complement**?

Use DeMorgan's Law to determine what happens to a product term when complemented

(ab)′ = a′ + b′

F(a, b, c) = a′bc **[0 1 1]**

F′(a, b, c) = (a′bc)′
= a + b′ + c′ **[1 2 2]**
[2 0 2]
[2 2 0]

Again switch every 1 & 0 entry, and make sure each row has only one 1/0 entry

Many other transformations exist (distance, consensus, etc..) we'll stick with the basic ones for now

Espresso Optimization Goal

- Espresso algorithm returns a "minimized cover"
  - What is the algorithm trying to minimize?

Φ = (NPT, NLI, NLO)

NLO = # of literals in output part

NLI - # of literals (non-2's) in input part of cover

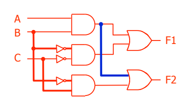
NPT - # of product terms in a cover

Espresso Subroutine

- Many smaller subroutines used in Espresso
- We will only cover a few
  - Unwrap
  - Unate Complement
  - Complement
  - Expand

Unwrap(F)

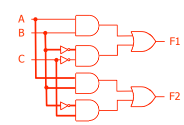
- Incoming data may have output sharing
  - Apply Unwrap(F) to the input so we start with a less biased starting point
  - When complete algorithm can decide what sharing is desirable



F1 = AB + B′C′
F2 = AB + B′C

[1 2 2 4 4]
[2 0 0 4 3]
[2 0 1 3 4]

Each cube feeding k different outputs are replaced with k cube feeding 1 output



F1 = AB + B′C′
F2 = AB + B′C

[1 2 2 4 3]
[1 2 2 3 4]
[2 0 0 4 3]
[2 0 1 3 4]

Coming Soon …

- Unate Complement
- Complement
- Expand