Digital Logic
I. *Basics*

Jalal Kawash

Outline

1. Introduction
   a. Binary Logic
   b. Logic Gates
   c. Boolean Functions & Logic Circuits
   d. Logic Circuits
2. Minterms and Gate-Level Minimization
   a. Minterms
   b. Boolean Sum of Products
   c. K-Maps

Section 1

Introduction

Section 1 Objectives

At the end of this section you will

1. Recall binary logic and logic operators
2. Recall the definition of AND, OR, NOT, and XOR
3. See the correspondence between gates and logic operators
4. Build Boolean functions
5. Build digital logic circuits that correspond to Boolean functions

Binary Logic

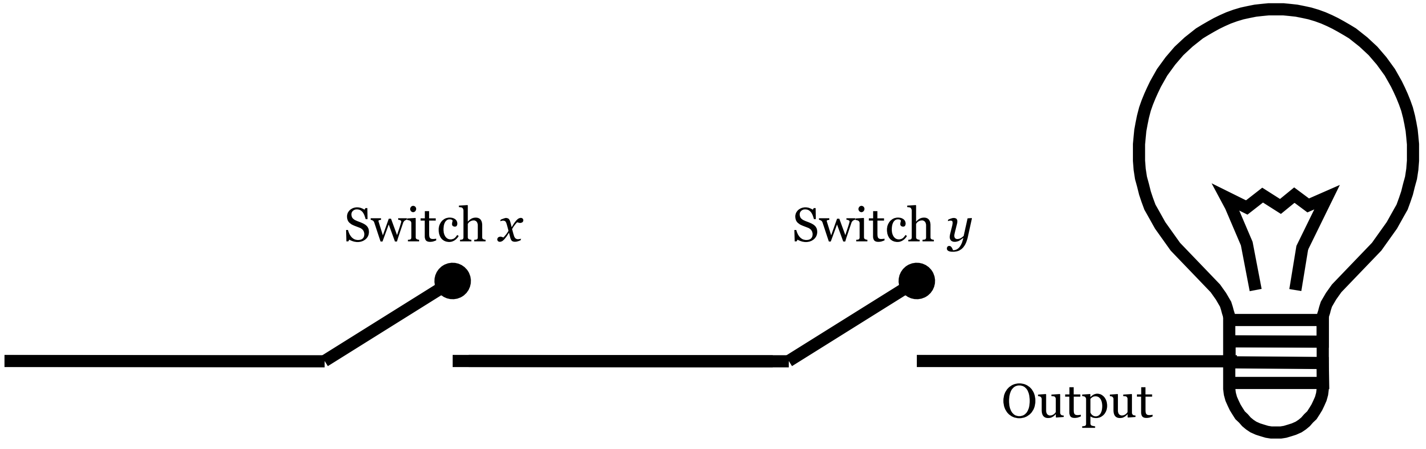
- Deals with:

1. Binary variables
   - Take values from {0, 1}

2. Logic operators
   - Three basic operators: AND, OR, and NOT

Logic Operators: AND

x **AND** y

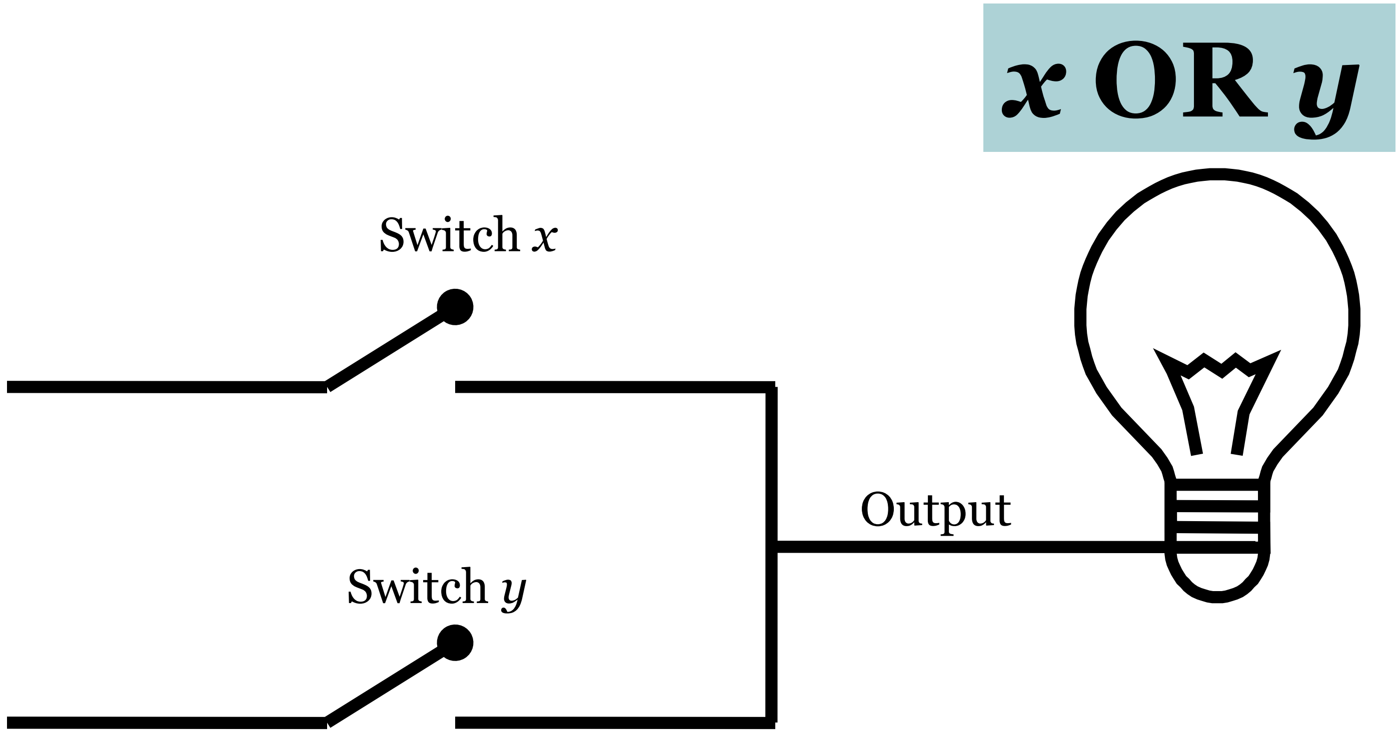


Logic Operators: AND

- Let x and y be binary variables, $x.y$ or xy is defined by:

x	y	xy
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

Logic Operators: OR



Logic Operators: OR

- Let x and y be binary variables, $x+y$ is defined by:

x	y	$x+y$
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**

Logic Operators: NOT

- Let x be binary variables, x' is defined by:

x	x'
0	1
1	0

Precedence

- Logic operators are applied in the following order:

1. Parentheses
2. NOT
3. AND
4. OR

- *Exercise: Create truth tables for $x+xy'$ and $x+(xy)'$*

Useful Logic Laws

$x+0 = x$	$x.1 = x$
$x+x' = 1$	$x.x' = 0$
$x+x = x$	$x.x = x$
$x+1 = 1$	$x.0 = 0$
$(x')' = x$	
$x+y = y+x$	$x.y = y.x$
$x + (y+z) = (x+y) + z$	$x(y.z) = (x.y)z$
$x(y + z) = x.y + x.z$	$x + y.z = (x+y)(x+z)$
$(x+y)' = x'.y'$	$(xy)' = x' + y'$
$x + xy = x$	$x(x+y) = x$

Useful Logic laws

- *Exercise: Verify these laws using truth tables*

Logic Gates

- A logic gate is an electronic circuit that operates on one or more input signals to produce one output signal
- Signals represent 0s and 1s
  - Typically 0-1V for 0 and 2-3V for 1
- Logic gates are basic building blocks for digital machines
- There is a gate for each logic operator

AND gate

x	y	xy
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

Think of it as a function :

`bit and( bit x, bit y)`

OR gate

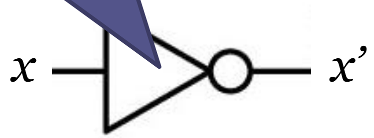
x	y	$x+y$
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**

Think of it as a function :
`bit or( bit x, bit y)`

NOT gate (invertor)

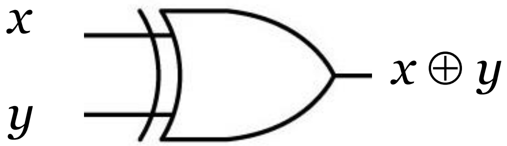
x	x'
0	1
1	0

Think of it as a function :
`bit not( bit x)`



More Gates: Exclusive OR (XOR)

x	y	$x \oplus y$
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**0**



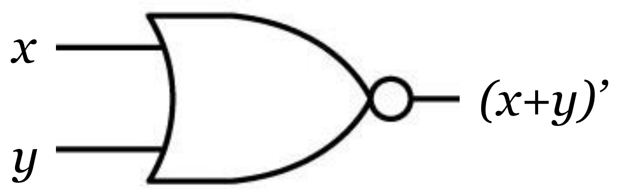
XOR is redundant

- $x \oplus y = x'y + y'x$
- $x \oplus y = (x+y)(xy)'$

- *Exercise: Verify these laws using truth tables*

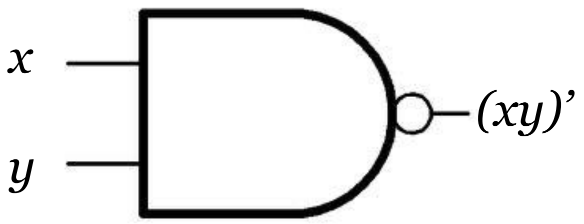
More Gates: NOT OR (NOR)

x	y	$x+y$	$(x+y)'$
0	0	0	**1**
0	1	1	**0**
1	0	1	**0**
1	1	1	**0**



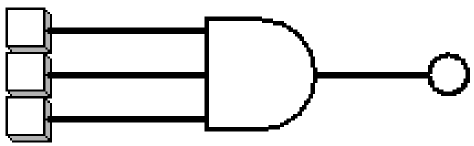
More Gates: NOT AND (NAND)

x	*y*	*xy*	*(xy)'*
0	0	0	**1**
0	1	0	**1**
1	0	0	**1**
1	1	1	**0**

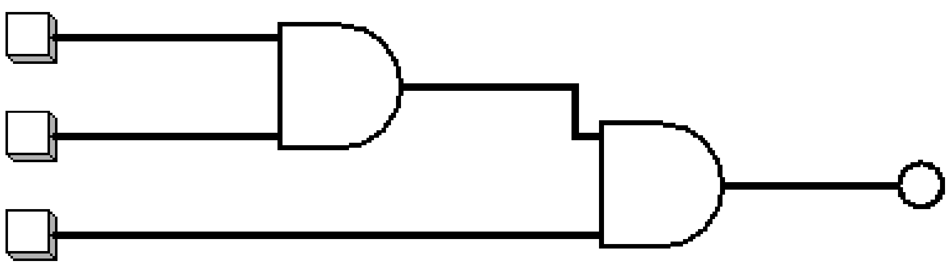


Gates with more inputs

- It is possible to feed a gate (except invertors) more than 2 inputs



- Note that this is equivalent to



Boolean Functions

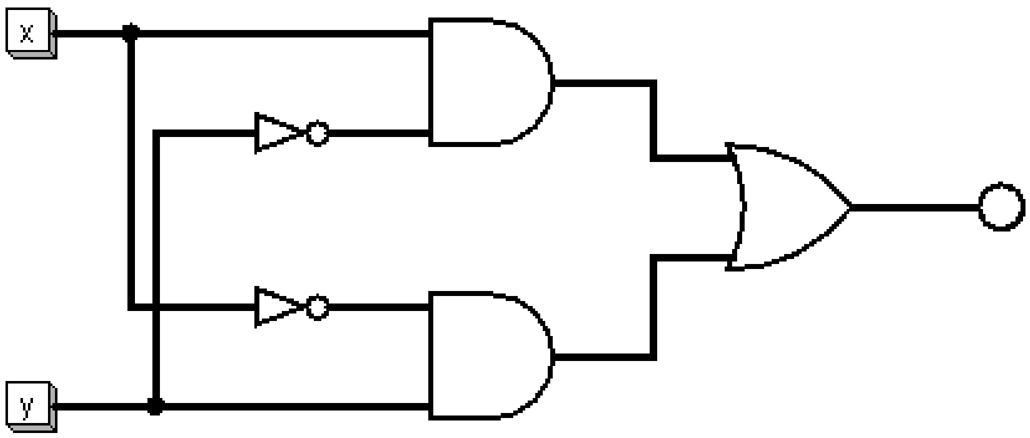
- A Boolean function combines binary variables using logic operators.
- Examples:
- $F1 = x + y'z$
- $F2 = x'y'z + x'yz + xy'$
  - *Exercise: Simplify to get $x'z + xy'$*

Boolean Function Exercises

- Simplify:
- $xyz' + x'yz + xyz + x'yz'$
  - *Answer:* y
- $(x+y'+z')(x'+z')$
  - *Answer:* $z' + x'y'$
- $yz(xz+x'+xz')$
  - *Answer:* yz

Logic Circuits

- A logic circuit is formed of logic gates and calculates a Boolean function
- Example: $x'y + y'x$

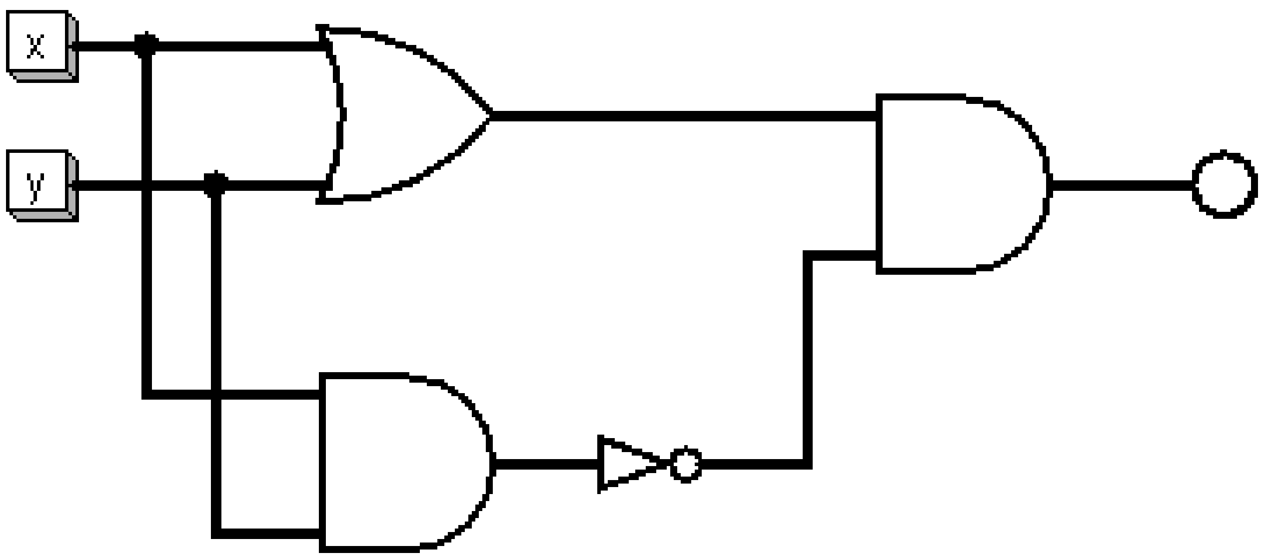


Source: DL1.circ (xor1)

Logisim

- Free simulator for logic circuits
- Downloadable from various places
  - http://sourceforge.net/projects/circuit/
- Demo!
  - XOR1 & XOR2

Another XOR Circuit



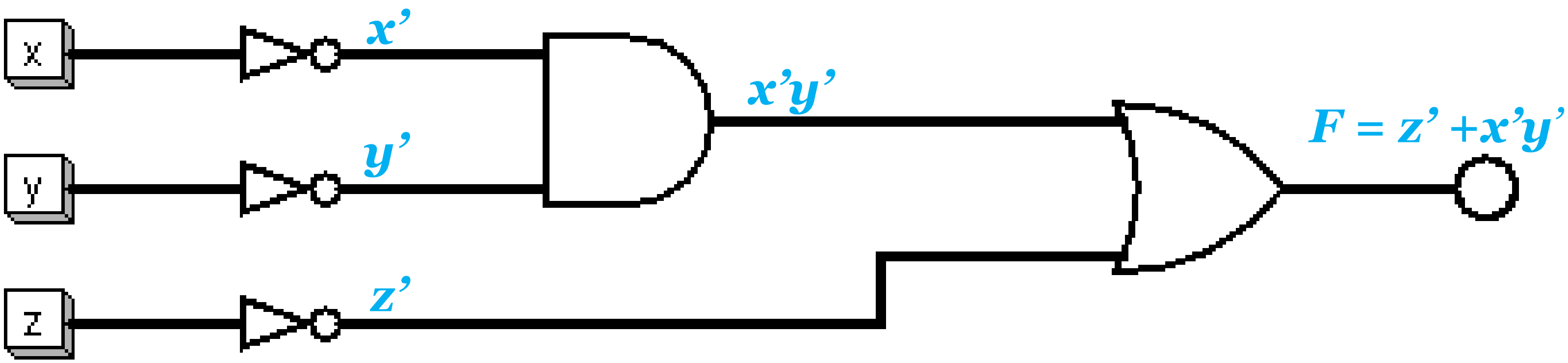
Source: DL1.circ (xor2)

$$x \oplus y = (x+y)(xy)'$$

Boolean Function Exercises

- Draw the corresponding logic circuit:
- $xyz' + x'yz + xyz + x'yz'$
  - Answer: y
- $(x+y'+z')(x'+z')$
  - Answer: $z' + x'y'$
- $yz(xz+x'+xz')$
  - Answer: yz

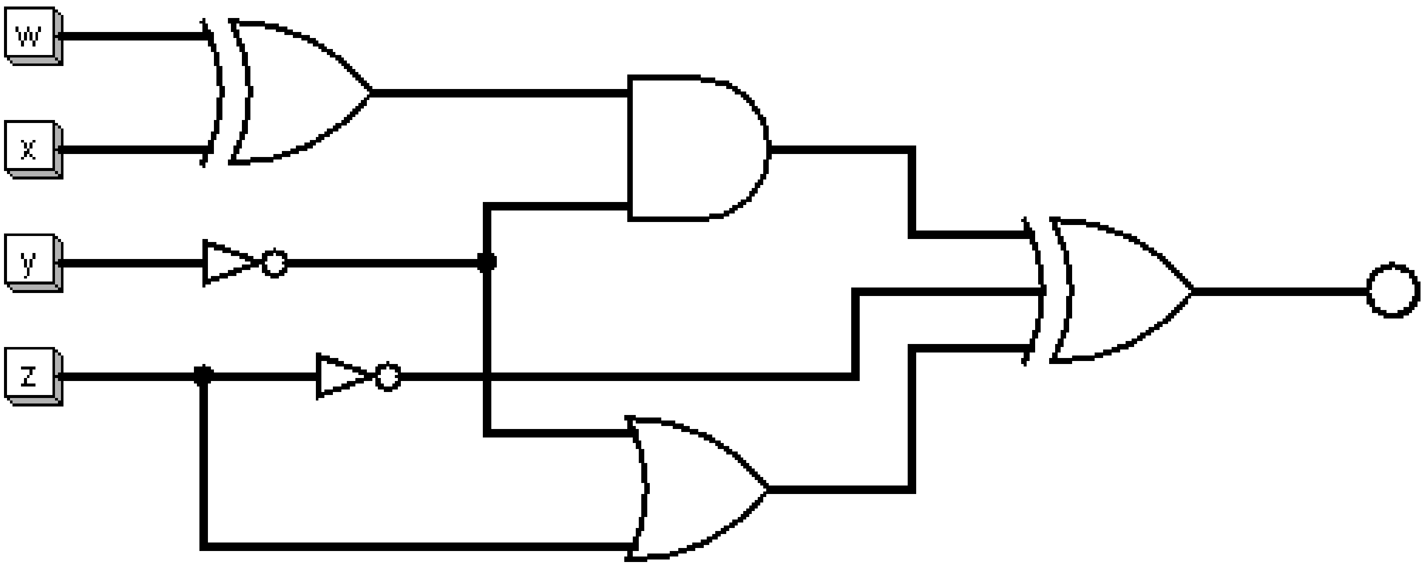
$$F = z' + x'y'$$



Source: DL1.circ (Example1 Function)

Boolean Function Exercise

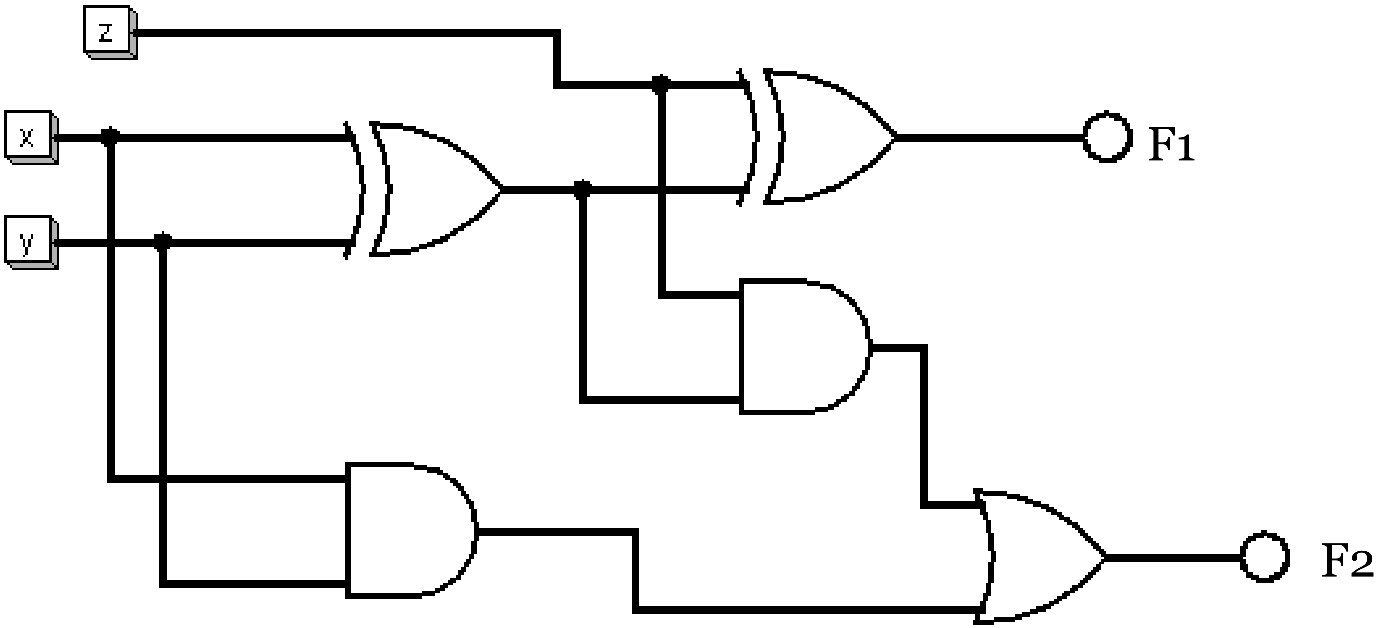
- What is the Boolean function corresponding to the following circuit?



Source: DL1.circ

Self-Test Quiz

- What is the Boolean functions corresponding to the following circuit?



Section 2

Minerms and Gate-Level Minimization

Section 2 Objectives

At the end of this section you will

1. Master the Boolean-Sum-of-Products algorithm
2. Convert truth tables to Boolean functions
3. Minimize minterms in a function using factorization
4. Minimize minterms in a function in canonical form using K-Maps

Boolean Sum of Products (BSP)

- Truth table = Boolean function(s) = Logic Circuit
- BSP Algorithm:
  - Given a truth table, convert each column C to an equivalent function F_C as follows:

1. For each output column, for each row R where the output is 1:
   - $f_R = empty$
   - For each input value x in R that is 1, $f_R = f_R.x$
   - For each input value x in R that is 0, $f_R = f_R.x'$
2. $F_C = \sum_R f_R$

BSP Example

x	y	$x \oplus y$
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**0**

$x'y$

xy'

$$F = x'y + xy' = x \oplus y$$

BSP Example

- *Convert the following table to 2 functions F1 and F2*
- *What do F1 and F2 represent?*

x	*y*	*z*	*F1*	*F2*
0	0	0	**0**	**0**
0	0	1	**1**	**0**
0	1	0	**1**	**0**
0	1	1	**0**	**1**
1	0	0	**1**	**0**
1	0	1	**0**	**1**
1	1	0	**0**	**1**
1	1	1	**1**	**1**

Minterms

- BSP is often called sum of *minterms*
- Example: minterms with three variables

Minterm symbol	Minterm	Corrsponding binary number
m_0	$x'y'z'$	000
m_1	$x'y'z$	001
m_2	$x'yz'$	010
m_3	$x'yz$	011
m_4	$xy'z'$	100
m_5	$xy'z$	101
m_6	xyz'	110
m_7	xyz	111

Understanding minterms

- m_i corresponds to the binary number i
- *(Assume a 3-variable function)*
- If i = 1, *i* in binary is 001
- m_1 is $x'y'z$
  - *That is, if the digit is 0, negate the corresponding variable (x' and y')*
  - *If the digit is 1, leave the corresponding variable as is (z)*

Minterms Canonical Form

- A function is in canonical form if it is a sum of minterms
- Examples:
- $F1(x, y, z) = x'yz' + x'yz + xy'z$
  - $F1(x, y, z) = m_2 + m_3 + m_5$
  - $F1(x, y, z) = \sum(2, 3, 5)$
- $F2(x, y, z) = x'y'z' + x'y'z + x'yz' + x'yz$
  - $F2(x, y, z) = m_0 + m_1 + m_2 + m_3$
  - $F2(x, y, z) = \sum(0, 1, 2, 3)$

Karnaugh-Maps

- A K-map is a visual representation of a Boolean function in canonical form
- It helps in the process of simplifying a function
- An *n*-variable function in canonical form will have up to 2^n minterms
- The form in which the variable appear in a term (such as *x* or *x'*) is called a *literal*
- In an *n*-variable function, each minterm consists of *n* literals (e.g. *x'yz'* in a 3-variable function)

2-Variable Functions

- Minterms:

m_0	$x'y'$	00
m_1	$x'y$	01
m_2	xy'	10
m_3	xy	11

- 2-variable map:

m_0	m_1
m_2	m_3

2-Variable Maps

- $F1 = x'y' + x'y + xy$ is

m_0	m_1
m_2	m_3

- $F2 = x'y' + xy'$ is

m_0	m_1
m_2	m_3

Adjacency

m_0	m_1
m_2	m_3

- Two squares in a map are said to be adjacent if the corresponding minterms differ in one literal only.
- Example m_0 and m_1 are adjacent
  - $m_0 = x'y'$ and m_1 is $x'y$
  - Differ in y
  - Similarly, m_0 and m_2 are adjacent
  - m_2 and m_3
  - m_1 and m_3

Simplifying Functions

- Two adjacent squares can be combined to eliminate the different literal
- Example: $x'y + xy = y(x+x') = y(1) = y$

- A map gives us clues as to which minterms should be combined

Example

m_0	m_1
m_2	m_3

- $F(x, y) = \sum(1, 2, 3)$
- Combine 1 and 3
- Combine 2 and 3
- $F = x'y + xy' + xy$
- $F = x'y + xy + xy' + xy$ $\quad$ (since $xy + xy = xy$)
- $F = y(x'+x) + x(y'+y)$
- $F = y + x$

2-Variable Map Observations

- The number of adjacent squares that can be combined is a power of 2

1. A single square represents a minterm with 2 literals (e.g. $x'y$)
2. 2 adjacent squares represent one minterm with 1 literal (e.g. x)
3. 4 adjacent squares represent the function $F = 1$

3-Variable Functions

m_0	$x'y'z'$
m_1	$x'y'z$
m_2	$x'yz'$
m_3	$x'yz$
m_4	$xy'z'$
m_5	$xy'z$
m_6	xyz'
m_7	xyz

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6

Adjacency

- Same definition

Adjacent too

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6

Adjacency

- 0 and 2 are adjacent
  - 000 and 010: they differ in one bit
  - The corresponding minterms x’y’z’ and x’yz’ only differ in y

- Note that 1 and 2 are not adjacent
  - 001 and 010: they differ in more than one bit
  - Hence, the first row of the map is drawn as 0, 1, 3, 2 instead of 0, **1**, **2**, 3

Example

- $F(x, y, z) = \sum(2, 3, 4, 5)$

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6

- To simplify, combine
  - 3 and 2
  - 4 and 5

Example

- $F(x, y, z) = \sum(2, 3, 4, 5)$
- $F = x'yz' + x'yz + xy'z' + xy'z$
- $F = x'y\,(z'+z) + xy'(z'+z)$
- $F = x'y + xy'$
- $F = x \oplus y$

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6

Example

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6

- $F(x, y, z) = \sum(3, 4, 6, 7)$
- $F = x'yz + xy'z' + xyz' + xyz$
- *Combine*
  - *4 and 6*
  - *7 and 6*
- $F = x'yz + xy'z' + xyz' + xyz + \mathbf{xyz'}$
- $F = x'yz + xz' + xy$

Example

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6

- *Also combine*
  - *3 and 7*
- $F = x'yz + xz' + xy$
- $F = xz' + xy + x'yz + xyz$
- $F = xz' + xy + yz$

3-Variable Map Observations

1. A single square represents 1 minterm with three literals (e.g., $x'yz'$)
2. 2 adjacent squares represent one minterm with 2 literals (e.g., xy')
3. 4 adjacent squares represent one minterm with 1 literal (e.g., z)
4. 8 adjacent squares represent the function $F = 1$

Example

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6

- $F = x'y'z' + x'y'z + x'yz + x'yz'$
- *Combine (0,1), (1,3), (3,2), (2,0)*
- $F = x'$
- *One literal*

4-Vraiable Maps

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6
m_{12}	m_{13}	m_{15}	m_{14}
m_8	m_9	m_{11}	m_{10}

Adjacency

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6
m_{12}	m_{13}	m_{15}	m_{14}
m_8	m_9	m_{11}	m_{10}

4-Variable Map Observations

1. A single square represents 1 minterm with 4 literals (e.g., $wx'yz'$)
2. 2 adjacent squares represent one minterm with 3 literals (e.g., $xy'z$)
3. 4 adjacent squares represent one minterm with 2 literal (e.g., $w'z$)
4. 8 adjacent squares represent one minterm with 1 literal (e.g., z')
5. 16 adjacent squares is the function $F = 1$

Self-Test Quiz

- Draw a 4-variable K-map and simplify the following functions:

$$F(w, x, y, z) = \sum(0, 1, 5, 8, 9)$$

$$F(w, x, y, z) = \sum(1, 3, 5, 6, 9, 10, 11, 15)$$

Higher-Order K-Maps

- A 5-variable map is 3-D
- More dimensions are needed for higher order maps
- Useful only with computer programs

Optional Topics

- Maxterms
- Boolean product of sums (product of maxterms)
- 5-variable and 6-variable maps