CO20-320241

Computer Architecture and Programming Languages

CAPL

Lecture 3 & 4

Dr. Kinga Lipskoch

Fall 2019

Binary to Decimal Conversion

Convert binary data to decimal
by summing the positions that contain a 1

$$\begin{array}{cccccc} 5 & 4 & 3 & 2 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1_2 \\ \times & \times & \times & \times & \times & \times \end{array}$$

$$2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0 = 32 + 8 + 2 + 1 = 43_{10}$$

Decimal to Binary Conversion (1)

Two methods to convert decimal data to binary:

1. Reverse process described before
2. Use repeated division

Decimal to Binary Conversion (2)

Reverse process described before

- Note that all positions must be accounted for

$$43_{10} = 2^5 + 0 + 2^3 + 0 + 2^1 + 2^0$$
$$1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1_2$$

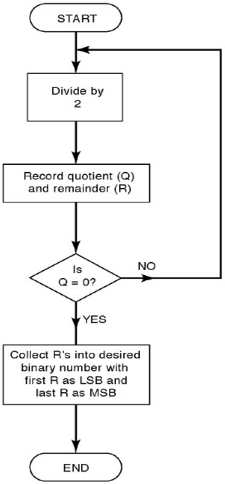
Decimal to Binary Conversion (3)

- Repeated integer division steps:
  - Divide the decimal number by 2
  - Write the remainder after each division until a quotient of zero is obtained
- The first remainder is the LSB (least significant bit) and the last one is the MSB (most significant bit)

Decimal to Binary Conversion (4)

Repeated integer division:
This flowchart describes the process and can be used to convert from decimal to any other number system

$43/2 = 21 \rightarrow 1$ (LSB)
$21/2 = 10 \rightarrow 1$
$10/2 = 5 \rightarrow 0$
$5/2 = 2 \rightarrow 1$
$2/2 = 1 \rightarrow 0$
$1/2 = 0 \rightarrow 1$ (MSB)



Hexadecimal Number System (1)

- Most digital systems deal with groups of bits in even powers of 2 such as 8, 16, 32, and 64 bits
- The hexadecimal system uses groups of 4 bits
- Base 16
  - 16 possible symbols
  - $0 - 9$ and $A - F$
- Allows for convenient handling of long binary strings

Hexadecimal Number System (2)

- Convert from hexadecimal to decimal by multiplying each hexadecimal digit by its positional weight
- Example:

$$\begin{aligned} 163_{16} &= 1\times(16^2)+6\times(16^1)+3\times(16^0) \\ &= 1\times 256+6\times 16+3\times 1 \\ &= 355_{10} \end{aligned}$$

Hexadecimal Number System (3)

- Convert from decimal to hexadecimal by using the repeated division method used for decimal to binary, decimal to octal conversion, etc.
- Divide the decimal number by 16
- The first remainder is the LSB and the last one is the MSB

Hexadecimal Numbers (1)

- Hexadecimal to binary conversion:

Decimal	0	1	2	3	4	5	6
Hexadecimal	0	1	2	3	4	5	6
Binary	0000	0001	0010	0011	0100	0101	0110

7	8	9	10	11	12	13	14	15
7	8	9	A	B	C	D	E	F
0111	1000	1001	1010	1011	1100	1101	1110	1111

- Example:

$$9F2_{16} = \begin{array}{ccc} 9 & F & 2 \\ 1001 & 1111 & 0010 \end{array} = 100111110010_2$$

Hexadecimal Numbers (3)

- Example of binary to hexadecimal conversion
  - Note the addition of leading zeroes

$$\begin{aligned} 1110100110_2 &= \mathbf{00}11 \quad 1010 \quad 0110 \\ &= \quad 3 \qquad A \qquad 6 \\ &= 3A6_{16} \end{aligned}$$

- Counting in hexadecimal requires a reset and carry after reaching F

Hexadecimal Numbers (4)

- Hexadecimal is useful for representing long strings of bits
- Understanding the conversion process and memorizing the 4 bit patterns for each hexadecimal digit will prove valuable later

Binary-Coded Decimal (BCD) (1)

- Another way to represent decimal numbers in binary form
- BCD is widely used and combines features of both decimal and binary systems
- Applications: electronic systems where a numeric value is to be displayed, small processors with limited computation power, date and time in BIOS
- Essence of BCD: Each digit is converted to its binary equivalent

Binary-Coded Decimal (BCD) (2)

- To convert the number 874_{10} to BCD:

  $$\begin{array}{ccc} 8 & 7 & 4 \\ 1000 & 0111 & 0100 = 100001110100_{BCD} \end{array}$$
- Each decimal digit is represented using 4 bits
- Each 4-bit group can never be greater than 9
- Reverse the process to convert BCD to decimal

Binary-Coded Decimal (BCD) (3)

- BCD is not a number system
- BCD is a decimal number with each digit encoded to its binary equivalent
- A BCD number is not the same as a straight binary number
- The primary advantage of BCD is the relative ease of converting to and from decimal

Bytes, Nibbles and Words

- 1 byte = 8 bits
- 1 nibble = 4 bits
- 1 word = size depends on data bus width
  - 32-bit system: 1 word = 32 bits
  - 64-bit system: 1 word = 64 bits

Alphanumeric Codes

- Represent characters and functions found on a computer keyboard
- ASCII – American Standard Code for Information Interchange
  - Seven-bit code: $2^7 = 128$ possible code groups
  - Was developed from telegraph code, first commercial use was as a seven-bit teleprinter code promoted by Bell data services (in 1960's)
  - Extended ASCII refers to eight-bit or larger character encodings
  - Table on next slide lists the standard ASCII codes
  - Examples of use: transfer information between computers, between computers and printers, and for internal storage

ASCII Table

32:		48:	0	64:	@	80:	P	96:	`	112:	p
33:	!	49:	1	65:	A	81:	Q	97:	a	113:	q
34:	"	50:	2	66:	B	82:	R	98:	b	114:	r
35:	#	51:	3	67:	C	83:	S	99:	c	115:	s
36:	$	52:	4	68:	D	84:	T	100:	d	116:	t
37:	%	53:	5	69:	E	85:	U	101:	e	117:	u
38:	&	54:	6	70:	F	86:	V	102:	f	118:	v
39:	'	55:	7	71:	G	87:	W	103:	g	119:	w
40:	(	56:	8	72:	H	88:	X	104:	h	120:	x
41:	)	57:	9	73:	I	89:	Y	105:	i	121:	y
42:	*	58:	:	74:	J	90:	Z	106:	j	122:	z
43:	+	59:	;	75:	K	91:	[	107:	k	123:	{
44:	,	60:	<	76:	L	92:	\	108:	l	124:	|
45:	-	61:	=	77:	M	93:	]	109:	m	125:	}
46:	.	62:	>	78:	N	94:	^	110:	n	126:	~
47:	/	63:	?	79:	O	95:	_	111:	o	127:	

Basic Logic Functions

- Information in digital computers is represented and processed by electronic networks called logic circuits
- These circuits operate on binary variables that assume one of two distinct values, usually called 0 and 1
- Boolean algebra is an important tool in describing, analyzing, designing, and implementing digital circuits

Boolean Constants and Variables

- Boolean algebra allows only two values 0 and 1
- Logic 0 can be:
  - false, off, low, no, open switch
- Logic 1 can be:
  - true, on, high, yes, closed switch
- Three basic logic operations: OR, AND, and NOT

Truth Tables

Examples of truth tables with 2, 3, and 4 inputs:

Inputs / Output

A	B	x
0	0	1
0	1	0
1	0	1
1	1	0

(a)

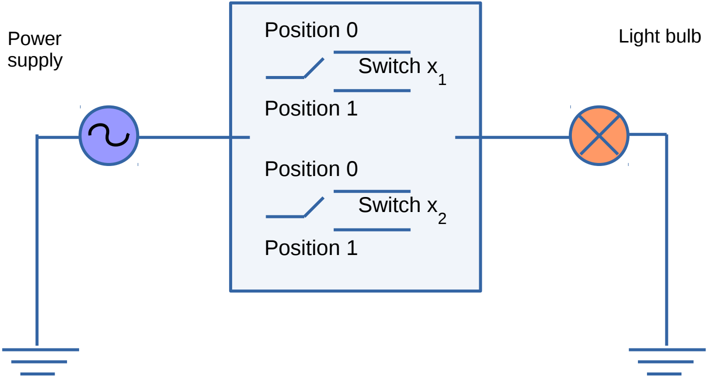
A	B	C	x
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

(b)

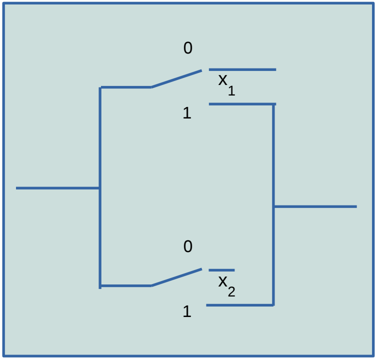
A	B	C	D	x
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

(c)

Simple Practical Problem



Parallel Connection (OR Control)

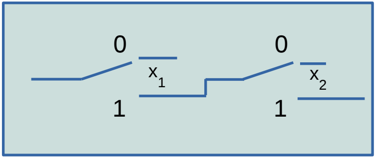


x_1	x_2	$f(x_1, x_2) = x_1 + x_2$
0	0	0
0	1	1
1	0	1
1	1	1

$$1 + x = 1$$

$$0 + x = x$$

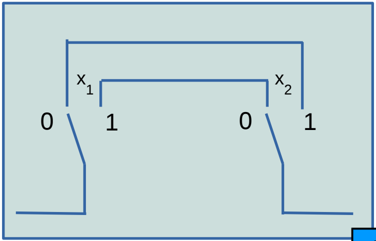
Series Connection (AND Control)



x_1	x_2	$f(x_1, x_2) = x_1 \bullet x_2$
0	0	0
0	1	0
1	0	0
1	1	1

$1 \bullet x = x$
$0 \bullet x = 0$

Exclusive-OR Connection (XOR Control)



$1 \oplus x = \overline{x}$
$0 \oplus x = x$

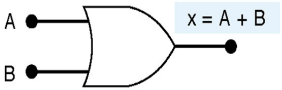
x_1	x_2	$f(x_1, x_2) = x_1 \oplus x_2$
0	0	0
0	1	1
1	0	1
1	1	0

OR Operation with OR Gates (1)

- The Boolean expression for the OR operation is $x = A + B$
- This is read as "x equals A OR B"
- $x = 1$ if $A = 1$ or $B = 1$
- Truth table and circuit symbol for a two input OR gate:

OR

A	B	x = A + B
0	0	0
0	1	1
1	0	1
1	1	1



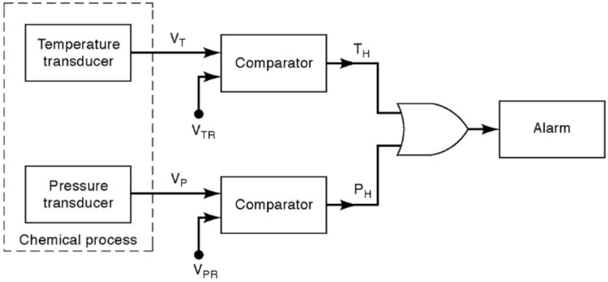
OR Gate

OR Operation with OR Gates (2)

- The OR operation is similar to addition, but if $A = 1$ and $B = 1$, the OR operation produces $1 + 1 = 1$
- $x = 1 + 1 + 1 = 1$
- In the Boolean expression
  $x = A + B + C$
  x is true (1) if
  - A is true (1) OR
  - B is true (1) OR
  - C is true (1)

OR Operation with OR Gates (3)

There are many examples of applications where an output function is desired if one of multiple inputs is activated

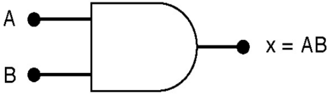


AND Operations with AND Gates (1)

- The Boolean expression for the AND operation is $x = A \cdot B$
- This is read as "x equals A AND B"
- $x = 1$ if $A = 1$ and $B = 1$
- Truth table and circuit symbol for a two input AND gate are shown
- Notice the difference between OR and AND gates

AND

A	B	$x = A \cdot B$
0	0	0
0	1	0
1	0	0
1	1	1



AND gate

AND Operation with AND Gates (2)

- The AND operation is similar to multiplication
- In the Boolean expression
  $x = A \cdot B \cdot C$
  $x = 1$ only if
  - $A = 1$ AND
  - $B = 1$ AND
  - $C = 1$

NOT Operation (1)

- The Boolean expression for the NOT operation is $x = \overline{A}$
- This is read as:
  - x equals NOT A, or
  - x equals the inverse of A, or
  - x equals the complement of A

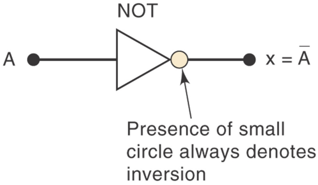
NOT Operation (2)

Truth table, symbol, and sample waveform for the NOT circuit

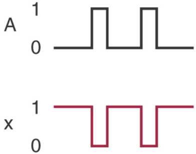
NOT

A	$x = \overline{A}$
0	1
1	0

(a)



(b)



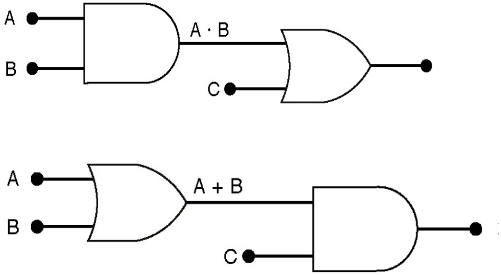
(c)

Describing Logic Circuits Algebraically (1)

- The three basic Boolean operations (OR, AND, NOT) can describe any logic circuit
- If an expression contains both AND and OR gates then the AND operation will be performed first, unless there is a parenthesis in the expression

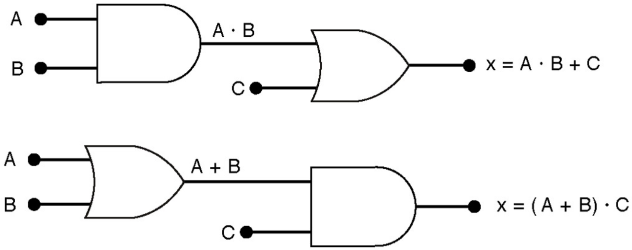
Describing Logic Circuits Algebraically (2)

Examples of Boolean expressions for logic circuits:



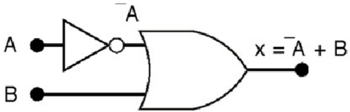
Describing Logic Circuits Algebraically (3)

Examples of Boolean expressions for logic circuits:



Describing Logic Circuits Algebraically (4)

- Input A through an inverter equals $\overline{A}$
- The output of an inverter is equivalent to the input with a bar over it
- Example using an inverter:



Evaluating Logic Circuit Outputs (1)

Rules for evaluating a Boolean expression:

- Perform all inversions of single terms
- Perform all operations within parenthesis
- Perform AND operation before an OR operation unless parenthesis indicates otherwise
- If an expression has a bar over it, perform the operations inside the expression and then invert the result

Evaluating Logic Circuit Outputs (2)

- Evaluate Boolean expressions by substituting values and performing the indicated operations
- Example:

$$A = 0, B = 1, C = 1, \text{and } D = 1$$

$$\begin{aligned} x &= \overline{A}BC\overline{(A + D)} \\ &= \overline{0} \cdot 1 \cdot 1 \cdot \overline{(0 + 1)} \\ &= 1 \cdot 1 \cdot 1 \cdot \overline{(0 + 1)} \\ &= 1 \cdot 1 \cdot 1 \cdot (\overline{1}) \\ &= 1 \cdot 1 \cdot 1 \cdot 0 \\ &= 0 \end{aligned}$$

Implementing Circuits From Boolean Expressions

- The expression
  $x = A \cdot B \cdot C$
  could be drawn as a three input AND gate
- A more complex example such as
  $x = AC + B\overline{C} + \overline{A}BC$
  - Could be drawn as two 2-input AND gates and one 3-input AND gate feeding into a 3-input OR gate
  - Two of the AND gates have inverted inputs

Truth Table for Some Circuit

$$f = \overline{x}_1 \cdot x_2 + x_1 \cdot \overline{x}_2 = x_1 \oplus x_2$$
$$f = ((\overline{x}_1) \cdot x_2) + (x_1 \cdot (\overline{x}_2)) = x_1 \oplus x_2$$

$x_1\ x_2$	$\overline{x}_1 \cdot x_2$	$x_1 \cdot \overline{x}_2$	f
0 0	0	0	0
0 1	1	0	1
1 0	0	1	1
1 1	0	0	0

Truth Table for Three Inputs

x_1	x_2	x_3	f_1	f_2
0	0	0	1	1
0	0	1	1	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	1
1	1	0	0	0
1	1	1	1	0

Synthesize Algebraic Function from Truth Table

Determine f_1 using AND, OR and NOT gates

- For each row where $f_1 = 1$ include product term (AND) in sum of products form
- Use x_i, if $x_i = 1$
- Use $\overline{x_i}$, if $x_i = 0$
- Fourth row: $(x_1, x_2, x_3) = (0, 1, 1)$
- Product term: $\overline{x_1}x_2x_3$
- $f_1 = \overline{x_1}\ \overline{x_2}\ \overline{x_3} + \overline{x_1}\ \overline{x_2}\ x_3 + \overline{x_1}\ x_2\ x_3 + x_1\ x_2\ x_3$

x_1	x_2	x_3	f_1	f_2
0	0	0	1	1
0	0	1	1	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	1
1	1	0	0	0
1	1	1	1	0

Minimization of Logic Expressions

- $w(y+z) = wy + wz$ (distributive rule)
- $w + \overline{w} = 1$ (identity)
- $w\overline{w} = 0$
- $w + w = w$
- $ww = w$
- $w + wy = w$
- $(w+y)(w+z) = w + yz$

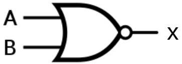
- $$\begin{aligned} f_1 &= \overline{x_1}\ \overline{x_2}\ \overline{x_3} + \overline{x_1}\ \overline{x_2}\ x_3 + \overline{x_1}\ x_2\ x_3 + x_1\ x_2\ x_3 \\ &= \overline{x_1}\ \overline{x_2}(\overline{x_3} + x_3) + (\overline{x_1} + x_1)x_2x_3 \\ &= \overline{x_1}\ \overline{x_2} \quad 1 \quad + \quad 1 \quad x_2x_3 \\ &= \overline{x_1}\ \overline{x_2} + x_2x_3 \end{aligned}$$

NOR Gates and NAND Gates (1)

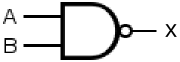
- Combine basic AND, OR, and NOT operations
- The NOR gate is an inverted OR gate
- An inversion "bubble" is placed at the output of the OR gate
- The NAND gate is an inverted AND gate
- The Boolean expression for NOR is:
  $x = \overline{A + B}$
- The Boolean expression for NAND is:
  $x = \overline{A \cdot B}$

NOR Gates and NAND Gates (2)

- NOR gate:



- NAND gate:



Rules of Binary Logic

Name	Algebraic identity	
Commutative	$w + y = y + w$	$wy = yw$
Associative	$(w + y) + z = w + (y + z)$	$(wy)z = w(yz)$
Distributive	$w + yz = (w + y)(w + z)$	$w(y + z) = wy + wz$
Idempotent	$w + w = w$	$ww = w$
Involution	$\overline{\overline{w}} = w$	
Complement	$w + \overline{w} = 1$	$w\overline{w} = 0$
de Morgan	$\overline{w + y} = \overline{w}\ \overline{y}$	$\overline{wy} = \overline{w} + \overline{y}$
	$1 + w = 1$	$0 \cdot w = 0$
	$0 + w = w$	$1 \cdot w = w$

DeMorgan's Theorems

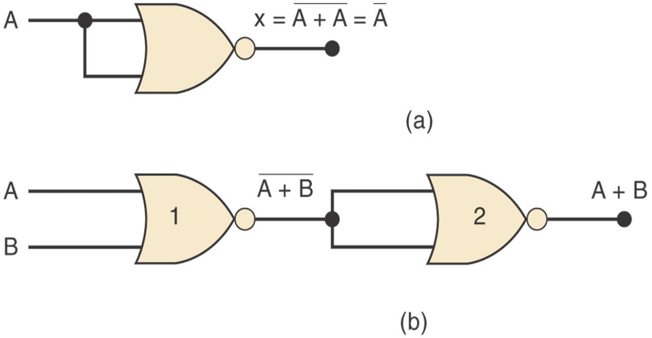
- When the OR sum of two variables is inverted, it is equivalent to inverting each variable individually and ANDing them
- When the AND product of two variables is inverted, it is equivalent to inverting each variable individually and ORing them

- A NOR gate is equivalent to an AND gate with inverted inputs
- A NAND gate is equivalent to an OR gate with inverted inputs

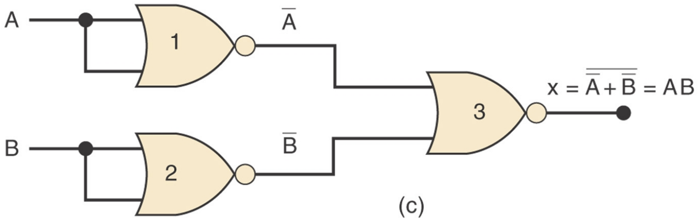
Universality of NAND and NOR Gates

- NAND or NOR gates can be used to implement the three basic logic functions/expressions (OR, AND, and INVERT)
- Next figures illustrate how combinations of NANDs or NORs are used to create the three logic functions
- This characteristic provides flexibility and is very useful in logic circuit design

Using NOR to Implement NOT, OR



Three NOR for Implementing AND



(c)

Summary of Methods to Describe Logic Circuits

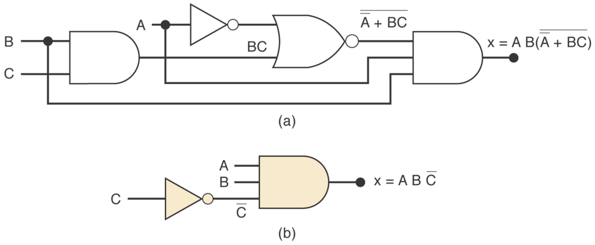
- Basic logic gate functions can be combined in combinational logic circuits
- Simplification of logic circuits can be done using Boolean algebra or a mapping technique (explained later)

Sum-of-Products Form

- A sum-of-products (SOP) expression will appear as two or more AND terms which are ORed together
- Examples:
  $A\ B\ C + \overline{A}\ B\ \overline{C}$
  $A\ B + \overline{A}\ B\ \overline{C} + \overline{C}\ \overline{D} + D$

Simplifying Logic Circuits

- The circuits below both provide the same output, but the lower one is clearly less complex



- We study simplifying logic circuits using Boolean algebra and Karnaugh-mapping

Algebraic Simplification (Minimization)

- Place the expression in SOP form by applying DeMorgan's theorems and multiplying terms
- Check the SOP form for common factors and perform factoring where possible
- Note that this process may involve some trial and error to obtain the simplest result

Designing Combinational Logic Circuits

Steps from problem to corresponding logic circuit:

- Interpret the problem and set up its truth table
- Write the AND (product) term for each case where the output equals 1
- Combine the terms in SOP form
- Simplify the output expression if possible
- Implement the circuit for the final, simplified expression

Gray Code

- The gray code is used in applications where numbers/values change rapidly
- Originally designed to prevent spurious output from electromechanical switches
- In the gray code, only one bit changes from each value to the next one
- Applications: used in digital television, genetic algorithms, plays important role in error correction, helps to label Karnaugh-maps, etc.

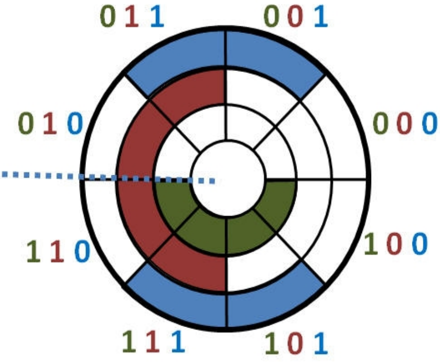
Gray Code on 2 Bits

Binary Counting	Gray Code
00	00
01	01
10	11
11	10

Gray Code on 3 Bits

Binary Counting	Gray Code
000	000
001	001
010	011
011	010
100	110
101	111
110	101
111	100

Gray Code Ring on 3 Bits



$Q_2Q_1Q_0$
000
001
011
010
110
111
101
100

Karnaugh-Map Method (1)

- A graphical method of simplifying logic expressions or truth tables
- It is also called a K-map
- Theoretically can be used for any number of input variables, but practically limited to up to 5 or 6 variables

Karnaugh-Map Method (2)

- The truth table values are placed in the K-map as shown on next slides
- Adjacent K-map cells differ in only one variable both horizontally and vertically
- The pattern from top to bottom and left to right must be in the form $\overline{A}\ \overline{B}$, $\overline{A}\ B$, $A\ B$, $A\ \overline{B}$
- A SOP expression can be obtained by ORing all cells that contain a 1

Karnaugh-Map Method (3)

- Looping adjacent groups of 2, 4, or 8 1s will result in further simplification
- When the largest possible groups have been looped, only the common terms are placed in the final expression
- Looping may also be wrapped between top, bottom, and sides
- Next slides illustrate this looping

Truth Table and K-map for Two Variables

$$x = \overline{A}\,\overline{B} + A\,B$$

A	B		x
0	0		1
0	1		0
1	0		0
1	1		1

$\rightarrow \overline{A}\,\overline{B}$

$\rightarrow A\,B$

	$\overline{B}$	B
$\overline{A}$	1	0
A	0	1

Truth Table and K-map for Three Variables

A	B	C		x	
0	0	0		1	$\rightarrow \overline{A}\,\overline{B}\,\overline{C}$
0	0	1		1	$\rightarrow \overline{A}\,\overline{B}\,C$
0	1	0		1	$\rightarrow \overline{A}\,B\,\overline{C}$
0	1	1		0	
1	0	0		0	
1	0	1		0	
1	1	0		1	$\rightarrow A\,B\,\overline{C}$
1	1	1		0	

	$\overline{C}$	C
$\overline{A}\,\overline{B}$	1	1
$\overline{A}\,B$	1	0
$A\,B$	1	0
$A\,\overline{B}$	0	0

Looping Groups of Two (1)

- Vertically adjacent 1s
- Can be looped (combined) to eliminate A
- Can also be applied to horizontally adjacent 1s

$$\begin{aligned} x &= \overline{A}\,B\,\overline{C} + A\,B\,\overline{C} \\ &= B\,\overline{C}\,(\overline{A} + A) \\ &= B\,\overline{C}\,(1) \\ &= B\,\overline{C} \end{aligned}$$

	$\overline{C}$	C
$\overline{A}\,\overline{B}$	0	0
$\overline{A}\,B$	1	0
$A\,B$	1	0
$A\,\overline{B}$	0	0

$$\begin{aligned} x &= \overline{A}\,B\,\overline{C} + A\,B\,\overline{C} \\ &= B\,\overline{C} \end{aligned}$$

Looping Groups of Two (2)

- In a K-map the top row and bottom row are considered to be adjacent
- Can also be looped (combined) to eliminate A
- Same applies to horizontally adjacent columns

	$\overline{C}$	C
$\overline{A}\,\overline{B}$	1	0
$\overline{A}\,B$	0	0
$A\,B$	0	0
$A\,\overline{B}$	1	0

$$x = \overline{A}\,\overline{B}\,\overline{C} + A\,\overline{B}\,\overline{C} = \overline{B}\,\overline{C}$$

Karnaugh-Map Method (1)

By looping (combining) a pair of adjacent 1s the variable that appears in both complemented and uncomplemented form can be eliminated

Looping Groups of Four (Quads) (1)

- A quad is a group of four adjacent 1s
- When a quad is looped, the resultant term contains only variables that do not change their form for all cells in the quad

	$\bar{C}$	C
$\bar{A}\,\bar{B}$	0	1
$\bar{A}\,B$	0	1
$A\,B$	0	1
$A\,\bar{B}$	0	1

$x = C$

$$\begin{aligned} x &= \bar{A}\,\bar{B}\,C + \bar{A}\,B\,C + A\,B\,C + A\,\bar{B}\,C \\ &= \bar{A}\,C(\bar{B} + B) + A\,C(B + \bar{B}) \\ &= \bar{A}\,C + A\,C \\ &= C(\bar{A} + A) = C \end{aligned}$$

Looping Groups of Four (Quads) (2)

	$\overline{C}\,\overline{D}$	$\overline{C}\,D$	$C\,D$	$C\,\overline{D}$
$\overline{A}\,\overline{B}$	0	0	0	0
$\overline{A}\,B$	0	1	1	0
$A\,B$	0	1	1	0
$A\,\overline{B}$	0	0	0	0

x = B D

x = try at home

Looping Groups of Four (Quads) (3)

- These 1s are all adjacent to each other
- Looping a variable of adjacent 1s eliminates the two that appear both in uncomplemented and complented form

	$\overline{C}\,\overline{D}$	$\overline{C}\,D$	$C\,D$	$C\,\overline{D}$
$\overline{A}\,\overline{B}$	1	0	0	1
$\overline{A}\,B$	0	0	0	0
$A\,B$	0	0	0	0
$A\,\overline{B}$	1	0	0	1

$$x = \overline{B}\,\overline{D}$$