Review for Test 1 : Ch1–5

October 5, 2006

Positional Numbers

$527.46_{10} = (5 \times 10^2) + (2 \times 10^1) + (7 \times 10^0) + (4 \times 10^{-1}) + (6 \times 10^{-2})$

$527.46_8 = (5 \times 8^2) + (2 \times 8^1) + (7 \times 8^0) + (4 \times 8^{-1}) + (6 \times 8^{-2})$

$527.46_5 =$ illegal – why?

$101011.11_2 = (1 \times 2^5) + (0 \times 2^4) + (1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) + (1 \times 2^{-1}) + (1 \times 2^{-2})$

This works for binary as well...

Positional Number Conversion

Decimal Base 10	Binary Base 2	Octal Base 8	Hexadecimal base 16
00	00000	00	00
01	00001	01	01
02	00010	02	02
03	00011	03	03
04	00100	04	04
05	00101	05	05
06	00110	06	06
07	00111	07	07
08	01000	10	08
09	01001	11	09
10	01010	12	0A
11	01011	13	0B
12	01100	14	0C
13	01101	15	0D
14	01110	16	0E
15	01111	17	0F
16	10000	20	10

Binary Coded Decimal (BCD)

Decimal	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Convert 2496_{10} to BCD code:

2 4 9 6

↓ ↓ ↓ ↓

0010 0100 1001 0110

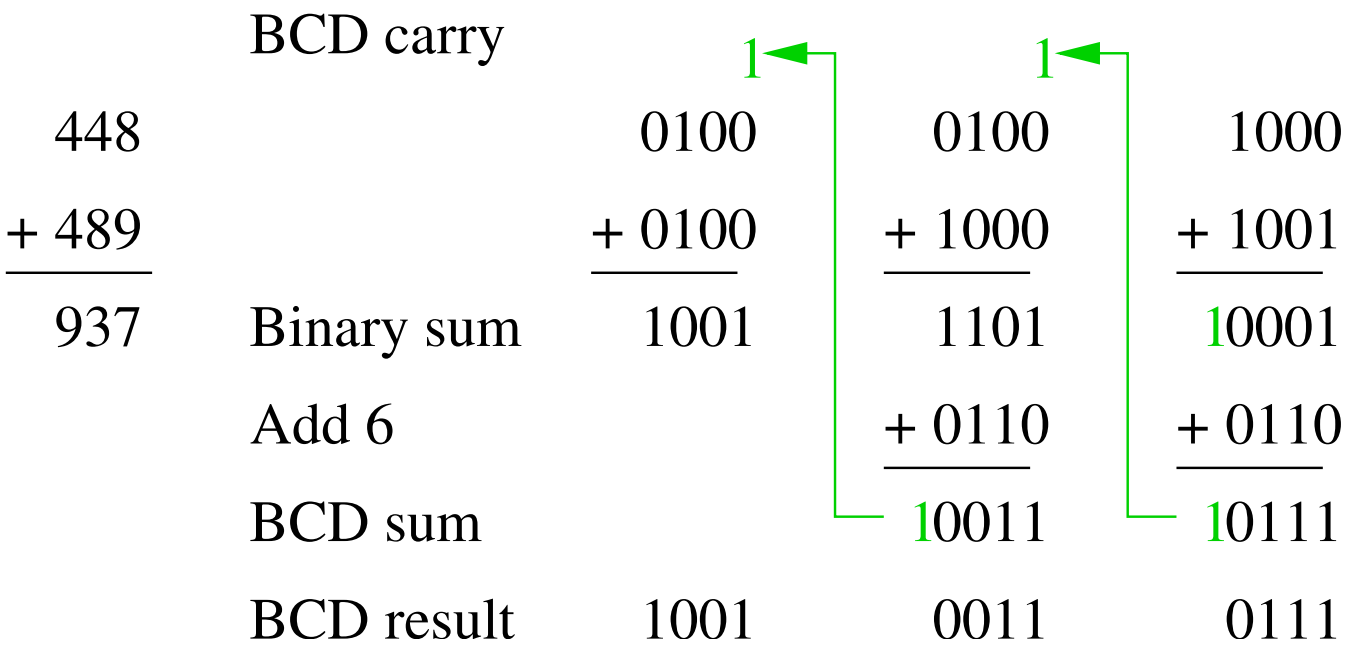
Not this is very different from converting to binary which yields:

$$100111000000_2$$

In BCD ...

$$0010010010010110$$

BCD Addition



Add each digit. If the result is greater than 9, add 6 and carry any overflow to the next digit. Repeat.

Binary Codes - ASCII

Character	ASCII Code
c	1 1 0 0 0 1 1
d	1 1 0 0 1 0 0
e	1 1 0 0 1 0 1
f	1 1 0 0 1 1 0
g	1 1 0 0 1 1 1
h	1 1 0 1 0 0 0
I	1 1 0 1 0 0 1
j	1 1 0 1 0 1 0
k	1 1 0 1 0 1 1
l	1 1 0 1 1 0 0
m	1 1 0 1 1 0 1
n	1 1 0 1 1 1 0
o	1 1 0 1 1 1 1
p	1 1 1 0 0 0 0
q	1 1 1 0 0 0 1

Convert "help" to ASCII

h	e	l	p
110100	1100101	1101100	1111000

Gray Codes

Number	Binary	Gray Code
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

- Only one bit changes with each number increment
- Not a weighted code
- Useful for interfacing to some physical systems

Boolean Algebra

Objectives

- Understand basic Boolean Algebra
- Relate Boolean Algebra to Logic Networks
- Prove Laws using Truth Tables
- Understand and Use First 11 Theorems
- Apply Boolean Algebra to:
  - Simplifying Expressions
  - Multiplying Out Expressions
  - Factoring Expressions

Truth Tables

A truth table provides a *complete enumeration* of the nputs and the corresponding output for a function.

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

If there n inputs, there will be 2^n rows in the table.

Unlike with regular algebra, full enumeration is poss ible (and useful) in Boolean Algebra.

Boolean Expressions

Boolean expressions are made up of variables and constants combined by AND, OR and NOT.

Examples: 1, A', $A \bullet B$, $C + D$, AB, $A(B + C)$, $AB + C$

$A \bullet B$ is the same as AB ($\bullet$ is omitted when obviou s) Parentheses are used like in regular algebra for grouping.

A **literal** is each instance of a variable or constant.

This expression has 4 variables and 10 literals:

$$a'bd + bcd + ac' + a'd'$$

Basic Boolean Algebra Theorems

Here are the first five Boolean Algebra theorems we will study and use :

$X + 0 = X$	$X \bullet 1 = X$
$X + 1 = 1$	$X \bullet 0 = 0$
$X + X = X$	$X \bullet X = X$
$(X')' = X$	
$X + X' = 1$	$X \bullet X' = 0$

Basic Boolean Algebra Theorems

While these laws don't seem very exciting, they can be very useful in simplifying Boolean expressions:

Simplify:

$$\underbrace{\underbrace{(MN' + M'N)\ P + P'}_{X} + 1}_{1}$$

Boolean Algebra Theorems

Commutative Laws

$$X \bullet Y = Y \bullet X \qquad X + Y = Y + X$$

Associative Laws

$$(X \bullet Y) \bullet X = X \bullet (\ Y \bullet Z) = X \bullet Y \bullet Z$$

$$(\ X + Y\ ) + Z = X + (\ Y + Z\ ) = X + Y + Z$$

Just like regular algebra

Distributive Law

$$X(Y+Z) = XY + XZ$$

Prove with a truth table:

X	Y	Z	Y+Z	X(Y+Z)	XY	XZ	XY + XZ
0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	0	0	0	0
1	0	0	0	0	0	0	0
1	0	1	1	1	0	1	1
1	1	0	1	1	0	0	1
1	1	1	1	1	1	1	1

Again, like algebra

Other Distributive Law

Proof: $X + YZ = (X + Y)(X + Z)$

$$\begin{aligned}
(X+Y)(X+Z) &= X(X+Z) + Y(X+Z) \\
&= XX + XZ + YX + YZ \\
&= X + XZ + XY + YZ \\
&= X \bullet 1 + XZ + XY + YZ \\
&= X(1 + Z + Y) + YZ \\
&= X \bullet 1 + YZ \\
&= X + YZ
\end{aligned}$$

NOT like regular algebra!

Simplification Theorems

$$X\,Y + X\,Y' = X \qquad (X + Y)(X + Y') = X$$
$$X + X\,Y = X \qquad X\,(X + Y) = X$$
$$(X + Y')\,Y = X\,Y \qquad X\,Y' + Y = X + Y$$

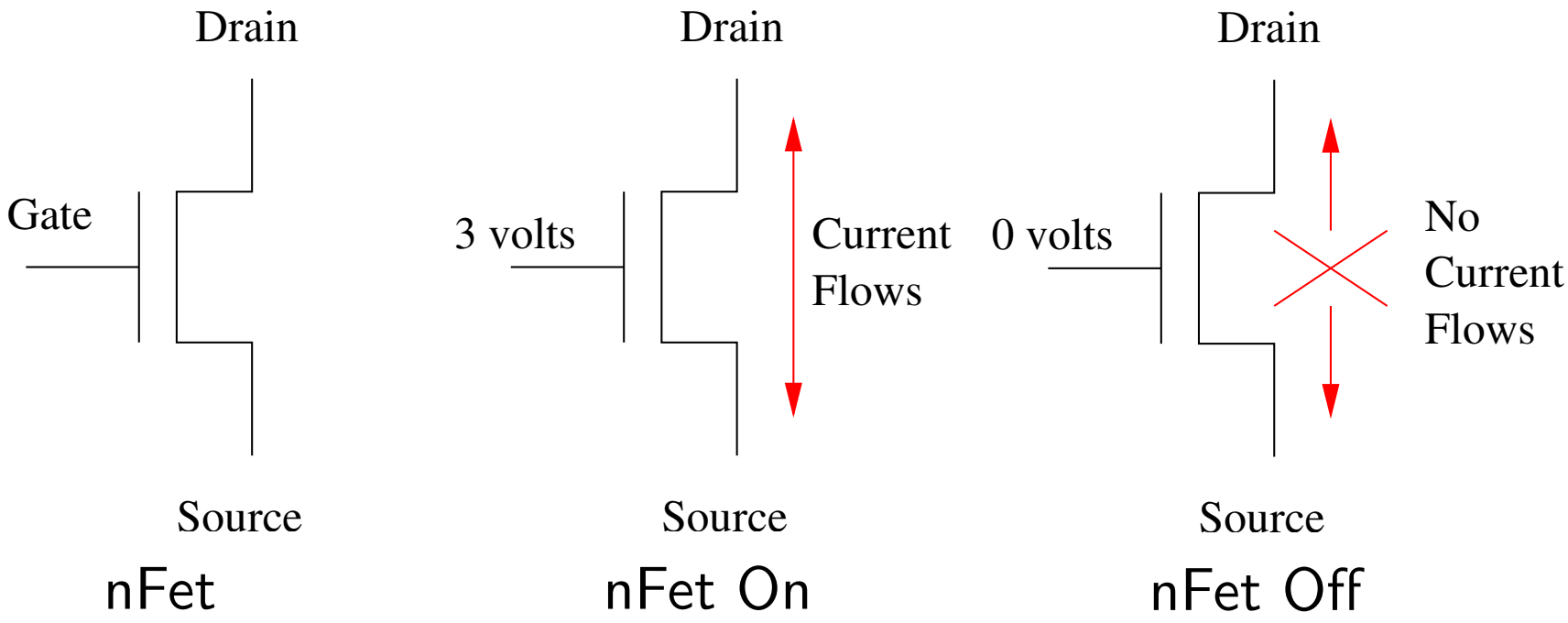
These are useful for simplifying Boolean Expressions.

The trick is to find X and Y.

$$(A' + B + CD)(B' + A' + CD)$$
$$(A' + CD + B)(A' + CD + B')$$
$$A' + CD$$

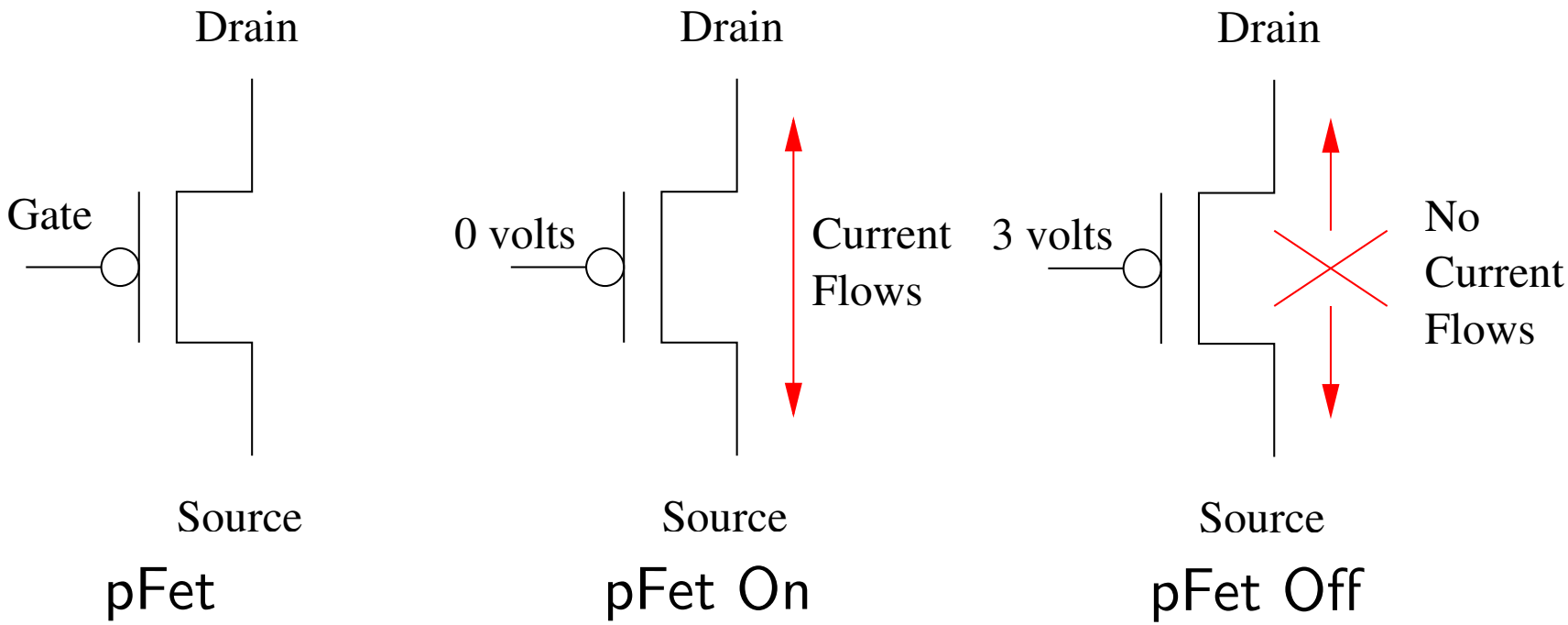
Using the rule at the top right.

Gates are built with Transistors



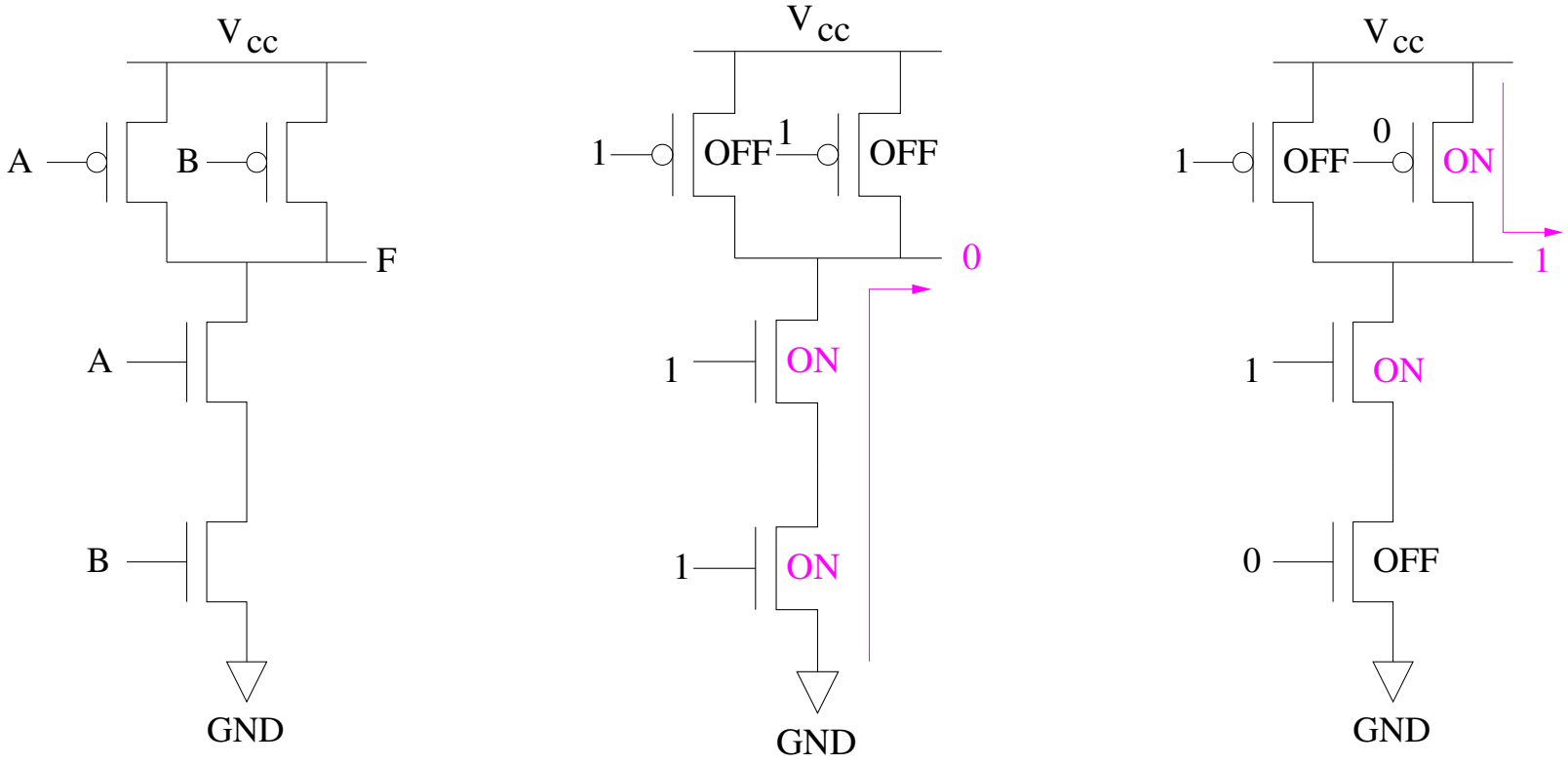
N-type field-effect transistor = nFet

Gates are built with Transistors



P-type field-effect transistor = pFet

FET-Based NAND Gate



DeMorgan's Laws - One Step Rule

$$(f(X_1, X_2, ...X_N, 0, 1, +, \bullet))' = f(X_1', X_2', ...X_N', 1, 0, \bullet, +)$$

1. Replace all variables with the inverse.

2. Replace $+$ with $\bullet$ and $\bullet$ with $+$.

3. Replace 0 with 1 and 1 with 0.

Be careful of hierarchy...

This is the biggest source of errors, when applying DeMorgan's Laws. Before beginning, surround all AND terms with parentheses.

Minterm Expansion

- A minterm expansion is *unique*.

$$f(A, B, C, D) = \sum \mathsf{m}(0, 2, 3, 7)$$

- Useful for:
  - Proving equality
  - Shorthand for representing boolean expressions

Maxterm Expansion

Any function can be written as a product of maxterms. This is called a:

Standard Product of Sums
(Standard POS)

Use the **Zeros** for f to write the POS:

A	B	C	f	
0	0	0	0	M0
0	0	1	1	M1
0	1	0	0	M2
0	1	1	0	M3
1	0	0	0	M4
1	0	1	1	M5
1	1	0	1	M6
1	1	1	1	M7

$$f(A,B,C) = M_0 M_2 M_3 M_4$$

$$f(A,B,C) = \prod M(0,2,3,4)$$

$$f = (A+B+C)(A+B'+C)(A+B'C')(A'+B+C)$$

Algebraic Simplification: Which Theorems To Use?

Essential Identities	
$X + 0 = X$	$X \bullet 1 = X$
$X + 1 = 1$	$X \bullet 0 = 0$
$X + X = x$	$X \bullet X = X$
$(X')' = X$	
$X + X' = 1$	$X \bullet X' = 0$
Essential Commutative, Associative, Distributive and DeMorgan's Laws	
$X + Y = Y + X$	$X \bullet Y = Y \bullet X$
$(X + Y) + Z = X + (Y + Z) = X+Y+Z$	$(XY)Z = X(YZ) = XYZ$
$X(Y + Z) = XY + XZ$	$X + YZ = (X + Y) (X + Z)$
$[f(X_1, X_2, ...X_N, 0, 1, +, \bullet)]' = f(X'_1, X'_2, ...X'_N, 1, 0, \bullet, +)$	
Essential	
$X Y + X Y' = X$	$(X + Y) (X + Y') = X$
$X + XY = X$	$X(X+Y)=X$
Useful, hard to remember, easy to re-derive	
$(X + Y') Y = XY$	$XY' + Y = X + Y$

Suggestions:

1. Focus on blue ones!
2. Create duals onright as needed.
3. Be familiar with the last group.

Four Methods of Algebraic Simplification

1. Combine terms
2. Eliminate terms
3. Eliminate literals
4. Add redundant terms

Converting English to Boolean Expressions

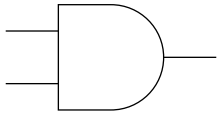
Review

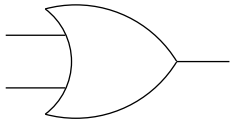
Converting English to Boolean

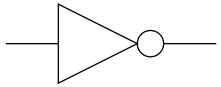
1. Identify phrases
2. Identify connective words
3. Construct a Boolean Expression
4. Draw the network

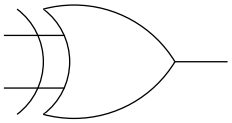
Types of gates

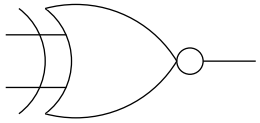
Gates already studied:

AND 

OR 

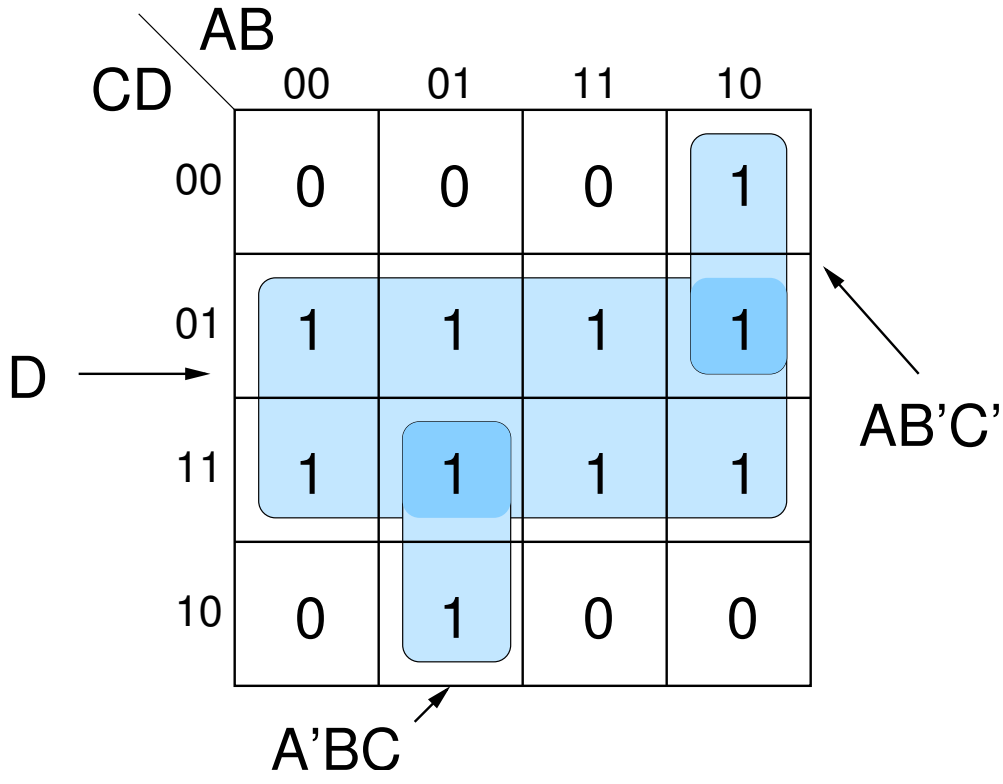
Inverters 

Exclusive–OR 

Equivalence 

Four Variable Karnaugh Map

CD \ AB	00	01	11	10
00	m0	m4	m12	m8
01	m1	m5	m13	m9
11	m3	m7	m15	m11
10	m2	m6	m14	m10



F = A'BC + AB'C + D

Note the row and column numbering. This is required for adjacency.

K-Map Solution Summary

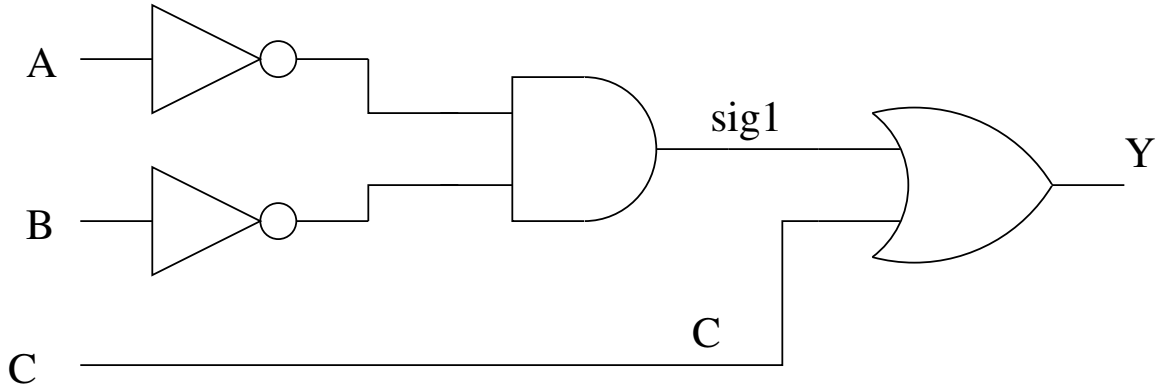
- Identify prime implicants.
- Add essentials to solution.
- Find minimum number non-essentials required to cover rest of map.

Design Hierarchy

- Design complexity requires a *divide and conquer* approach
- Circuit $\longrightarrow$ *blocks*
- Each block is a distinct function
- Blocks are interconnected.
- Complex blocks are broken down into simpler blocks.
- Blocks are combined to form a *system*.

Basic VHDL building Blocks

Consider the following circuit:




```
-- ENTITY
entity few_gates is
  port(
     a : in std_logic;
     b : in std_logic;
     c : in std_logic;
     y : out std_logic
    );
end fewgates;
```

```
-- ARCHITECTURE
architecture behavior of fewgates is
  signal sig1 : std_logic;
 begin
     sig1 <= (not a) and (not b);
     y <= c or sig1;
end behavior;
```

Design Procedure

- Specification
- Formulation - create truth table or boolean equations.
- Optimization - reduce requirements to achieve goal
- Technology Mapping - transform logic diagram to a new diagram or netlist using available technology
- Verification - check the correctness of the final design

Binary Arithmetic

- Binary Addition and Subtraction
- Overflow
- Sign-Magnitude
- One and Twos-complement
- Binary Adder/Subtractors

Binary Arithmetic Comparison

	Sign Magnitude	One's Complement	Two's Complement
Negative Number	Easiest to Understand Simple to Compute	Easy to Compute	Hardest to Compute
Zeroes	2 Zeroes	2 Zeroes	1 Zero
Largest Number	Same number of + and - Numbers	Same number of + and - Numbers	One Extra Negative Number
Logic Required	Requires Adder and Subtracter Extra Logic to Identify Larger Operand, Compute Sign, etc.	Only Adder Required Carry Wraps Around	Only Adder Required -
Overflow Detection	Overflow: Carry from High Order Adder Bits	Overflow: Sign of Both Operands is the Same and Sign of Sum is Different	Overflow: Sign of Both Operands is the Same and Sign of Sum is Different

ROM, Decoders and Muxes

Know them all!

Programmable Logic

- Read Only Memory (ROM) – a fixed array of AND gates and a programmable array of OR gates.
- Programmable Array Logic (PAL) – a programmable array of AND gates feeding a fixed array of OR gates.
- Programmable Logic Array (PLA) – a programmable array of AND gates feeding a programmable array of OR gates.
- Complex Programmable Logic Device (CPLD)/Field- Programmable Gate Array (FPGA) - complex enough to be called "architectures"

Adders and Multiplication

- Iterative combinational circuits
- Binary adders
  - Half and full adders
  - Ripple carry and carry lookahead adders
- Binary subtraction
- Binary adder-subtractors
- Binary multiplication

- Other arithmetic functions (constant inputs)