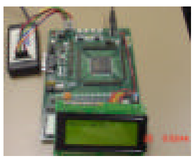
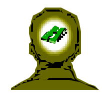
CPE/EE 422/522
Advanced Logic Design

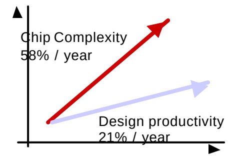
Electrical and Computer Engineering
University of Alabama in Huntsville



Motivation

- Benefits of HDL-based design
  - Portability
  - Technology independence
  - Design cycle reduction
  - Automatic synthesis and Logic optimization
- … But, the gap between available chip complexity and design productivity continues to increase

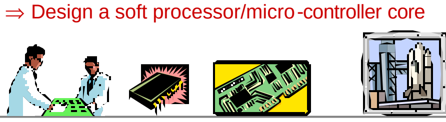




Educators Mission

- Educate future generations of designers
  - Emphasis on hierarchical IP core design
  - Design systems, not components!
  - Understand hardware/software co-design
  - Understand and explore design tradeoffs between complexity, performance, and power consumption

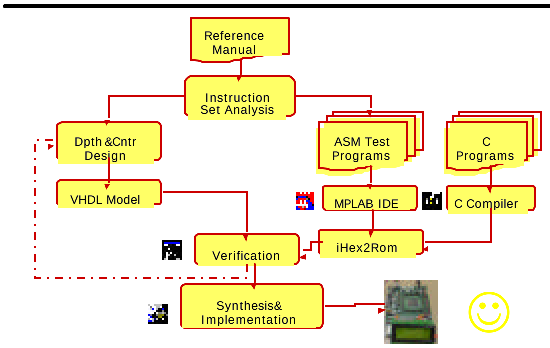
$\Rightarrow$ Design a soft processor/micro-controller core



UAH Library of Soft Cores

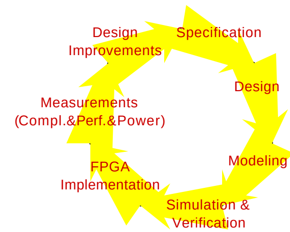
- Microchip's PIC18 micro-controller
- Microchip's PIC16 micro-controller
- Intel's 8051
- ARM Integer CPU core
- FP10 Floating-point Unit

Design Flow



Benefits

- Proposed project-based approach encompasses the whole engineering cycle



- Put together knowledge in digital design, HDLs, computer architecture, programming languages
- State-of-the-art devices
- Work in teams

PIC18 Greetings

Outline

Review of Logic Design Fundamentals

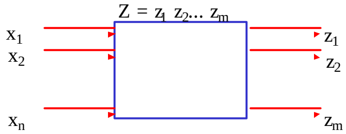
- Combinational Logic
- Boolean Algebra and Algebraic Simplifications
- Karnaugh Maps
- Combinational-Circuit Building Blocks

Combinational Logic

- Has no memory =>
  present state depends only on the present input

$X = x_1\ x_2 \ldots x_n$

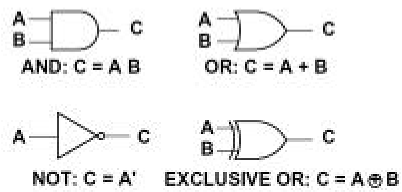
$Z = z_1\ z_2 \ldots z_m$



$$Z(t) = F(X(t))$$

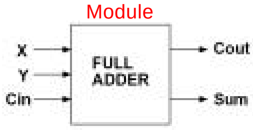
Note:
Positive Logic – low voltage corresponds to a logic 0, high voltage to a logic 1
Negative Logic – low voltage corresponds to a logic 1, high voltage to a logic 0

Basic Logic Gates



Full Adder

Module



Truth table

X	Y	Cin	Cout	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Algebraic expressions
F(inputs for which the function is 1):

Minterms

$Sum = X'Y'Cin + X'YCin' + XY'Cin' + XYCin$

$Cout = X'YCin + XY'Cin + XYCin' + XYCin$

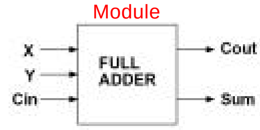
m-notation

$Sum = m_1 + m_2 + m_4 + m_7 = \sum m(1, 2, 4, 7)$

$Cout = m_3 + m_5 + m_6 + m_7 = \sum m(3, 5, 6, 7)$

Full Adder (cont'd)

Module



Truth table

X	Y	Cin	Cout	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Algebraic expressions
F(inputs for which the function is 0):

Maxterms

$Sum = (X + Y + Cin)(X + Y' + Cin')(X' + Y + Cin')(X' + Y' + Cin)$

$Cout = (X + Y + Cin)(X + Y + Cin')(X + Y' + Cin)(X' + Y + Cin)$

M-notation

$Sum = M_1 \cdot M_3 \cdot M_5 \cdot M_6 = \prod M(1, 3, 5, 6)$

$Cout = M_0 \cdot M_1 \cdot M_2 \cdot M_4 = \prod M(0, 1, 2, 4)$

Boolean Algebra

- Basic mathematics used for logic design
- Laws and theorems can be used to simplify logic functions
  - Why do we want to simplify logic functions?

Laws and Theorems of Boolean Algebra

Operations with 0 and 1:

$X + 0 = X$ (1-5) $X \cdot 1 = X$ (1-5D)

$X + 1 = 1$ (1-6) $X \cdot 0 = 0$ (1-6D)

Idempotent laws:

$X + X = X$ (1-7) $X \cdot X = X$ (1-7D)

Involution law:

$(X')' = X$ (1-8)

Laws of complementarity:

$X + X' = 1$ (1-9) $X \cdot X' = 0$ (1-9D)

Commutative laws:

$X + Y = Y + X$ (1-10) $XY = YX$ (1-10D)

Associative laws:

$(X + Y) + Z = X + (Y + Z) = X + Y + Z$ (1-11) $(XY)Z = X(YZ) = XYZ$ (1-11D)

Distributive laws:

$X(Y + Z) = XY + XZ$ (1-12) $X + YZ = (X + Y)(X + Z)$ (1-12D)

Laws and Theorems of Boolean Algebra

Simplification theorems:

$XY + XY' = X$ (1-13) $(X + Y)(X + Y') = X$ (1-13D)

$X + XY = X$ (1-14) $X(X + Y) = X$ (1-14D)

$(X + Y')Y = XY$ (1-15) $XY' + Y = X + Y$ (1-15D)

DeMorgan's laws:

$(X + Y + Z + \cdots)' = X'Y'Z'\cdots$ (1-16) $(XYZ\cdots)' = X' + Y' + Z' + \cdots$ (1-16D)

$[f(X_1, X_2, \ldots, X_n, 0, 1, +, \cdot)]' = f(X_1', X_2', \ldots, X_n', 1, 0, \cdot, +)$ (1-17)

Duality:

$(X + Y + Z + \cdots)^D = XYZ\cdots$ (1-18) $(XYZ\cdots)^D = X + Y + Z + \cdots$ (1-18D)

$[f(X_1, X_2, \ldots, X_n, 0, 1, +, \cdot)]^D = f(X_1, X_2, \ldots, X_n, 1, 0, \cdot, +)$ (1-19)

Theorem for multiplying out and factoring:

$(X + Y)(X' + Z) = XZ + X'Y$ (1-20) $XY + X'Z = (X + Z)(X' + Y)$ (1-20D)

Consensus theorem:

$XY + YZ + X'Z = XY + X'Z$ (1-21) $(X + Y)(Y + Z)(X' + Z) = (X + Y)(X' + Z)$ (1-21D)

Simplifying Logic Expressions

- Combining terms
  - Use XY+XY'=X, X+X=X

$$Cout = X'YCin + XY'Cin + XYCin' + XYCin$$
$$= (X'YCin + XYCin) + (XY'Cin + XYCin) + (XYCin' + XYCin)$$
$$= YCin + XCin + XY$$

- Eliminating terms
  - Use X+XY=X
- Eliminating literals
  - Use X+X'Y=X+Y
- Adding redundant terms
  - Add 0: XX'
  - Multiply with 1: (X+X')

Theorems to Apply to Exclusive-OR

$X \oplus 0 = X$

$X \oplus 1 = X'$

$X \oplus X = 0$

$X \oplus X' = 1$

$X \oplus Y = Y \oplus X$ (Commutative law)

$(X \oplus Y) \oplus Z = X \oplus (Y \oplus Z)$ (Associative law)

$X(Y \oplus Z) = XY \oplus XZ$ (Distributive law)

$(X \oplus Y)' = X \oplus Y' = X' \oplus Y = XY + X'Y'$

Karnaugh Maps

- Convenient way to simplify logic functions of 3, 4, 5, (6) variables
- Four-variable K-map
  - each square corresponds to one of the 16 possible minterms
  - 1 - minterm is present; 0 (or blank) – minterm is absent;
  - X – don't care
    - the input can never occur, or
    - the input occurs but the output is not specified
  - adjacent cells differ in only one value => can be combined

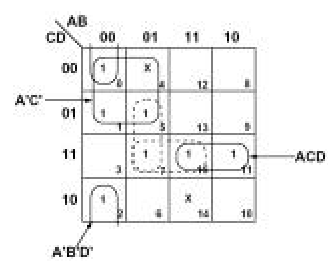
Location of minterms

CD \ AB	00	01	11	10
00	0	4	12	8
01	1	5	13	9
11	3	7	15	11
10	2	6	14	10

Sum-of-products Representation

- Function consists of a sum of prime implicants
- Prime implicant
  - a group of one, two, four, eight 1s on a map represents a prime implicant if it cannot be combined with another group of 1s to eliminate a variable
- Prime implicant is essential if it contains a 1 that is not contained in any other prime implicant

Selection of Prime Implicants



Two minimum forms

F = A'C' + A'B'D' + ACD + A'BD

or

F = A'C' + A'B'D' + ACD + BCD

Procedure for min Sum of products

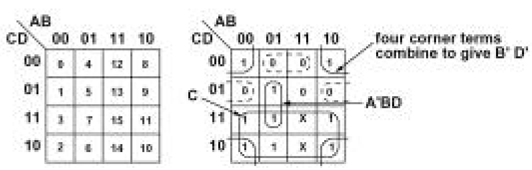
- 1. Choose a minterm (a 1) that has not been covered yet
- 2. Find all 1s and Xs adjacent to that minterm
- 3. If a single term covers the minterm and all adjacent 1s and Xs, then that term is an essential prime implicant, so select that term
- 4. Repeat steps 1, 2, 3 until all essential prime implicants have been chosen
- 5. Find a minimum set of prime implicants that cover the remaining 1s on the map. If there is more than one such set, choose a set with a minimum number of literals

Products of Sums

- F(1) = {0, 2, 3, 5, 6, 7, 8, 10, 11}
  F(X) = {14, 15}

Karnaugh Maps

- Example

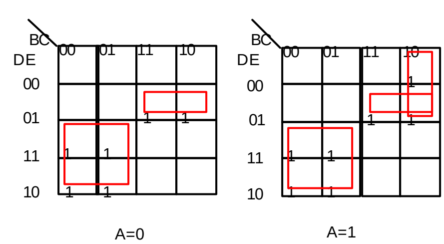


Sum of products F = C + B'D'+A'BD

Product of sums F = (A'+B')(A'+C + D')(B'+C + D)

Five variable Karnaugh Map

- f(1) = {2,3,6,7,9,13,18,19,22,23,24,25,29}



Six Variable Karnaugh Map

AB=00

EF \ CD	00	01	11	10
00	1			1
01		1	1	
11		1	1	
10	1			1

AB=01

EF \ CD	00	01	11	10
00	1			1
01		1		
11		1		
10	1			1

AB=10

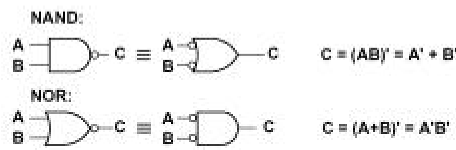
EF \ CD	00	01	11	10
00	1			1
01		1		
11		1		
10	1			1

AB=11

EF \ CD	00	01	11	10
00	1			1
01			1	
11			1	
10	1			1

Designing with NAND and NOR Gates (1)

- Implementation of NAND and NOR gates is easier than that of AND and OR gates (e.g., CMOS)

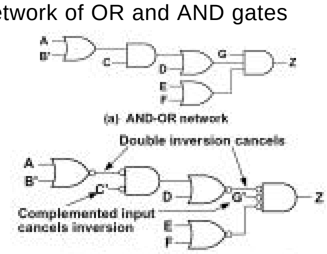


Designing with NAND and NOR Gates (2)

- Any logic function can be realized using only NAND or NOR gates => NAND/NOR is complete
  - NAND function is complete – can be used to generate any logical function;
  - 1: a I (a | a) = a | a' = 1
  - 0: {a I (a | a)} | {a I (a | a)} = 1 | 1 = 0
  - a': a | a = a'
  - ab: (a | b) | (a | b) = (a | b)' = ab
  - a+b: (a | a) | (b | b) = a' | b' = a + b

Conversion to NOR Gates

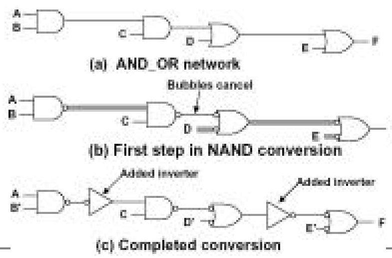
- Start with POS (Product Of Sums)
  - circle 0s in K-maps
- Find network of OR and AND gates



(b) Equivalent NOR-gate network

Conversion to NAND Gates

- Start with SOP (Sum of Products)
  - circle 1s in K-maps
- Find network of OR and AND gates



Tristate Logic and Busses

- Four kinds of tristate buffers
  - B is a control input used to enable and disable the output

B	A	C
0	0	Hi-Z
0	1	Hi-Z
1	0	0
1	1	1

(a)

B	A	C
0	0	Hi-Z
0	1	Hi-Z
1	0	1
1	1	0

(b)

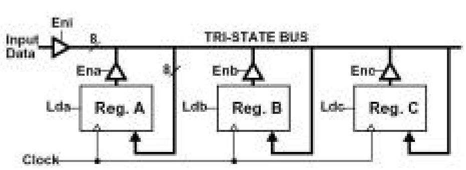
B	A	C
0	0	0
0	1	1
1	0	Hi-Z
1	1	Hi-Z

(c)

B	A	C
0	0	1
0	1	0
1	0	Hi-Z
1	1	Hi-Z

(d)

Data Transfer Using Tristate Bus

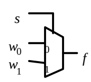


Combinational-Circuit Building Blocks

- Multiplexers
- Decoders
- Encoders
- Code Converters
- *Comparators*
- *Adders/Subtractors*
- *Multipliers*
- *Shifters*

Multiplexers: 2-to-1 Multiplexer

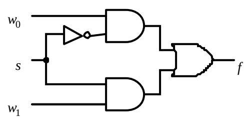
- Have number of data inputs, one or more select inputs, and one output
  - It passes the signal value on one of data inputs to the output



(a) Graphical symbol

s	f
0	w_0
1	w_1

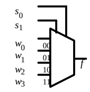
(b) Truth table



(c) Sum-of-products circuit

$$f = s'w_0 + sw_1$$

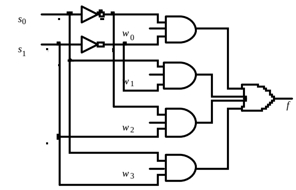
Multiplexers: 4-to-1 Multiplexer



(a) Graphic symbol

s_1	s_0	f
0	0	w_0
0	1	w_1
1	0	w_2
1	1	w_3

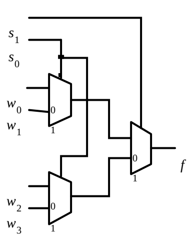
(b) Truth table



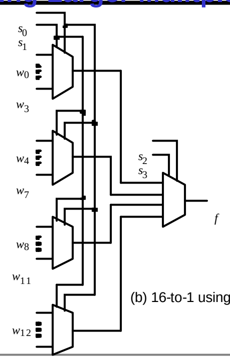
(c) Circuit

$$f = s_1's_0'w_0 + s_1's_0w_1 + s_1s_0'w_2 + s_1s_0w_3$$

Multiplexers: Building Larger Mulitplexers



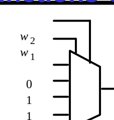
(a) 4-to-1 using 2-to-1



(b) 16-to-1 using 4-to-1

Synthesis of Logic Functions Using Muxes

w_1	w_2	f
0	0	0
0	1	1
1	0	1
1	1	0

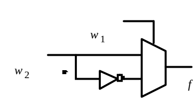


(a) Implementation using a 4-to-1 multiplexer

w_1	w_2	f
0	0	0
0	1	1
1	0	1
1	1	0

w_1	f
0	w_2
1	$\overline{w}_2$

(b) Modified truth table



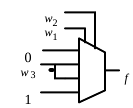
(c) Circuit

Synthesis of Logic Functions Using Muxes

w_1	w_2	w_3	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

w_1	w_2	f
0	0	0
0	1	w_3
1	0	w_3
1	1	1

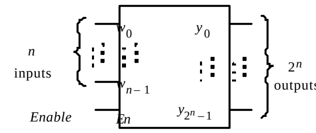
(a) Modified truth table



(b) Circuit

Decoders: n-to-2^n Decoder

- Decode encoded information: n inputs, 2^n outputs
- If En = 1, only one output is asserted at a time
- One-hot encoded output
  - m-bit binary code where exactly one bit is set to 1



$y_0 = w_{n-1}'..w_1' w_0' En$

$y_1 = w_{n-1}'..w_1' w_0 En$

$y_2 = w_{n-1}'..w_1 w_0' En$

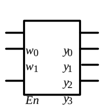
...

$y_{2^n-1} = w_{n-1}..w_1 w_0 En$

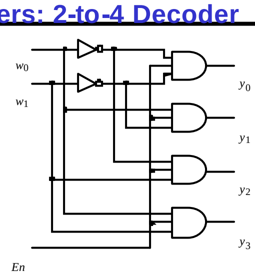
Decoders: 2-to-4 Decoder

En	w_1	w_0	y_0	y_1	y_2	y_3
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1
0	x	x	0	0	0	0

(a) Truth table

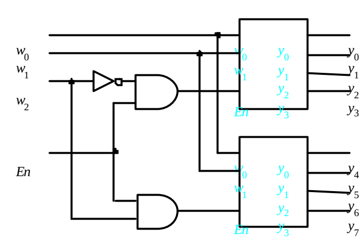


(b) Graphic symbol

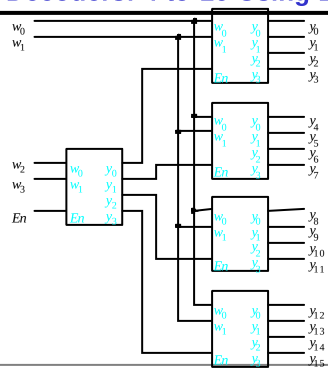


(c) Logic circuit

Decoders: 3-to-8 Using 2-to-4

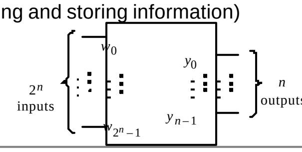


Decoders: 4-to-16 Using 2-to-4



Encoders

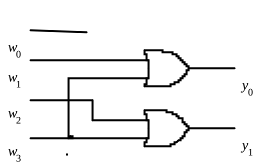
- Opposite of decoders
  - Encode given information into a more compact form
- Binary encoders
  - 2^n inputs into n-bit code
  - Exactly one of the input signals should have a value of 1, and outputs present the binary number that identifies which input is equal to 1
- Use: reduce the number of bits (transmitting and storing information)



Encoders: 4-to-2 Encoder

w_3	w_2	w_1	w_0	y_1	y_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

(a) Truth table



(b) Circuit

Encoders: Priority Encoders

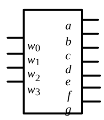
- Each input has a priority level associated with it
- The encoder outputs indicate the active input that has the highest priority

(a) Truth table for a 4-to-2 priority encoder

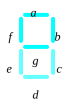
w_3	w_2	w_1	w_0	y_1	y_0	z
0	0	0	0	d	d	0
0	0	0	1	0	0	1
0	0	1	x	0	1	1
0	1	x	x	1	0	1
1	x	x	x	1	1	1

Code Converters

- Convert from one type of input encoding to a different output encoding
  - E. g., BCD-to-7-segment decoder



(a) Code converter



(b) 7-segment display

w_3	w_2	w_1	w_0	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1

(c) Truth table

To Do

- Textbook
  - Chapter 1.3, 1.4, 1.13
- Read
  - Altera's MAX+plus II and the UP1 Educational board: A User's Guide, B. E. Wells, S. M. Loo
  - Altera University Program Design Laboratory Package