ECE 545—Digital System Design with VHDL
Lecture 1A

Digital Logic Refresher

Part A – Combinational Logic Building Blocks

Lecture Roadmap – Combinational Logic

- Basic Logic Review
  - Basic Gates
  - De Morgan's Law
- Combinational Logic Building Blocks
  - Multiplexers
  - Decoders, Demultiplexers
  - Encoders, Priority Encoders
  - Arithmetic circuits
  - ROM. Implementing combinational logic using ROM.
  - Tri-state buffers.

Textbook References

- Combinational Logic Review
  - Stephen Brown and Zvonko Vranesic,

    Fundamentals of Digital Logic with VHDL Design, 2nd or 3rd Edition
    - Chapter 2 Introduction to Logic Circuits (2.1-2.8 only)
    - Chapter 6 Combinational-Circuit Building Blocks (6.1-6.5 only)
  - **OR** your undergraduate digital logic textbook (chapters on combinational logic)

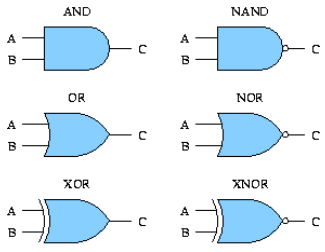
Basic Logic Review

some slides modified from:
S. Dandamudi, "Fundamentals of Computer Organization and Design"

Rules

- If you believe that you know a correct answer, please raise your hand
- I will select *one or more* students (independently whether an answer given by the first student is correct or incorrect)
- Please, identify yourself by first name and give an answer
- **Correct answer = 1 bonus point**

Basic Logic Gates (2-input versions)



A	B	AND	OR	XOR	NAND	NOR	XNOR
0	0	0	0	0	1	1	1
0	1	0	1	1	1	0	0
1	0	0	1	1	1	0	0
1	1	1	1	0	0	0	1

Basic Logic Gates Generalized

- Simple logic gates
  - AND → 0 if one or more inputs is 0
  - OR → 1 if one or more inputs is 1
  - NAND = AND + NOT
    - 1 if one or more inputs is 0
  - NOR = OR + NOT
    - 0 if one or more input is 1
  - XOR → 1 if an odd number of inputs is 1
  - XNOR → 1 if an even number of inputs is 1
  - NAND and NOR gates require fewer transistors than AND and OR in standard CMOS
- Functionality can be expressed by a truth table
  - A truth table lists output for each possible input combination

7

Number of Functions

- Number of functions
  - With *N* logical variables, we can define
    2^{2^N} functions
  - Some of them are useful
    - AND, NAND, NOR, XOR, …
  - Some are not useful:
    - Output is always 1
    - Output is always 0
  - "Number of functions" definition is useful in proving completeness property

8

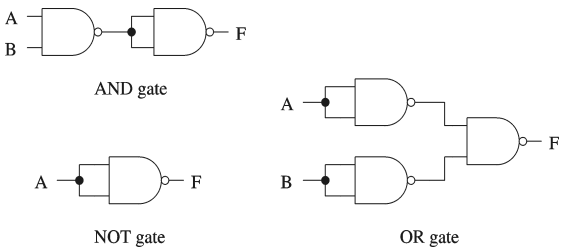
Complete Set of Gates

- Complete sets
  - A set of gates is complete
    - if we can implement any logic function using only the type of gates in the set
  - Some example complete sets
    - {AND, OR, NOT} ← Not a minimal complete set
    - {AND, NOT}
    - {OR, NOT}
    - {NAND}
    - {NOR}
  - Minimal complete set
    - A complete set with no redundant elements.

9

NAND as a Complete Set

- Proving NAND gate is universal



10

Logic Functions

- Logic functions can be expressed in several ways:
  - Truth table
  - Logical expressions
  - Graphical schematic form
  - HDL code
- Example:
  - Majority function
    - Output is one whenever majority of inputs is 1
    - We use 3-input majority function

11

Alternative Representations of Logic Function

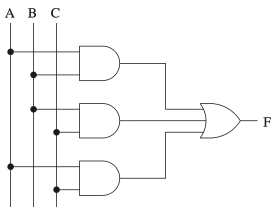
Truth table

A	B	C	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

Logical expression form

F = A B + B C + A C

Graphical schematic form



HDL code: F <= (A AND B) OR (B AND C) OR (A AND C) ;

12

Boolean Algebra

Boolean identities

Name	AND version	OR version
Identity	x·1 = x	x + 0 = x
Complement	x· x' = 0	x + x' = 1
Commutative	x·y = y·x	x + y = y + x
Distribution	x· (y+z) = xy+xz	x + (y· z) = (x+y) (x+z)
Idempotent	x·x = x	x + x = x
Null	x·0 = 0	x + 1 = 1

13

Boolean Algebra (cont' d)

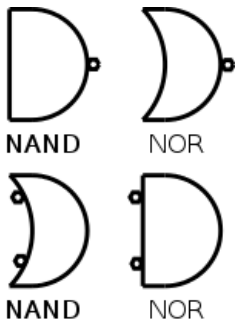
- Boolean identities (cont' d)

Name	AND version	OR version
Involution	x = (x')'	---
Absorption	x· (x+y) = x	x + (x·y) = x
Associative	x·(y· z) = (x· y)·z	x + (y + z) = (x + y) + z
de Morgan	(x· y)' = x' + y'	(x + y)' = x' · y'

(de Morgan's law in particular is very useful)

14

Alternative symbols for NAND and NOR



15

Deriving Equivalent Expressions

- Using NAND gates
  - Get an equivalent expression

    A B + C D = (A B + C D)' '
  - Using de Morgan's law

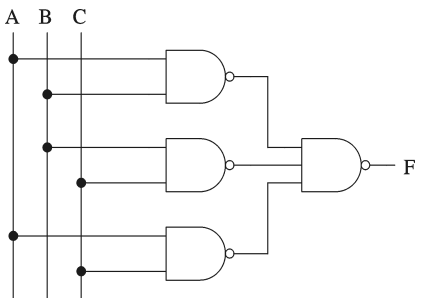
    A B + C D = ((A B)' · (C D)')'
  - Can be generalized
    - Example: Majority function

      A B + B C + AC = ((A B)' · (B C)' · (AC)')'

16

Majority Function Using Other Gates

- Majority function



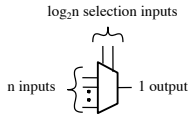
17

Combinational Logic Building Blocks

Some slides modified from:
S. Dandamudi, "Fundamentals of Computer Organization and Design"
S. Brown and Z. Vranesic, "Fundamentals of Digital Logic"

18

Multiplexers



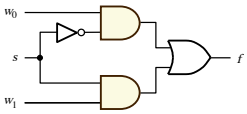
- multiplexer
  - n binary inputs (binary input = 1-bit input)
  - $\log_2 n$ binary selection inputs
  - 1 binary output
  - Function: one of n inputs is placed onto output
  - Called **n-to-1** multiplexer

2-to-1 Multiplexer

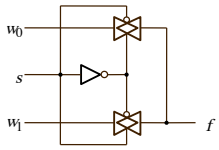
(a) Graphical symbol

s	f
0	w_0
1	w_1

(b) Truth table



(c) Sum-of-products circuit



(d) Circuit with transmission gates

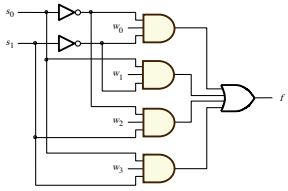
Source: Brown and Vranesic

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

Source: Brown and Vranesic

Multi-bit 4-to-1 Multiplexer

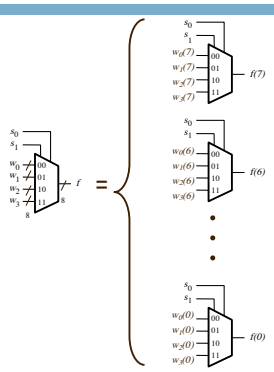
(a) Graphic symbol

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

(b) Truth table

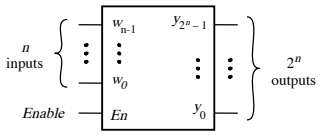
- When drawing schematics, can draw **multi-bit** multiplexers
- Example: 8-bit 4-to-1 multiplexer
  - 4 inputs (each 8 bits)
  - 1 output (8 bits)
  - 2 selection bits
- Can also have multi-bit 2-to-1 muxes, 16-to-1 muxes, etc.

8-bit 4-to-1 Multiplexer



An 8-bit 4-to-1 multiplexer is composed of eight [1-bit] 4-to-1 multiplexers

Decoders

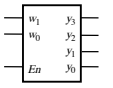


- Decoder
  - n binary inputs
  - 2^n binary outputs
  - Function: decode encoded information
    - If enable=1, one output is asserted high, the other outputs are asserted low
    - If enable=0, all outputs asserted low
  - Often, enable pin is not needed (i.e. the decoder is always enabled)
  - Called **n-to-2^n** decoder
    - Can consider n binary inputs as a single n-bit input
    - Can consider 2^n binary outputs as a single 2^n-bit output
  - Decoders are often used for RAM/ROM addressing

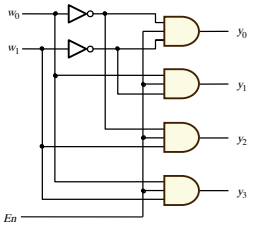
2-to-4 Decoder

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

(a) Truth table



(b) Graphical symbol



(c) Logic circuit

Source: Brown and Vranesic

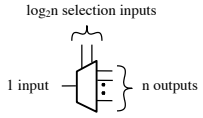
25

Problem 8

Show how to implement a decoder that recognizes the following 4 ranges of a 16-bit address A, and generates the corresponding enable signals `e0,e1,e2,e3`:

For A in:	Assert
`C000-CFFF:`	`e0`
`D000-DFFF:`	`e1`
`E000-EFFF:`	`e2`
`F000-FFFF:`	`e3`

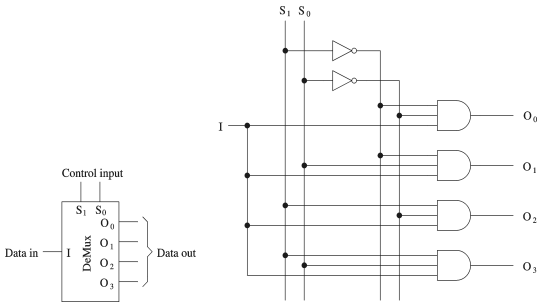
Demultiplexers



- Demultiplexer
  - 1 binary input
  - n binary outputs
  - $\log_2 n$ binary selection inputs
  - Function: places input onto one of n outputs, with the remaining outputs asserted low
  - Called **1-to-n** demultiplexer
- Closely related to decoder
  - Can build 1-to-n demultiplexer from $\log_2 n$-to-n decoder by using the decoder's enable signal as the demultiplexer's input signal, and using decoder's input signals as the demultiplexer's selection input signals.

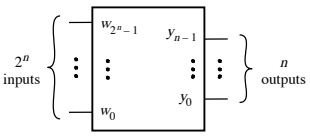
27

1-to-4 Demultiplexer



28

Encoders



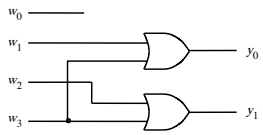
- Encoder
  - 2^n binary inputs
  - n binary outputs
  - Function: encodes information into an n-bit code
  - Called **2^n-to-n** encoder
    - Can consider 2^n binary inputs as a single 2^n-bit input
    - Can consider n binary output as a single n-bit output
- Encoders only work when **exactly one** binary input is equal to 1

29

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

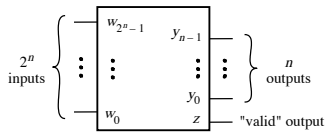
(a) Truth table



(b) Circuit

30

Priority Encoders

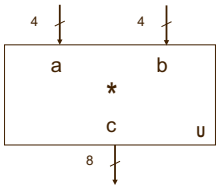


- Priority Encoder
  - 2^n binary inputs
  - n binary outputs
  - 1 binary "valid" output
  - Function: encodes information into an n-bit code based on priority of inputs
  - Called **2^n-to-n** priority encoder
- Priority encoder allows for multiple inputs to have a value of '1', as it encodes the input with the highest priority (MSB = highest priority, LSB = lowest priority)
  - "valid" output indicates when priority encoder output is valid
  - Priority encoder is more common than an encoder

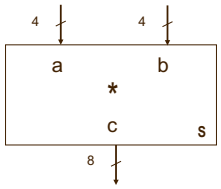
4-to-2 MSB Priority Encoder

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

4x4-bit Unsigned Multiplier



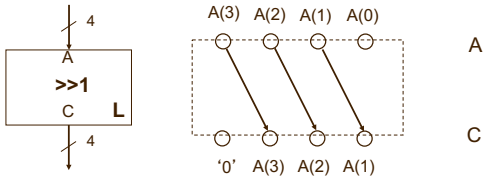
4x4-bit Signed Multiplier



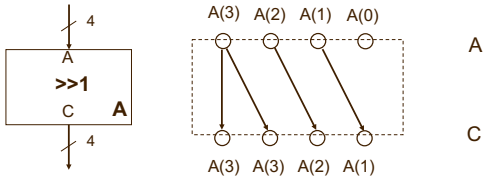
Unsigned vs. Signed Multiplication

Unsigned		Signed	
1111	15	1111	-1
x 1111	x 15	x 1111	x -1
11100001	225	00000001	1

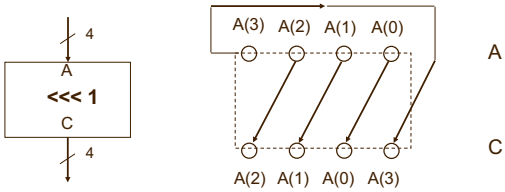
Logical Shift Right



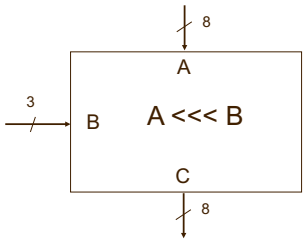
Arithmetic Shift Right



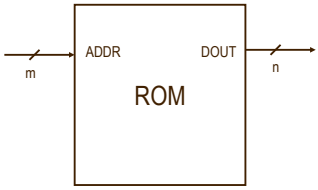
Fixed Rotation



8-bit Variable Rotator Left

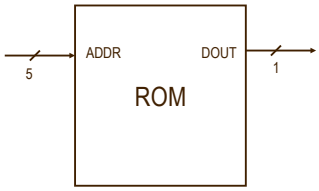


Read Only Memory (ROM)

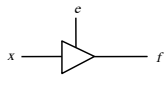


Implementing Arbitrary Combinational Logic Using ROM

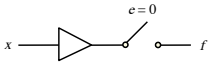
X_5	X_4	X_3	X_2	X_1	Y
0	0	0	0	0	0
0	0	0	0	1	1
0	0	0	1	0	0
0	0	0	1	1	0
0	0	1	0	0	1
0	0	1	0	1	1
0	0	1	1	0	0
0	0	1	1	1	0
0	1	0	0	0	1
0	1	0	0	1	0
0	1	0	1	0	0
0	1	0	1	1	1
0	1	1	0	0	1
0	1	1	0	1	1
0	1	1	1	0	1
0	1	1	1	1	1
1	0	0	0	0	0
1	0	0	0	1	0
1	0	0	1	0	0
1	0	0	1	1	0
1	0	1	0	0	0
1	0	1	0	1	0
1	0	1	1	0	0
1	0	1	1	1	1
1	1	0	0	0	0
1	1	0	0	1	1
1	1	0	1	0	0
1	1	0	1	1	1
1	1	1	0	0	0
1	1	1	0	1	1
1	1	1	1	0	0
1	1	1	1	1	0

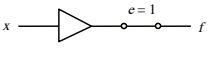


Tri-state Buffer



(a) A tri-state buffer



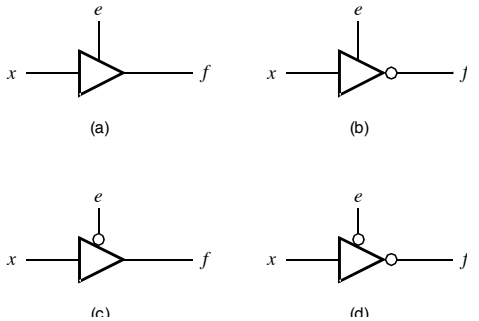


(b) Equivalent circuit

e	x	f
0	0	Z
0	1	Z
1	0	0
1	1	1

(c) Truth table

Four types of Tri-state Buffers



43