
CEG 3320 - Digital System Design

Instructor: Travis Doom, Ph.D.
331 Russ Engineering Center
775-5105
travis.doom@wright.edu
<http://www.wright.edu/~travis.doom>

Lecture slides created by T. Doom for Wright State University's course in Digital System Design. Some slides contain fair use images or material used with permission from textbooks and slides by R. Haggard, F. Vahid, Y. Patt, J. Wakerly, M. Mano, and other sources.

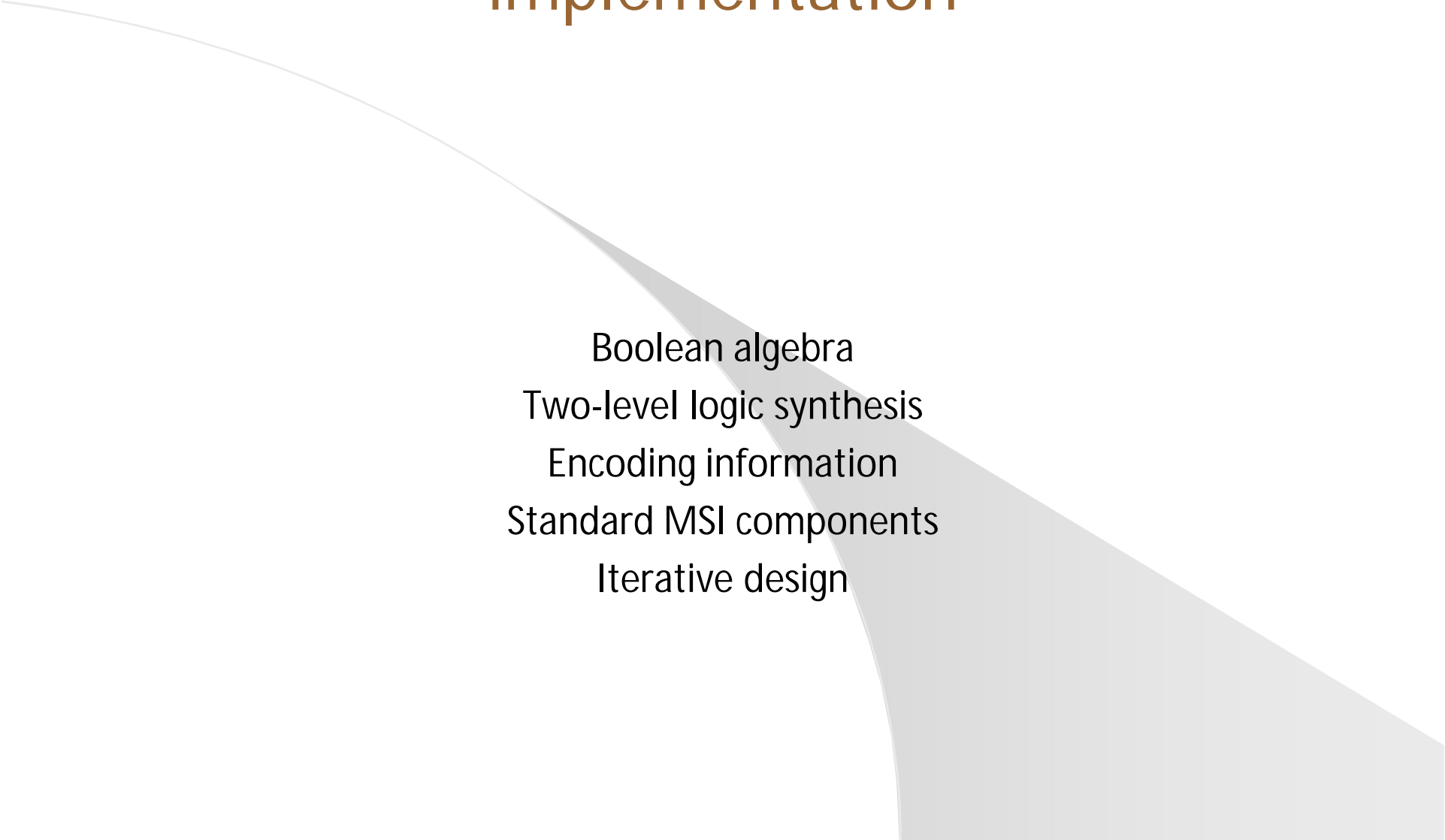


How is the pace/difficulty of the course so far?

- We are moving too slowly; bring it on!
- Things are about right, we could maybe do a bit less group work/problems
- Things are about right, we could maybe do a bit more group work/problems
- We are moving too quickly; please slow down!
- I have seen this material before/elsewhere so my vote doesn't count



Module II: Combinational design and implementation

A large, light gray, curved shape that starts from the left edge of the slide and curves downwards and to the right, ending near the bottom right corner. It has a smooth, organic, wave-like border.

Boolean algebra
Two-level logic synthesis
Encoding information
Standard MSI components
Iterative design

Boolean Algebra

A large, light gray, curved shape that starts from the left edge of the slide and curves downwards and to the right, ending near the bottom right corner. It has a smooth, organic, wave-like border.

Terminology

Properties

Canonical forms

Optimization via K-map

Boolean Algebra: Basic Operators

-
- | Operator | Symbol | Alternate Symbol(s) |
|----------|----------|---------------------|
| – NOT | — | ' |
| – AND | . | ^ |
| – OR | + | v |
| – XOR | \oplus | |
- Order of precedence:
 - (1) Not and Parenthesis
 - (2) AND
 - (3) OR
 - Examples
 - Truth table for: $F = A + BC$
 - Truth table for : $F = A + B'C$
-



2 operand Boolean functions

A	B	F ₀	F ₁	F ₂	F ₃	F ₄	F ₅	F ₆	F ₇	F ₈	F ₉	F _A	F _B	F _C	F _D	F _E	F _F
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
		F A L S E	A N D		A		B	X O R	O R	N O R	X N O R	B'		A'		N A N D	T R U E

- Any Boolean function can be represented using a combination of AND, OR, and NOT



Boolean Algebra: Terminology

- Variables: Represents an input value of 0 or 1.
 - e.g. A, B, Sensor
- Literal: A variable in true or complemented form.
 - e.g. A, A'
- Product term: A product (AND) of literals.
 - e.g. $a'bc$
- Sum of products (SOP): Two or more product terms OR-ed together.
 - e.g. $AB + A'C + BC$
 - Any Boolean expression can be represented as a sum of products
 - SOP is a standard canonical form (analogous to a simplified form in normal algebra)
- Product of Sums (POS): Two or more product terms OR-ed together.
 - e.g. $(A+B)(B+C)$



Boolean algebra: Properties

Property	Rule
Commutative	$A + B = B + A$ $AB = BA$
Associative	$(A + B) + C = A + (B + C)$ $(AB)C = A(BC)$
Identity	$0 + A = A$ $1 \cdot A = A$
Null / Annihilator	$A + 1 = 1$ $A \cdot 0 = 0$
Laws of Tautology (Idempotent)	$A + A = A$ $A \cdot A = A$
Laws of Complementation (Involution)	$A + A' = 1$ $AA' = 0$ $(A')' = A$



Boolean algebra: Advanced Properties

Property	Rule
Laws of Absorption	$A + (AB) = A$ $A(A+B) = A$
Distributive Laws	$A(B + C) = AB + AC$ $A + (BC) = (A+B)(A+C)$
DeMorgan's Theorems	$(A + B)' = A'B'$ $(AB)' = A' + B'$



In-class exercises: Boolean algebra

- True or False?
 - $abc' = c'ba$
 - $abc + abc' = ab$
 - $x + x'z = x + z$
 - $(w'x + yz)' = (w+x')(y'+z')$
 - $wx' + wx'yz = wx' + yz$
 - $w'x + w'xy = w'x + w'xz$
 - Convert to SOP
 - $F = ab(c+d)$
 - $F = wx(x'y + zy' + xy)$
 - $F = x(x' + y(x'+y'))$
 - Implement as two-level logic
-



Minimization goals: Canonical SOP form

- Substance

- Reduce to a standard form (generally SOP) $F = ab(ac+ab'd)(b+dc)$
- Minimize number of terms $F = abac(b+dc) + abab'd(b+dc)$
- Minimize number of literals
- Use only AND, OR, NOT
 - Do NOT use XOR

$$F = abacb + abacdc + abab'db + abab'ddc$$

- Style

- Present literals “in order” in each term $F = abc + abdc + 0 + 0$
- organize terms “in order” by the first literal present in each term, break ties with second literal, and so on. $F = abc + abcd$
 $F = abc$





Karnaugh maps (K-map)

- Boolean n-space

- Office building example
- 4 rooms per floor, 4 floors, 4 buildings

1a	1c
1b	1d

- Hamming distance: the number of positions at which 2 strings differ
 - Implicant: A product term whose minterms all have the value 1.
 - Prime Implicant: An implicant not wholly contained by a larger implicant
 - Essential Implicant: An implicant whose inclusion is necessary to cover the function
 - Map simplification: Cover the function with as few implicants as possible.
 - Two-variable
 - Three-variable
 - Four-variable
 - Don't care conditions
-





In-class exercises: K-maps

- Perform algebraically and with K-map to SOP
 - (1) $F(a,b,c) = ab'c + abc + a'bc + abc'$
 - (2) $F(a,b,c) = a + a'b'c + a'c$
 - (3) $F(a,b,c,d) = a'bc' + abc'd' + abd$
 - (4) $F(a,b,c,d) = ab + a'b'd'$
 - (5) $a'b'c + abc$ (if $a'bc$ and $ab'c$ are don't care)

 - Design as two-level implementation from minimized SOP
-





Encoding information

There are 10 types of people:

- *People that understand this*
- *People that do not*

Data encoding
binary

Hexadecimal

Unsigned magnitude representation for natural numbers

Data representation in binary

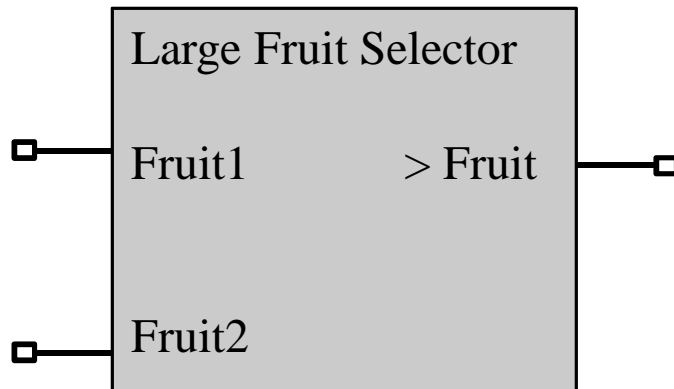
- We need a way to represent information (data) in a form that is mutually comprehensible by human and machine.
 - We have to develop schemes for representing all conceivable types of information – integers, characters, floating point numbers, language, images, actions, etc.
- Binary digITs (Bits): sequences of 0's and 1's that help humans keep track of the current flows stored/processed in the computer
 - Using base-2 provide us with two symbols to work with: we can call them *on* & *off*, or (more usefully) *0* and *1*.
 - We group bits together to allow representation of more complex information
 - For ease of use, Computer scientists usually represent quartets of bits in **Hexadecimal**
 - eg. xA13F vs. 1010000100111111

Hex	Binary
x0	0000
x1	0001
x2	0010
x3	0011
x4	0100
x5	0101
x6	0110
x7	0111
x8	1000
x9	1001
xA	1010
xB	1011
xC	1100
xD	1101
xE	1110
xF	1111



Encoding information digitally

- In order to represent human understandable concepts digitally, an engineer must assign meaning to patterns of binary digits (bits).



Fruit	
Cranberry	
Plum	
Apple	
Watermelon	

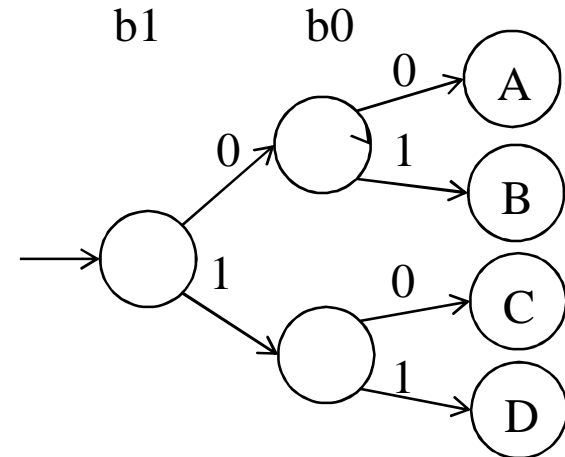
How do you represent Apple?

- Data encodings must be specific and unique
- Order/position in the encoding is relevant
 - Left hand side is the Most Significant bit (MSB)
 - Right-hand side is the Least Significant bit (LSB)



Encoding information

- How many bits must be used in order to represent:
 - 2 difference concepts?
 - 8?
 - 9?
 - In general?
- When can I use more bits?
- When can I use less?
 - Information Theory

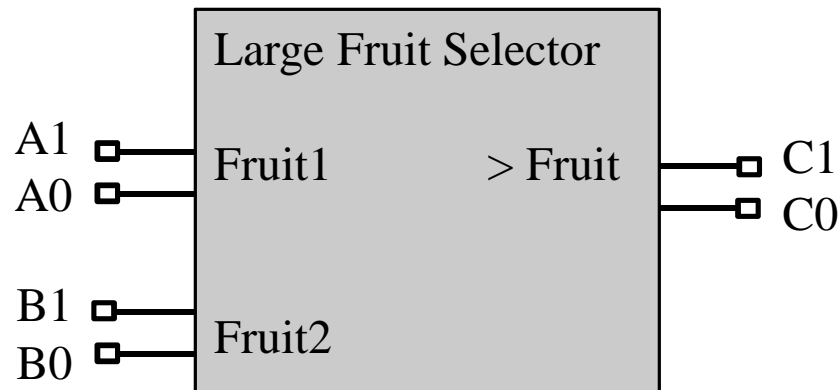


Decision Tree



Encoding information digitally

- In order to represent human understandable concepts digitally, an engineer must assign meaning to patterns of binary digits (bits).



Fruit	Encoding
Cranberry	00
Plum	01
Apple	10
Watermelon	11

- Data encodings must be specific and unique
- Order/position in the encoding is relevant
 - Left hand side is the Most Significant bit (MSB)
 - Right-hand side is the Least Significant bit (LSB)

Exercise:

What is the truth table for this device?



Unsigned Binary Integers

$$Y = \text{"abc"} = a.2^2 + b.2^1 + c.2^0$$

(where the digits a, b, c can each take on the values of 0 or 1 only)

N = number of bits
Range is: $0 \leq i < 2^N - 1$

$$U_{\min} = 0$$
$$U_{\max} = 2^N - 1$$

	3-bits	4-bits	8-bits
0	000	0000	00000000
1	001	0001	00000001
2	010	0010	00000010
3	011	0011	00000011
4	100	0100	00000100



More design examples

- 4-bit prime number detector
- 3-bit fuel gauge, assert empty light if gauge reads 3 or less
 - Design Challenge: Implement using only NAND gates
- Fruit sorter:
 - Inputs
 - Size: small , big [$>10\text{lbs}$]
 - Color: green, yellow, red, orange
 - Outputs
 - Apple, Orange, Bannana, Watermelon
- Turn signal
 - Inputs:
 - Left, Right, and Hazzard (left and right cannot be asserted simultaneously)
 - Output: Blink Left, Blink Right



Design techniques: Standard MSI/LSI devices

Standard MSI combinational components

Hierarchical design techniques

Iterative design techniques

Decoders

Encoders

Multiplexors

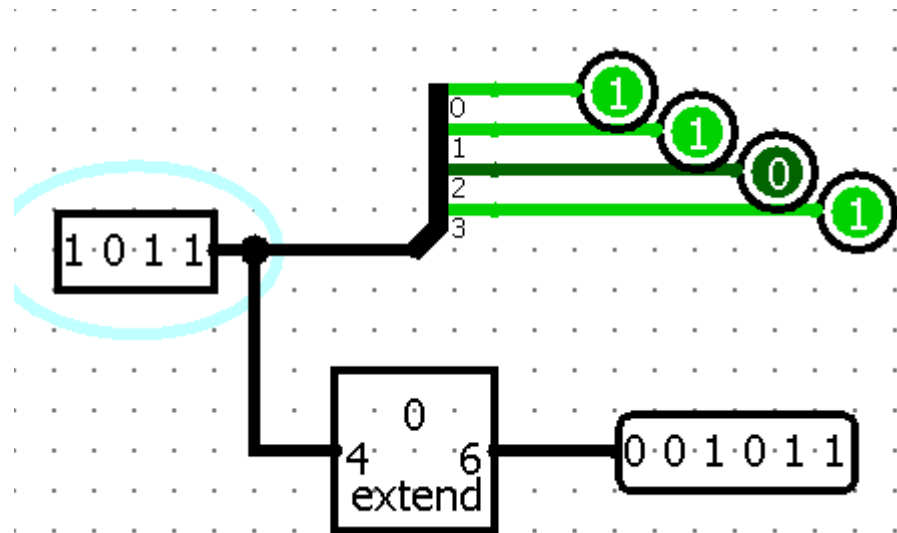
Busses & Splitters

Enables

Three state devices

Increasing complexity: Multibit inputs

- Busses
- Splitters
- Extenders

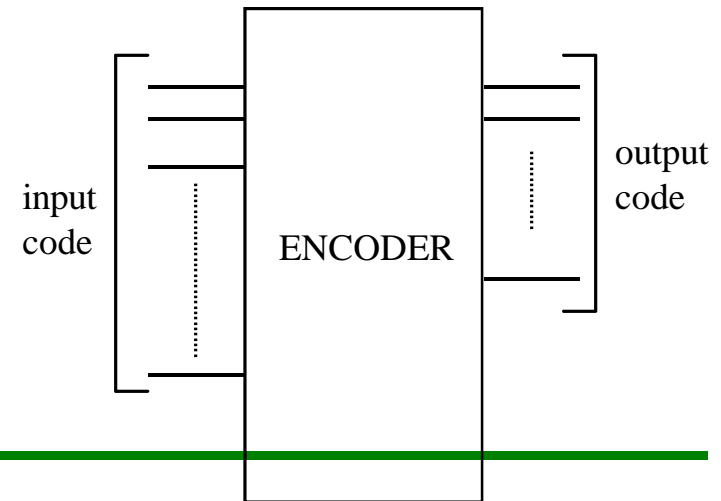
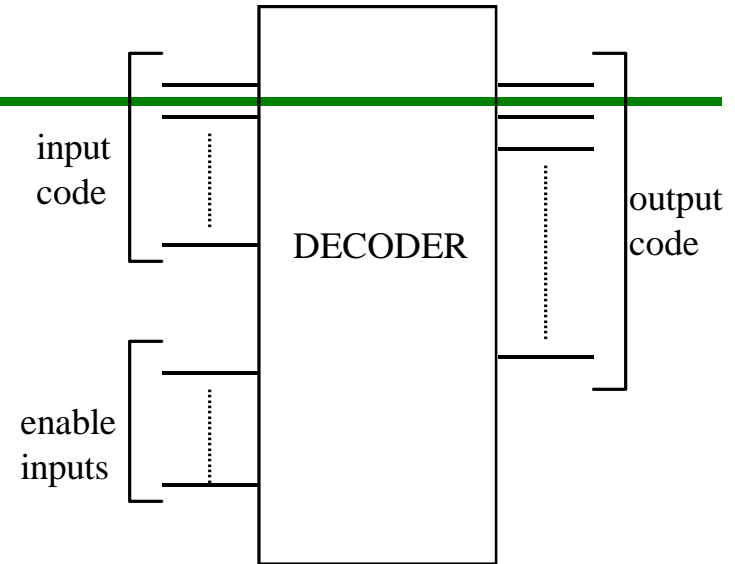


- Notation for multibit inputs
- The individual values in a 4-bit bus input A are commonly denoted as
 - $A_3A_2A_1A_0$
 - $A[3]A[2]A[1]A[0]$
 - $A[3:0]$



Decoder

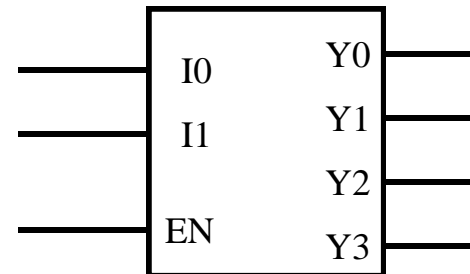
- Multiple-input/multiple-output device.
- Decoder - Inputs (n) < outputs (m).
- Encoder - Inputs (n) > outputs (m)
- Converts input code words into output code words.
- One-to-One mapping :
 - Each input code produces only one output code.
- Input codes :
 - Binary Code
 - Gray Code
 - BCD Code
 - Your encoding!



Binary Decoder

- n -to- 2^n decoder: n inputs and 2^n outputs.
- Input code : Binary Code.
- Output code : 1-out-of- 2^n , One output is asserted for each input code.
- Example : $n=2$, 2-to-4 decoder

Inputs			Outputs			
EN	I1	I0	Y3	Y2	Y1	Y0
0	x	x	0	0	0	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



Hierarchical and Iterative design

- We have an implementation of a 2-bit binary decoder.
- Does that help us implement a 4-bit binary decoder?
- An 8-bit decoder?

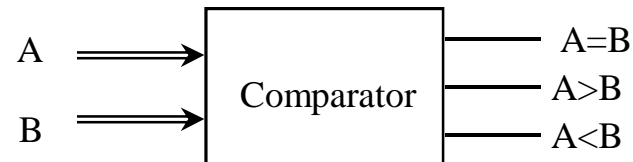


Comparators

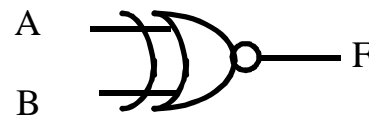
- Compares Two binary words and indicate if they are equal



- Advanced Comparators :



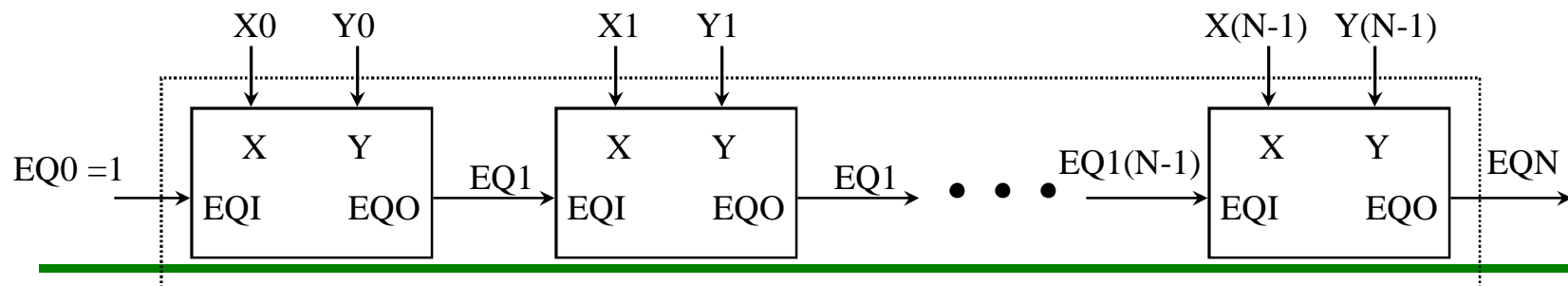
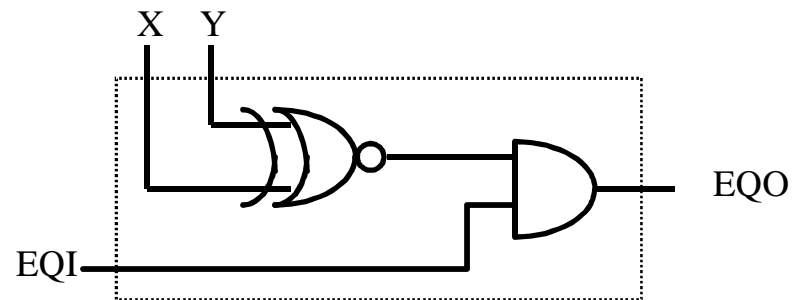
- 1-bit Comparator : XNOR gate , the Output is 1 if $A=B$



Iterative Combinational Logic

- Iterative logic array: A device consisting of identical sub-circuits connected together in a chain to perform a larger overall function
- Iterative Comparator : cascaded 1-bit comparators
- 1-bit comparator :

Function Table			
EQI	X	Y	EQO
0	x	x	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1



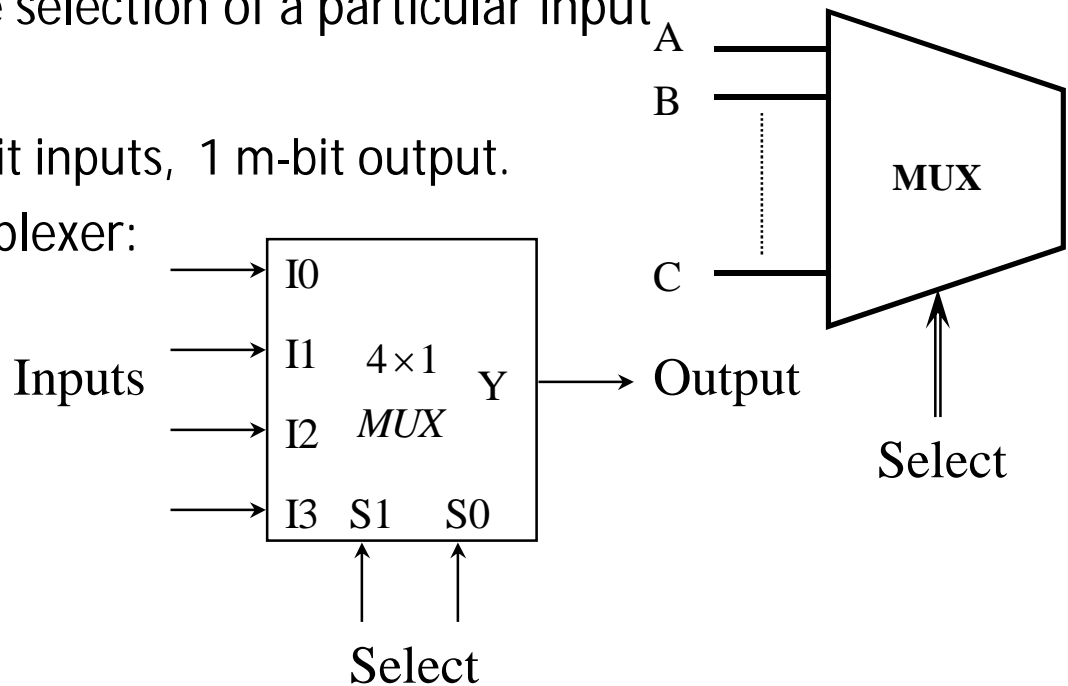
Multiplexers

- Multiplexing: transmitting large number of signals over a small number of channels or lines
- Digital multiplexer (MUX): selects one of many input lines and directs it to a single output (often a bus or “party line”).
- Selection lines controls the selection of a particular input
- m-bit $2^n \times 1$ multiplexer:
 - n selection lines, 2^n m-bit inputs, 1 m-bit output.

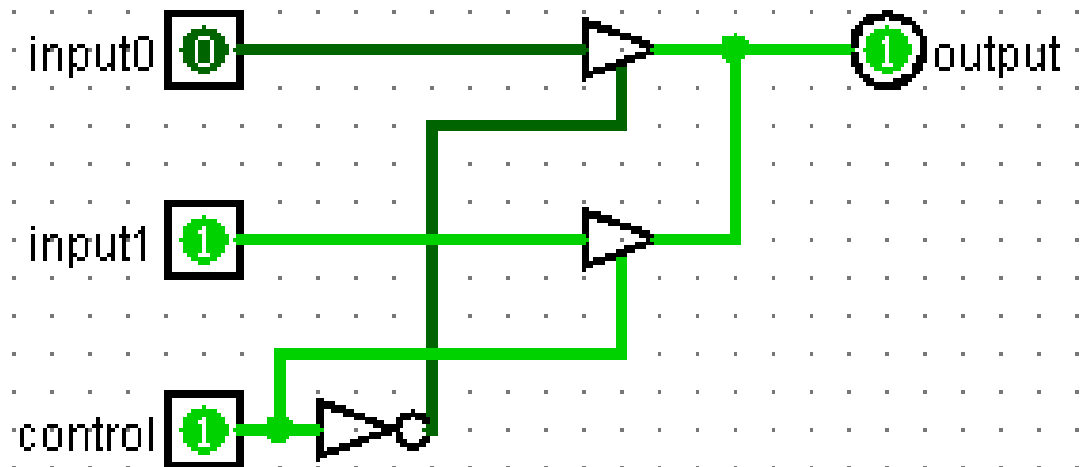
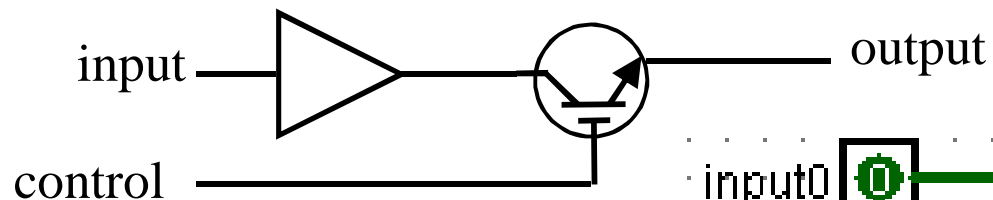
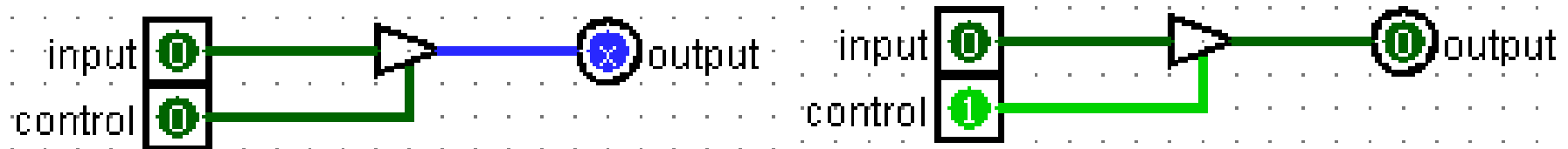
- Example : 4-to-1 line multiplexer:

Function Table :

S1	S0	Y
0	0	I0
0	1	I1
1	0	I2
1	1	I3



Controlled Buffers (the 3-state buffer)



- High Impedance state
- Can often be used instead of muxes



Design optimization and style

Design styles & trade-offs

Two level Vs. Multi level logic

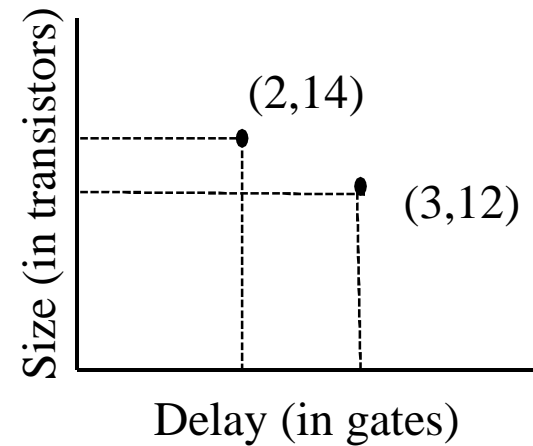
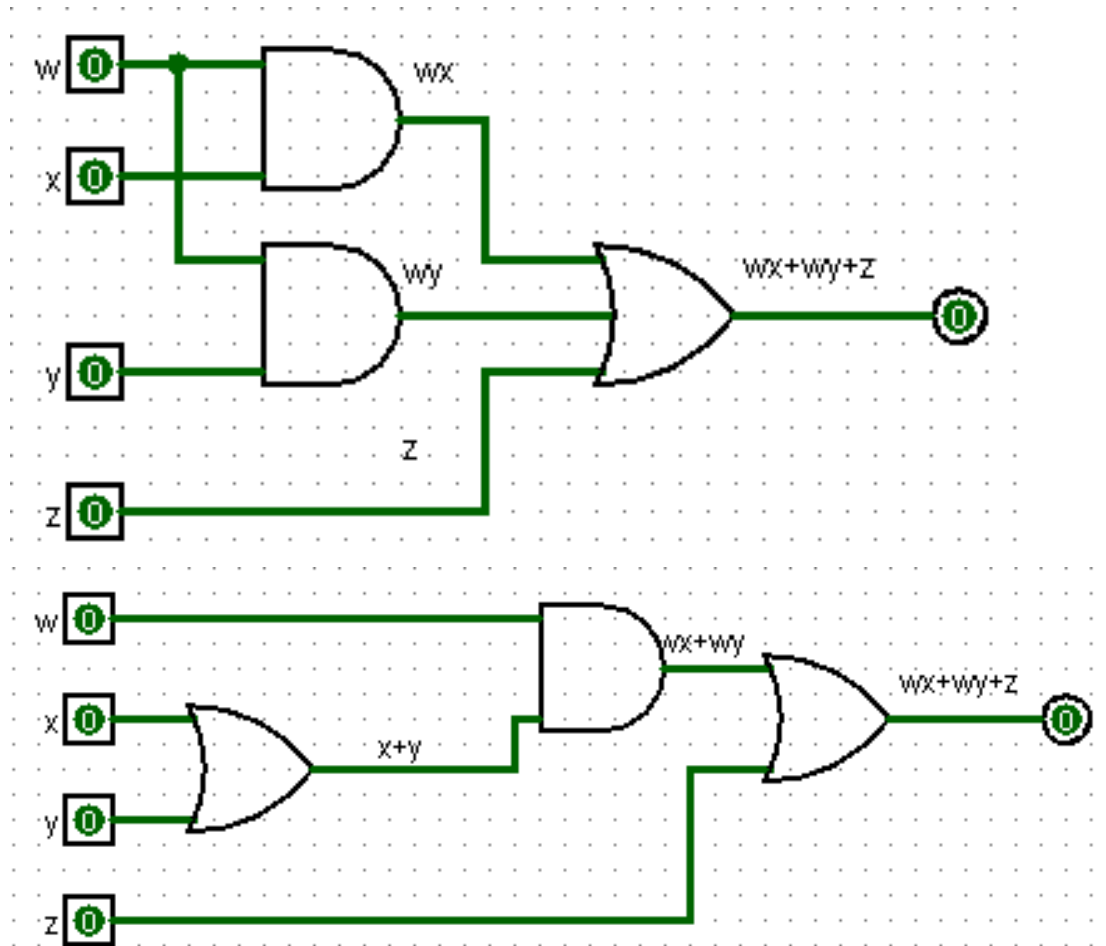
Universal Devices

Designing with NAND gates

Designing with Multiplexors (Shannon's Theorem)

Functional Decomposition

Which implementation is better?



- Which is better?
 - Assume 2 transistors per gate input
- Assume Pareto points



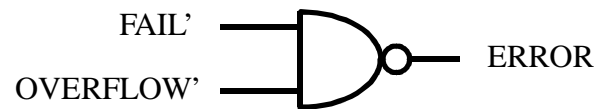
Design

- Design approaches
 - Hierarchical Design
 - Combinational iterative design
- Implementation trade offs
 - Size / Time / Cost
 - Speed
- Technology mapping
 - NAND is a universal logic device
 - NOR is a universal logic device
 - Multiplexors are universal logic devices



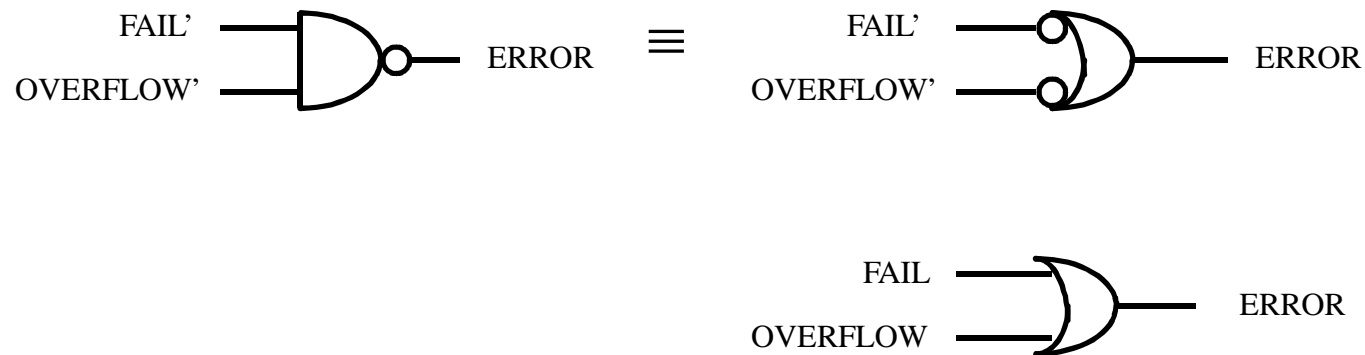
Bubble-to-Bubble Logic Design

- Make the function of the Logic circuit easy to understand!



Bubble-to-Bubble Logic Design

- Make the function of the Logic circuit easy to understand!

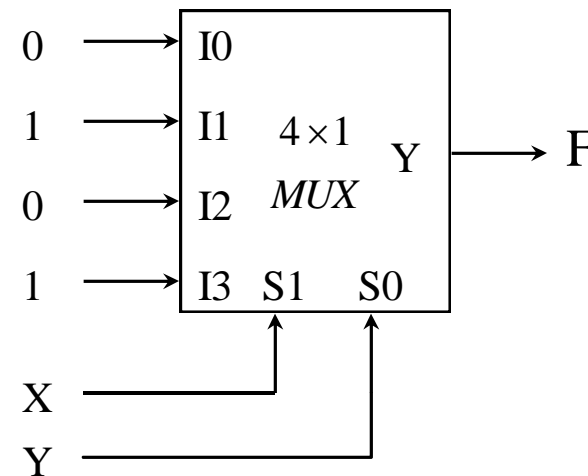


Implementing Logic Functions

- Any n -variable logic function can be implemented using a 2^n -to-1 MUX
 - (In fact, MUXes are a universal device! Any combinational function can be implemented using only MUXes).
- The inputs variables are connected to the select input.
- The function value for each input combination (0 or 1) is connected to the corresponding input of the MUX

- Example:

Row	X	Y	F
0	0	0	0
1	0	1	1
2	1	0	0
3	1	1	1



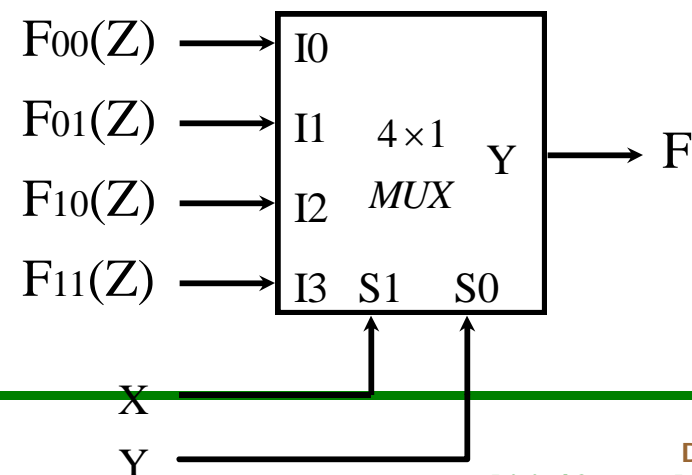


Functional Decomposition

- Effective way for using MUX to implement Logic Functions.
- n-variable truth table can be simplified using any MUX :
 - Use one or more variables as control inputs
 - Decompose the remaining logic function in terms of the remaining inputs
- For 3-variable Logic Function , the decomposed truth table is :

Row	X	Y	Z	F
0,1	0	0	x	$F_{00}(Z)$
2,3	0	1	x	$F_{01}(Z)$
4,5	1	0	x	$F_{10}(Z)$
6,7	1	1	x	$F_{11}(Z)$

Values of $F_{xx}(Z) = 0$ or 1 or Z or Z'



In class examples

- Design with NANDS/NORs
 - Implement NOR with NAND
 - Draw a logic diagram for the function $F(A,B,C,D,E) = AB(C+D'+E')$ using only 2-input NAND and NOR gates. Use good documentation practices.
 - Implement a 2-to-1 1-line MUX with only NAND/NOR
- Design with MUXes
 - Implement an inverter
 - Implement NAND
 - Implement $F(A,B,C,D,E) = AB(C+D'+E')$ using a 4-to-1 1-line MUX



