CSC 411

Computer Organization (Fall 2024)
Lecture 18: Boolean algebra, decoders, multiplexers

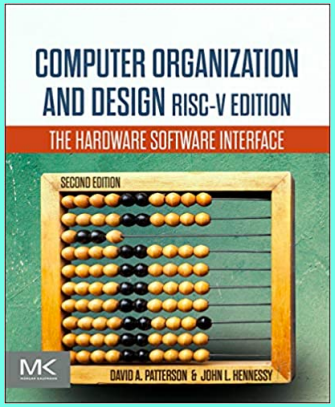
Prof. Marco Alvarez, University of Rhode Island

Disclaimer

Some figures and slides are adapted from:

Computer Organization and Design (Patterson and Hennessy)

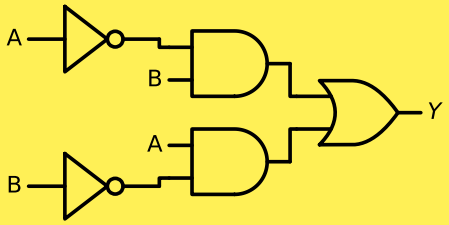
The Hardware/Software Interface



Practice

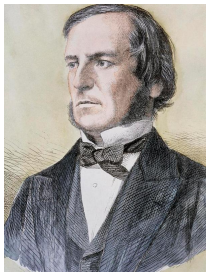
- Which gate is this?
  - not, or, and, xnor, xor, nand, nor

A	B	Y
0	0	
0	1	
1	0	
1	1	



Boolean algebra

- Operates on the binary system
  - all variables on a function have the values of 0 or 1
  - there are three basic operators: **NOT, AND, OR** which can be combined to form more complex Boolean functions
- Defines axioms, laws, and principles
  - digital circuits (logic functions) can be expressed with Boolean algebra equations/expressions
  - Boolean expressions can be **manipulated or simplified** using the Boolean algebra theory
- Foundation for the design and implementation of complex digital circuits



George Boole (1815–1864) was an English mathematician, philosopher, and logician. He is best known as the author of The Laws of Thought (1854) which contains Boolean algebra, laying the foundations for the Information Age.

https://en.wikipedia.org/wiki/George_Boole

The following slides are just a brief introduction to boolean algebra, not covering all topics in the field, there are many more complex concepts and applications that are beyond the scope of this brief overview.

Axioms and laws

AND is typically represented using the multiplication symbol ($\cdot$), however, the dot is often omitted.For example, the expression $A \cdot B$ can be simply written as AB.

OR is typically represented using the addition symbol ($+$).

NOT typically represented using an overline or the symbols ($\neg$) or($'$). For example, the negation of A can be written as $\overline{A}$ or $\neg A$ or A'.

Identity:	$A + 0 = A$
	$A \cdot 1 = A$
Annulment:	$A + 1 = 1$
	$A \cdot 0 = 0$
Inverse:	$A + \overline{A} = 1$
	$A \cdot \overline{A} = 0$
Commutative:	$A + B = B + A$
	$A \cdot B = B \cdot A$
Associative:	$A + (B + C) = (A + B) + C$
	$A \cdot (B \cdot C) = (A \cdot B) \cdot C$

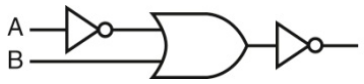
Axioms and laws

Idempotent:	$A + A = A$
	$A \cdot A = A$
Double negation:	$\overline{(\overline{A})} = A$
Absorption:	$A \cdot (A + B) = A$
	$A + A \cdot B = A$
Distributive:	$A \cdot (B + C) = (A \cdot B) + (A \cdot C)$
	$A + (B \cdot C) = (A + B) \cdot (A + C)$
DeMorgan's:	$\overline{A \cdot B} = \overline{A} + \overline{B}$
	$\overline{A + B} = \overline{A} \cdot \overline{B}$

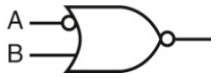
Practice

- Draw the logic gate implementation of $\overline{\overline{A} + B}$

- using explicit invert gates



- using bubbled inputs and outputs

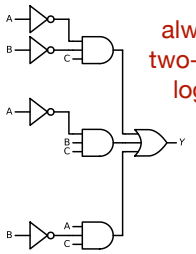


Sum of products

- Starting from a **truth table** …
  - any Boolean function can be represented as the sum (OR) of one or more product (AND) terms
  - a.k.a. DNF (disjunctive normal form)

$$Y = \overline{A}\overline{B}C + \overline{A}BC + A\overline{B}C$$

A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0



always two-level logic

Simplifying sums of products

$$Y = \overline{A}BC + A\overline{B}\overline{C} + A\overline{B}C + AB\overline{C} + ABC \quad \text{canonical form}$$

use distributive rule

$$= \overline{A}BC + A\overline{B}(\overline{C} + C) + AB(\overline{C} + C)$$

use inverse rule

$$= \overline{A}BC + A\overline{B} + AB$$

use distributive rule

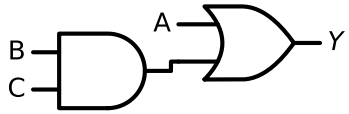
$$= \overline{A}BC + A(\overline{B} + B)$$

use complementary rule

$$= \overline{A}BC + A$$

use distributive rule

$$= BC + A \quad \text{minimal form}$$



A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Boolean algebra also defines **product of sums**, i.e., expressing a function as the product of sums of literals. Both sums of products and products of sums are canonical forms used to simplify and represent Boolean expressions.

Practice

- Simplify this function and draw the final logic implementation

$$Y = \overline{A}\overline{B}\overline{C} + A\overline{B}\overline{C} + A\overline{B}C$$

Boolean algebra and logic design

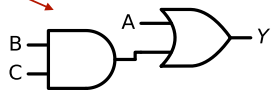
A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Define a function (inputs/outputs) using a truth table

Express and simplify the function using Boolean algebra

$$Y = \overline{A}BC + A\overline{B}\overline{C} + A\overline{B}C + AB\overline{C} + ABC$$
$$= \overline{A}BC + A\overline{B}(\overline{C} + C) + AB(\overline{C} + C)$$
$$= \overline{A}BC + A\overline{B} + AB$$
$$= \overline{A}BC + A(\overline{B} + B)$$
$$= \overline{A}BC + A$$
$$= BC + A$$

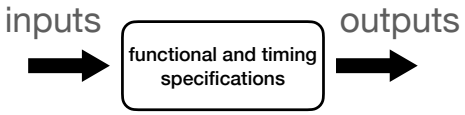
Implement it using digital circuits



Logic circuits

Logic circuits

Fundamental **building blocks** in digital systems



- **Inputs and outputs**
  - input signals represent the information or data that the circuit needs to process
  - output signals represent the result of that processing
- **Functional specification**
  - defines the mapping between the input signals and the output signals, typically represented using Boolean expressions
  - different Boolean expressions for the same function can lead to different circuit designs, each with its own trade-offs in terms of hardware area, cost, latency, power consumption, and other performance metrics
- **Timing specification**
  - defines the temporal behavior of the circuit, including the latency between the changes in input signals and the corresponding changes in output signals
  - crucial for ensuring that the circuit operates correctly and meets the requirements of the larger system it is a part of

Logic circuits

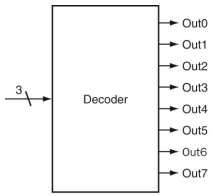
The relationship between the functional specification (Boolean expressions) and the resulting circuit design is particularly important. By applying Boolean algebra techniques, such as simplification and optimization, designers can arrive at more efficient circuit implementations that meet the desired performance goals.

- Two main types
  - **Combinational logic**
    - memoryless, meaning their output is solely a function of the current input signals, without any dependence on previous inputs or internal state
    - examples include adders, multiplexers, decoders
  - **State (sequential) logic**
    - have memory, meaning their outputs depend not only on the current input signals but also on the previous states or history of the circuit
    - examples include flip-flops, registers, counters, and finite state machines

Decoders

Decoder

- Logic circuit that converts coded inputs into coded outputs
  - have n input lines and a maximum of 2^n unique output lines
  - each output line corresponds to a **unique combination** of the input lines
  - only one of the output lines is active (equal to 1) at any given time, while all the others are 0
- Decoders are used for tasks like memory addressing, peripheral device selection, and data routing



a. A 3-bit decoder

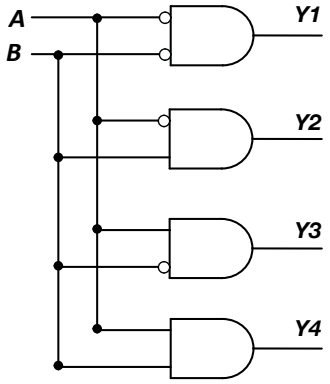
Inputs			Outputs							
I2	I1	I0	Out7	Out6	Out5	Out4	Out3	Out2	Out1	Out0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

b. The truth table for a 3-bit decoder

Practice

- Complete the following truth table

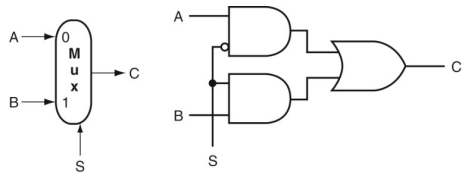
A	B	Y1	Y2	Y3	Y4
0	0				
0	1				
1	0				
1	1				



Multiplexers

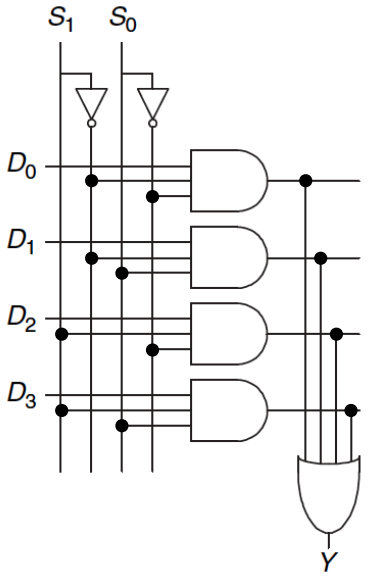
Multiplexer (MUX)

- Logic circuit that selects one of several input signals and forwards it to a single output line
  - has the following key components:
    - data input lines, selection input lines, a single output line
  - selection input lines determine which of the data input lines gets transmitted to the output
- Commonly used in data selection, address decoding, etc.



2-to-1 mux

4-to-1 Mux

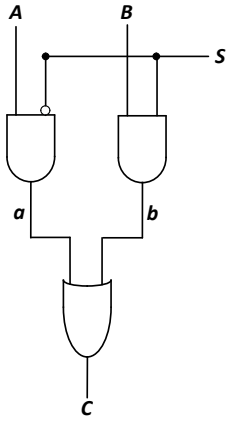


Credit: Digital Design & Computer Architecture, Our Mutlu, ETH, 2023

Practice

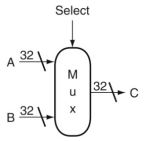
- Complete the following truth table

S	A	B	C
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

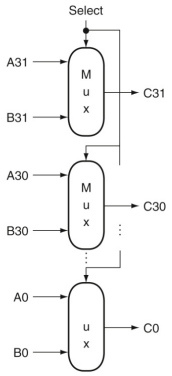


32-bit wide 2-to-1 Multiplexer

- Uses an array of 1-bit multiplexers
- Operates on an entire collection of inputs
  - 32 inputs in this case, often referred to as a **bus**
  - a **bus** consists of multiple parallel wires or signal lines, these individual lines are grouped together and treated as a single logical unit or signal



a. A 32-bit wide 2-to-1 multiplexor



b. The 32-bit wide multiplexor is actually an array of 32 1-bit multiplexors