CS201 W'00 Lecture Notes
Lecture 5

Gates

Gates are building blocks of circuits

Each basic gate is made from two or three transistors

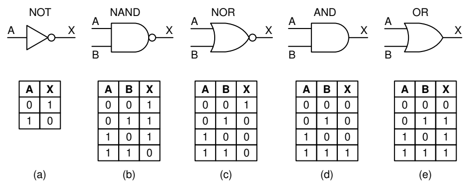


Figure 3-2. The symbols and functional behavior for the five basic gates.

Each gate has small but non-zero **delay** (usually a few nanoseconds) between time when inputs stabalize and time when outputs stabalize.

Wires connecting gates also have non-zero delay.

Combinational Logic

A **combinational circuit** is one whose outputs are pure **Boolean functions** of its inputs.

• Output depends solely on input; no internal state. (Contrast **sequential** logic)

• Circuits made from basic gates wired up **without cycles**.

• Can always be described by a truth table for each output in terms of inputs, e.g.

A	B	C	Z
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	1
1	1	1	0

• Each output can be written as a **sum of products** of inputs.

– Just describe the truth table rows producing a 1.

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

Equivalent Circuits

A given boolean function can be represented by many different circuits, **optimizing** for different properties, e.g.,

• Low gate count, which gives low area.

• Low gate fan-in, which technology may require

= low number of literals in boolean formula

• Low number of logic levels, which gives shorter delay.

All circuits are described by Boolean formulas that can be proven equivalent using Boolean algebra.

Direct implementation of sum of products:

$Z_0 = \overline{A}\overline{B}C + \overline{A}BC + A\overline{B}C + AB\overline{C}$

Fewer literals, gates, and smaller fan-in:

$Z_1 = AB\overline{C} + \overline{A}C + \overline{B}C$

Three-level implementation has fewer gates but longer delay:

$Z_2 = T\overline{C} + \overline{T}C$
$T = AB$

Various forms of **minimization** are a big topic of study.

Programmable Logic Arrays

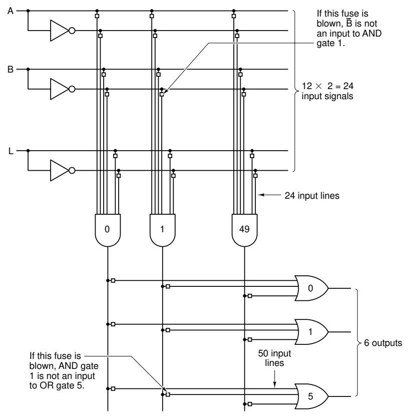


Figure 3-15. A 12-input, 6-output programmable logic array. The little squares represent fuses that can be burned out to determine the function to be computed. The fuses are arranged in two matrices: the upper one for the AND gates and the lower one for the OR gates.

Example: Multiplexer (MUX)

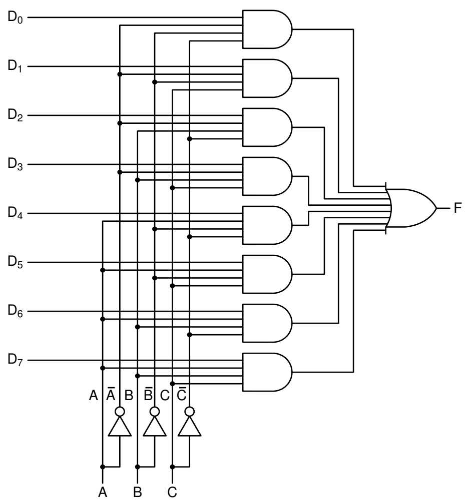


Figure 3-11. An eight-input multiplexer circuit.

Example: Decoder

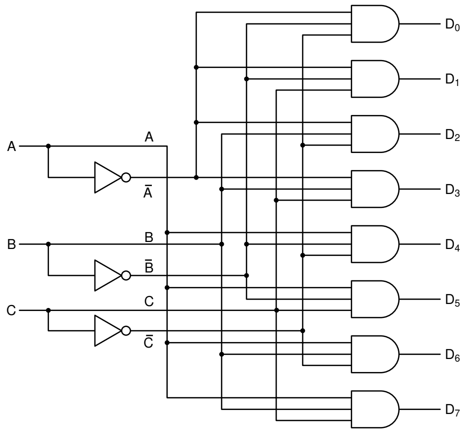


Figure 3-13. A 3-to-8 decoder circuit.

Clocks

Left to itself, a hardware circuit will always "compute" as fast as possible (and in parallel!)

To group computations into phases, we use **clock** signals. These are just signals that alternative regularly between 0 and 1.

The period while the clock signal is 1 is called a clock **pulse**. The pulse and the interval between pulses is precisely regulated.

The time between two consecutive pulse edges is called the **clock cycle time**.

Typically events can be associated with:

• leading or trailing edge of pulse (**edge-triggered** clocking); or

• state of clock signal (low or high)

Typical clock-based events including loading of **memory elements** (latches, flip-flops, registers, etc.)

Key point: If we want to compute some Boolean function within a single clock cycle, total delays in combinational circuit must be less than cycle time.

Adders

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

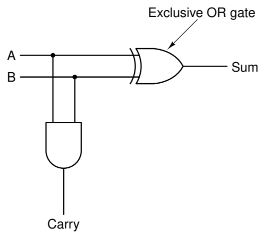
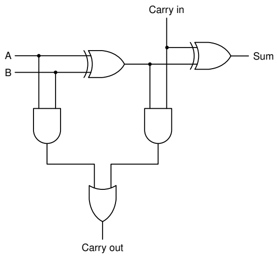


Figure 3-17. (a) Truth table for 1-bit addition. (b) A circuit for a half adder.

A	B	Carry in	Sum	Carry out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

(a)



(b)

Figure 3-18. (a) Truth table for full adder. (b) Circuit for a full adder.

ALU bitslice

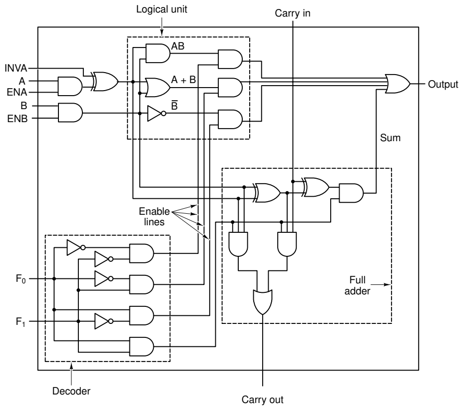


Figure 3-19. A 1-bit ALU.

Full ALU

In principle can compose from bit slices:

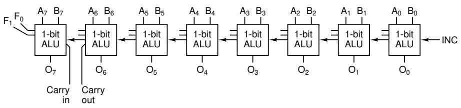


Figure 3-20. Eight 1-bit ALU slices connected to make an 8-bit ALU. The enables and invert signals are not shown for simplicity.

But in practice, cascading of carries from one bit to the next makes overall circuit too slow.

Also, need some whole-word outputs, e.g,

$Z = \overline{O_0 + O_1 + O_2 + \ldots + O_7}$

Note that shifter is typically a separate unit.

Full Datapath

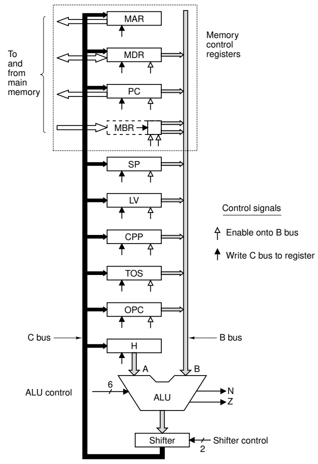


Figure 4-1. The data path of the example microarchitecture used in this chapter.