Architecture et Système

Stefan Schwoon

Cours L3, 2014/15, ENS Cachan

Logical circuits in computer architecture

Several levels of abstraction:

physical (transistor) level

logical-gate level

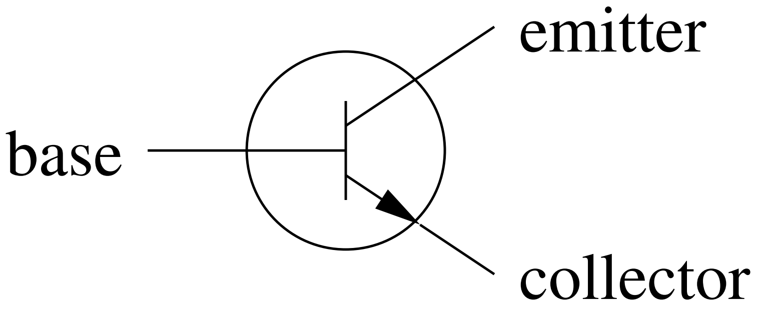
register (word) level

processor level

Static and dynamic circuits (time, storage)

Physical level

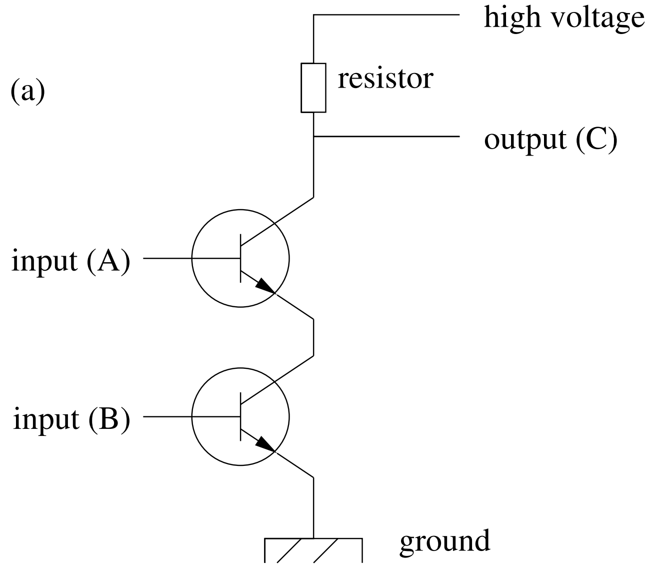
Most computers are built on (various types of) transistors:

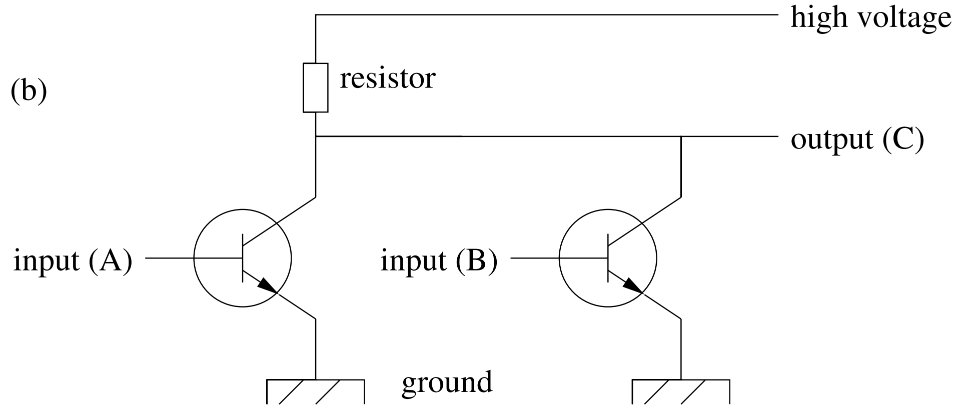


Voltage flows from emitter to collector when voltage at base exceeds a certain threshold (“is high”).

Logical gates using transistors

NAND (a) and NOR (b) gates built from transistors:





Logical gates

NAND and NOR gates are important for two reasons:

They can easily be realized using transistors.

All other logical functions can be built from either operator, e.g.:

$\neg A \equiv A \barwedge A$;

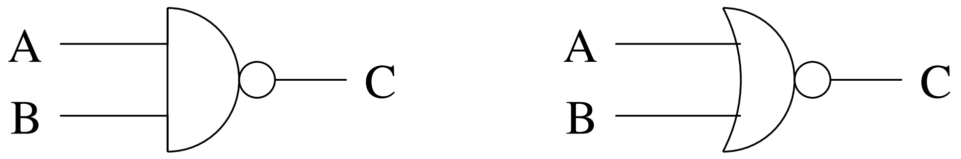
$A \wedge B \equiv (A \barwedge B) \barwedge (A \barwedge B)$;

$A \vee B \equiv (A \barwedge A) \barwedge (B \barwedge B)$.

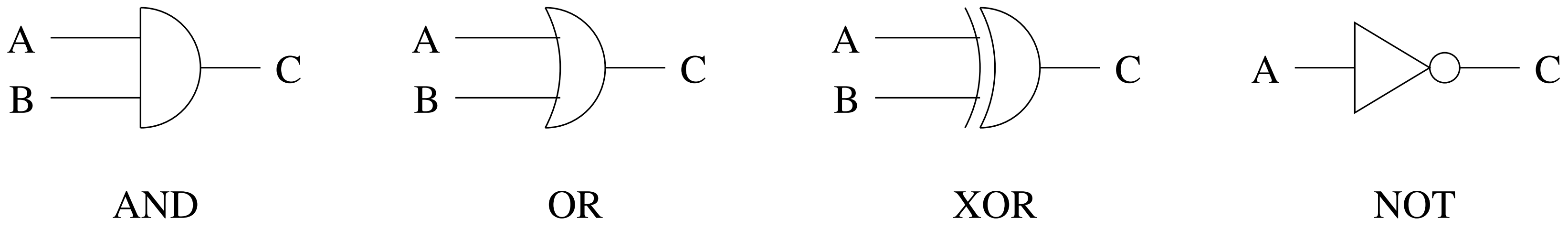
To abstract from physical details, we use a different type of diagram with logical gates. Here, we are concerned with the treatment of individual bits.

Logical-gate diagrams

Diagrams for NAND (left) and NOR (right):



Derived logical diagrams, possibly realised by combining several NAND/NOR gates at the physical level:



Complexity of logical circuits

Two characteristics are interesting in logical circuits:

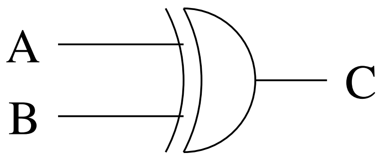
the size, i.e. how many transistors/gates/wires are present;
this determines the cost of a circuit.

the depth, i.e. the largest number of transistors/gates on a path between an input and an output; this determines the speed of the circuit.

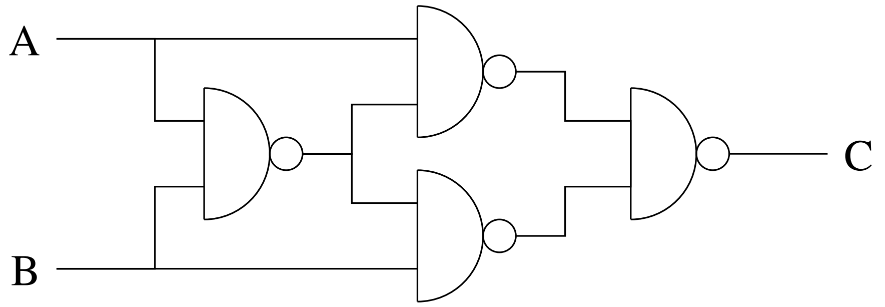
Minimising both size and depth are usually contradictory goals!

Complexity and derived diagrams

Consider the XOR gate:



Recall that it is physically realized by, e.g., multiple NAND gates:

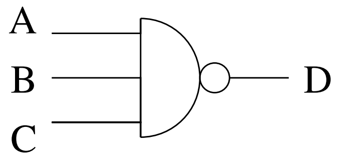


The first diagram suggests a size/depth of 1.
The second diagram suggests a size of 4 and a depth of 3.
The number of transistors is 8.

But this may change if only NOR gates are available . . .

Complexity and fan-in

Some functions (AND/OR etc) can be generalised to more than two inputs.



At transistor level, the size and depth can be affected as follows:

if only binary circuits are avaiable, the circuit above can be realized by sequentialising two NAND circuits (4 transistors/depth 2);

if the physical layout allows to sequentialise multiple transistors, then the circuit can be physically realised with 3 transistors and depth 1.

Conventions

In the following, we will be interested by functions of n bits, where n is a variable.

We will wish to know how the size/depth of a circuit realizing a certain function grows as n increases (i.e. their asymptotic complexity).

Typical goal: size $\mathcal{O}(n)$, depth $\mathcal{O}(\log n)$.

Convention: We allow ourselves fixed fan-in (arbitrary, but independent of n) and the previously given logical gates.

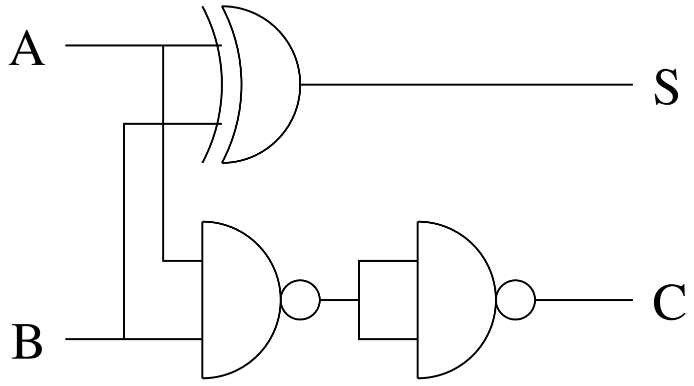
> At the transistor level, such gates have some constant size and depth (i.e. independent of n).
>
> Thus, the asymptotic complexity at transistor level and at gate level coincide.

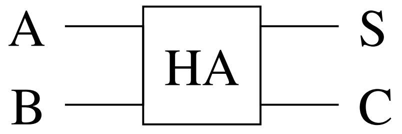
Arithmetic functions: Half adder

We first propose a so-called half adder:

- two input bits, A and B;
- two output bits, C (the *carry*) and S (the *sum*);
- desired result: $(C.S)_2 = A + B$ (where . denotes concatenation).

Possible realisation:



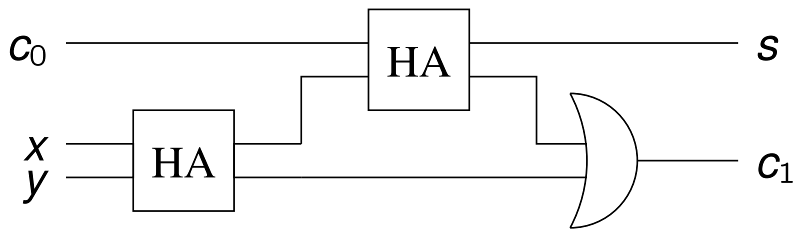


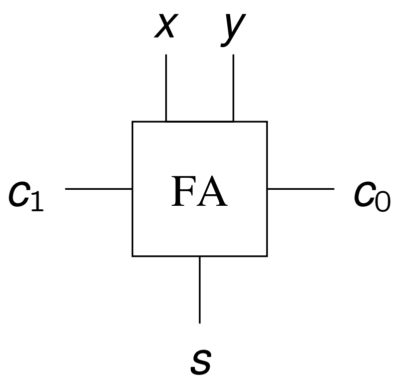
Full adder

A full adder produces the addition of three bits, we call the inputs x, y, and c_0, where c_0 may be the carry of a previous addition.

Desired result: $(c_1.s)_2 = x + y + c_0$

Possible realisation:





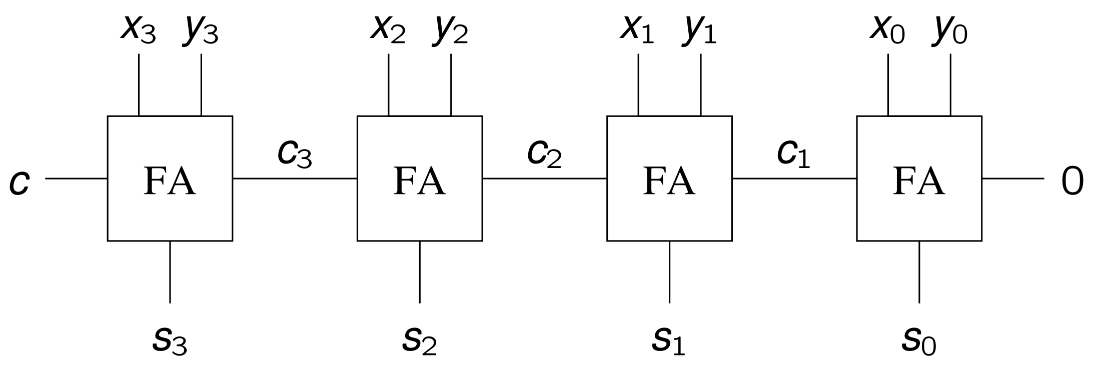
Other realisations are possible, for instance...?

Adding two integers

Suppose that we have two integers represented as vectors of, e.g., 4 bits.
$x = (x_3.x_2.x_1.x_0)_2$ and $y = (y_3.y_2.y_1.y_0)_2$.

We wish to compute the sum $s = x + y$ as $s = (c.s_3.s_2.s_1.s_0)_2$.

Realisation by sequentialising four adders:



Suppose that we generalise the principle of the sequential adder to n bits. Then the size of the circuit is $\mathcal{O}(n)$ and its depth also!

The latter is bad because it would mean that a computer would become (much) slower when its register size is increased.

We shall study a solution with *logarithmic* depth.

Adder based on propagate-generate bits

Let $0 \leq i \leq j < n$. We consider the input bits from position i to j:

> the block $i..j$ is said to generate a carry if $c_j = 1$ *independently* of the bits at other positions;
>
> the block $i..j$ is said to propagate a carry if $c_i = 1$ implies $c_{j+1} = 1$.

Computing these two bits depends only on $(x_i, y_i) \cdots (x_j, y_j)$:

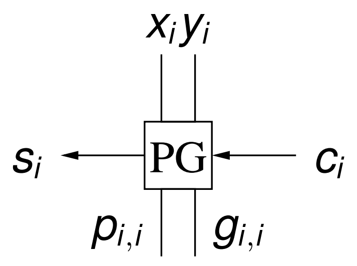
When $i = j$, then $g_{i,j} = 1$ iff $x_i + y_i = 2$ and $p_{i,j} = 1$ iff $x_i + y_i \geq 1$.

Otherwise, let $i \leq k < j$, then:

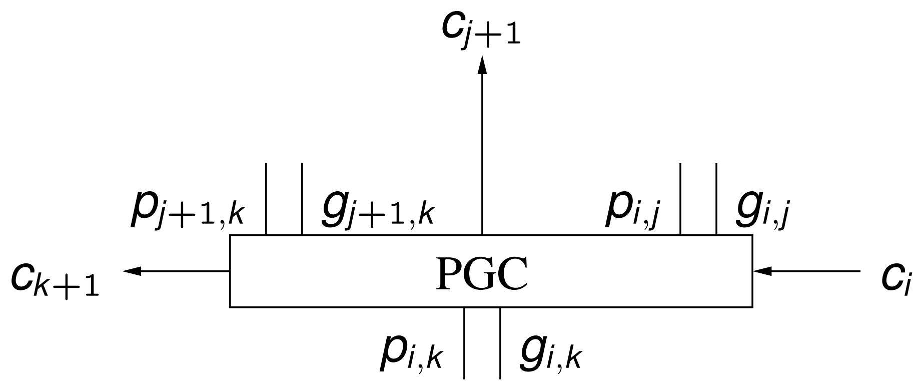
$g_{i,j} = g_{k+1,j} \vee (g_{i,k} \wedge p_{k+1,j})$ and $p_{i,j} = p_{i,k} \wedge p_{k+1,j}$

Also, $c_{j+1} = g_{i,j} \vee (c_i \wedge p_{i,j})$.

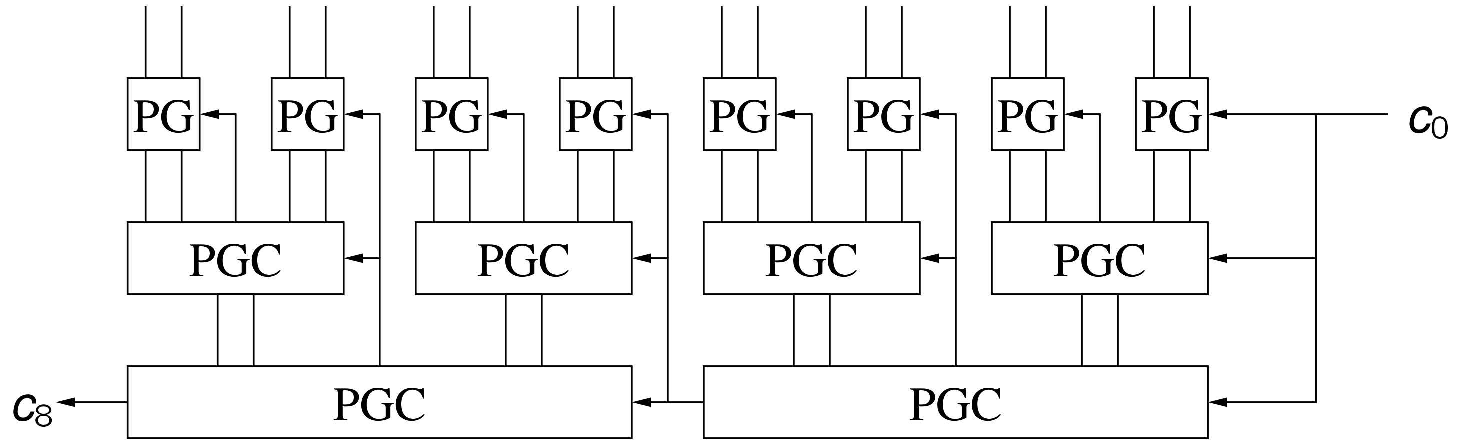
Symbol for a one-bit full adder extended with p/g bits:



Symbol for combining p/g bits from two blocks:



Simplified diagram for 8 bits, only the propagation of carries is detailed:



Notice that the p/g bits of an *m*-th level PGC block are ready after a delay of $\mathcal{O}(m)$. Once the p/g bits *and* the input carry are ready, the output carry can be computed.

The longest path (in terms of PGC blocks) for a carry is approximately $2 \log n$.

Further arithmetic/logical functions

Subtraction, multiplication, division, . . .

Comparison, test for zero, . . .

Vector-wise AND, OR, XOR, . . .

Shifting left or right

Multiplexer: input n bits, select one as output

Decoder: input i, output $y_i = 1$ and $y_j = 0$ for $j \neq i$