From Boolean Algebra to Arithmetics

ECE_3TC31_TP/INF107

Tarik Graba Ulrich Kühne Guillaume Duc

2024

Boolean Algebra

Logic Variables and Functions

Logic Variable

- A logic variable takes on one of two values: 0 (*false*) or 1 (*true*).
- We denote the set of Boolean values by $\mathbb{B} = \{0, 1\}$

Logic Function

- A logic function takes one or more logic variables as inputs and returns 0 or 1:

$$
\begin{aligned}
F : \mathbb{B} \times \mathbb{B} \ldots \times \mathbb{B} &\rightarrow \mathbb{B} \\
x_0, x_1, \ldots, x_n &\rightarrow y = F(x_0, x_1, \ldots, x_n)
\end{aligned}
$$

Hardware Implementation of Logic Variables

In hardware, to represent the two values of a logic variable, we use:

- 2 different voltages (0 V/5 V, -12 V/+12 V...)
- 2 different electric currents
- Presence/absence of light in an optical fiber

Representation of Logic Functions

A logic function can be represented in different ways:

- With a *truth table*: table that lists the value of the function for all possible inputs
- With an *equation*
- With a *diagram*: a graphical representation using normalized symbols
- Using an *Hardware Description Language* (HDL): a computer language designed to be easily interpreted by a computer program

Combinational and Sequential Logic

Combinational Logic

The output depends only on the present value of the inputs

$$\forall t, y(t) = F(x_0(t), x_1(t), \dots, x_n(t))$$

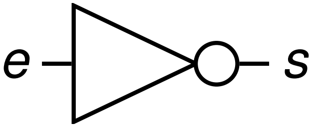
Sequential logic

The output depends on the present value of the input and on the sequence of past inputs

$$y(t) = F(x_0(t), x_1(t), \dots, x_n(t), x_0(t-1), x_1(t-1) \dots)$$

Basic Logic Gates

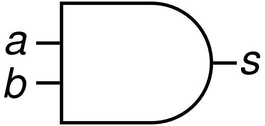
NOT (Inverter)



$$s = \overline{e}$$

e	**s**
0	1
1	0

AND

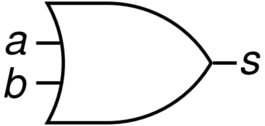


$$s = a \cdot b$$

a	b	s
0	0	0
0	1	0
1	0	0
1	1	1

Note: $x \cdot 0 = 0$ et $x \cdot 1 = x$, so an AND gate can be use to produce, from a signal x, a signal that equals to x or 0 depending on a command signal

OR

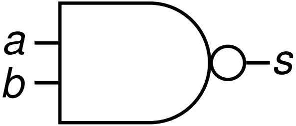


$$s = a + b$$

a	b	s
0	0	0
0	1	1
1	0	1
1	1	1

Note: $x + 0 = x$ et $x + 1 = 1$, so an OR gate can be use to produce, from a signal x, a signal that equals to x or 1 depending on a command signal

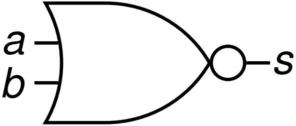
NAND (Not AND)



$$s = \overline{a \cdot b}$$

a	b	s
0	0	1
0	1	1
1	0	1
1	1	0

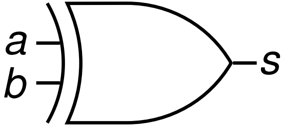
NOR (Not OR)



$$s = \overline{a + b}$$

a	b	s
0	0	1
0	1	0
1	0	0
1	1	0

XOR (Exclusive OR)



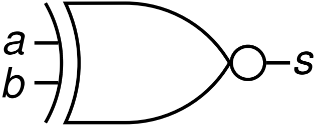
$$s = a \oplus b$$

$$s = a \cdot \overline{b} + \overline{a} \cdot b$$

a	b	s
0	0	0
0	1	1
1	0	1
1	1	0

Note: $x \oplus 0 = x$ et $x \oplus 1 = \overline{x}$, so an XOR gate can be use to produce, from a signal x, a signal that equals to x or $\overline{x}$ depending on a command signal

XNOR (Not Exclusive OR, Equality)



$$s = \overline{a \oplus b}$$

$$s = a \cdot b + \overline{a} \cdot \overline{b}$$

a	b	s
0	0	1
0	1	0
1	0	0
1	1	1

Boolean Algebra

Boolean Algebra

The set $\mathbb{B}$ with conjunction ($a \cdot b$), disjunction ($a + b$), and negation($\overline{a}$) forms a *Boolean Algebra*

Associativity

$$a + (b + c) = (a + b) + c \quad \text{(AssocOr)}$$
$$a \cdot (b \cdot c) = (a \cdot b) \cdot c \quad \text{(AssocAnd)}$$

Commutativity

$$a + b = b + a \quad \text{(CommOr)}$$
$$a \cdot b = b \cdot a \quad \text{(CommAnd)}$$

Idempotence

$$a + a = a \quad \text{(IdemOr)}$$
$$a \cdot a = a \quad \text{(IdemAnd)}$$

Distributivity

$$a + (b \cdot c) = (a + b) \cdot (a + c) \quad \text{(DistrOr)}$$
$$a \cdot (b + c) = (a \cdot b) + (a \cdot c) \quad \text{(DistrAnd)}$$

More Rules

Neutral elements

$$a + 0 = a \quad \text{(Zero)}$$
$$a \cdot 1 = a \quad \text{(One)}$$

Annihilation

$$a + 1 = 1 \quad \text{(AnnOr)}$$
$$a \cdot 0 = 0 \quad \text{(AnnAnd)}$$

Absorption

$$a + (a \cdot b) = a \quad \text{(AbsOr)}$$
$$a \cdot (a + b) = a \quad \text{(AbsAnd)}$$

Complementary

$$a + \overline{a} = 1 \quad \text{(ComplOr)}$$
$$a \cdot \overline{a} = 0 \quad \text{(ComplAnd)}$$

Double negation

$$\overline{\overline{a}} = a \quad \text{(DoubleNeg)}$$

De Morgan

$$\overline{a + b} = \overline{a} \cdot \overline{b} \quad \text{(DeMorgan1)}$$
$$\overline{a \cdot b} = \overline{a} + \overline{b} \quad \text{(DeMorgan2)}$$

Rewriting Logic Terms by Applying Boolean Algebra Rules

Showing the equivalence of two different terms for exlusive or:

$$
\begin{aligned}
& (a+b)\cdot(\overline{a}+\overline{b}) && \text{by (DistrAnd)} \\
&= (a+b)\cdot\overline{a} + (a+b)\cdot\overline{b} && \text{by (DistrAnd)} \\
&= a\cdot\overline{a} + b\cdot\overline{a} + a\cdot\overline{b} + b\cdot\overline{b} && \text{by (ComplAnd)} \\
&= 0 + b\cdot\overline{a} + a\cdot\overline{b} + 0 && \text{by (Zero)} \\
&= b\cdot\overline{a} + a\cdot\overline{b} && \text{by (CommAnd)} \\
&= \overline{a}\cdot b + a\cdot\overline{b}
\end{aligned}
$$

From Truth Table to Boolean Equation

- Each row corresponds to a *minterm*

a	b	c	minterm
0	0	0	$\overline{a} \cdot \overline{b} \cdot \overline{c}$
0	0	1	$\overline{a} \cdot \overline{b} \cdot c$
0	1	0	$\overline{a} \cdot b \cdot \overline{c}$
0	1	1	$\overline{a} \cdot b \cdot c$
1	0	0	$a \cdot \overline{b} \cdot \overline{c}$
1	0	1	$a \cdot \overline{b} \cdot c$
1	1	0	$a \cdot b \cdot \overline{c}$
1	1	1	$a \cdot b \cdot c$

- Take the conjunction of minterms leading to 1

a	b	c	z
0	0	0	**1**
0	0	1	**1**
0	1	0	**0**
0	1	1	**0**
1	0	0	**1**
1	0	1	**0**
1	1	0	**0**
1	1	1	**1**

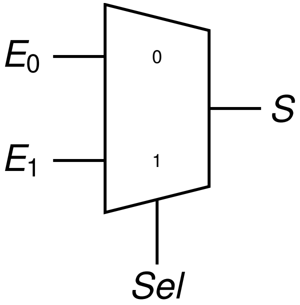
$$z = (\overline{a} \cdot \overline{b} \cdot \overline{c}) + (\overline{a} \cdot \overline{b} \cdot c) + (a \cdot \overline{b} \cdot \overline{c}) + (a \cdot b \cdot c)$$

More complex gates

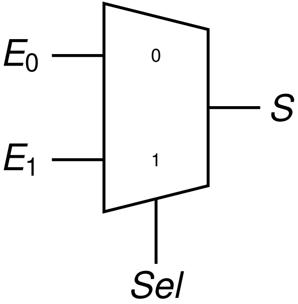
2-to-1 Multiplexer

We want to build a function that selects one of its two inputs (E_0 or E_1) depending on a third "selection" input (Sel):

- $S = E_0$ if $Sel = 0$
- $S = E_1$ if $Sel = 1$



2-to-1 Multiplexer

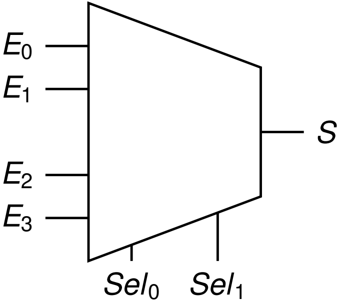


$$S = \overline{Sel} \cdot E_0 + Sel \cdot E_1$$

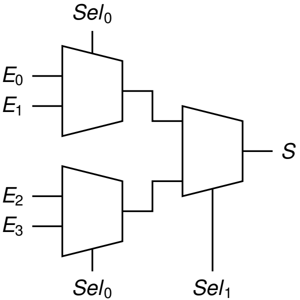
Sel	E_0	E_1	S
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

4-to-1 Multiplexer

A 4-to-1 multiplexer (4 inputs so 2 selection inputs)



Can be built from 3 2-to-1 multiplexers



$$S = \overline{Sel_1} \cdot \overline{Sel_0} \cdot E_0 + \overline{Sel_1} \cdot Sel_0 \cdot E_1 + Sel_1 \cdot \overline{Sel_0} \cdot E_2 + Sel_0 \cdot Sel_1 \cdot E_3$$

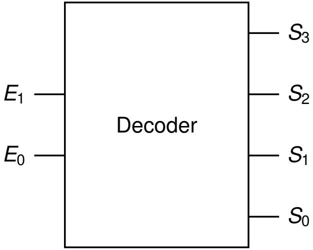
n-to-1 Multiplexer

A n-to-1 multiplexer (with $n = 2^p$):

- needs p selection inputs
- can be built with $n - 1$ 2-to-1 multiplexer organized in p layers

Decoder

A decoder has n inputs and 2^n outputs. Only one output (selected by the value of the inputs) is at 1, all others are at 0.



E_0	E_1	S_3	S_2	S_1	S_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Representation of numbers

Representation of positive integers

A positive integer N can be represented in base b by a vector $(a_{n-1}, a_{n-2}, \ldots, a_1, a_0)$, such as:

$$N = a_{n-1} \cdot b^{n-1} + a_{n-2} \cdot b^{n-2} + \ldots + a_1 \cdot b^1 + a_0 \cdot b^0$$

where:

- $a_i \in \{0, 1, \ldots, b-1\}$
- a_{n-1} is the most significant digit
- a_0 is the least significant digit

Commonly used bases

- Decimal: $b = 10$, $a_i \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- Hexadecimal: $b = 16$, $a_i \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$
- Octal: $b = 8$, $a_i \in \{0, 1, 2, 3, 4, 5, 6, 7\}$
- Binary: $b = 2$, $a_i \in \{0, 1\}$

Binary representation of positive integers

A positive integer N can be represented in base 2 by a vector $(a_{n-1}, a_{n-2}, \dots, a_1, a_0)$, such as:

$$N = a_{n-1} \cdot 2^{n-1} + a_{n-2} \cdot 2^{n-2} + \dots + a_1 \cdot 2^1 + a_0 \cdot 2^0$$

where:

- $a_i \in \{0, 1\}$
- a_i is a binary digit (bit)
- a_{n-1} is the most significant bit
- a_0 is the least significant bit

Exercise

Give the binary representation of 54_{10}

Conversion between binary and hexadecimal representations

In base 2 (we suppose that n is a multiple of 4):

$$N = a_{n-1} \cdot b^{n-1} + a_{n-2} \cdot b^{n-2} + ... + a_1 \cdot b^1 + a_0 \cdot b^0$$

As $2^4 = 16$, we also have:

$$N = \sum_{k=0}^{n/4-1} (a_{4k+3} \cdot 8 + a_{4k+2} \cdot 4 + a_{4k+1} \cdot 2 + a_{4k}) \cdot 16^k$$

So it is easy to convert between hexadecimal and binary representation (each hexadecimal digit corresponds to 4 bits). In addition, the hexadecimal representation is more compact than the binary representation.

Exercise

- Convert $7A_{16}$ in binary
- Convert 11111100_2 in hexadecimal

Binary representation

In a digital circuit (a processor for instance), the number of bits used for representing numbers is limited.

For n bits:

- There are 2^n values that can be represented
- We can represent numbers in $[0, 2^n - 1]$
- Arithmetic is performed modulo 2^n

Binary representation

- With 4 bits, we can represent numbers from 0 to $15 = 2^4 - 1$
- The arithmetic is modulo $2^4 = 16$:
  - $15 + 1 = 0$
  - $0 - 1 = 15$

Decimal	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

Decimal	Binary
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

Representation of integers

- With 4 bits, we can represent numbers from 0 to $15 = 2^4 - 1$
- The arithmetic is modulo $2^4 = 16$:
  - $15 + 1 = 0$
  - $0 - 1 = 15$
- **How to keep the same behaviour and represent negative and positive integers?**

Decimal	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

Decimal	Binary
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

Representation of integers

- With 4 bits, we can represent numbers from 0 to $15 = 2^4 - 1$
- The arithmetic is modulo $2^4 = 16$:
  - $15 + 1 = 0$
  - $0 - 1 = 15$
- **We can interpret 1111 as -1 instead of 15**

Decimal	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

Decimal	Binary
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
-1	1111

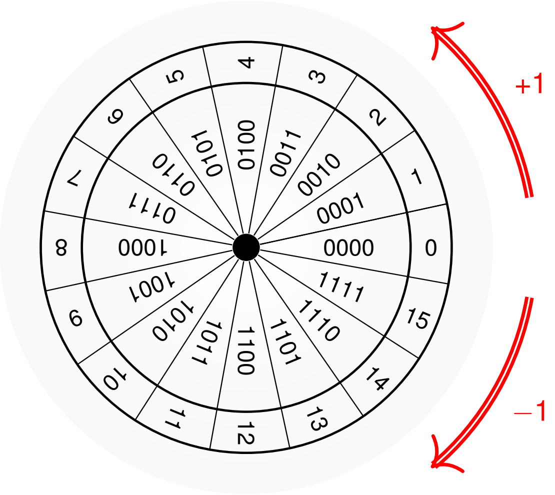
Representation of integers

- With 4 bits, we can represent numbers from 0 to $15 = 2^4 - 1$
- The arithmetic is modulo $2^4 = 16$:
  - $15 + 1 = 0$
  - $0 - 1 = 15$
- **We can interpret half the numbers as negatives**

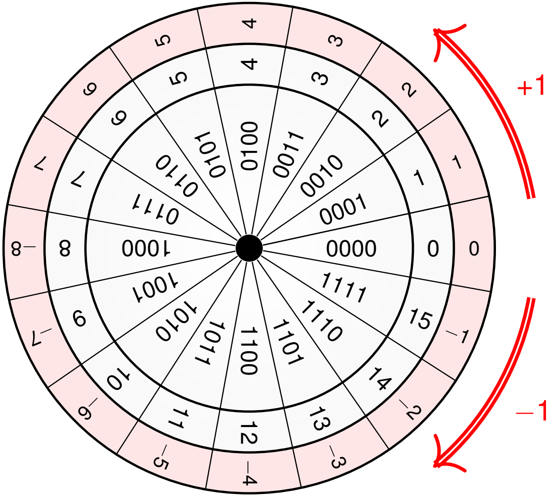
Decimal	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

Decimal	Binary
-8	1000
-7	1001
-6	1010
-5	1011
-4	1100
-3	1101
-2	1110
-1	1111

Representation of integers



Representation of integers



Two's complement

The two's complement is a way to represent signed numbers (it is the most commonly used but not the only one)

The most significant bit holds an information about the sign (0: positive, 1: negative)

The value A of an n-bit integer $a_{N-1}a_{N-2}\dots a_0$ in two's complement is:

$$A = -a_{n-1}2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i$$

The two's complement can represent integers in the range $[-2^{n-1}, 2^{n-1}-1]$

Exercises

- Represent -8 and +8 in two's complement
  - Using 4 bits
  - Using 5 bits
- Represent -1
  - Using 1 bit
  - Using 2 bits
  - Using 3 bits

Sign extension

If $N = a_{n-1}, a_{n-2}, \dots, a_0$ a signed integer represented in two's complement with n bits, how to represent N with $n+1$ bits?

$$N = -a_{n-1}2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i = -a_{n-1}2^{n-1} \cdot (2-1) + \sum_{i=0}^{n-2} a_i 2^i = -a_{n-1}2^{n} + \sum_{i=0}^{n-1} a_i 2^i$$

So $N = a_{n-1}, a_{n-1}, a_{n-2}, \dots, a_0$ with $n+1$ bits: the most significant bit is duplicated

Additive inverse/The opposite

If N is a signed integer represented in two's complement with n bits, and if $-N$ can be represented in two's complement with n bits:

$$-N = \overline{N} + 1$$

The opposite (proof)

$$
\begin{aligned}
N &= -a_{n-1}2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i \\
-N &= a_{n-1}2^{n-1} - \sum_{i=0}^{n-2} a_i 2^i \\
&= a_{n-1}2^{n-1} + \sum_{i=0}^{n-2} (-a_i) 2^i
\end{aligned}
$$

If b is a bit, $(1 - b = \overline{b})$ or $(-b = -1 + \overline{b})$, so:

$$
-N = a_{n-1}2^{n-1} + \sum_{i=0}^{n-2} (-1 + \overline{a_i}) 2^i
$$

The opposite (proof)

$$\begin{aligned}
-N &= a_{n-1}2^{n-1} - \sum_{i=0}^{n-2} 2^i + \sum_{i=0}^{n-2} \overline{a_i}2^i \\
&= a_{n-1}2^{n-1} - (2^{n-1} - 1) + \sum_{i=0}^{n-2} \overline{a_i}2^i \\
&= (a_{n-1} - 1)2^{n-1} + \sum_{i=0}^{n-2} \overline{a_i}2^i + 1 \\
&= -\overline{a_{n-1}}2^{n-1} + \sum_{i=0}^{n-2} \overline{a_i}2^i + 1 \\
&= \overline{N} + 1
\end{aligned}$$

Arithmetic operators

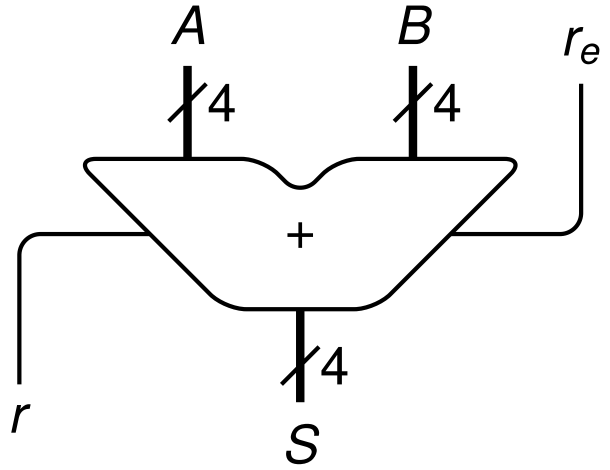
Addition

Do the addition of two 4-bit numbers

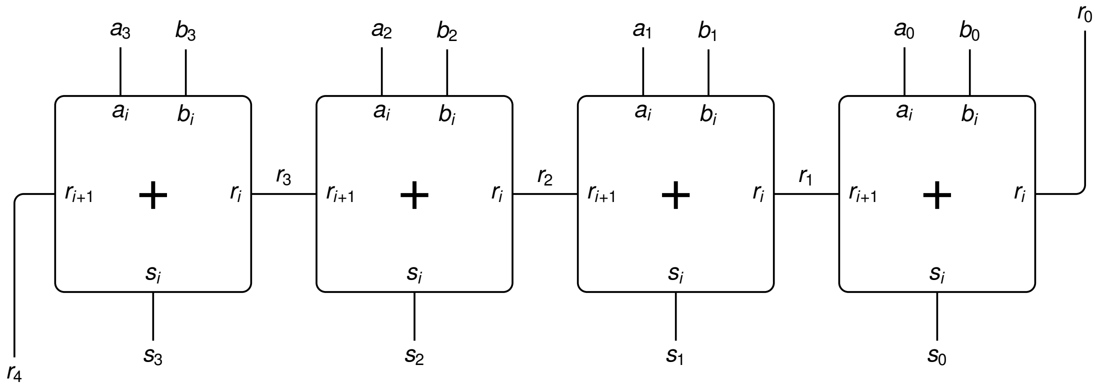
Addition

The addition of two binary numbers can be decomposed into several elementary addition on 1 bit.

Ripple-carry adder (carry-propagate adder)



Ripple-carry adder



Full adder (1 bit)

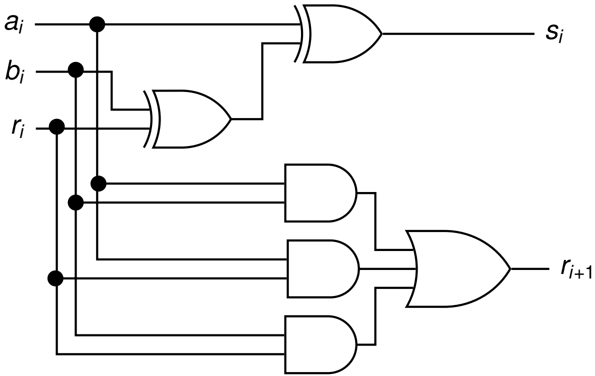
Arithmetically: $a_i + b_i + r_i = 2 \cdot r_{i+1} + s_i$

a_i	b_i	r_i	r_{i+1}	s_i	Decimal
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	1	1
0	1	1	1	0	2
1	0	0	0	1	1
1	0	1	1	0	2
1	1	0	1	0	2
1	1	1	1	1	3

Full adder (1 bit)

$$s_i = a_i \oplus b_i \oplus r_i$$

$$r_{i+1} = a_i \cdot b_i + a_i \cdot r_i + b_i \cdot r_i$$



Addition (natural integers)

If A and B are two natural integers represented with n bits:

$$\begin{aligned} A &\leq 2^n - 1 \\ B &\leq 2^n - 1 \\ A + B &\leq 2^{n+1} - 2 < 2^{n+1} \end{aligned}$$

So the result of $A + B$ can always be represented with $n + 1$ bits

Addition (two's complement)

If A and B are two integers represented in two's complement with n bits:

$$\begin{array}{rcrcl} -2^{n-1} & \leq & A & \leq & 2^{n-1}-1 \\ -2^{n-1} & \leq & B & \leq & 2^{n-1}-1 \\ -2^{n} & \leq & A+B & \leq & 2^{n}-2 < 2^{n} \end{array}$$

So the result of $A + B$ can always be represented with $n + 1$ bits

Addition (two's complement)

Addition of two integers represented on 3 bits:

					unsigned	$2'sC$
		1	1	1	7	-1
$+$		0	0	1	1	1
$=$	1	0	0	0	8	0 or $-8?$

					unsigned	$2'sC$
		0	1	1	3	3
$+$		0	0	1	1	1
$=$	1	1	0	0	4	-4 or $+4?$

					unsigned	$2'sC$
		1	1	1	7	-1
$+$		1	0	0	4	-4
$=$	1	0	1	1	11	$+3$ or $-5?$

Addition (two's complement)

There is an issue with the interpretation of the carry in two's complement.

The simple solution to always have the correct answer is to sign extend the operands to one more bit and then do the addition. The resulting carry can be discarded.

$$
\begin{array}{cccccc|r}
 & & 1 & 1 & 1 & 1 & -1 \\
+ & & 1 & 1 & 0 & 0 & -4 \\
\hline
= & \cancel{1} & 1 & 0 & 1 & 1 & -5
\end{array}
$$

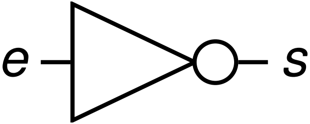
$$
\begin{array}{cccccc|r}
 & & 1 & 1 & 1 & 1 & -1 \\
+ & & 0 & 0 & 0 & 1 & 1 \\
\hline
= & \cancel{1} & 0 & 0 & 0 & 0 & 0
\end{array}
$$

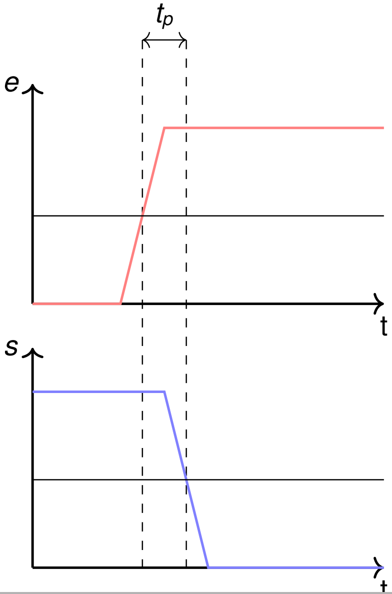
Propagation time

Propagation time of a gate

- When the input of a gate changes, its output cannot change instantaneously.
- The **propagation time** is the time between the instant when the inputs of a gate change and the instant when the output of the gate stabilizes to the correct value.
- During this time, the output of a gate may be invalid (with regards to the current value of its inputs).

Example

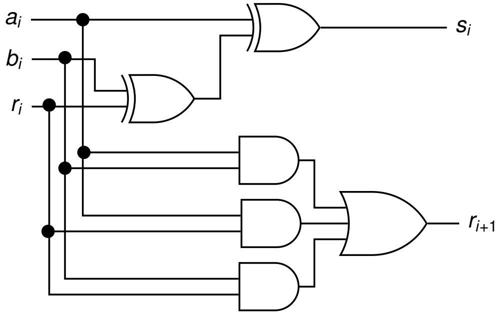




Propagation time of a complex function

- For a given technology, the propagation times of basic gates are given
- From these values, we can compute the propagation time of more complex functions by adding the individual propagation time
- The propagation time of a complex function is the propagation time on the longest path

Example: full adder



- We consider the following propagation times:
  - AND and OR gates: 1 ns
  - XOR gates: 2 ns
- What is the propagation time from each inputs to each output?