

Shared Resources and Blocking in Real-Time Systems

Giuseppe Lipari

`http://feanor.sssup.it/~lipari`

Scuola Superiore Sant'Anna – Pisa

April 1, 2008

Outline

1 Boolean algebra

2 Binary systems

Outline

1 Boolean algebra

2 Binary systems

An algebra for logic

- Domain: {true, false}
- Basic operations: {and, or, not}
- Truth tables:

a and b	false	true
false	false	false
true	false	true

a or b	false	true
false	false	true
true	true	true

a	not a
false	true
true	false

Other operators

- $a \text{ nand } b \equiv \text{not } (a \text{ and } b)$
- $a \text{ nor } b \equiv \text{not } (a \text{ or } b)$
- $a \rightarrow b \equiv \text{not } (a \text{ and not } b)$
- $a \text{ xor } b \equiv (a \text{ or } b) \text{ and not } (a \text{ and } b)$
- It can be shown that every operator can be derived by either nand or nor

Properties

- It is an algebra, thus it has the following properties:
 - the identity for `and` is `true`
 - the identity for `or` is `false`
 - the null element for `and` is `false`
 - commutativity. ex: $a \text{ and } b \equiv b \text{ and } a$
 - associativity. ex: $a \text{ or } (b \text{ or } c) \equiv (a \text{ or } b) \text{ or } c$

Boolean algebra in digital electronic systems

- It is possible to build electronic logic gates that
 - Interpret high voltage as `true` and low voltage as `false`
 - Implement logic operations like `nand` and `nor`

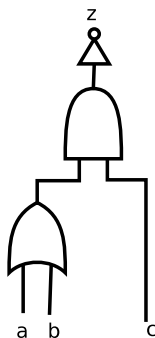


Figure: A logic circuit that implements `not ((a or b) and c)`

Outline

1 Boolean algebra

2 Binary systems

Boolean algebra in computers

- In digital electronic systems, high and low voltages are interpreted as two different symbols, 1 and 0 respectively
- It is possible to build arithmetic using binary encoding of numbers and symbols
- Definitions:
 - one binary digit (0 or 1) is a *bit*
 - a group of 8 binary digits is a *byte*
 - a *word* in current processor is 4 bytes (32 bits)

Encoding integer numbers

- Translation from decimal to binary and viceversa
 - Let's start from positive integer numbers
 - the minimum number is 0000 0000 (0 in decimal)
 - the maximum number is 1111 1111 (255 in decimal)
 - how to translate a binary number:

0100 1011 =

$$1 \cdot 2^0 + 1 \cdot 2^1 + 0 \cdot 2^2 + 1 \cdot 2^3 + 0 \cdot 2^4 + 0 \cdot 2^5 + 1 \cdot 2^6 + 0 \cdot 2^7 = 75$$

0011 0110 =

$$0 \cdot 2^0 + 1 \cdot 2^1 + 1 \cdot 2^2 + 0 \cdot 2^3 + 1 \cdot 2^4 + 1 \cdot 2^5 + 0 \cdot 2^6 + 0 \cdot 2^7 = 54$$

Summing integer numbers

- By using boolean logic, we can implement binary adders
- Truth table of an adder: $s = x + y$, plus the carry

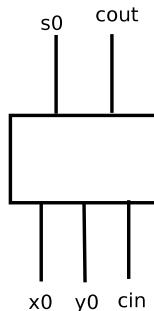
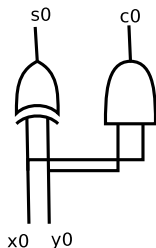
$x \backslash y$	0	1
0	0	1
1	1	0

$x \backslash y$	0	1
0	0	0
1	0	1

- $s = x \text{ xor } y$
- $c = x \text{ and } y$

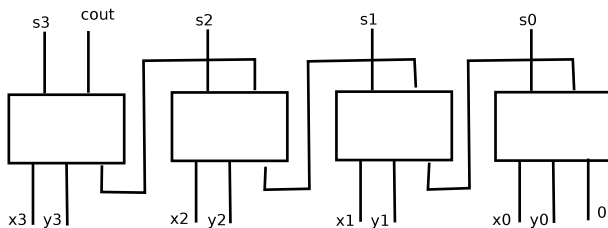
Basic adder and full adder

The following diagram represent a 2-bit adder



Full adders

To implement a 4-bit adder, we compose 4 full-adders:



How to represent negative numbers

There are many systems to represent negative integers

- 1 use the first bit as a sign: 0 is positive, 1 is negative
 - 0111 1111 corresponds to 127
 - 1111 1111 corresponds to -127
 - Problem: zero is represented twice, 1000 0000 and 0000 0000
 - Not possible to directly use this representation in sums
- 2 Two's complement
 - represent positive numbers up to 127 normally
 - represent negative numbers as the positive, negated (bit by bit) plus 1

Example: represent -58 on 8 bits:

- 58 is: 0011 1010
- negation is: 1100 0101
- plus 1: 1100 0110

Hence, the representation of -58 is 1100 0110

Representing characters

- It is possible to represent characters and string using an appropriate encoding
- The ASCII encoding assigns each character a number between 0 and 255
- Some example of character encoding:

bin	dec	glyph
011 0000	48	'0'
011 0001	49	'1'
011 0010	50	'2'
011 0011	51	'3'
011 0100	52	'4'
011 0101	53	'5'
011 0110	54	'6'
011 0111	55	'7'
011 1000	56	'8'
011 1001	57	'9'

bin	dec	glyph
110 0001	97	a
110 0010	98	b
110 0011	99	c
110 0100	100	d
110 0101	101	e
110 0110	102	f
110 0111	103	g
110 1000	104	h
110 1001	105	i
110 1010	106	j

Representing decimal numbers

Two possible systems:

- Fixed point representation: a fixed number of bits are for the integer part, the remaining for the rational part
 - Used in some embedded system (DSP) because calculations are usually faster
 - fixed precision, limited range
- Floating point representation: a fixed number of bits to represent the mantissa, and the remaining to represent the exponent
 - Used in modern PCs
 - very wide range, variable precision

IEEE 754 standard for floating point

