

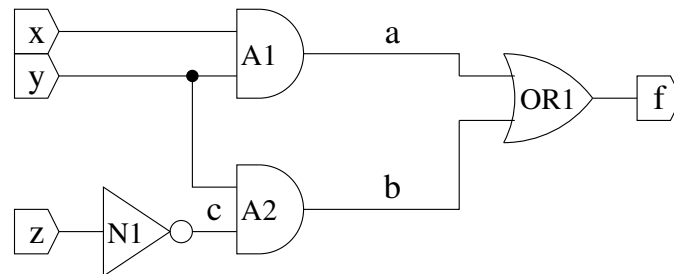
17 STRUCTURAL DESCRIPTIONS

A behavioral description formally defines what the digital system is supposed to do, but does not say anything about how the behavior is to be achieved. There are many possible ways to obtain that behavior, and each way is referred to as a “realization” or *structural description* of the digital system.

A structural description identifies how the behavior of a digital system can be realized with existing digital components. Typically provided by a schematic or textual description:

- schematic: A circuit or logic diagram.
- HDL: A textual description of a schematic, expressed using a “hardware description language (HDL),” of the components and their interconnections.

The following is an example of a structural description:



The purpose of “digital design” is to obtain a structural description from a behavioral description.

Complex digital systems design is “top down”:

1. The entity is first resolved into major functional components
2. Each functional component is either:
 - resolved to “inventory available” components or
 - resolved into smaller functional units.
3. If any functional units are not inventory available, then go to step 1 with the entity for each such functional unit.

Before attempting to design a complex system, it is important to become familiar with the behavioral descriptions of the inventory available components and how to design digital systems using them.

18 DESIGN WITH GATES

A “gate” is a digital system that implements a Boolean operator. That is, gates can be used to implement the C Boolean functions: `&&` (and), `||` (or), and `!` (not).

18.1 Gate Entities

Gate entities occur so frequently in applications that, by convention, they are given special graphical symbols that uniquely represent their entities. The table below provides examples of the graphic symbols for a number of gate entities: i


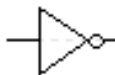





	Buffer		NOT (inverter)
	2-input AND		2-input NAND
	2-input OR		2-input NOR
	2-input XOR		

Figure 1: The Basic Gate Entities

Of these, the “2 input And”, “2 input Or”, 2-input XOR, and “Not” entities represent the digital components that implement the corresponding C Boolean operators.

The functional specification of a simple combinational system can be expressed with a “function table.” This table is simply a list of all possible binary input sequences and for each sequence, the corresponding binary output sequence. The function tables associated with the four gate entities are:

- NOT : function table is given by:

x	!x
0	1
1	0

- 2-input AND : function table is given by:

x	y	x && y
0	0	0
0	1	0
1	0	0
1	1	1

- 2-input OR : function table is given by:

x	y	x y
0	0	0
0	1	1
1	0	1
1	1	1

- 2-input XOR: function table is given by:

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

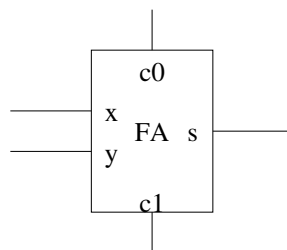
19 DESIGNING DIGITAL SYSTEMS WITH GATES

The goal of digital design is to obtain a structural description from a behavioral description. All gate-level structural descriptions represent implementations of behavioral descriptions that can be expressed as a set of Boolean equations. To construct a schematic from a set of such equations:

1. For each operator, F, introduce the appropriate gate, G.
2. For gate G, representing operator F:
 - (a) If the operand is a label, it represents an external input.
 - (b) If the operand is an expression:
 - i. Construct a subcircuit for the expression
 - ii. Connect its output to an input port of G.
 - (c) Repeat for all operands of F.
3. Repeat until each gate replaces an operator.

Example: A behavioural description is given by:

Entity Definition :

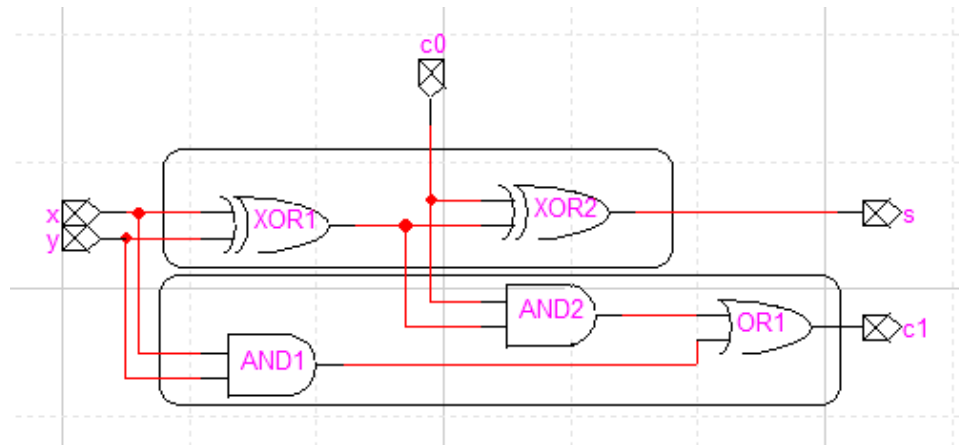


Functional Specification :

$$\begin{aligned}
 s &= (x \oplus y) \oplus c0 \\
 c1 &= (x \&\& y) \vee (c0 \&\& (x \oplus y))
 \end{aligned}$$

While the conversion can begin with any operator and any equation, it is usually easiest to start with the “innermost” expressions. Furthermore, common terms can be shared by more than one circuit. In this case the expression $x \oplus y$ is common to both boolean expressions. Therefore the XOR gate used to implement this expression is shared by both circuits.

The resulting circuit is as follows. The gates that define each output have been enclosed in boxes with rounded corners:



Notational Changes

While the use of C symbols for the Boolean operators results in Boolean expressions expressed in C source code and follows the convention of the text, these symbols are not the convention in the digital design community. The following symbols are equivalent:

Operator	C Notation	standard Notation
NOT(x)	! x	\bar{x} or x'
OR(x, y)	x y	$x + y$
AND(x, y)	x && y	$x \cdot y$ or xy

METHOD 2: From a function table:

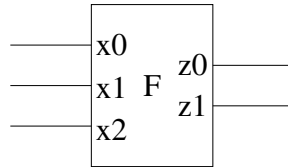
Literal: A variable or the complement of a variable.

For each output column:

1. Select a row for which the output is '1'.
2. Construct a conjunction of inputs as follows:
 - (a) if input $x = 1$ then x is a literal of the conjunction
 - (b) if input $x = 0$ then \bar{x} is a literal of the conjunction
 - (c) Repeat for all inputs
3. repeat for all rows where the output is '1'.
4. A Boolean equation for the output is the disjunction of all the conjuncts.

Example: A behavioural description is given by:

Entity Definition :



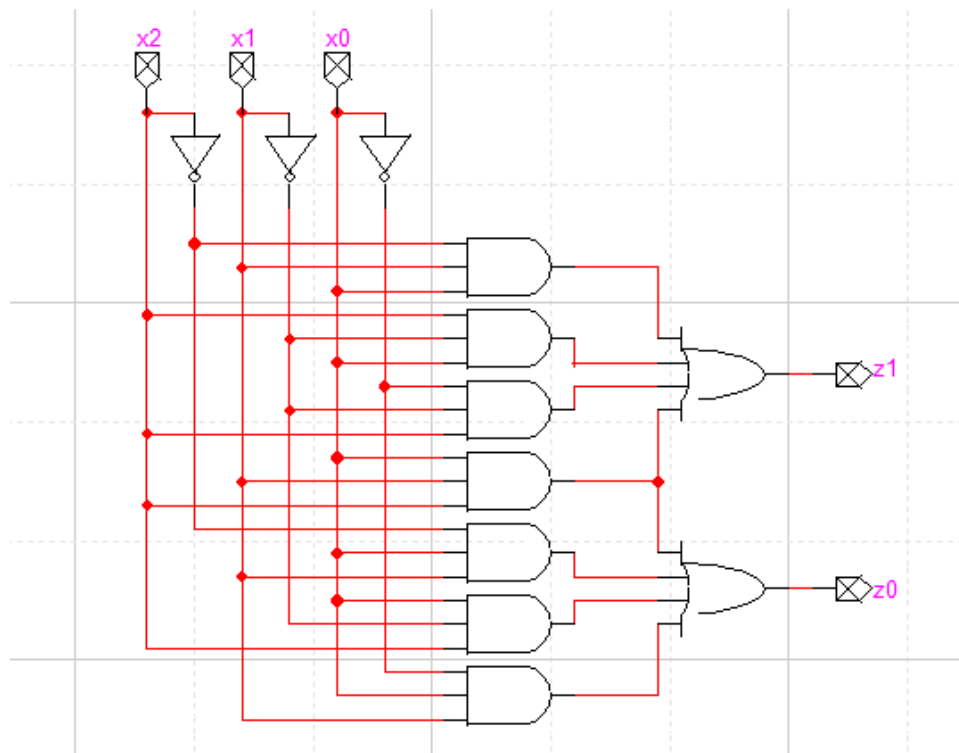
Functional Specification :

FUNCTION TABLE					CONJUNCTS	
x2	x1	x0	z1	z0	conjuncts for z1	conjuncts for z0
0	0	0	0	0		
0	0	1	0	1		$\overline{x2} \cdot \overline{x1} \cdot x0$
0	1	0	0	1		$\overline{x2} \cdot x1 \cdot \overline{x0}$
0	1	1	1	0	$\overline{x2} \cdot x1 \cdot x0$	
1	0	0	0	1		$x2 \cdot \overline{x1} \cdot \overline{x0}$
1	0	1	1	0	$x2 \cdot \overline{x1} \cdot x0$	
1	1	0	1	0	$x2 \cdot x1 \cdot \overline{x0}$	
1	1	1	1	1	$x2 \cdot x1 \cdot x0$	$x2 \cdot x1 \cdot x0$

The Boolean equations are obtained by taking the disjunction of the conjuncts in each column of the table:

- $z0 = \overline{x2} \cdot \overline{x1} \cdot x0 + \overline{x2} \cdot x1 \cdot \overline{x0} + x2 \cdot \overline{x1} \cdot \overline{x0} + x2 \cdot x1 \cdot x0$
- $z1 = \overline{x2} \cdot x1 \cdot x0 + x2 \cdot \overline{x1} \cdot x0 + x2 \cdot x1 \cdot \overline{x0} + x2 \cdot x1 \cdot x0$

The circuit obtained from this pair of Boolean equations is given on the next page.



19.1 Functional Equivalency

An important aspect of digital design is finding the “best” implementation of a behavioral description. A good design is one that optimizes any or all of the following factors:

- **Performance:** Minimizing the propagation delay of a circuit provides the circuit with the quickest response time.
- **Cost:** The number of gates in the circuit affects its cost. Therefore designs that minimize the number of gates can be more economical.
- **Simplicity:** Gates are packaged in groups called “chips.” Reducing the number of chips reduces not only fabrication costs, but also simplifies fabrication.
- **Availability:** An existing inventory of gates may not provide the gates in a given design. Therefore an alternate design may be needed that uses only gates in the available inventory.

Fundamental to the determination of different implementations of a given behavioral description is that it can be realized by different circuits. This is because two circuits are equivalent if they have the same behavioural descriptions. If a functional specification is given by one or more Boolean equations, then different but equivalent circuits can be obtained if there exist transformation rules that change a given Boolean expression into a different but equivalent one.

Finding equivalent gate-level circuits consists of applying the laws of Boolean algebra:

1. Obtain a function specification as a Boolean expression.
2. Transform the expression using the laws of Boolean Algebra
3. Construct the schematic for the derived Boolean expression.

19.2 Laws of Boolean Algebra Important to Digital Design**C Notation**

LAW		DUAL
Identity:	$x \parallel 0 = x$	$x \&\& 1 = x$
Complement:	$x \parallel ! x = 1$	$x \&\& ! x = 0$
Commutative:	$x \parallel y = y \parallel x$	$x \&\& y = y \&\& x$
Associative:	$x \parallel (y \parallel z) = x \parallel (y \parallel z)$	$x \&\& (y \&\& z) = (x \&\& y) \&\& z$
Absorption:	$x \parallel (x \&\& y) = x$	$x \&\& (x \parallel y) = x$
Idempotent:	$x \parallel x = x;$	$x \&\& x = x$
Involution:	$! (! x) = x$	
De Morgan:	$! (x \parallel y) = ! x \&\& ! y$	$! (x \&\& y) = ! x \parallel ! y$
Dominance:	$x \parallel 1 = 1$	$x \&\& 0 = 0$
Distributive:	$x \&\& (y \parallel z) = (x \&\& y) \parallel (x \&\& z)$	$x \parallel (y \&\& z) = (x \parallel y) \&\& (x \parallel z)$

Traditional Notation

LAW		DUAL
Identity:	$x + 0 = x$	$x \cdot 1 = x$
Complement:	$x + \bar{x} = 1$	$x \cdot \bar{x} = 0$
Commutative:	$x + y = y + x$	$x \cdot y = y \cdot x$
Associative:	$x + (y + z) = x + (y + z)$	$x \cdot (y \cdot z) = (x \cdot y) \cdot z$
Absorption:	$x + (x \cdot y) = x$	$x \cdot (x + y) = x$
Idempotent:	$x + x = x;$	$x \cdot x = x$
Involution:	$\overline{(\bar{x})} = x$	
De Morgan:	$\overline{(x + y)} = \bar{x} \cdot \bar{y}$	$\overline{(x \cdot y)} = \bar{x} + \bar{y}$
Dominance:	$x + 1 = 1$	$x \cdot 0 = 0$
Distributive:	$x \cdot (y + z) = (x \cdot y) + (x \cdot z)$	$x + (y \cdot z) = (x + y) \cdot (x + z)$

19.2.1 The Duality Principle

Boolean algebras possess a very important property that “expands” the ways in which expressions can be manipulated while still preserving equivalence. That property is the “Principle of Duality”.

The *Dual* of a function f , denoted by f_D is obtained as follows:

1. Replace all ‘0’s by ‘1’s and all ‘1’s by ‘0’s.
2. Replace all ‘AND’s by ‘OR’s and all ‘OR’s by ‘AND’s

The Principle: Let f be a Boolean function. Whatever is a true statement about f is also a true statement about f_D

The usefulness of the Principle is that for every equivalence between two Boolean expressions, there is an equivalence between their respective duals. For example, consider the properties and laws of Boolean algebra that were introduced previously. Except for the Involution Law, there are two versions. Inspection of the two versions of each property and law reveals that one is the dual of the other. Therefore for every property or law there is an equivalent dual property or law.