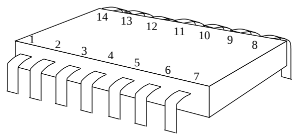
Chapter 2

Realizing Logic in Hardware

The examples in this chapter and part of the next are based on early-generation integrated circuits from the *74LS TTL* (low-power Schottky transistor-transitor logic) family of *integrated circuit* (*IC*) components. This family has come to be known as *SSI*, which stands for "small scale integration," because ICs contain relatively few transistors compared to later generation packages.

SSI components or their counterparts are still available and are occasionally used as "glue" logic for composing larger-scale devices. That is not why we are looking at them, though. They are ideal for understanding the extremely important concept of *mixed logic*. While the benefits of knowing about these older technologies grow increasingly marginal, the concepts we employ them to introduce are not; to the contrary, these concepts grow increasingly important as higher capacity technologies compel designers to work at ever higher levels of abstraction.

A typical early SSI device is packaged in a *dual in-line package* (DIP) configuration illustrated below.



Operating power enters the chip through two pins, Vcc and Gnd, usually at opposite corners of the chip (e.g. pins 7 and 14). A chip contains one or more independent logic *gates*.

2.1 Mixed Logic: Realizing *and*, *or* and *not*

We may represent logical truth by *either* of the two voltage levels in a digital electronic device. If we always bear in mind that the representation is the *de-*

signer's choice, a powerful and beautiful design method emerges. If, on the other hand, we permit the hardware to dictate our choice, we relinquish flexibility and make the task of design more rigid and difficult.

The important point is that these binary devices conform to a particularly useful conceptual view—as formalized by boolean algebra—and *it is vital to take advantage of this conceptual view as freely as possible.* Of course, this by no means new to a computer scientist, who has been taught from the very outset to:

> *Principle of Abstraction:.* Defer concrete representation decisions long as is practical.

A switch may be two-valued and it may seem intuitive, in some sense, to say "*up* means 1 and *down* means 0." But the designer <u>should not</u> commit to that correspondence arbitrarily. The *up* position can just as well represent 0, and that may—indeed, certainly will—turn out to be the better choice in some circumstances.

Let us now undertake the development of clear and systematic ways of building and describing circuits for logic expressions. Efforts at solving digital problems yield logic equations and logical structures; the hardware must faithfully embody those equations and those structures. Furthermore, we certainly wish the documentation of our hardware (our *circuit diagrams*) to convey the spirit of the solution to the original problem. We would like to be able to readily see that the implementation we have designed satisfies the intended specification.

The foregoing thoughts suggest some criteria for drafting methods:

(a) We wish to synthesize (create) a physical realization of any logic expression directly from the logic, in a straightforware, natural, and rigorous manner.

(b) We wish to be able to analyze (pick apart) a physical realization and directly recover the logical expression.

These are strong conditions; many digital drafting and construction techniques in use today to not meet them. The conditions require that the circuit diagram clearly and fully display *both* the logic *and* the hardware. This goal has several implications:

(i) The drafting notation should represent the boolean expressions in *and*, *or* and *not*—the natural way we develop our logic.

(ii) The correspondence between logical value (T or F) and its voltage counterpart (H and L) should be evident.

(iii) The notation should clearly identify each physical device in the circuit.

The key to satisfying these requirements is a representation called *mixed logic*. This notation was first published in a coherent form in 1971,[1] although

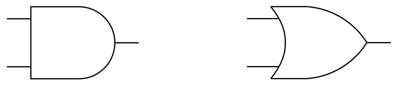
[1] P. M. Kintner, Mixed logic: a tool for design simplification. *Computer Design*, August 1971, pp. 55–60; and F. Prosser and D. Winkel, "Mixed logic leads to maximum clarity with minimum hardware. *Computer Design*, May 1977, 111–117.

mixed logic was used as early as 1957 in the Philco TRANSAC computers, and the technique is likely older than that.

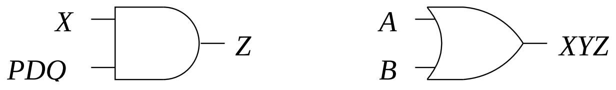
We shall develop mixed-logic methodology carefully, since the principle is vital to clear, top-down design and has far reaching applications beyond gate-level implementation. Since we will eventually move beyond designing at *and-or-not* levels however, it is perhaps most important to remember the larger lesson of mixed logic: that the purpose of abstraction is to serve the conceptual needs of the designer, rather than to impose an arbitrary uniformity on the design object.

2.1.1 Mixed Logic

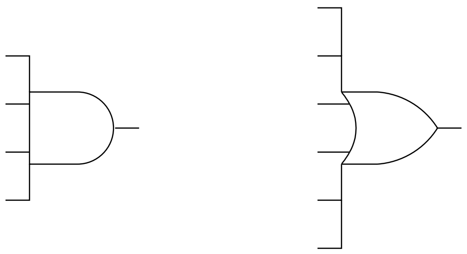
Showing the logic. We choose a unique symbol for each of the logic/boolean operators. The shapes below traditionally represent *and* and *or*; we shall discuss *not* and some other boolean functions later.



Whenever we see these shapes, we know that we are representing a logical *and* (left) or a logical *or* function (right). Furthermore, each and every *and* and *or* operation in our original expression will appear in the circuit diagram as the corresponding shape. For example, the notations



represent equations $X = X \cdot PDQ$ and $XYZ = A + B$. For convenience, we shall allow our schematic symbols to have more than two inputs. Below, for example shows an *and* symbols with 4 inputs and an *or* symbol with and 6 inputs.



Logic Conventions. There are two logic values, T and F. There are two voltage levels, H and L, So two useful possibilites exist:

(a) We could use H to represent T (leaving L to represent F). This interpretation is called *positive logic.*

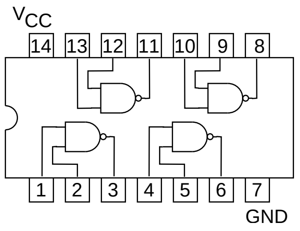
(b) The alternative is to use L to represent T (leaving H to represent F). This interpretation is called *negative logic*.

If one choses to impose a single interpretation, positive logic for instance, across an entire circuit we would call this a positive *logic convention* or a negative logic convention for that circuit.

A *mixed logic convention*—which is used throughout this book—allows us to apply either positive or a negative logic at any point in a design, as we desire. Although this might seem ambiguous, somehow, on first thought, the opposite is true in the extreme. The advantages of mixed logic are immediately apparent, as you will soon see, and leave one to wonder why any other mode of interpretation would be considered.

And yet, the prevalent interpretation is a uniform convention, most often positive logic. While this may be regretable, it is not really wrong, and since it is pervasive, we must learn to live with and accomodate it. But we do not need to accept or conform to its practice.

Showing the circuit. In circuit schematics, the graphical symbols represent physical devices. Throughout this section, the devices involved are logic gates like those discussed in the first chapter. For illustration purposes we shall use gates from the SSI 74LS family. Our first example is the first chip in this family, the the 74LS00 Quad Two-Input Nand Gate chip. The 14-pin dual in-line 74LS00 chip contains four *nand* gates, one of whose voltage tables is shown to the right of the chip schematic, below



A	B	A *nand* B
L	L	H
L	H	H
H	L	H
H	H	L

A	B	$A \wedge B$
F	F	F
F	T	F
T	F	F
T	T	T

Section **??** contains a discussion of the physical realization of a TTL *nand* device, of which the 74LS00 is an example. Chapter **??** gives more detail about the transistor-level behavior of this device. The logical function of one 74LS00 gate is shown to the far right, above. As discussed earlier, this functionality is given by the shape of the symbol used to document the gate.

The name "*nand*" commonly used for this device stems from the view that it implements a *not-and* function in *postive logic*; recall this means an interpretation of voltage under which H always means T. We shall not accept this conventional interpretation, but since it is prevalent, we must learn to co-exist with it. The 74LS00 chip contains four *not-and* gates only if one stipulates from the begining that H *always* means *true* and L *always* means *false*.

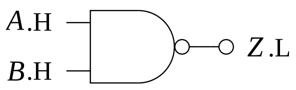
The interpretation of voltage is simply that: an interpretation. Above all, an interpretation should serve the purpose of clarity. It certainly doesn't matter to the electrons whether they represent 0 or 1. We are not much interested in expressing logic in terms of *not-and*s, and shall not be forced into it by adopting positive logic.

Instead, we shall interpret the "bubbles" on the gate symbol to indicate the *polarity* of the logic, that is, whether

— H = T at that point in the circuit—in which case there is no bubble, or

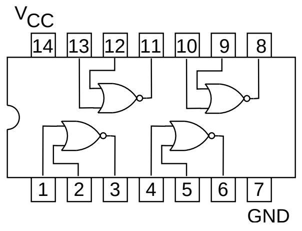
o–o L = *T* at that point in the circuit—in which case there is a bubble.

Thus we shall read the symbol



as an *and* gate with positive logic inputs and a negative logic output.

Another early SSI device is the 74LS02 Quad Two-input Nor Gate,



A	*B*	*AnorB*
L	L	H
L	H	L
H	L	L
H	H	L

A	*B*	$A \vee B$
F	F	F
F	T	T
T	F	T
T	T	T

Again, the name "Nor" comes from the historical, positive-logic interpretation of the device as a *not-or* gate. The mixed logic interpretation, given by the shape of the symbol, is that it is an *or* gate with positive-logic inputs and negative logic output.

The 74LS00 is an *or* gate, too.

The logical function of a device follows from our choice of polarity for its inputs and outputs. For example, if we elect to interpret the inputs of an 'LS00 gate as *active-low* (i.e., T = L), then we derive

voltage

A	*B*	*Z*
L	L	H
L	H	H
H	L	H
H	H	L

⇒

logic

A.H	*B*.H	*Z*.L
0	0	0
0	1	0
1	0	0
1	1	1

⇒

logic

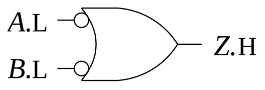
A.L	*B*.L	*Z*.H
1	1	1
1	0	1
0	1	1
0	0	0

Placing the truth table on the right into canonical form, we get

logic

A.L	*B*.L	*Z*.H
0	0	0
0	1	1
1	0	1
1	1	1

⇒ $Z.\mathrm{H} = A.\mathrm{L} \vee B.\mathrm{L}$ ⇒



So a 74LS00 is also an *or* gate with low-active inputs and high-active output.

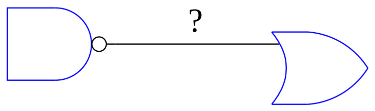
While it may be logically valid to say that a *nand* gate can also be seen as an "*orn*" gate from an application of DeMorgan's Law,

$$\overline{A \cdot B} = \overline{A} + \overline{B},$$

it is merely a rationalization from the mixed-logic perspective We would not choose to engage logical manipulations in so unmeaningful a way *except* to rectify mixed logic co-exist with positive-logic should it becomes absolutely necessary. It sometimes does become necessary, for instance, when one needs to use a data sheet or explain a circuit to someone unfortunate enough to have learned circuit design the hard way.

2.2 Implementing *not* with a wire.

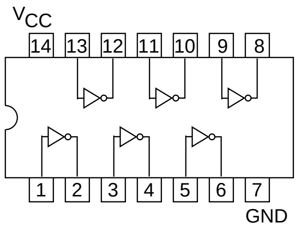
What can we make of the situation when a negative-logic output is connected to a postive-logic input?



The line is understood to represent an electrical connection between the two points, a *wire*, and this conductive connection makes the voltage, V equal at every point on the wire, hence on the line. The bubble says only that we have changed the interpretation:

V	V.H	V.L
L	0	1
H	1	0

2.2.1 The 74LS04 Inverter



voltage	
IN	*OUT*
L	H
H	L

voltage	
IN.H	*OUT*.L
0	0
1	1

voltage	
IN.L	*OUT*.H
1	1
0	0

$\boxed{Z(A,B) = \overline{A \wedge B}}$

A.H	B.H	Z.H
0	0	1
0	1	1
1	0	1
1	1	0

$\boxed{Z(A,B) = A \supset B}$

A.H	B.L	Z.H
0	0	1
0	1	1
1	0	0
1	1	1

$\boxed{Z(A,B) = B \supset A}$

A.L	B.H	Z.H
0	0	1
0	1	0
1	0	1
1	1	1

$\boxed{Z(A,B) = A \vee B}$

A.L	B.L	Z.H
0	0	0
0	1	1
1	0	1
1	1	1

$\boxed{Z(A,B) = A \wedge B}$

A.H	B.H	Z.L
0	0	0
0	1	0
1	0	0
1	1	1

$\boxed{Z(A,B) = \overline{A \wedge B}}$

A.H	B.H	Z.H
0	0	0
0	1	0
1	0	0
1	1	1

Figure 2.1: Six functions of the 74LS00 gate, expressed in terms of *and*, *or*, *not* and *implies*.