NAND Rules The World

Paul S. Wang, Sofpower.com

May 15, 2023

Digital computers are logic machines. They use bits to store information. A bit is nothing but a switch with two states, on and off. In computer speak, we use 1 to represent the on state, 0 the off state. In logic speak, 1 is called true, and 0 false. Combining the two, a bit can either be in the state on/1/true or in the state off/0/false. In other words, inside the computer 1 means true and 0 means false.

A CPU is a thumbnail-size chip containing *Integrated Circuits* (IC) formed with transistors and capacitors. Modern CPUs are extremely complicated using VLSI (Very Large-Scale Integration) containing many basic units called *logic gates*. A logic gate takes truth value inputs and produces truth value outputs. Computations on truth values have simple rules specified by Boolean algebra.

This article shall look under the hood and reveal the heart of the CPU built with logic gates. Among the seven logic gates, `NAND` is the most outstanding and, in fact, a universal building block. We shall explain in simple and easy-to-understand terms.

This post is part of our *Computational Thinking* (CT) blog where you can find many other interesting and useful articles.

Digital Electronic Circuits

Modern computers process digital signals represented by the presence (1) or absence (0) of an electric current or voltage. Digital electronic circuits process signals represented by 0s and 1s, receiving input truth values, performing logic operations on them, then producing output truth values. CPU registers hold input signals as well as store output signals.

Treating 1 as *true* and 0 as *false*, the basic logical operations on the input *truth values*, `A` and `B`, are as follows.

- `(A AND B)` is true only when `A` and `B` are both true.
- `(A OR B)` is true if at least one of `A` and `B` is true.
- `(NOT A)` is true if `A` is false and is false if `A` is true.
- `(A XOR B)` is true if only one of `A` and `B` is true.
- `(A NAND B)` is false only when both `A` and `B` are true.
- `(A NOR B)` is false if at least one of `A` and `B` is true.
- `(A XNOR B)` is false if only one of `A` and `B` is true.

We see seven logic operators listed. The complete behavior of each logic operator can be described by a *truth table* specifying the output for all possible input values.

Table 1: Truth Tables

A	B	A AND B	A OR B	A XOR B
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Table 1 shows the truth tables for `AND`, `OR`, and `XOR`. Try it yourself and write down the truth tables for the other logical operators.

Here is an advise to computational thinkers: *Be careful about the precise meaning of "and", "or", "not", and "nor". Their meaning as logic operators may not be the same as in day-to-day usage. Guard against their imprecise use when receiving communication from others. Better be careful than sorry.*

For example, the menu in a restaurant may state "mashed potatoes, French fries, or steamed broccoli". It usually means you can choose only one. Whereas, the job requirement "a Bachelor or Master degree" means either or both.

Logic Gates

A *gate* is a basic building block in integrated circuits constructed with CMOS (Complementary Metal–Oxide–Semiconductor) or similar technologies. A *logic gate* implements one of the basic logic operations as we have just described. A logic gate takes one or two inputs and produces a single output. The seven basic gates are as follows. Each gate, except the `NOT` gate, takes two inputs.

- `AND` gate—Outputs 1 only if both inputs are 1. Otherwise outputs 0.
- `OR` gate—Outputs 0 only if both inputs are 0. Otherwise outputs 1.
- `NOT` gate—Outputs the opposite of the input.
- `XOR` (exclusive `OR`) gate—Outputs 1 only if the two inputs are different. Otherwise outputs 0.
- `NAND`, `NOR`, or `XNOR` gate—Outputs the opposite of the corresponding `AND`, `OR`, or `XOR` gate.

Standardized graphical symbols can be used to design simple digital circuits (Figure 1). Note that a little circle at the tip of a gate indicates an

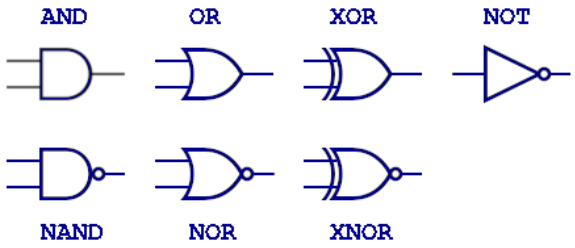

Figure 1: Distinctive Gate Symbols

inversion, turning a 1 to 0 and a 0 to 1.

More complicated digital circuits are built by wiring together these logic gates. Pulse signals, typically produced by a crystal oscillator, provide a processing rhythm. The next processing step won't start until a new clock pulse arrives. The pulse interval is needed for all digital signal transitions to complete and the circuit components to settle into their new states. Typical clock rates for modern CPUs are in the GHz range.

Building CPUs with Logic Gates

Computations in a CPU are done with clever combinations of logic gates. We can get a taste of this by looking at simple arithmetic operations such as adding two binary numbers. For example, adding 011 (the number three) to 001 (the number one) must result in 100 (the number four). A quick Web search will land you more information on binary numbers.

Half Adder

An *adder* circuit provides the ability to add binary numbers which in turn can be used repeatedly to perform multiplication of binary numbers. But first, we will build a *half adder*. A *half adder* takes two input bits `A` and `B` and outputs a sum bit `S` and a carry bit `C` (carry to the next bit position) (Figure 2). Here we used an `XOR` gate to produce the sum bit and an `AND`

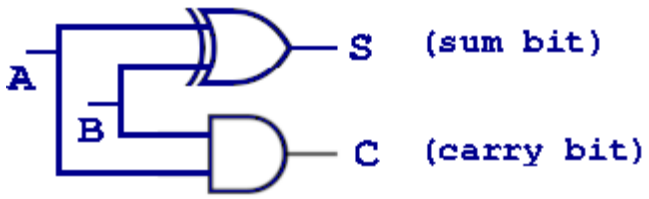

Figure 2: A Half Adder

gate to compute the carry bit. Table 2 lists the input and output values for the half adder. The half adder shows how logic gates are used to add two

Table 2: Half Adder I/O

A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

bits.

Full Adder

The half adder can be used to build a more capable *full adder* that can take three input bits A, B, and C_{in} (carry in from the previous bit position) and will produce a sum bit S_{out} and a carry bit C_{out}. A full adder consists two half adders and an extra `OR` gate (Figure 3).

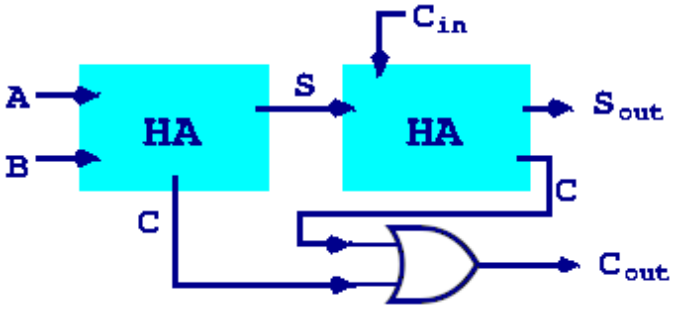

Figure 3: A Full Adder

The Bottom-Up Approach

Building simple components first then combine them into more complicated components is a solution method that often works well. This leads to the

> **CT concept–*bottom-up solution***: *Consider the bottom-up approach for problem solving. Start by solving the small and most basic problems. Then combine those solutions to solve bigger problems.*

The toy LEGO™ is a vivid illustration of the bottom-up approach—using tiny blocks to build anything you can imagine.

Here, we first built a half adder. Then we combine two half adders to build a full adder. Then we can combine a number of full adders to solve the more complicated problem of adding two n-bit numbers.

NAND Is A Universal Gate

A logic gate is universal if it alone can be used to build all other logic gates. The `NAND` and `NOR` gates are both universal. Because `NAND` and `NOR` gates are

economical and easier to fabricate, they are the basic gates used in all IC (integrated circuit) digital logic families. For many reasons, the `NAND` gate is used in most CPU designs.

To give you an idea, here is how `NAND` can be used to build an `OR` gate (Figure 4). The input `A` is fed as the two inputs to the first `NAND` gate

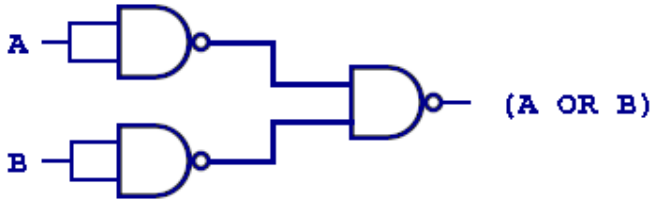

Figure 4: An OR Gate Using NAND Gates

and similarly `B` to the second `NAND` gate. The third `NAND` gate produces the correct output for `A OR B`. It may be interesting for the reader to verify that this implementation works for all possible values of `A` and `B`.

The take away here is: modern CPUs use `NAND` to implement logic gates that in turn can build all other functions.

One Good Turn Deserves Another

Feedback Loop

Computer hardware design is a complicated engineering activity. Software tools are used to help automate various phases of the design, testing, synthesis, and fabrication of integrated circuits. Better hardware leads to faster software which leads to even better hardware, a wonderful virtuous cycle indeed (Figure 5). *Take advantage of new knowledge, new tools, new circumstances and apply them everywhere possible. Don't miss the chance to create a positive feedback loop!*

Iteration

A more general concept is *iteration*—repeated application of the same procedure. The power of iteration cannot be overstated. Let's revisit our favorite example.

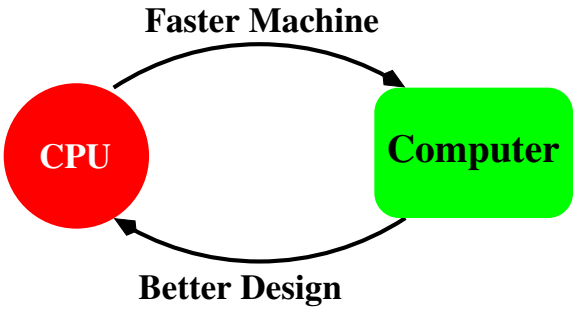

Figure 5: A Virtuous Cycle

Iteration has led to the invention of the *polymerase chain reaction* (PCR), a technique in molecular biology to generate thousands to millions of copies of a particular DNA sequence.

Developed by Dr. Kary Mullis in 1983, PCR is now indispensable in medical and biological research and applications, including DNA testing and genetic fingerprinting. The impact of automated PCR is huge and far-reaching. Mullis was awarded the 1993 Nobel Prize in Chemistry for his part in the invention of PCR.

In recounting his invention, Kary Mullis wrote in his book *Dancing Naked in the Mind Field*:

> *I knew computer programming, and from that I understood the power of a reiterative mathematical procedure. That's where you apply some process to a starting number to obtain a new number, and then you apply the same process to the new number, and so on. If the process is multiplication by two, then the result of many cycles is an exponential increase in the value of the original number: 2 becomes 4 becomes 8 becomes 16 becomes 32 and so on.*
>
> *If I could arrange for a short synthetic piece of DNA to find a particular sequence and then start a process whereby that sequence would reproduce itself over and over, then I would be close to solving my problem.*

At the time of the invention, the "polymerase" and other related DNA duplication techniques were already known. It was the "chain reaction" part that was missing. Well, we have Dr. Mullis and his computational thinking to thank for the invention. And what a significant invention! The *New*

York Times described it as "highly original and significant, virtually dividing biology into the two epochs of before P.C.R. and after P.C.R."

Duplicating DNA sequences is key to modern use of DNA technologies. For instance, COVID-19 virus tests depend on PCR.

Finally

Looking under the hood of a modern CPU, we found tens of billions of transistors forming logic gates (mostly `NAND`) that perform computations at lightening speed. All this is made possible by using computer software to design next generation CPUs—a virtuous cycle that is still on-going.

Now, imagine a computer running to perform any task such as playing a video game or navigating a map. It all comes down to a dizzying blur of numerous `NAND` operations in the CPU! And the task is done. Isn't this a

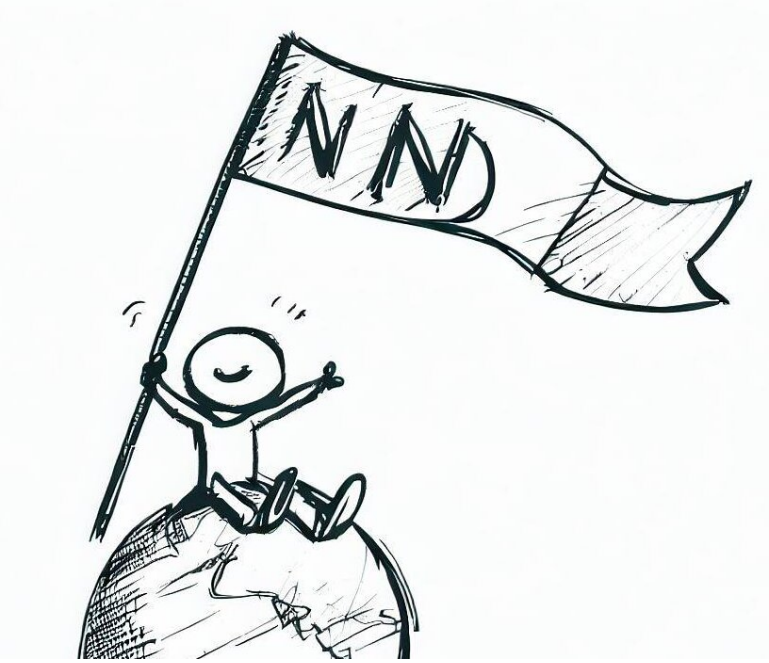

Figure 6: NAND Rules the World

concrete example of the power of the bottom-up problem solving method—combining `NAND` gates to do almost anything? Thus, in a real sense, *NAND Rules The World* (Figure 6). How mysteriously wonderful.

Bonus: N-Bit Adder

For motivated readers, here is a bonus: how full adders are combined to add two n-bit binary numbers (unsigned or two's complement). Basically, we combine n full adders, each taking its Cin from the $Cout$ of the adder to its right. Here we see how this works for input `X` ($X_nX_{n-1}...X_0$) and `Y` ($Y_nY_{n-1}...Y_0$) producing a result `S` ($S_nS_{n-1}...S_0$) (Figure 7). Because the

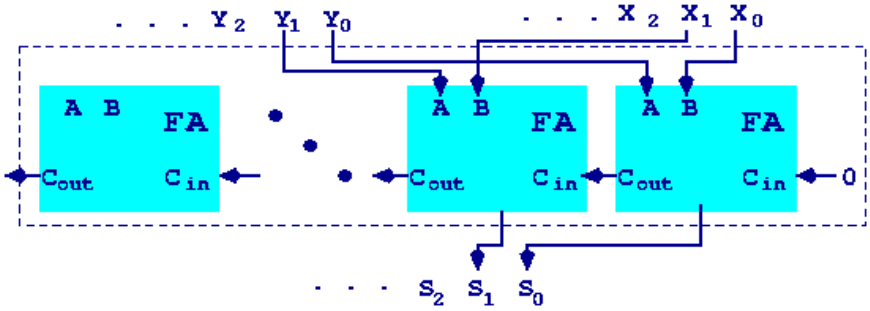

Figure 7: N-Bit Adder

carry bit may need to propagate from the rightmost adder all the way to the leftmost, a delay proportional to n is needed until the last output bit settles down. Such an adder is known as a *ripple adder*. More complicated adders can reduce the delay and make binary addition faster, especially if addition is performed as a subprocess of multiplication.