Ternarylogic

LOGIC, SWITCHES AND COMPUTER CIRCUITS

Logic is the part of a computer that makes it cerebral, rather than just storing data. Logic appears to do something that otherwise only humans are supposed to do: perform some form of reasoning that leads to a result. That is what makes computers so smart, of course.

Human logic

Logic is generally considered to mean human logic and from ancient time is associated with human reasoning. Reasoning leads to a conclusion or a result. Formal logic leads to a valid formal conclusion or result.

Computer logic

Computer logic, a term often used in common language, refers generally to an algebra for manipulating variables defined in a truth space (like False or True). Functions in Boolean algebra can be expressed in truth tables that provide an output value as a result of input values in a truth space.

And this is where we are forced to leave "logic" and move to switching. Because in reality, computers do **not** perform logic. Computers are state dependent switching machines. They change their switching states at an incredible speed, and are enabled to process signals and to giving the impression of intelligence or reasoning. Computers are deterministic signal switching machines, with (currently) no capability for intuition or awareness.

Signals are merely physical phenomena. Signals by themselves do not mean anything unless given some meaning by …. humans.

This also tells us that what we consider to be "intelligence" in a machine is actually a combination of utterly predictable steps that can easily be performed by a machine.

Switching algebra

Switching is a physical operation and is performed by a device or a material in a first state and puts a property of the device or material in a second state different from the first state. The change in state can be detected, preferably automatically, by another device or material. This allows a state of a device to be propagated to a next device, which is required for a computer to perform steps that are combinations of smaller steps.

The state propagation generally takes place by electrical signals in electronic computer circuits. However, biological state machines are also well known wherein a state is transported from one "device" to another "device" by a material.

The switching performance of individual switching devices, certainly the binary ones, is extremely simple. That is, based on some input state, or perhaps two input states, an output state is either ON or OFF. A two input/ 1 output switching function is generally identified as a "logic function." A single input/single output switching function is generally identified as an "inverter."

In order to describe (and design) a complex switching machine that uses many different individual devices, some descriptive language would be useful. As the language describes switching states, one may call that language for instance a "switching algebra," wherein for instance each individual switching device is represented as a term in a switching algebraic expression. Examples of different binary adder switching designs can be viewed at this website.

The welcome discovery/invention by Claude Shannon in his Master's Thesis was that Boolean Algebra is an appropriate tool to be used as a Switching Algebra.

It is again noted that machines do not reason, they merely switch. So, while we use the term logic, we really mean switching.

Realization vs. Switching Algebra

The fact that the description of the input and output states is extremely simple (ON and OFF, 0 or 1) and is represented by a simple algebraic statement does not mean that the realization is simple. In many cases the realization of computer devices is not simple at all.

This distinction between switching description and physical realization formed a huge barrier that hindered engineers to efficiently create actual computer circuits. One can, on paper or in a computer simulation program, very well design very complex computers. However, building or realizing these designed machines is an entirely different matter. Well into the 1970s huge bottlenecks existed for insufficiently trained engineers who were unable to translate computer designs into digital integrated circuits.

The solution to that problem is what is now called the Mead & Conway Revolution. Basically, it changed the switching design of a computing machine into a design of a realizable chip. In a way, it removed the barrier that existed between switching design (or logic design) and the physical realization or circuit design. It allowed the creation of realizable logic libraries, from the smallest memory or switching entity to functional entities.

The distinction between realization and switching becomes important, again

The distinction between logic design and switching device has faded in practice. One reason is that in computer design function libraries are used that translate the functional design immediately into a realization design. One example of that is the Field Programmable Gate Array (FPGA), which is supported by huge libraries for instance as provided by companies such as Altera and Xilinx. ARM is an example of a company that provides logic designs (IP) often related to foundry related processes.

Ternarylogic LLC believes that in the future physical **non-binary** switches will be used. Current research focused on new switching materials and phenomena that can potentially replace the binary silicon based switches, indicates that non-binary switching is inherent to most of the considered materials. See for instance at Link1, Link2, Link3 and Link4.

The switching rules for non-binary switches are fundamentally different from the binary switching rules. There is no simple translation from binary switching rules to non-binary rules. Non-binary is not a trivial variant of binary with just some more states. Non-binary switching requires a complete logical redesign of almost any circuit presently in use.

What is Different in Our World to Require Non-binary Switching?

So far, binary switching has served us well, very well actually. Why the need for change?

If one looks in the past of computing, what is striking is that how little is actually done per unit of time by the computer. This is especially true in applications like machine intelligence and image processing.

To arrive at even a very modest result, a computer has to run sometimes for minutes if not hours or even days. For instance in image processing, very intensive arithmetical processing takes place on all pixels in an image, which may contain millions of pixels. Not only that, the computer usually has to perform similar operations on previous images, which may run in up to 30 images per second or more.

The lack of processing speed means that very interesting applications in real time that would form the next generation of real-time useful applications are currently not possible. Some break-through in processing is required. We are not talking about an increase by 50%, but more likely about a factor 10 to 100.

Similar constraints apply to data transmission and data storage.

A Threat and an Opportunity

The threat of a change-over from binary to non-binary has many aspects. One threat is a lack of understanding of basic non-binary switching methods. Imagine a situation, wherein physicists will hand us technology that provides easy to realize and combine non-binary switching. Most design engineers will react by saying: what am I supposed to do with this?

Another threat is the lack of available designs to be realized. Almost every functional binary circuit has to be redesigned. Chaos will ensue.

This, of course, offers also a great opportunity for new companies that have anticipated the above threats and own the Intellectual Property on new non-binary designs. Non-binary

switching, if and when it becomes the leading implementation of computer designs, will be recognized as a true computer industry "inflection point" as described by Andy Grove of Intel. That inflection point is occurring now. While still under the radar for many engineers, and certainly for the general public, researchers all over the world are designing and developing non-binary switching mechanisms based on novel uses of materials.

When the actual non-binary technology appears in switching products, it will be one of the most disruptive technologies that we have seen in a while. It will put the current industrial infrastructure in an upheaval that will remind us of the 20 year period that created "Silicon Valley." One difference will be that we know, at least partially, what we are looking for and what type of functionality is required.

Non-binary Virtual Valley

The emergence of non-binary switching may lead to a new technology community which may not be in California or even in the USA. A new Non-binary Valley will most likely not be an imitation of Silicon Valley, but a Virtual Valley: a conglomerate of research institutions, risk capital, IP filings and application firms spread over a broad geographic area that will create an organized transformation of the electronics industry. It will probably not depend on installed manufacturing capability, but largely on creative intellectual capabilities.

ARCHITECTURE, IMPLEMENTATION AND REALIZATION

The architecture of a computing device is what the (system) programmer sees as functionality. For instance an addition of an addend B to an augend A creates a sum S in an accumulator. For instance, the programming statement may be S=SUM(A,B). The programmer usually works in a decimal system (or in a hexa-decimal system) and is not concerned with how the accumulator works.

The implementation of the accumulator is the logical design of the addition operation. It describes in detail which switching functions have to be applied and how the inputs and outputs of these functions are connected to achieve the result. A computer by itself does not know how to add. It has to be designed or configured to do so. There are different logical designs for achieving a sum of multi-digit numbers. Remember that a computer does not "know" numbers, it only switches signals. An easy to understand adder is the ripple adder, which generates iteratively carry digits that "ripple" through the intermediate results, hence the name. However, this is a relatively slow adder design. Other designs, such as "carry predict" designs are possible. Most of these designs are for binary adders, as explained here. The significance of implementation is commonly ignored to focus on architecture and realization.

The realization of the accumulator is formed by the actual switching devices. A switching device may be an electro-mechanical relay, an electronic tube, a transistor or a CMOS gate. The implementation determines how the realization is applied.

AN EXAMPLE COMPARISON

The difference between binary and non-binary switching tables (and thus switching) is illustrated with the following example.

One commonly used binary function, in arithmetical circuits and in encryption is the XOR function. The XOR function is one of only 16 theoretically possible 2 input/1 output binary switching functions.

The truth table of the binary XOR is provided in the following table:

output *c*		input *b*	
	XOR	0	1
input *a*	0	0	1
	1	1	0

This table shows that if two inputs values of *a* and *b* are identical (both 0 or both 1) then the output *c* has a value 0. If, however, *a* and *b* have different values, (for instance *a* = 0 and *b*=1) then the value of output *c* is 1.

However, the XOR function has additional properties that are of interest:
a) the XOR function is the modulo-2 addition;
b) the XOR function is commutative;
c) the XOR function is associative;
d) you can form a distributive pair of XOR with another function such as the AND;
e) the XOR function is reversible (invertible) and self-reversing;
f) the combination of identical input values/states is always 0;
g) all rows are orthogonal to each other, so are the columns (the Sudoko rule).

There are more than 4 billion theoretically possible 2 input/1 output 4-state switching functions. The truth tables of the following 4-state or 4-valued functions are provided:

c		*b*			
	mod−4	0	1	2	3
	0	0	1	2	3
a	1	1	2	3	0
	2	2	3	0	1
	3	3	0	1	2

c		*b*			
	*rev*4	0	1	2	3
	0	0	3	2	1
a	1	3	2	1	0
	2	2	1	0	3
	3	1	0	3	2

c		*b*			
	*GF*4	0	1	2	3
	0	0	1	2	3
a	1	1	0	3	2
	2	2	3	0	1
	3	3	2	1	0

Mod-4 is the modulo-4 addition. This truth table does not have the property wherein combinations of all identical inputs provide output state 0. Mod-4 is not self-reversing.

Rev4 is a self-reversing function, but is not associative.
GF4 is an addition over a 4-state finite field. It is self-reversing and has the property wherein identical input values/states provide output state 0.

The above already indicates how different working in 4-state switching may be compared to binary switching.

Peter Lablans,

www.ternarylogic.com
For info: admin@ternarylogic.com

Literature:
G.A. Blaauw - Digital System Implementation, Prentice Hall, 1976
Peter Lablans - The Logic of More, Applications in Non-binary Machine Logic, Ternarylogic LLC, 2014.