Dual-Edge Signaling Logic

David R Resnick
Extreme Scale Computing Group, 1420
Center for Computing Research, Sandia National Laboratories

The single-pulse nature of Josephson junction (JJ) logic functions makes logic design harder to do and not at all familiar to pretty much all people who might want to design using JJs—in effect, logic zeros are missing and the logic must be somewhat more complex in implementation to 'get around' that. One result of the single-sided logic is that a constant clock function is needed for most all the individual logic functions and gates in current JJ design styles to work. The resultant clock distribution is thus large and takes a large proportion of the logic energy.

Here is a proposal for a different kind of logic design style and function that may well overcome most of the difficulties mentioned above. *If the logic functionality shown here can actually be brought to life, it would be interesting to assess its impact in area, in energy usage, ease of design, etc*.

A JJ design and implementation issue, that of restricted fan-out, is not addressed here.

The Basic Idea

All logic signals are generated and sent as signal pairs. A 2-in AND gate has two pairs of input signals and a pair of output signals. One of signals in a pair has a pulse when the signal-pair indicates going from a 0 to a 1, and the other signal carries a pulse when the signal-pair indicates going from a 1 to a 0. This means that if the receiver of a signal captures the pulses there is never a need to retransmit the signal; only changes are sent. The state of the signal pair is that of the line carrying the last signal pulse.

At the input of most logic functions it is expected that the 1-going signal of a pair will set a JJ in one direction and the other 0-going signal will set that JJ in the opposite direction (or simply cancel the magnetization in the JJ?). In receiving from a signal pair, there is thus a JJ that functions as a set/clear latch that holds the state of the received signal pair. *Each side of signal pair is also used separately, as needed and along with the JJ latches, to implement the logic function of the gate*.

Extending the Semantics of Boolean Equations

There are four pieces of information needed to describe what is going on with respect to a signal-pair. In the table below, will use 'N' as a generic name to be replaced as needed.

	Symbol	Meaning
1.	N_L	Pulse on the 0→1 side of a signal-pair
2.	N_T	Pulse on the 1→0 side of a signal-pair
3.	N^+	The symbol JJ latch is set in the 0→1 direction
4.	N^-	The symbol JJ latch is set in the 1→0 direction

'L' and 'T' stand for 'Leading' (as in Leading Edge) and 'Trailing' respectively

These symbol extensions are then used in writing the equations that describe the logical functions of logic gates. An example from an AND gate: $O_L = A^+ \cdot B_L + A_L \cdot B^+$

This means that if the A latch is set when a pulse on the B-pair leading edge arrives, generate a pulse to the O (Output) leading-edge signal wire. In addition, a pulse is sent out O_L when the B-latch is set and a pulse on the leading edge of A is received.

Back to the Basic Idea

The leading-, trailing-edge idea is used in developing the output signals of any logic function. But it cannot simply complement the output of the logic that implements the logic function to become the second/trailing-edge signal of an output pair because of the pulse logic. Instead, using a 2-in AND as an example, while the 'leading' side of gate implements the needed logic equation—AND here, the 'trailing' side is generated from other side of the logic inputs, and uses a De Morgan-law form of the generation equation—thus !A OR !B (using '!' for negation).

Outside of some control signals that enable or disable the L and T sides at the same time, to a gate logic designer an L output signal is generated using L side of an input term if the term used as true and the T side of an input term if used in complement form, and ANDs and ORs as indicated in the equation be implemented. The T output signal is generated using the opposite side of the input terms from the L side, and with ORs in place of ANDs and ANDs in place of ORs.

Both the A and the B signal-pairs into the AND must 'set' and 'clear' their respective JJ state latches.

$A_L \rightarrow A^+$
$A_T \rightarrow A^-$
$B_L \rightarrow B^+$
$B_T \rightarrow B^-$

The output equations are:

$O_L = A \cdot B = A^+ \cdot B_L + A_L \cdot B^+$
$O_T = !(A \cdot B) = (!A + !B) = A_T + B_T$

Note that the AND terms in the defining Boolean expand to two terms, one for each possible input order. The pulse arriving first is used in an equation in latch state form; the second as a pulse. In most all cases the logic must be able to handle all possible input orders—thus two terms for this 2-in gate.

Note that if A and B go to zero in succession (adjacent T pulses, one from each pair), that there are two trailing-edge output pulses; this has no logical effect.

Example: 2-in XOR

The basic Boolean equations for an XOR gate driving a signal-pair output are:

$O_L = A \cdot !B + !A \cdot B$
$O_T = A \cdot B + !A \cdot !B$

The equation for the trailing signal is the complement (jiggered De Morgan) of the output for the leading edge side.

The equations turn into the following pair. Each of the AND terms in the above equations turn into a pair of terms as it is assumed here that there is no knowledge of the order of the A and B inputs.

$O_L = A^+ \cdot B_T + B^- \cdot A_L + A^+ \cdot B_L + B^+ \cdot A_T$
$O_T = A^+ \cdot B_L + B^+ \cdot A_L + A^- \cdot B_T + B^- \cdot A_T$

Means What?

There are some interesting things that result from doing this:

- To this point in exploring the design space there is no need for logic inversion (NOT) gates. Inversion is built into the structure. Signal pairs can be connected in both senses, and both pulse and JJ senses are available to use in implementing the logic functions.
- Except for timing issues, state saving, and similar concerns, outputs can be generated directly from the input signals; there is no need for clocking or resetting JJs in logic-only gates. Gates should be able to directly connect together in most cases without clocking. This should greatly reduce the amount of clock fan-out and functions. More on this below.
- In most cases, a single gate type can be used for multiple different logic functions. The AND gate above actually can be used for multiple different logic functions: AND, NAND, OR, NOR, A·!B, A+!B. The XOR gate is also an EQV gate (Equivalence),
- There is more logic function in one of these gates then in previous logic types. One result is that there are more components in the gate—a bigger area. This trades-off in a reduction of the total number of gates needed to implement a specific function in current JJ logic types (as far as the current logic capabilities are known). Another result is that implementation logic is built from only a few basic functions. This means that it may well be possible to have gate-array or standard-cell logic in which a general structure of components is metalized into any of a wide library of gates interconnected as needed. IF this is reasonable, then JJ logic design just became MUCH easier to do.

With respect to the third bullet: The 2-in AND gate can be used in multiple different ways, depending on which side of each signal pair is considered the 'true' side. If the output side of the gate is used so that its L and T senses are reversed (the T side is hooked up to the L side of the gate being driven and the L side connected to the T input side), the result is a NAND gate. If the input and output sides of the gate are reversed then the result is a full OR gate. And if one of the input pair is reversed then the function is A·!B; if the output pair is also then reversed, the function becomes !A+B.

Interconnection

There being no clocking logic in the gate implementations discussed to this point means that logic can be chained and interconnected in multi-level asynchronous blocks. Data flow is thus unconstrained, though there likely needs to be timing computer-aided design tool to verify that signals can be kept in order where that is needed in addition to meeting setup and hold times in the appropriate places. There is then a need for latch/flip-flop functions to maintain order and also to create function blocks that have outputs that are insensitive to changes in input state.

Making Latches and Flip-Flops

For the purposes here, a latch holds a logic level at its output when the latch signal is active, otherwise enabling input data to flow to the gate output. A flip-flop (FF) captures its input signal at a specific (leading- or trailing-) edge of a clocking signal, transferring that data to the FF's output.

For the purposes here a latch simply prevents any following output pulses when the latch signal is active (so between a leading edge latch signal and a following trailing edge). Most full FFs in other logic types are back-to-back latches, which each latch holding data on opposite levels of the clock signal. For this logic, where signals are pulses, a latch or FF is about the simplest of all possible gates (inverters excepted here because there aren't any.

For the equations below, the following pin description:

D: Data input, C: Clock or latch enable signal, O: Output

Not shown again are the JJ input latch equations resulting in D^+ and D^-, which are standard for all dual-edge gates.

Output Equations for a Flip-Flop

$O_L = D^+ \cdot C_L$ The state of the data input is strobed to the output
$O_T = D^- \cdot C_L$

The FF can be used to capture on the trailing edge of the clocking signal by reversing the C input signal connection.

Output Equations for a Latch

$O_L = D_L \cdot C^-$
$O_T = D_T \cdot C^-$

Each latch equation enables the output to follow the input when the latch is simply passing signals thru. If the latch signal is true then the output does not change from what it was before the signal became true; no further pulses are passed while C is true.

Most any gate can be 'upgraded' to include latch or FF functionality by expanding the output equations to include the C terms shown here.

And …

Some logic will have to be initialized at start-up and/or include some sort of master-clear, both for clearing latches and FFs, and also for the states of the JJs in the logic gates. In a lot of cases the logic should be largely self-clearing, for example in a set of pipe stages, so that the clearing network should not be hugely expensive in area or complexity.

Note: Not counting fully differential signaling, there have been 2-wire signal implementations done before for logic functions. One example of this is where the second wire of a pair is used to indicate validity of the first wire; this kind of signal-pair connection has been done for fully asynchronous logic implementations (no clocks at all).

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.