ALU Transistor Level Description

Robert Plachno
December 1999

http://www.elqc.com/RobertPlachno/alu_description.pdf

Table of Contents

1 Summary

This published Arithmetic Logic Unit design combines older NMOS techniques for a CMOS implementation. This documents and explains the techniques used.

Refer to the related papers including the description of the single cycle RISC (Plachno, A True Single Cycle RISC Processor without Pipelining, 2009)and the RISC timing document (Plachno, Processor Design, 1999).

2 Design Concepts

2.1 Origin

This ALU was first used in the American Information Technology 64-bit processor designed in the mid-1980s. This 64-bit wide design was hardware compatible to several widely used processor architectures. The ALU was also used in a smaller embedded processor for a couple of audio product lines during the 1990s. This embedded processor design used a novel timing architecture and was cross assembled to be assembly level compatible with other processors. These chips had high volumes averaging 2.5 million units per month over seven years. This design was successfully produced in high volume.

2.2 Design Technique

These processors used a “datapath” design technique. The datapath circuits are custom designed and laid out in a rectangular array. The control logic is implemented as standard cells and interface to the datapath in metal 1. The data buses run vertically over the datapath in metal 2 while the control lines run horizontally in metal 1. Since the control logic is typically place-and-routed this is called a semi-custom approach. The processor contains EBOX, IBOX, and MBOX datapaths for the execution unit, instruction unit, and memory unit.

While a “datapath” design reflects the layout technique, these designs also are designed at the transistor level to optimize cost, performance, and current dissipation. A lower transistor count typically improves all three.

CMOS designs will dissipate AC current by the basic equation learned in physics class as:

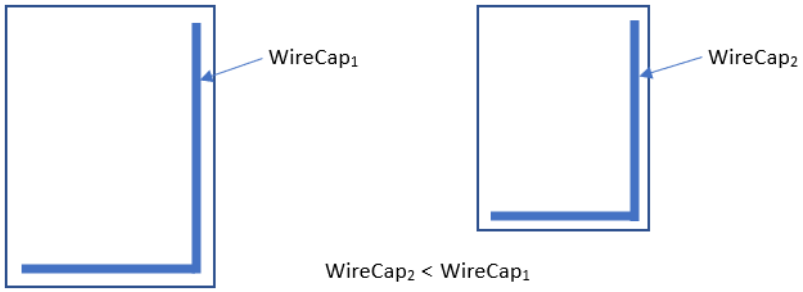
$$i = c * \frac{dv}{dt}$$

where “i” is the current dissipation, “c” is the capacitance of a node, and “dv/dt” is the first derivative of the voltage swing over time. Since CMOS should be transitioning rail to rail between ground and the power supply, (dv) becomes the power supply. (1/dt) is the inverse of the cycle time count (or average frequency). Both a positive and a negative transition is needed to complete the current dissipation from the supply to ground. For example, a low-to-high transition on a CMOS inverter output will charge the load capacitance from the power supply but a high-to-low transition is required to complete the discharge to ground.

Any decrease of the capacitance, voltage, or number of transitions will decrease the current dissipation.

Reducing the transistor count should result in a smaller area. A smaller area equates to lower cost since more die can be arrayed across a wafer. The wafer cost is divided across more gross

die. If two competing companies have designed similar products with the same functionality, then the company with the smaller die size (using the same process technology) will have a lower manufacturing cost.

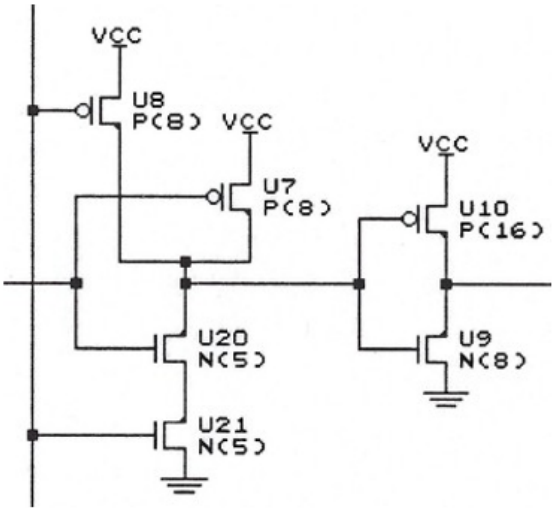


With everything else equal, if a layout can be reduced in area, then the interconnect capacitance is also reduced. A lower node capacitance results in both a lower current dissipation and less delay (higher speed performance). This is an important concept.

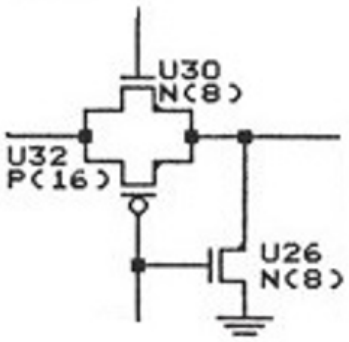
A custom transistor level design can sometimes eliminate logic transitions. For example, a standard cell implementation of an “AND” gate would have two inverting stages. First is a “NAND” gate followed by an “INVERTER”. This standard cell implementation consists of 6 transistors with two transitioning nodes (excluding the input nodes). A mux-logic implementation consists of a transmission gate followed by a pull-down nchan transistor. This mux-logic implementation only has 3 transistors (half of the gate implementation) and has one less transitioning node (also half). The mux-logic implementation is non-inverting while the standard cell implementation requires two inverting nodes to transition rail-to-rail. The mux-logic will have lower current dissipation and it will be faster (if designed correctly).

AND

Logic Gate Implementation
2 transitions, 6 transistors

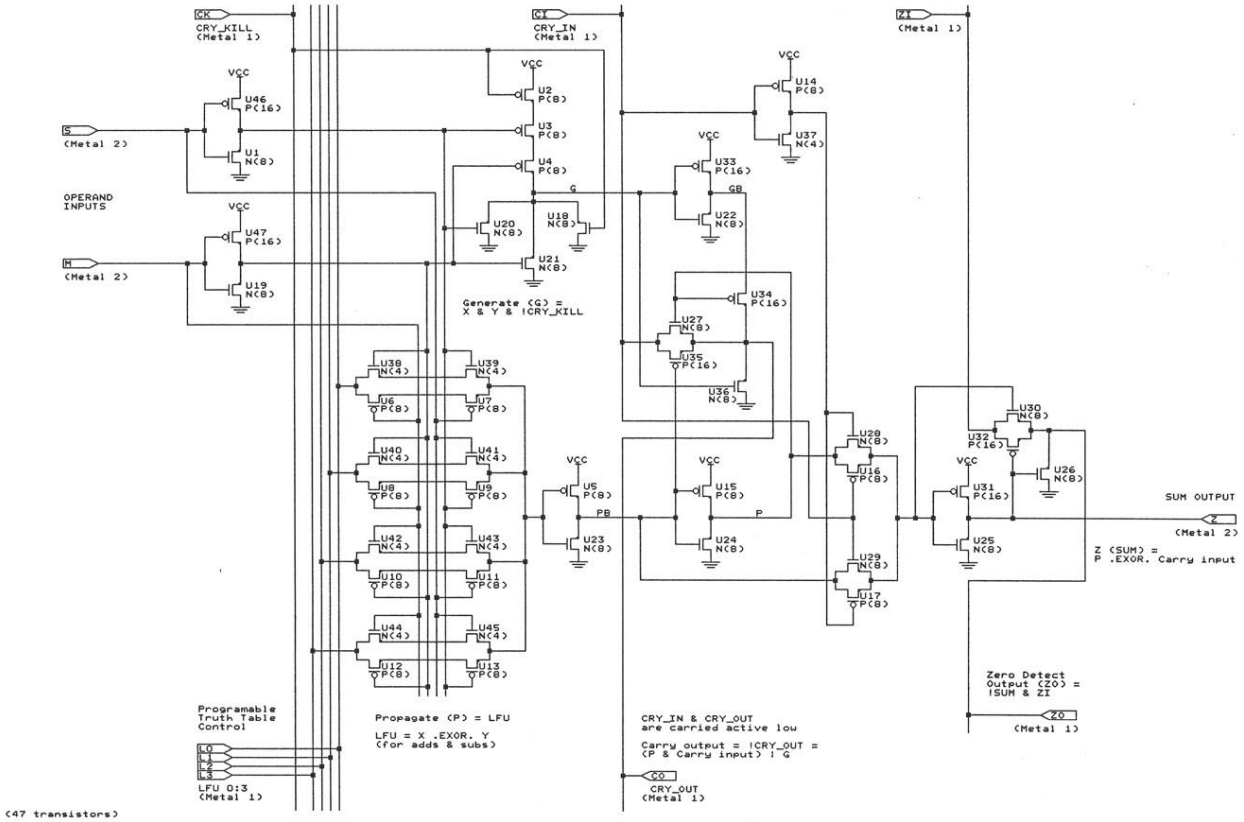


Mux-Logic Implementation
1 transition, 3 transistors



The above schematic for the mux-logic version is from the actual ALU zero detect. Mux-logic is non-buffered so the engineer must be careful how the circuit is driven and how much load it drives. For datapaths the interfaces are contained allowing for mux-logic designs.

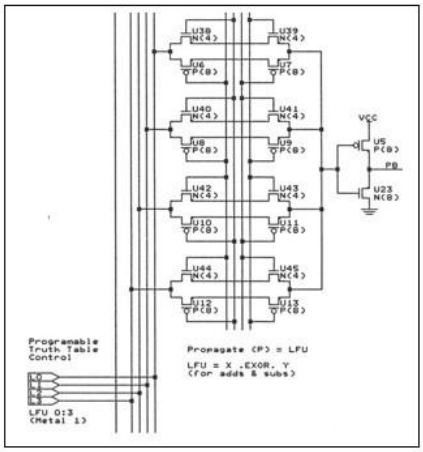
3 Schematic

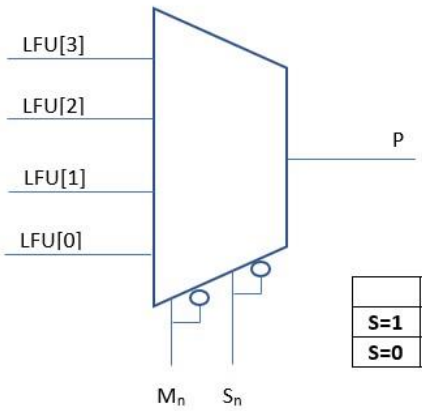


4 LFU

4.1 Mux

The ALU contains an LFU or Logical Function Unit. This LFU is a programable truth table implementation. Since there are only 2 operands for the ALU input, there are only 4 truth table values. These are shown as L3,L2,L1,L0 or LFU[3], LFU[2],LFU[1],LFU[0] in the bottom left of the ALU schematic.





Truth Table

	M=1	M=0
S=1	LFU[3]	LFU[1]
S=0	LFU[2]	LFU[0]

Notice that the operand combinations select the LFU[3:0]. The LFU[3], LFU[2],LFU[1],LFU[0] control is generated by the instruction decode and are bussed through the ALU horizontally in metal 1. These signals need to be well buffered.

The mux is implemented by a pair of nchans and a pair of pchans for each LFU[i] input. This saves on transistors but will add some capacitance. The internal nodes do not need to be tied together since the nchans only pull low and the pchans only pull high.

4.2 LFU Instruction Decode

Not shown but in the instruction decode is the generation of the LFU[3:0] control. This is straight forward logic.

Mnemonic	Opcode	LFU[3]	LFU[2]	LFU[1]	LFU[0]
ADD	0x60	0	1	1	0
ADDC	0x64	0	1	1	0
SBC	0x68	0	1	1	0
CP	0x6C	0	1	1	0
XOR	0x7C	0	1	1	0
AND	0x70	1	0	0	0
TM	0x74	1	0	0	0
OR	0x78	1	1	1	0

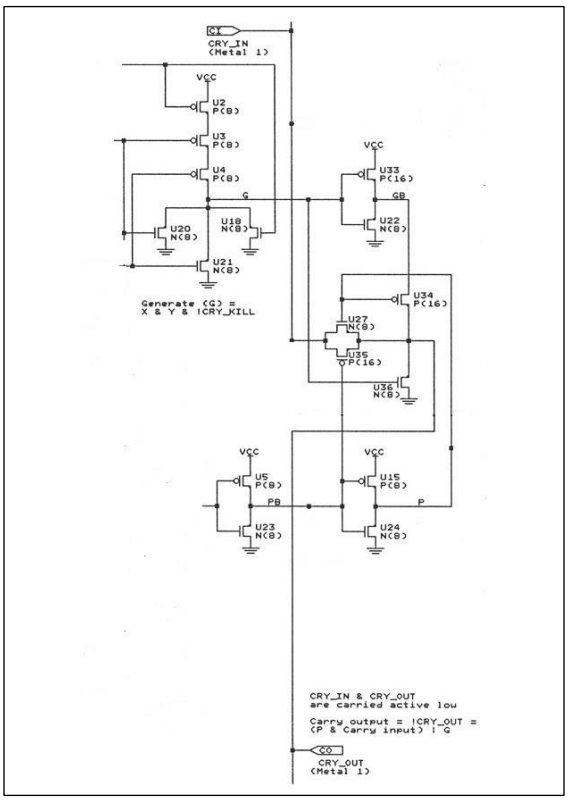
The CP, compare, is a subtract that does not write a result. The TM, test mask, is an AND that does not write a result (only the condition codes are set).

5 Carry Chain

5.1 Generate – Propagate

Without reviewing a transistor level implementation, the terminology of generate – propagate would not be well understood. For the old nchan ALU designs, Carry-In was passed to Carry-Out using a single nchan pass transistor. For a CMOS implementation a full transmission gate is used to pass Carry-In to Carry-Out. This is gated by “propagate” which is the EXOR of the two input operands.

The concept is to only propagate the carry when necessary. Otherwise, the Carry-Out signal is forced using the generate control. When you generate, the previous carry input is ignored thus the logic ripple for carry is interrupted. Obviously, this is pattern dependent and the worst case still has the full carry ripple. The advantage is the use of mux-logic over a gate level design. If properly designed the propagation will be faster than the multiple transitions of the gate level implementation.



5.2 Carry Polarity

The carry chain will need to be buffered for wider datapaths such as the 64-bit version (AIT) of the ALU. It is common to design datapath cells for both polarities of carry. The buffer can then be a single inverter between two groups of opposite cell polarities. This minimizes the transitions.

5.3 Carry Save

“Carry Save” is a type of carry look ahead. This is used on the wider 64-bit version.

The carry chain is duplicated for carry save and calculated in groups (such as 4, 8, 16 bits). For the MSB side of the chain, one carry group calculates the results for a carry input fixed as ‘1’ and the duplicated chain calculates for a carry input fixed as ‘0’.

As the actual carry is calculated from LSB to MSB, the determined carry is used to mux the next MSB precalculated carry group as the result to be used. Thus, the carry ripple is not bit by bit but rather using multi-bit groups. This trades off transistor count versus propagation delay.

Since some propagation time is used for the first LSB carry determination, the sizes of the groups can be increased moving from LSB to MSB. The size of these groups should be optimized using Spice simulations. Thus, the lower part of the chain may use 4-bit groups and then these

may feed 8-bit groups to eventually feed 16-bit groups. The 16-bit groups have more time to calculate their results while waiting for the real carry to be determined from the LSBs.

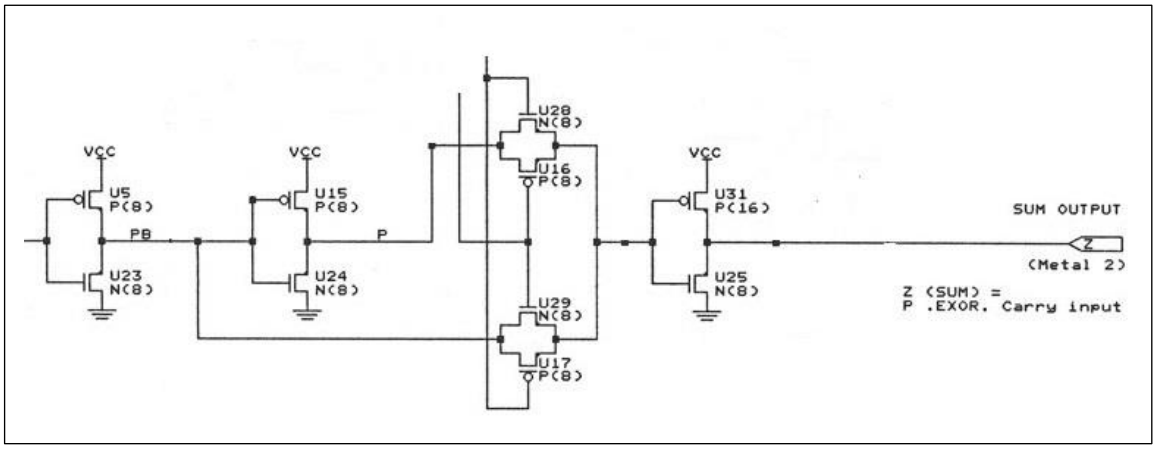
5.4 Carry Kill

Since the ALU performs logical functions such as AND, OR, EXOR, a control line (CRY_KILL) is generated by the instruction decode to kill the carry chain.

6 Sum EXOR

The output of the LFU is “propagate” which will be the EXOR of the two input operands for math instructions. The sum output still needs another EXOR with the carry input.

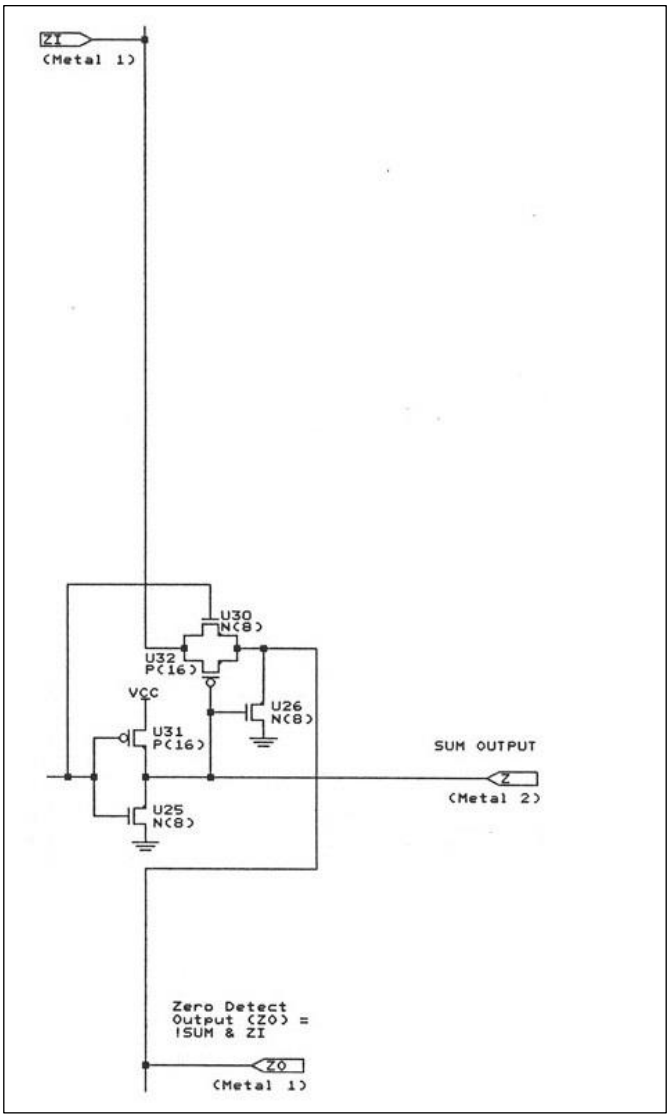
As most engineers know, an EXOR gate can be used as an inversion selection. Using one input as a control, if ‘0’ pass the other input to the output, if ‘1’ invert the other input to the output.



7 Zero Detect

The processor contains several “condition codes” for conditional branch instructions such as Z (zero), C (carry), S (sign), and G (greater-than) and hardware externally defined conditions. The zero detection is built into the ALU datapath cell and ripples alongside the carry chain.

The zero-detect is just a mux-logic AND gate. Alternatively, a CMOS PLA-like circuit of one nchan and one pchan connected with two wires to form a distributed NAND gate could be implemented.



8 Bibliography

Plachno, R. (1999, December). *Processor Design.* Retrieved from Robert S. Plachno: http://www.elqc.com/RobertPlachno/RP_RISC_timing.pdf

Plachno, R. (2009, March). A True Single Cycle RISC Processor without Pipelining. *ESS Design White Paper – RISC Embedded Controller*. Retrieved from http://www.elqc.com/RobertPlachno/RP_RISC.pdf