A 320ps 16-bit ALU using Static OPL in a 130nm process

Bharan Giridhar
bharan@umich.edu

Bhavitavya Bhadviya
bhavi@umich.edu

Faissal Sleiman
sleimanf@umich

Hassan Ghaed
ghaed@umich.edu

ABSTRACT

We present a 16-bit ALU designed using Static Output Prediction Logic (OPL) in the IBM 130nm process. The ALU used a modified Kogge Stone adder. The ALU was found to have a worst case delay of 320ps (nominal)

Keywords

Kogge Stone adder, Output Prediction Logic, Static CMOS

1. INTRODUCTION

The ALU is one of the primary components determining microprocessor performance. Output Prediction Logic (OPL) is a recently developed circuit technique that is twice as fast as domino logic, and several times faster than static CMOS logic. We present a 16-bit ALU implemented in static OPL.

In most cases, static CMOS logic is the best option for the vast majority of CMOS circuits because of its noise immunity and no static power dissipation. Dynamic circuit families, which are another alternative to static OPL, have commonly been used to improve circuit performance (due to reduced input capacitance, lower switching thresholds and circuit implementations that typically use fewer levels of logic due to the use of efficient and wide gates), have notable disadvantages – the main one being increased noise sensitivity compared to static CMOS.

Output prediction logic is a technique that can be applied to a variety of inverting logic families to increase speed while retaining their other attributes (such as noise margins). OPL relies on the alternating nature of the logical output values for inverting gates on a critical path. OPL applied to static CMOS in the past was shown to yield gains of 2-3x compared with conventional static CMOS implementation. Circuits implemented varied from a chain of gates to complete datapath circuits, and to random logic benchmarks. These speedups were obtained while retaining, for the most part, the noise margins of static CMOS.

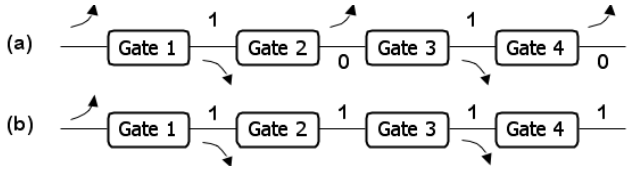

Figure 1. (a) Static CMOS worst-case behavior (b) OPL [1]

Figure 1 illustrates OPL's basic concept. A series of gates are precharged to a value of 1. Only one (low-skew, i.e., fast) pull-down event has to occur for every two consecutive clock separations, because at most one of two adjacent gates in a critical path has to transition. OPL aims to optimally time the release of each gate in the series. For that, well-timed clocks must be generated for the component (Figure 2). An in-depth analysis of the concept of OPL was presented in [1].

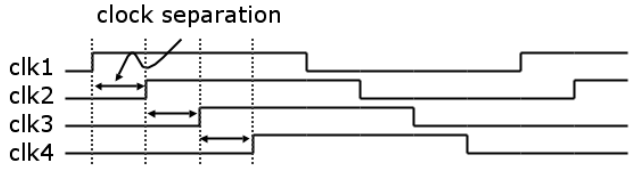

Figure 2. Clocking Scheme for OPL

1.1 Design Context

We have designed our OPL-based ALU in the context of a two-stage pipeline. Within one clock cycle, its operation begins after instruction decode, register file read and input muxing, altogether taking 1.5ns. This will bear on the timing of our initial OPL clock.

With static CMOS, we achieved a worst-case ALU delay of 650ps. Based on results and speculation from [1], we aimed for a speed-up of 2x, i.e. ~320ps.

2. ALU ARCHITECTURE

Our ALU comprises an adder (that can produce XOR, AND, OR functions of its inputs as a by-product) and an output multiplexor. We chose a logarithmic adder architecture because of its inherent division into stages.

As our clock generator design ran in parallel with the ALU design, we produced preliminary results on variations of producible clock separations. We saw that our ALU would have to sustain up to 10ps clock variation.

2.1 Adder Architecture

The reference adder implemented was a radix-2 Han-Carlson adder. The adder had 8 stages of logic. To meet our 320ps requirement, the stages need to be given clocks separated at 40ps. This happened to be the optimal clock separation for the circuit, but variation of the clock means we would have to separate the clocks by more, leading to a failure to meet the requirement. We note that if we have *n* stages, then increasing a clock by an amount, produces *n* times that amount of delay to the output. In other words the more the stages, the more the delay is sensitive to clock separation. We proceeded by exploring circuits with fewer larger stages.

We explored radix-3 adder which had relatively more complex gates, thereby reducing the probability of glitching. However, when applying a radix-3 topology to for a 16-bit adder, the sizes of gates from stage to stage drastically reduce. In OPL, smaller gates tend to glitch more, where glitching leads to a subsequent

snowball effect on delay. In conclusion, with radix 3, we have fewer stages, for which we need to increase clock separation (because of large gates), and for which we do not tolerate much clock variation (because of small gates).

Thus the final choice of adder architecture was a hybrid of different of different radices to achieve evenly sized gates for 16 bits. We will present this in greater detail in Section 3.

2.2 Output Mux

We evaluated two output mux topologies – the transmission gate based mux and the tristate mux. However, the tristate mux is faster because of the higher capacitance seen by the input. So we went with the tristate 4:1 mux to choose between nand, nor, xor and sum.

3. OPL ADDER IMPLEMENTATION

Earlier in [2] the author had suggested merging the generate/propagate signal generation block with the carry generation (Gi,Pi) first stage block for making the gate robust to initial clock arrival time by merging the two Boolean equations to get a single expression, resulting in a complex gate. Based on this and [3], we extended this concept to merge the further stages, two at a time to the extent of getting gates of roughly equal sizes and delays, which is beneficial for OPL circuit with a uniform clock separation between the stages. This resulted in a very custom (hybrid radix) adder, with each stage having roughly the same amount of complexity.

3.1 Sizing

One of the primary reasons to switch to the hybrid radix was that the complexity of the gates provided inherent robustness to glitching. However, a key factor here is the tradeoff between speed and robustness of the gate. In order to size a gate for speed, we were primarily interested in the falling transition of the gate, i.e. to say, we designed the cell to have a dominant NMOS pull down stack and a weak PMOS pull up stack. The NMOS stack was sized based on the fact that 0.69*Rnmos*Cout must be approximately equal to my clock separation between the stages, By doing so, we try to prevent the circuit from becoming either data limited or clock limited and operate at an intermediate optimal point. Also, as by making the gates complex, we also end up with large NMOS stack depths; we note that the transistors are sized fairly large due to the fall time constraint mentioned. The NMOS stack is also tapered when the intrinsic MOS capacitance itself dominates the output capacitance of the gate.

The PMOS pull-up stack, on the other hand must be sized purely for robustness. We observed earlier that glitching at the output causes a slowdown for the entire chain and has a snowball effect on delay. Thus for robust operation, we must ensure that no stage glitches below the V_{IH} of the next gate. Based on this theory, we also suggest that for sluggish gates, the pull-up to pull-down (p/n) ratio can be significantly smaller than for static CMOS operation. Thus we improve the delay at the expense of noise margin. However for gates that are small and fast, we would have to size the PMOS stack up for robustness. In our design, for sluggish gates, p/n ratio was about 0.8 while the p/n ratio was about 1.5 for fast gates.

The clock PMOS is sized up for allowing the circuit to operate at higher frequencies. Since the circuit has been sized keeping the falling transition in mind, the rising transition is also required to keep pace with my falling transition to allow for high frequencies of operation.

We spoke about sizing for speed and robustness till this point. We now propose an input ordering scheme to further increase the robustness of the OPL gate while still adding to speed! Inputs which arrive late must be positioned higher up (closer to the output) in the NMOS stack, as we are primarily interested in a rising transition on the NMOS and robustness is not the concern for a rising transition on the NMOS as the output is supposed to fall – so here the speed is improved. Also, at the falling transition, we focus on the PMOS – the later the signal arrives, the higher up (farther from the output) it must be on the PMOS stack – as a late arriving falling transition will expose more of the charged nodes to the pull-down stack which now has more charge to discharge thereby reducing the glitch.

3.2 Timing

We designed the adder for robust operation with a 40ps clock separation. But our clock generation circuit was designed for a nominal best-case clock separation of 50ps. So, we have ‘overdesigned’ our circuit for robustness – given the 10ps clock variation that we observed earlier. Also, note that if a fast stage precedes a slow stage, then there is a possibility to group their clocks together – this is because the slow stage would not react to the fast switching quickly thereby offering to keep the output glitch in check. However, whenever a slow stage precedes a fast stage, the clocks must be separated for robustness.

3.3 Layout

The stages of the ALU were organized in a U-turn fashion. This enables the output to be produced near the input, which distributes the burden of the wire capacitance seen when routing back to the rest of the datapath. Since we are trying to design stages for even delays, such a distribution is suitable.

We matched a bitslice of 6.4um in all our ALUs. In effect, only the other dimension changes between layouts. In going from the static-CMOS Radix-2 Han-Carlson ALU to the static-OPL Hybrid Kogge Stone ALU our ALU's height increased from 60um to 88um. Within the new ALU, 15um of height were dedicated to OPL's clock tree. As far as the ALU itself is concerned, its area increased by around 25%. This was primarily because with larger transistor stacks, the transistors have to be sized up.

The final OPL ALU's area was approximately 9200um^2.

4. CLOCKING

Generating properly-spaced clocks is the most critical issue in OPL circuits. In order to achieve a reliable clocking scheme a number of methods have been proposed in literature. In general, all accurate delay-generating can be divided into two categories:

1. Open-Loop Delay Generation: These methods use either inverter chains or differently-sized buffers to achieve desired clock separation. They may use reduced-swing buffers to achieve delays less than FO1 delay.

2. Close-Loop Delay Generation: These techniques add DLL or PLL-based feedback to open-loop methods so that more reliable clock separation is achieved.

In our project, we first looked into closed-loop methods. Among them, PLL-based delay controllers were not suitable for our project for two reasons:

1. They require a proper VCO as their core element whose design was way beyond the scope of the course, and, more important than that, designing a VCO that can operate at frequencies of GHz range is very difficult.

2. They accumulate jitter.

Thus, we investigated the possibility of using DLLs. A delay-locked loop can generate *n* evenly-separated clocks using *n* controlled delay elements with nominal delay of T/n in which T is the clock period. We simulated a 20-stage voltage-controlled inverter chain in closed loop as well as one closed loop copy of the same thing and ran Monte Carlo on the clock separation. Note that for the purpose of out project; we used ideal op-amps instead of charge pumps that are essential elements of DLLs. As we expected, the results were promising: the feedback could lock the delay to the T/20 of the clock period with a sigma of 3ps whereas the inverter-chain was susceptible to the changes in PVT.

One might think that closed-loop is preferred for OPL. However, if we investigate what we really need from the clocks we find out that the opposite is true. OPL needs the clocks to arrive when the previous stage is done with the derivation of the result. If the whole chip runs at a slower corner, we expect our adding to become slower, and, as a result we need the clock to arrive at a later time. This is where closed-loop disadvantage becomes obvious: it will still lock to T/N of the system clock which is a constant. This can easily make the circuit glitch and we will use all the advantages of OPL.

Among the open-loop methods for generating the clock, we could either use series inverter chain or parallel inverters with different sizes. We chose the series inverter solution because:

1. Using inverter chain, the 2nd delay element will be triggered "exactly" at the moment the first clock has switched, so the clk2-to-clk3 separation will have a mean and distribution which is independent of the clk1-clk2 delay. So the series circuit is better as far as robustness is concerned.

2. It resulted in smaller transistor sizes compared to the parallel method. For example, the parallel scheme nee

3. It gives us more flexibility in designing a buffer for each stage clock.

After choosing the simple inverter chain as our delay-generation method we had to decide on the separation. There are some reduced swing buffers proposed in literature that can give delays less than 2xFO1 delay. The idea is to bias the node at some middle voltage so that the transistor can switch faster. Apart from the static current it burns, these RSBs have one major disadvantage, that is, their sensitivity to Vt variations. Our simulations showed that they generate delays with sigmas twice as big as simple inverters (10 instead of 5ps) in Monte Carlo analysis with same-size transistors.

Switching to radix-4 adder enabled us to use larger clock separations and as a result, the required clock by the adder, i.e. 60ps was higher than 2xFO1 delay. So, we selected simple inverter chain over RSB.

We obviously needed to buffer the clocks before delivering them to OPL adder. Moreover, we chose to add a separate buffer for each bit of every stage for two reasons:

1. Without these dedicated clock buffers, we needed a big driver for the whole stage. Hand-calculations showed that since one of the stages had a clock capacitor load of 600f, the driver for that needed to be more than 20microns wide!

2. Using the dedicated clock buffer, we were able to size the buffers so that the rising time of ALL 16x5 clock bits were equal. That gave us a big advantage in maintaining robustness in Monte Carlo simulations.

5. SIMULATION

We performed two kinds of analysis – one being corner analysis at the slow corner and the other being local mismatch. Parameters varied were ACLW and ACLV and Vt, which in turn depended on doping and temperature.

5.1 Simulation Results

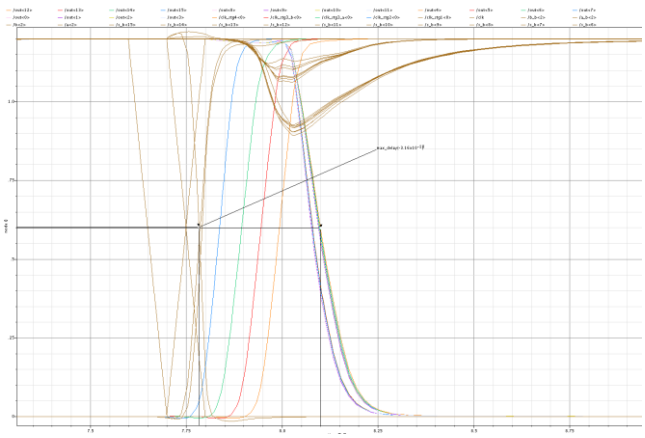

Figure 3. The Static OPL based ALU operating at ~50ps clock separation – worst case delay: ~320ps

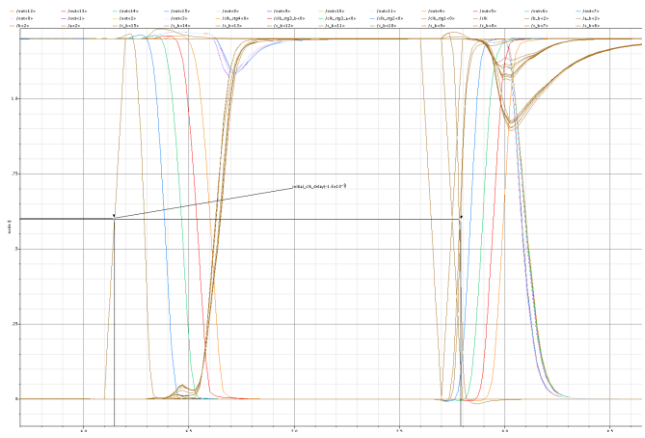

Figure 4. Illustrating operation in the context of out processor

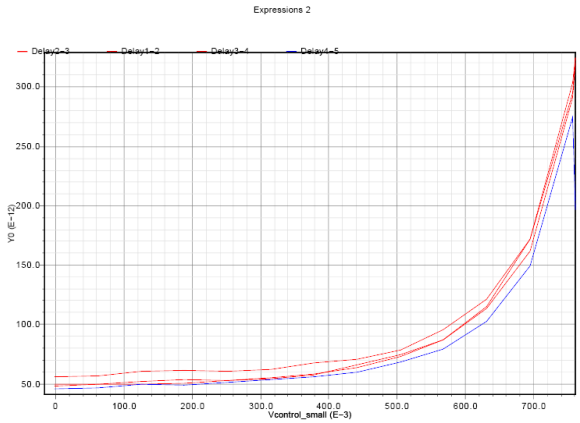

Figure 5. Clock separation as a function of voltage control

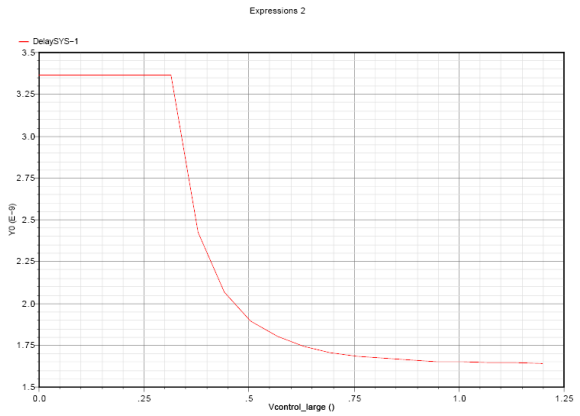

Figure 6. Initial clock delay as a function of voltage control

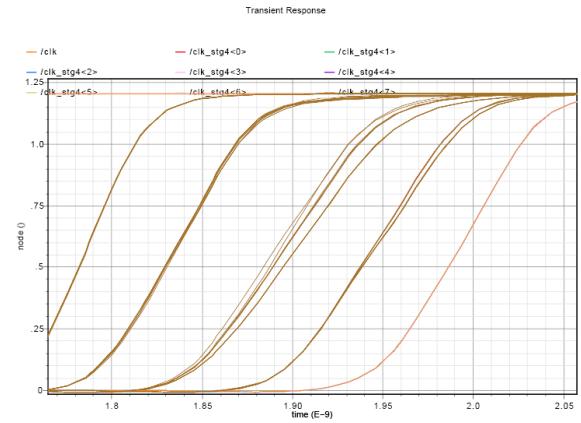

Figure 7. Demonstrating the clock separations generated by clock generation circuit

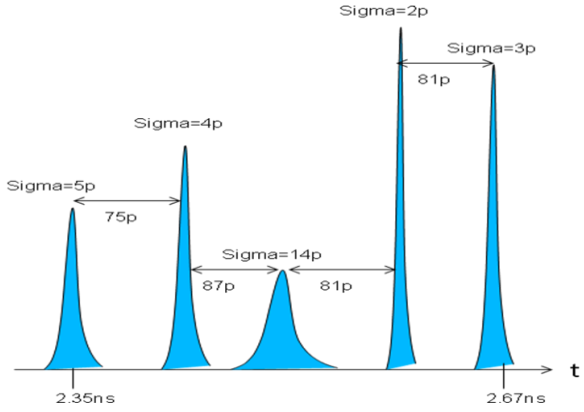

Figure 8. Effect of variation in whole ALU at slow corner

5.2 Layout view

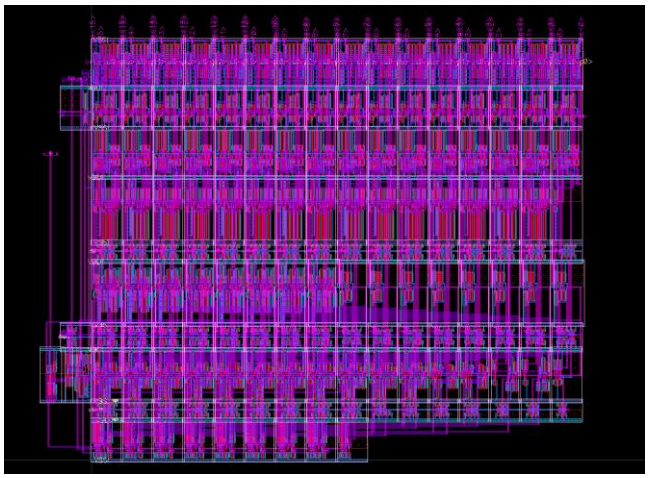

Figure 9. OPL ALU layout

6. REFERENCES

[1] L. McMurchie, S. Kio, G. Yee, C. Sechen, Output Prediction Logic: A High-Performance CMOS Design Technique, Proceedings of ICCD, Sept. 2000.

[2] S. Sun, et. al., 409ps 4.7 FO4 64b Adder Based on Output Prediction Logic in 0.18um CMOS, University of Washington, Seattle, WA, 2004

[3] Y. Leblebici et. al., Higher-radix Kogge StoneParallel Prefix Adder Architectures, ICAS 2000