Relativistic Hardware Design

Frans Skarman
Linköping University

Astrid Lauenstein
Linköping University

Abstract

As a community, we are very good at making large circuits that compute things very quickly. However, sometimes a small and efficient circuit is more desirable, and this is something that is less well explored in the community. To take this idea to its logical conclusion we built a circuit consisting only of a single time multiplexed NAND-gate. The circuit can emulate any digital hardware using very little space, though it is of course predictably slow.

1. Balancing Space and Time

Hardware design is all about trade-offs, and perhaps the most important tradeoff is that of time and space. In a simple processor we dedicate space on our chip to an ALU, control logic, some registers, and a memory interface. We also almost always choose to scale these to operate on one word of data at a time. As our performance requirements increase, we use additional space by adding pipelining, branch predictors, caches, and prefetches in order to make our processor run faster. If our thirst for performance is still not satiated, we start diverging from the general purpose processor to add custom instructions tailored to our application or accelerators that can offload the work for our processor. Eventually, we may ditch the processor entirely and build a fully custom ASIC for our application. In each case, we make a decision to trade space on our chip for time.

The reason a fully pipelined ASIC can be so much faster than a processor is that the compute unit of the processor is shared over time by multiple operations – it is time multiplexed. In a fully pipelined ASIC, there is no such sharing, each operation our algorithm needs to perform has dedicated logic on the chip.

As a community, are very good at taking a relatively fast sequential program and making it *extremely* fast with custom hardware, but going the other direction is much less well explored. In this work we will change that, and explore how we can trade computation time to build tiny hardware that is still powerful enough for complex computation. There are good reasons to explore this, if our performance requirements are satisfied with small hardware, building the larger hardware simply wastes resources. Examples of this include circuits that poll sensors occasionally, circuits only used at startup such as DRAM initialization, and novel semiconductor substrates where space is at a premium.

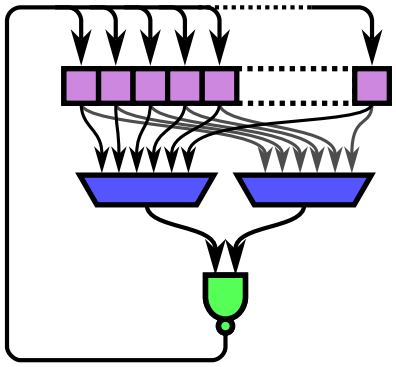

Figure 1: A time multiplexed NAND-gate

2. A Time Multiplexed NAND-gate

We know from our introduction to digital logic that any combinational circuit can be built from just NAND-gates and state elements connected together. Typically, we exploit this by spreading our NAND-gates out over space, filling our chip and connecting them together with wires. However, this is not our only option, in the same way that we can make our fully custom ASIC smaller by building a CPU and time multiplexing our arithmetic operations, we can spread our NAND-operations out over time instead of space. In the extreme case, that means computing all NAND-operations on a single NAND gate, as shown in Figure 1. By controlling the multiplexers to choose two operands from the flip-flops to feed to the NAND gate, and writing the result back to a third flip-flop, this circuit can emulate *any* boolean circuit, as long as the number of bits of state does not exceed the size of the "memory".

3. Screaming Into the Mercury

While this circuit is certainly smaller than most circuits it could implement, it is not as small as it could be. The compute logic is hard to shrink, and so is the state, but the multiplexers used to control the circuit are relatively large, and grow as the number of flip-flops grows. In order to address this, we need to find a different way to get data to and from the NAND gate. Here, we can draw inspiration from early computing history where hardware was simply much cooler than today. In particular, some computers stored data by placing a microphone and speaker on either side of a tube filled with mercury, encoding the data as sound, playing it into one end and reading it when it came back on the other.

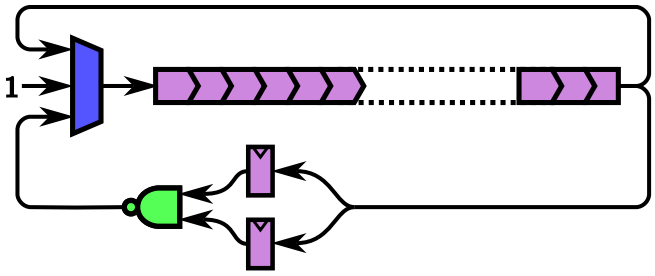

Figure 2: A time multiplexed NAND-gate with delay line memory

Sadly, putting tubes of mercury on a silicon chip is not practical, but we do have shift registers at our disposal.

Based on this, Figure 2 contains a second iteration of the time multiplexed NAND-gate. Here the state is stored in a constantly "spinning" shift register. The giant multiplexers of the previous version are gone, and are now replaced with two registers with enable signals. In addition, the newly added multiplexer controls what gets written into the shift register. In order to perform a NAND operation, a controller simply has to wait for the operands to appear at the end of the shift register to store them in one of the buffer registers. Once both operands are captured, the controller waits for the destination to come around to write the result back.

4. It is Small and Slow

In order to evaluate the performance of this circuit, it was implemented in Spade [1] and taped out on TinyTapeout [2]. The allocated space of 200 × 160µm on sky130 was enough to fit an 880 bit shift register in addition to an 8 bit shift-register for output. The circuit has 4 inputs: a clock, a 2-bit control signal for the multiplexer, and two enable signals, one for each buffer register.

While programming the circuit manually is possible, doing so is extremely tedious as one has to keep track of where along in the shift register each operation should take place. In order to make it easier program the circuit, we developed a yosys [3] backend[1] which compiles the input circuit into control signals for the NAND-gate.

In order to demonstrate this capability, we synthesized a 5 stage pipelined RISC-V core[2] which ends up requiring 1 753 839 NAND-instructions per RISC-V clock cycle. At 2.5 MHz[3] this means that the emulated circuit runs at a blazing 1.43 Hz! This may seem slow, but it is worth remembering that it emulates a almost 3000 discrete NAND gates and 350 registers using only a single NAND gate.

The performance characteristics of the circuit mean that a parallel emulated circuit can be much faster than a sequential one as the circuit can compute other values while waiting for the shift register to spin around. As an example, we were able to synthesize 8 parallel non-pipelined RISC-V cores and run them in the same time as the single pipelined core. Though this may also be caused by yosys being able to perform more optimizations when the instruction memory is fixed and there are no registers to block optimizations.

[1] https://git.sr.ht/~acqrel/nand-shift

[2] To fit this we had to limit it to 4 general purpose registers

[3] Limited by signal integrity of the control signals.

5. Reality Check

The delay line time multiplexed NAND-gate is most likely not practical in any real world application, it is simply too slow, and while the circuit itself is small, feeding it control signals is not trivial. Instead, this is meant as a data point on the pareto frontier of the time/space tradeoff in hardware design. As mentioned in the introduction, we are very good at building appropriately fast circuits where we take all the arithmetic operations we want to perform and spread them out over space and time to fit our needs. But in cases where we have time to spare, even with a small processor we often fail to claim the potential space savings we can get by further spreading whole-word arithmetic operations into multiple clock cycles.

While the time multiplexed NAND-gate is not practical, there are two interesting projects that do explore this idea: SERV[4] and FazyRV [4]. Both of these are Risc-V processors which compute a few bits of output every clock cycle, rather than a full word. SERV is bit-serial, it computes one bit per cycle, while FazyRV allows configuring the word length, allowing precise tuning of the processor to the time space target.

Bibliography

[1] F. Skarman and O. Gustafsson, "Spade: an expression-based HDL with pipelines," in *Proc. Workshop Open-Source Des. Automat.*, Apr. 2023.

[2] M. Venn, "Tiny Tapeout: A shared silicon tape out platform accessible to everyone," *IEEE Solid-State Circuits Magazine*, vol. 16, no. 2, pp. 20–29, 2024, doi: 10.1109/mssc.2024.3381097.

[3] D. Shah, E. Hung, C. Wolf, S. Bazanski, D. Gisselquist, and M. Milanovic, "Yosys+nextpnr: An open source framework from Verilog to bitstream for commercial FPGAs," in *Proc. Int. Symp. Field-Programmable Custom Comput. Machines*, 2019, pp. 1–4. doi: 10.1109/FCCM.2019.00010.

[4] M. Kissich and M. Baunach, "FazyRV: Closing the Gap between 32-Bit and Bit-Serial RISC-V Cores with a Scalable Implementation," in *Proceedings of the 21st ACM International Conference on Computing Frontiers*, in CF '24. ACM, May 2024, pp. 240–248. doi: 10.1145/3649153.3649195.

[4] https://github.com/olofk/serv