Enhancements to Adiabatic Logic for Quantum Computer Control Electronics

Technical report ZF002 v1.03, August 5, 2020

Erik P. DeBenedictis
Zettaflops, LLC, Albuquerque, NM 87112
erikdebenedictis@zettaflops.org

Abstract

This report improves upon what was originally the T-Gate shift register[1] and renamed to 2-Level Adiabatic Logic (2LAL) by Mike Frank when he enhanced it into a logic family.[2, 3] This report also offers improvements to the SCRL logic family.[4]

2LAL and SCRL have been proposed as energy efficient alternatives to room-temperature CMOS, a goal that depends upon high throughput and high speed. However, this report applies to quantum computer control electronics where Josephson junctions are available for logic functions but transistor circuitry such as 2LAL and SCRL provide much higher density for memory and subfunctions requiring high complexity. Since high speed and high throughput are available from Josephson junctions, the transistorized logic has less need for these attributes.

To better serve quantum computer control applications, this report includes improvements to the clocking structure to support inverting gates without doubling the number of logic rails and reducing density.

The clocking improvements also make the design of data permutation logic more efficient. This is an important optimization because some common digital building blocks are just data permutations, such as busses, multiplexers, and addressing logic.

This report also proposes a way to use Fully Depleted Silicon on Insulator (FDSOI) transistors to eliminate half the transistors from some stages.

Overview

This report builds on the quantum computer control system described in ref. 5, resulting in the enhanced system illustrated in fig. 1. To minimize the refrigeration load from high-power signals in the cryogenic environment, fig. 1 shows waveform generators and DC power supplies in the 300 K room temperature environment. The signals pass through the temperature gradient to a convenient temperature for *in situ* electronics, such as 4 K.

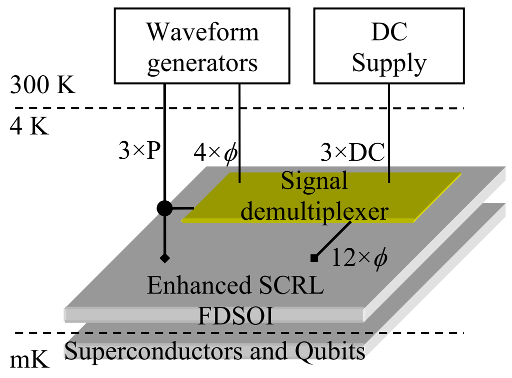


Fig. 1. System overview. Room temperature signal generators drive a signal demultiplexer and enhanced cryogenic adiabatic transistor circuits. Objective is to drive qubits on a lower layer.

By using the 3-phase SCRL signal demultiplexer described in this report, the number of wires going through the temperature gradient drops from 3 dual-rail P signals + 6 dual-rail clocks (= 18 wires) to 3 single-rail P signals + 4 clocks + 3 DC supplies (= 10 wires). The demultiplexer has features that move key sources of heat to the waveform generators, an optimization that is important for cryogenic operation.

The cryogenic environment is expected to contain a semiconductor control chip and a qubit chip, which may be layered on top of each other as in ref. 5, side-by-side, or in some other arrangement.

This report describes the control chip as including the signal demultiplexer and control electronics, the function of the latter described in ref. 5. This report describes control electronics that may include transistors with tailored or dynamically changeable threshold voltages. The latter is a property of FDSOI transistors. The reminder of the semiconductor chip may contain 2LAL or SCRL circuits, with this report describing circuit enhancements for inversion and permutation networks.

Baseline 2LAL

Fully pipelined (shift register) 2LAL is defined for four non-overlapping clocks, as illustrated in fig. 2a. Each clock must be available in true and complement form. However, the complement of each of these clocks is another one of the clocks, so no additional signals are needed.

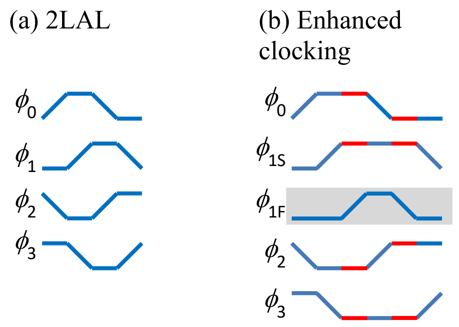


Fig. 2. 2LAL requires four non-overlapping clocks, but they don't need to be perfectly symmetric. For this report, the flat top of ϕ_1 is extended on either end, yielding ϕ_{1S} (slow) and ϕ_{1F} (fast). The resulting clock will drive all existing 2LAL circuits, albeit somewhat more slowly.

The original transmission gate, or T-gate, shift register is illustrated in fig. 3a and fig. 3d, which moves data streams a and b left-to-right. Each wire represents a dual-rail signal and each rectangle represents a pair of T-gates. Each T-gate in turn comprises an nFET and a pFET. Frank found that by tapping bits in data streams a and b, fig. 3b constructs a new data steam c where each bit $c_k = a_k \mid b_k$ is computed by the red circuitry; likewise fig. 3c constructs stream d where $d_k = a_k$ ^ b_k using the blue circuitry, where | and ^ are the logical OR and AND operators. New streams c and d must be decomputed at some point to avoid excessive power dissipation, which is shown on the right of fig. 3b and fig. 3c.

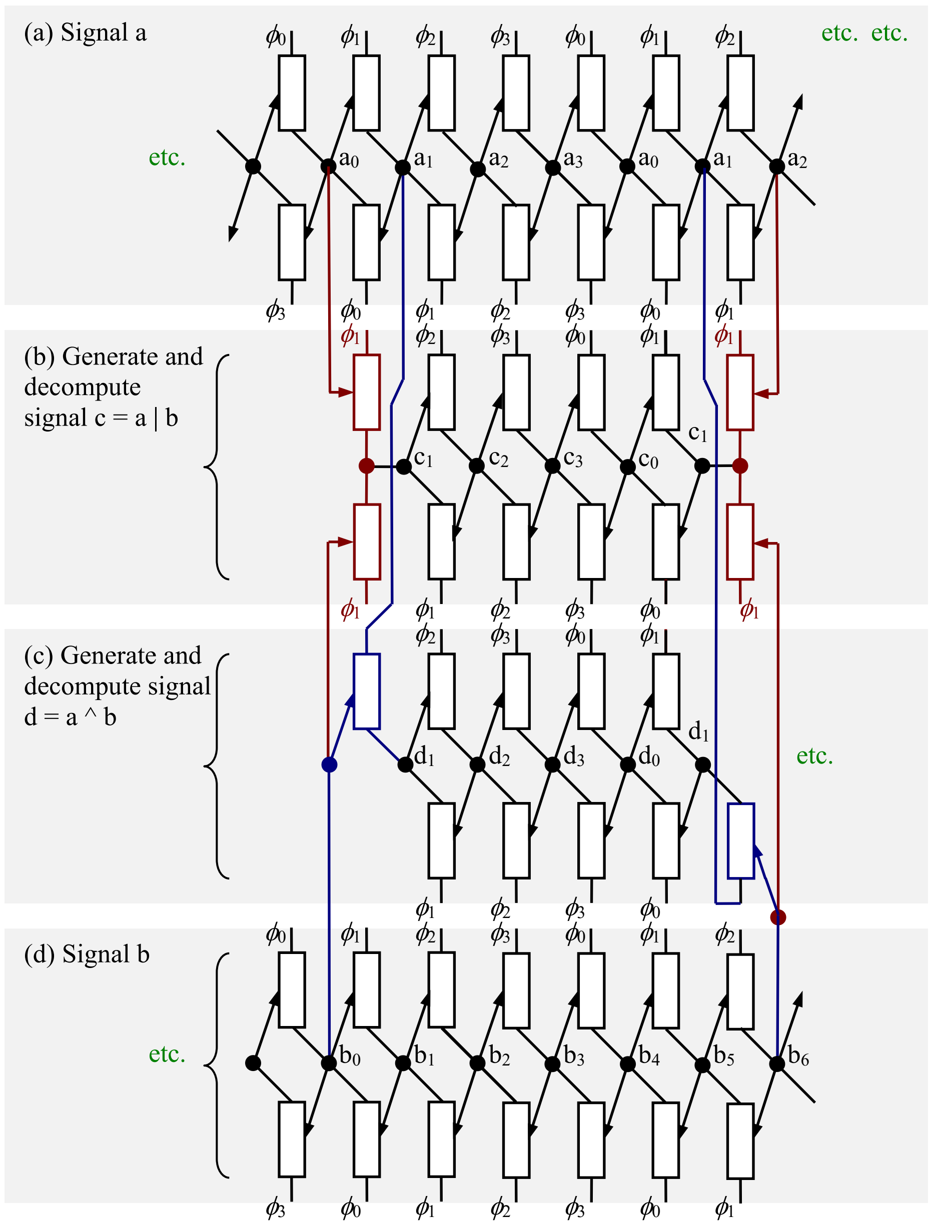

(a) Signal a
etc. etc.
etc.
ϕ_0 ϕ_1 ϕ_2 ϕ_3 ϕ_0 ϕ_1 ϕ_2
a_0 a_1 a_2 a_3 a_0 a_1 a_2
ϕ_3 ϕ_0 ϕ_1 ϕ_2 ϕ_3 ϕ_0 ϕ_1
(b) Generate and decompute signal c = a | b
c_1 c_2 c_3 c_0 c_1
(c) Generate and decompute signal d = a ^ b
d_1 d_2 d_3 d_0 d_1
etc.
(d) Signal b
etc.
b_0 b_1 b_2 b_3 b_4 b_5 b_6

Fig. 3. (a) A and (d) Decomputing a signal, or generating a new signal propagating to the left, or backwards in time.

In addition to logical AND and OR, Frank's approach can generate any AND-OR tree. However, 2LAL, by Frank's strict definition, cannot invert signals and hence is not a form of universal logic. If two logically complementary copies of a 2LAL logic network are created, and all external input during operation involves delivering complementary values to corresponding gates on the two networks, swapping corresponding bits between the two logic networks has the same effect as inversion. This is an effective approach for making the logic universal but has the disadvantage of doubling the number of circuit elements.

Enhancing 2LAL

We expand the clock waveforms by adding a gap on either side of the flat top of ϕ_1 in fig. 2a, which is equivalent to stopping all four clocks in two places. This is illustrated by the red segments in fig. 2b. Clock ϕ_1 is renamed ϕ_{1S}, S for slow, consistent with the terminology in ref. 4. We also add a variant of ϕ_{1S} called ϕ_{1F}, F for fast, with a pulse that fits entirely within the flat top of ϕ_{1S}.

Using either ϕ_{1S} or ϕ_{1F} as ϕ_1, the clocks in fig. 2b have the non-overlapping property required by fully pipelined 2LAL and can drive existing circuits. However, the new clocking comes at a cost:

- There are six instead of four transitions per cycle, so there will be less throughput for a given transition time—and the energy efficiency of adiabatic circuits is proportional to transition time.
- Six instead of four clock signals will be required. The complements of each of the clocks ϕ_0, ϕ_{1S}, ϕ_2, and ϕ_3 is another clock in the group, so only four distinct signals are needed for these four clocks and their complements, yet two new clock signals will be needed for ϕ_{1F} and its complement.
- The implementation in this report only illustrates inversion during one of the four clock phases. Increasing the number of phases that could support inversion allow the logic family to do more work per clock phase, but the system would slow further and require even more clocks.

Each stage in the description of 2LAL in published papers[2, 3] comprises two T-gates in a diagonal configuration, as shown in fig. 4a. To better represent the enhancements in this report, fig. 4b rearranges the circuit so T-gates driven by the same clock phase are vertically stacked.

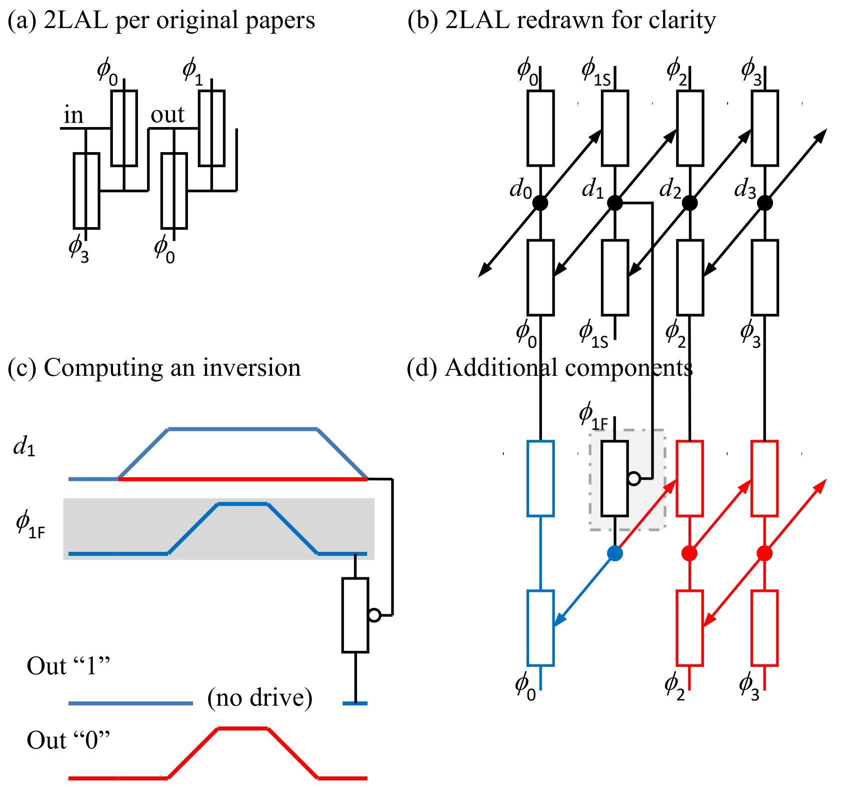

(a) 2LAL per original papers
(b) 2LAL redrawn for clarity
(c) Computing an inversion
(d) Additional components
in
out
ϕ_0
ϕ_1
ϕ_3
ϕ_{1S}
ϕ_2
d_0
d_1
d_2
d_3
ϕ_{1F}
Out "1"
(no drive)
Out "0"

Fig. 4. (a) The orignal publications for 2LAL use an offset geometry, which would confuse this exposition. (b) 2LAL redrawn so T-gates with the same clock phases are stacked on top of each other. This leads to a geometry where each stage's interaction with others is readily apparent. (c) An additional stage computes the logical complement of the original signal. (d) The red pass gates create an additional data stream with the logical complement while the blue pass gates "uncompute" a stream containing the logical complement, recovering signal energy. The red and blue structures would not appear on the same gate (see text).

Inverter

Data signal d_1 in fig. 4c shows a red DC voltage for a 0 value and a blue pulse for a 1 value. The inversion of d_1 should be a pulse if input is the red DC signal and a low DC output value if the input is a pulse. This can be accomplished with the inverting T-gate and the ϕ_{1F} clock in fig. 4c, as follows:

If the T-gate's inverting input is ground, the T-gate drives the ϕ_{1F} clock as a data signal, which is the desired behavior.

If the T-gate's inverting input is the d_1 signal, which is the same shape as ϕ_{1S}, the T-gate's output drive will weaken when d_1 leaves ground, becoming a reliably high impedance when the d_1 signal reaches the positive supply voltage. Inspection of the graph shows that the T-gate's upper input will stay at ground for the entire time the drive weakens. As a result, the T-gate blocks d_1 entirely, leaving the natural circuit capacitance to hold the output at ground, representing a 0 value. The same is true during the signal strengthening at the end of the ϕ_{1S} pulse. This is the desired behavior.

The red T-gates in fig. 4d illustrate the construction of a new inverted signal and two additional buffer stages for that signal. To avoid excessive dissipation, the new signal must be uncomputed at some point, which is the purpose of the blue T-gates. The red and blue T-gates are driven by the same signal in fig. 4d for convenience of illustration, but sensible circuits would not use both the red and blue T-gates on the same signal.

Generalization to Other Gates

Fig. 5a repeats the specific inversion logic in fig. 3d for reference, with fig. 5b extending it to an XOR gate. Since more than the single input d_1 is needed, the extension adds a second input δ_1 that would be created by a replica of the shift register in fig. 3b. The generalization of fig. 5b is an and-or tree based on a charge recovery logic (CRL)[4] circuit on inputs d_1, δ_1… that gates ϕ_{1F} to the output. This generalization will work for any number of inputs in principle, but scalability has not been analyzed.

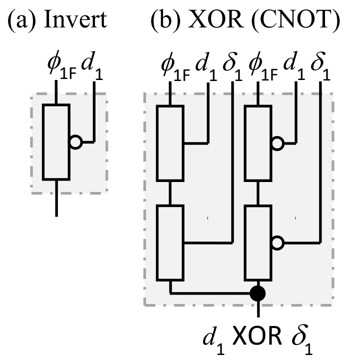


Fig. 5. General gates. (a) Inverter, passing the "0" pulse when d_1 is low. (b) XOR, or CNOT when in association with a saved signal. If d_1 and δ_1 are both 0, the two pass gates on the left will become conductive, passing the pulse. The pulse will also pass if they are both 1.

Using ϕ_{1S} and ϕ_{1F} in design

The ability to invert data signals during ϕ_1 provides a practical remedy for the lack of inversion in Frank's AND-OR trees. For example, a memory built from strictly defined 2LAL would have to store every bit and its complement, doubling the transistor count. This seems like a lot of overhead for a memory. However, a single inverter at the output of the memory could complement of each bit on the fly, creating the third and fourth rails and enabling universal logic.

Inversion during just ϕ_1 enables a general set of optimizations that a software circuit synthesizer could use to significant advantage. Inversion that naturally occurs during ϕ_1 would be realized directly. Where inversion is needed during ϕ_0, ϕ_2, and ϕ_3, the need for the inverted signals would be propagated backwards until such a signal is available or could be produced. The backwards propagation is guaranteed to stop at a previous ϕ_1. Decomputing would involve forward propagation and would stop at the next ϕ_1.

Permutation stages

Permutations stages generally apply to both SCRL and 2LAL, so they will be introduced with notation that covers both.

The diagram in fig. 6a is from ref. 7 and shows how an arbitrary function *f* can be realized in a reversible SCRL pipeline, provided that the pipeline also includes f^{-1}. The concept applies to 2LAL as well. The reader is encouraged to consult that reference for background information. However, this structure requires separate logic networks for compute *f* and f^{-1}, doubling transistor count. Let's see if we can do better.

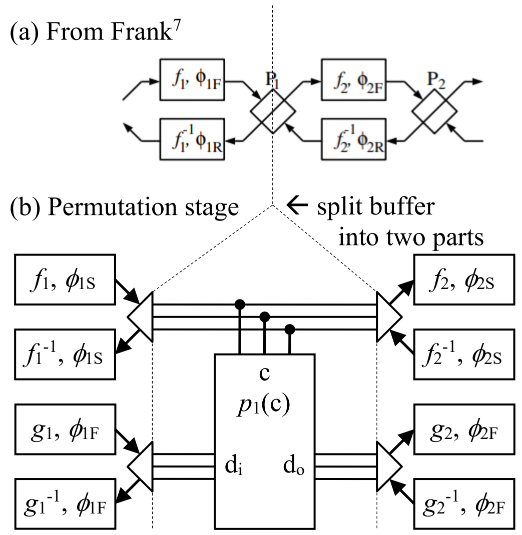


Fig. 6. Permutation stage. Requires fast and slow clocks, yet occurs independently of the invertible stages in the literature.

Like CMOS, 2LAL and SCRL gates can be composed into higher-level logic blocks with wires. Based on the terminology in this report, the bidirectional latches are split through the middle with the two halves connected by single wires (SCRL) or dual-rail signals (2LAL). The splitting of the latches is illustrated in fig. 6b, which shows 3-bit signals between stages.

Wires can define the topology of the logic block, like a netlist, but the logic block could be reconfigurable as well, like an FPGA. Fig. 6b shows functions g_1 and g_2 connected by a permutation $p_1(c)$, where the signals in *c* control the permutation. Thus, changing *c* rewires the circuit.

Reconfiguration should not take place when signals are passing through the permutation network because of the possibility of dissipation in transistors that are in the process of being turned on or off. We discussed previously that existing 2LAL circuits will function correctly with either ϕ_{1S} or ϕ_{1F} as ϕ_1, and the same principle applies to SCRL. To assure the permutation network is stable when data flows through it, the permutation should be selected by signals derived from the ϕ_{1S} clock and data passing through the network must be generated from the ϕ_{1F} clock. This is illustrated in fig. 6b.

The routing function p_1(c) must be reversible, which creates concerns at several levels. The structure in fig. 6b has been deliberately designed to pass the permutation control signals c without change, which assures the ϕ_{1S} and ϕ_{1F} signals are separate by design. The c signals will need to be uncomputed, but this should occur naturally as part of standard reversible circuit design. There will be no problem if p_1 is truly a permutation because backwards operation of the circuit will have the same c signal and hence the same permutation albeit operating backwards.

If the goal is to create a reversible FPGA, the c signals would be created by the configuration logic, which could use ϕ_{1S} everywhere. Of course, the FPGA design software would be obligated to create legal reversible circuits.

However, permutation stages can implement some important standard logic functions quite efficiently:

The Fredkin gate in fig. 7 can be viewed as a control signal that either wires two signals straight through with the blue components or swaps them with the red ones.

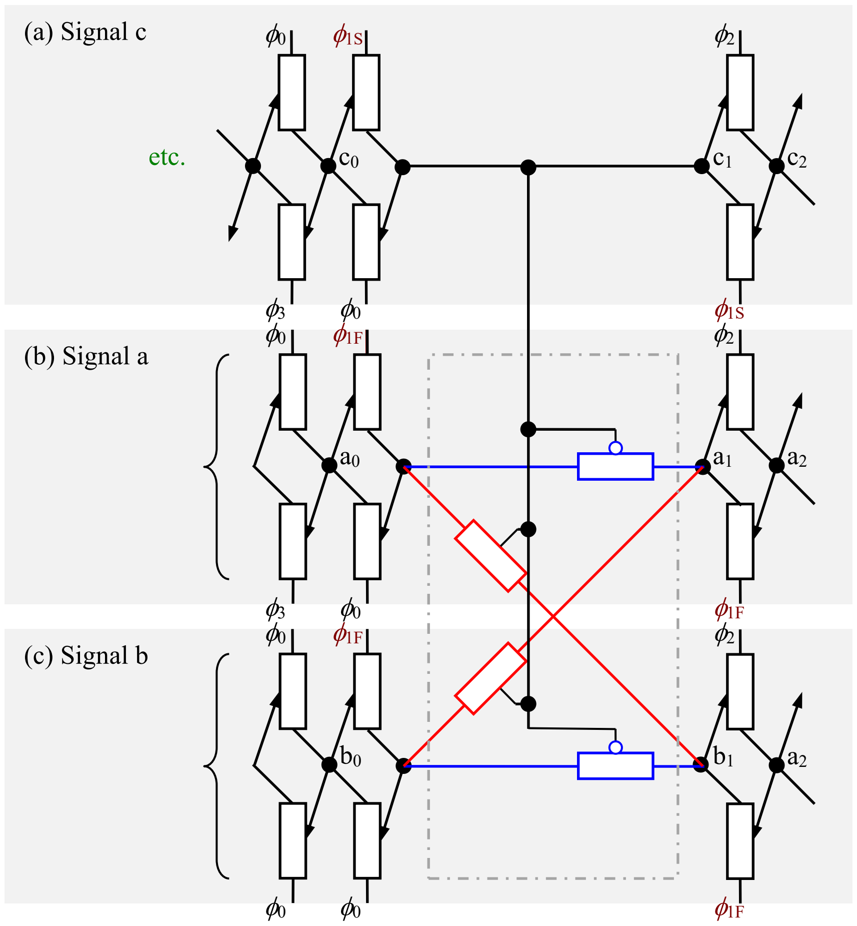


Fig. 7. Fredkin gate. Permutation control from ϕ_{1S}; bits to swap from ϕ_{1F}. If c input is 0, blue path conveys a and b straight through. If c input is 1, red path swaps a and b.

A 4-bit barrel shifter can be viewed as two sequential permutations, specifically a controlled rotation by one place followed by a controlled rotation of two places. The four combinations of the two controls can be viewed as a 2-bit binary number representing the number of places to rotate. Fig. 8 shows such a circuit. Each of the smaller blocks is a two-input multiplexer controlled by one of the two bits of shift count c, where a zero on the control bit enables the straight-through blue path and a one on the control bit enables the red diagonal path. The wiring pattern in the first crossover region shifts each bit down one position (with wrap around) and the wiring pattern in second crossover region shifts two places. The gray block structure comprises a test harness discussed in the appendix.

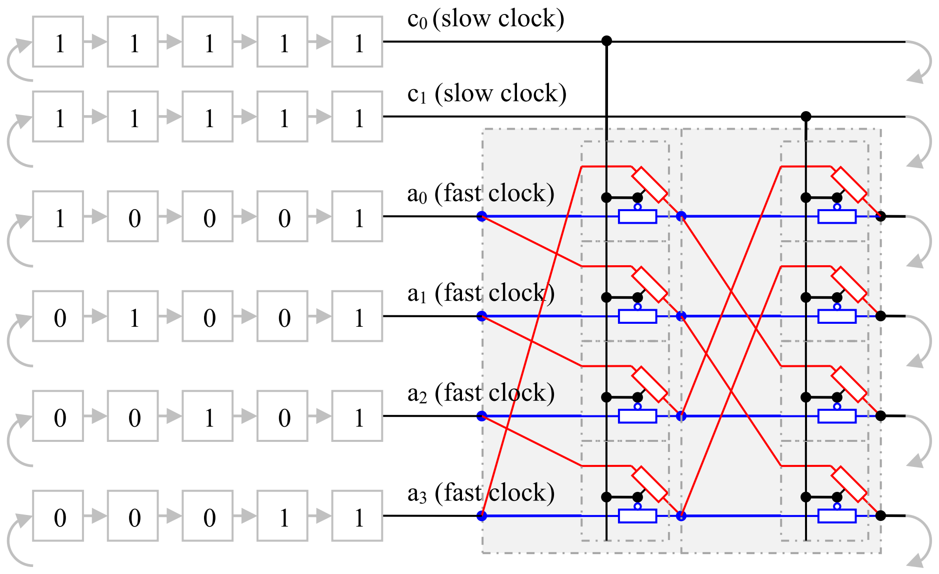


Fig. 8. 4-bit barrel shifter. The smaller outlined structures are 2-input multiplexers and the larger ones are shifters. The result is a complex reversible module. Test harness and initial data for the simulation in the appendix appear on the left.

The circuit in fig. 8 has some novelty. It is intuitively clear that a permutation of four wires is reversible. Furthermore, a controlled rotation will be reversible as long as the rotation doesn't change while data is flowing. However, a two-input multiplexer is not reversible, because one of the inputs gets lost. It's also obvious that fig. 8 can be scaled to any number of inputs, so fig. 8 is a template for a scalable reversible structure made from non-reversible subcomponents.

However, p_1 could be more than a permutation. For example, two input signals could be directed to a single output signal—as long as the two input signals are guaranteed to always have the same value. As shown in fig. 6, the permutations are in a different part of the circuit than the *f*s, *g*s, so it is possible to build a circuit synthesis program that implements a circuit behavior as *f*s, *g*s, and *p*s all at once. Details on such a synthesis program are beyond the scope of this report.

In summary, the advance here is that only one copy of the permutation's logic circuits is required because the signal pathways are electrically bidirectional. This is in contrast to the *f*s and *g*s, which only conduct in one direction.

Clock speed changes

Say we have 2LAL circuits operating a frequencies *f* and 2*f*. The clocks can be engineered so that the ϕ_1's sometimes occur at the same time. At such a common point in time, groups of four stages, such as fig. 9a, each contain a bit in the middle two stages, but the influence of the bit is cut off by the outer stages, making the input irrelevant and leaving the output at ground

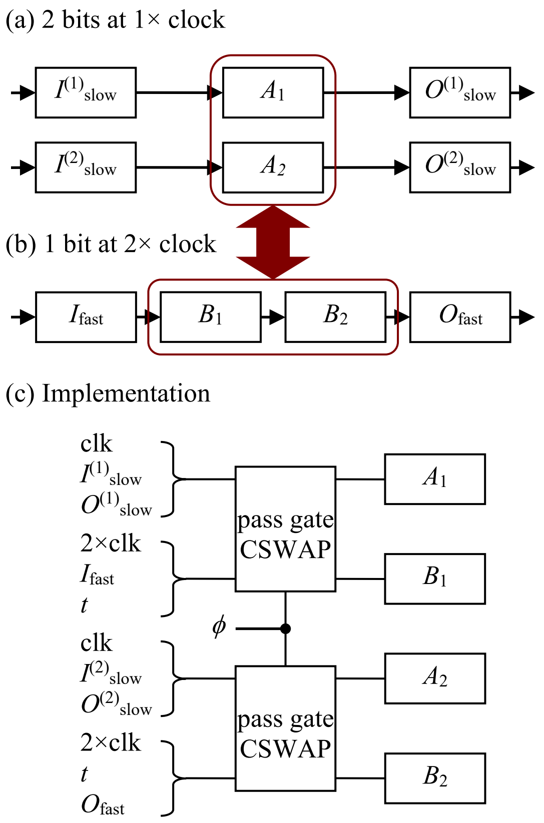


Fig. 9. Clock rate doubler. Top: Two domains at 2:1 clock rate swap data. (b) Implementation is to swap the gates at the point where their outputs are at ground.

Fig. 9 illustrates the clock speed change circuit. Fig. 9a is a two-bit wide shift register at speed *f* while fig. 9b is one bit wide at speed 2*f*. At the common point in time, the two slow bits are swapped with the two fast bits, as shown in red, switching each pair between serial and parallel. The clocks switch as well. When the clock continues, the two circuits will pick up where they left off, but with different bits.

The implementation in fig. 9b shows how the bits can be swapped by pass gates (like fig. 8, but with one control bit).

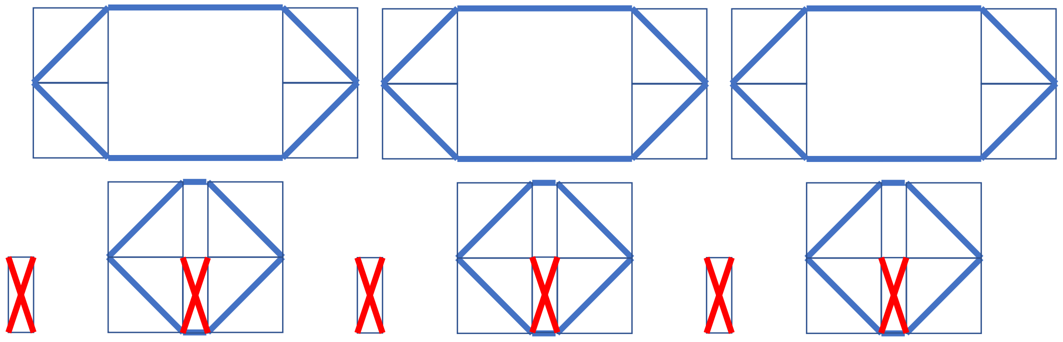
The circuit in fig. 9 can recover energy as expected from reversible circuits, but engineering attention will be required. The circuit in fig. 9 does not create or destroy information; it just swaps information between two clock domains. Information must be decomputed as always, but this may require moving information back to the clocking domain where it was originally created.

This discussion used clocks at rates *f* and 2*f* as an example, but it should be obvious how to extend the clock ratio to any integer. Less obvious is that fact that the clock ratio could be any rational number—or in fact, any clocks that somehow have a common point in time where they are both in the reset stage.

Cryogenic clock implementations

This section discusses cryogenic implementations of 3-phase SCRL[4] and 2LAL[2] clocks. Fig. 10 visually depicts the clock requirements of both. Voltage ramps in the diagram are drawn with consistent slopes, allowing eyeball-level comparison of the relative speeds of the two approaches.

(a) 3-phase SCRL. ϕ's in blue, P's in red.



(b) 2LAL

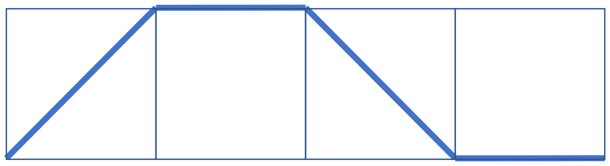
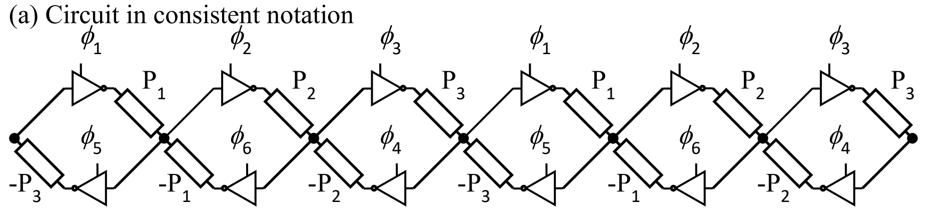


Fig. 10. Clocks for SCRL and 2LAL can be built from ramps. While SCRL has many more clocks than SCRL, only one split pair transitions at a time. This makes it possible to use a single pair of ramp waveforms from room temperature for ϕ, tying all clocks to a DC value when not undergoing transition. SCRL's Ps can have a faster ramp in a cryogenic environment without excessive power dissipation because the external wires only drive transistor gates. Furthermore, the Ps can be used as time dividers, leading to three or six (for dual rail) additional wires. The SCRL ramps only transit half the supply voltage whereas 2LAL ramps transit all of it. Thus, for the same slope, each SCRL ramp takes half the time. The diagram is a helpful heuristic of the sequence of ramps, which would be applied to different clocks. SCRL is still slower, but not as much as might be expected from a first glance.

As shown in fig. 10a, SCRL has six split-rail ϕ clock waveforms with a 0 V idle level and linear ramps in opposite directions to half the power supply voltage, later reversing the process. This yields the full supply voltage between the two signals yet reaching the full supply voltage in half the time as 2LAL.

Fig. 11a redraws 3-phase SCRL circuit in Ref. 4 rigorously and fig. 11b illustrates the sequence of transitions, revealing three-way symmetry with no more than one split-rail waveform making a transition at a time. In addition, the Ps taken as a 3-bit number undergo a gray-code count sequence that divides the timeline into six phases. This opens the possibility to a multiplexing scheme:

(a) Circuit in consistent notation



(b) Timing diagram, read left-to-right, top-to-bottom

P_1 ↑ 101	ϕ_1 on	ϕ_5 on	P_3 ↓ 001	ϕ_3 off	ϕ_4 off
P_2 ↑ 011	ϕ_2 on	ϕ_6 on	P_1 ↓ 010	ϕ_1 off	ϕ_5 off
P_3 ↑ 110	ϕ_3 on	ϕ_4 on	P_2 ↓ 100	ϕ_2 off	ϕ_6 off

Fig. 11. Three-phase SCRL (a) circuit design from Ref. 5 in consistent notation. (b) Timing sequence. The P values taken as a 3-bit number are shown in gray.

The split-rail ϕs can be generated by a pair of externally generated waveforms and a switching network in the cryogenic environment to route the waveform to the proper internal clock wire. The ideal solution would allow each internal clock wire to be connected to one of two DC voltages when not undergoing a ramp.

It makes a big difference in a cryogenic implementation whether heat is dissipated in the refrigerated environment or at room temperature. The blue lines at 45° in fig. 10a are all standard clock-power signals that will lead to dissipation in the cryogenic environment. However, the red P signals are generated at room temperature and terminate on transistor gates in the cryogenic environment, creating an almost purely capacitive load. While increasing the ramp rate of the P signals will increase dissipation, almost all of that dissipation will be in the signal generator that is at room temperature where it is not subject to cooling overhead. This allows optimizing SCRL throughput versus energy efficiency by increasing the ramp rate of the P signals.

Fig. 12 is a possible implementation of a 10-transistor clock demultiplexer. Based on the P signals, the circuit routes a clock waveform designated ϕ_k or the DC values ±V/2 or GND to the output at the bottom. The other clocks can be generated by rotating the P signals, applying suitable ϕ_k signals, and flipping the sign of the signal designated ±V/2.

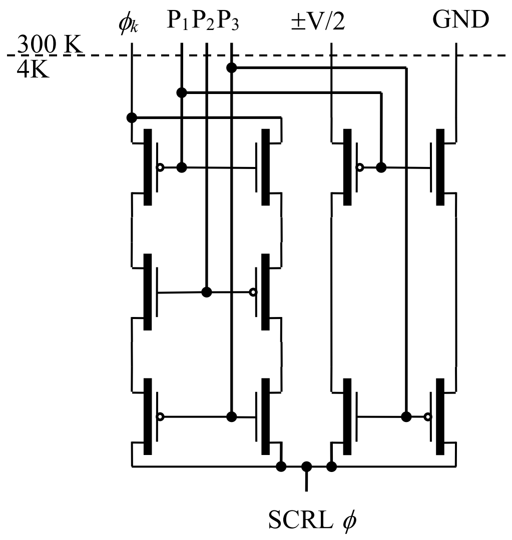


Fig. 12. SCRL 3ϕ signal demultiplexer. Signal generators at 300 K apply a gray code count sequence to $P_{1..3}$. The code selects a ramp waveform ϕ or one of two DC voltages ±V or GND. All the necessary signals can be generated with two waveforms, ±voltages, and three rotating permutations of $P_{1..3}$. The circuit's energy efficiency is due to (a) fast ramping voltages from 300 K are applied only to transistor gates and (b) voltages that go through transistor channels have slow ramps.

Fig. 10 conveys semi-quantitative information about the relative clock rate of SCRL and 2LAL. An SCRL cycle comprises 12 ϕ ramps of V/2 plus 6 P ramps of a steeper slope. In comparison, a 2LAL cycle has 4 ϕ ramps of V, which is equivalent to 8 ϕ ramps of V/2. This makes SCRL 33% slower by due to ϕs and slower still due to Ps. While this section explains important issues related to the length of a clock cycle, the amount of work accomplished per clock cycle varies between approaches, so the clock period is not the only consideration.

nFET-only Stages

The 2LAL adiabatic logic family[2, 4] is remarkably energy efficient but uses more transistors per equivalent function than CMOS. The larger transistor count is due in part to 2LAL's exclusive use of two-transistor T-gates.

This section describes a way to replace some T-gates by single transistors, albeit transistors with tailored (non-standard) or adjustable threshold voltages. Adjustable threshold voltages are a feature of Fully Depleted Silicon on Insulator (FD-SOI) processes.

The idea is to tune the clock voltages and devices on a per-stage basis, leading to the two scenarios in fig. 13, specifically with one nFET stage and with two. An nFET-only stage has half as many transistors as a normal stage, which is its advantage. In a shift register with the same number of T-gates in every stage, the two scenarios save 12.5% and 25% of overall device count for that circuit—which is not very much.

However, most logic is designed with a relatively large amount of combinational logic sandwiched between register stages. If standard logic design is applied to 2LAL and the nFET-only stages are used for the combinational logic, the overall device count savings could be much larger than 25%.

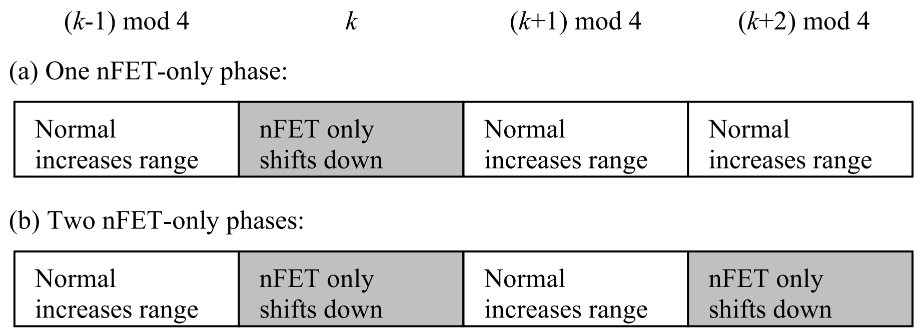


Fig. 13. The approach in this report deletes the pFETs from the pass gates associated with clock phase *k*, yielding an nFET-only stage. The alternative in (b) also deletes pFETs from stage *k*+2. nFET-only stages shift voltages, requiring adjustments to clocks

Deleting all the pFETs in a 2LAL stage, or clock phase, will shift voltages generated by the stage downward and reduce their range. Each 2LAL phase *k* = 0..3 connects only to phases ahead and behind, specifically (*k*+1) mod 4 and (*k*-1) mod 4. The normal 2LAL stages on each side can compensate for the shifted voltages by their natural ability to increase signal range. This report will quantify the voltage shifts.

The voltage shift and range reduction is caused by the voltage required to turn a transistor on, V_{on}, being larger than the voltage that leaves it off, V_{off}, which is strongly related to threshold voltages and subthreshold slope.

As will be quantified later, nFET-only stages need large-magnitude negative thresholds for pFETs and small-magnitude positive thresholds for the nFETs. While this could be achieved by tailoring (reengineering) a CMOS process, doing so would have undesirably high up-front costs. However, recent Fully Depleted Silicon on Insulator (FD-SOI) transistors have a fourth terminal connected to a thin Buried Oxide (BOX). This substrate bias terminal VB(V) alters the threshold voltage Vth(V) as illustrated in fig. 14. The circuit design strategy would be to bias the pFETs negatively and nFETs positively, yielding a threshold voltage spread of about 0.3 V (in magnitude; the sign of nFET and pFET thresholds are opposite in this case).

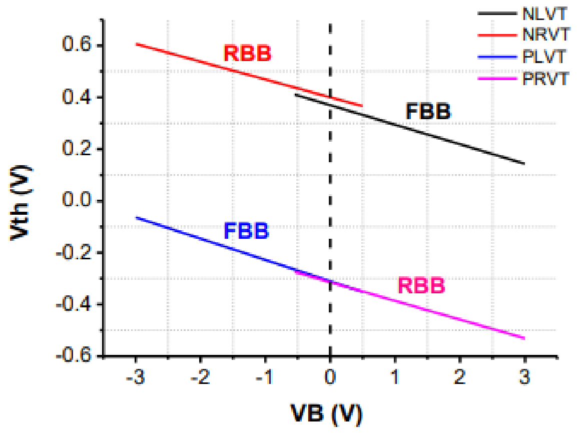


Fig. 14. From http://soiconsortium.eu/wp-content/uploads/2017/08/AnalogRF_28FDSOI_ACathelin_19092017.pdf

Cryogenic operation is also important to the utility of this idea. Cryogenic operation reduces subthreshold slope by more than a factor of 2 (for 4 K operation) and 10 (for 100 mK), reducing the difference between V_{on} and V_{off} and the amount of threshold variance required.

Circuit analysis

Two-transistor T-gates are necessary to drive the standard 2LAL waveforms to full amplitude, but let's consider circuit changes that will make the pFETs in phases 1 and 3 unnecessary. We will find the allowable voltage ranges for the clock waveforms algebraically, specifically finding one range for ϕ_1 and another range for ϕ_0 and ϕ_2. Given these clock ranges, we solve for transistor parameters $V^P{}_{on}$, $V^P{}_{off}$, $V^N{}_{on}$, and $V^N{}_{off}$ algebraically, where P and N represent pFET and nFET, and where the parameters represent the gate voltage that turns the transistors fully on and fully off. Note that V^P's are negative by convention.

Fig. 15a shows the minimum data swing for clock phase voltage ϕ_1, a stage with full T-gates. With $D_k{}^H$ and $D_k{}^L$ being the high and low voltages of ϕ_k and hence data signal k, $D_1{}^H - D_1{}^L$ must be large enough that both the nFET and pFET are fully turned on at the middle of the ramp, which requires a voltage range of $V^N{}_{on} - V^P{}_{on}$.

If the pFETs are removed from the phase 1 T-gates, turning them into single nFETs, the ϕ_1 voltage levels must be shifted and reduced in magnitude as illustrated in fig. 15b. If $d_0 = 1$ and the ϕ_1 waveform were to rise above $D_0{}^H - V^N{}_{on}$, the single nFET would not be fully turned on and unable to drive the d_1 waveform as required for full operating speed. If $d_0 = 0$ and the ϕ_1 waveform falls below $D_0{}^L - V^N{}_{off}$, the nFET will not be fully turned off, allowing the system to leak current backwards and create increased power dissipation. Thus, a phase with pFETs removed must be followed by a stage with clock range reduced by $V^N{}_{on} - V^N{}_{off}$, or shifted by the negative of that value $V^N{}_{off} - V^N{}_{on}$, and the DC value of the waveform shifted by $(V^N{}_{off} - V^N{}_{on})/2$.

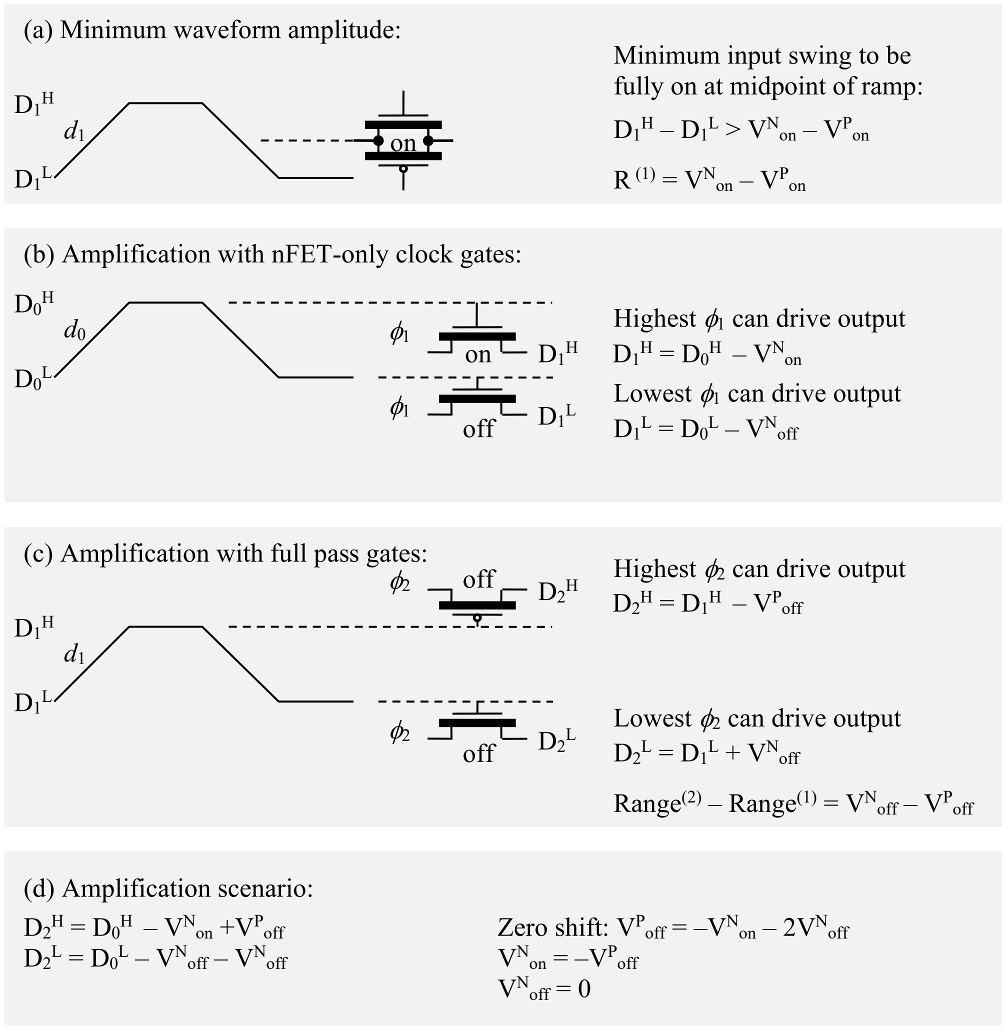

(a) Minimum waveform amplitude:
D_1^H
d_1
D_1^L
on
Minimum input swing to be fully on at midpoint of ramp:
$D_1^H - D_1^L > V^N_{on} - V^P_{on}$
$R^{(1)} = V^N_{on} - V^P_{on}$
(b) Amplification with nFET-only clock gates:
D_0^H
d_0
D_0^L
ϕ_1
on
D_1^H
ϕ_1
off
D_1^L
Highest ϕ_1 can drive output
$D_1^H = D_0^H - V^N_{on}$
Lowest ϕ_1 can drive output
$D_1^L = D_0^L - V^N_{off}$
(c) Amplification with full pass gates:
ϕ_2
off
D_2^H
D_1^H
d_1
D_1^L
ϕ_2
off
D_2^L
Highest ϕ_2 can drive output
$D_2^H = D_1^H - V^P_{off}$
Lowest ϕ_2 can drive output
$D_2^L = D_1^L + V^N_{off}$
$\text{Range}^{(2)} - \text{Range}^{(1)} = V^N_{off} - V^P_{off}$
(d) Amplification scenario:
$D_2^H = D_0^H - V^N_{on} + V^P_{off}$
$D_2^L = D_0^L - V^N_{off} - V^N_{off}$
Zero shift: $V^P_{off} = -V^N_{on} - 2V^N_{off}$
$V^N_{on} = -V^P_{off}$
$V^N_{off} = 0$

Fig. 15. Let D_k^L and D_k^H be the low and high levels of the clock and data at stage k, ϕ_k and d_k. Transistors will be fully conducting with gate voltages V^N_{on} and V^P_{on} and fully off with gate voltages V^N_{off} and V^P_{off}. (a) The minimum drive amplitude must be enough to turn on both transistors, which occurs at the middle of the ramp. (b) If clock gating is only nFETs, the clock (ϕ_1) may not rise closer to the input waveform than V^N_{on}, otherwise the transistor, expected to be on, will weaken. Likewise, if ϕ_1 drops more than V^N_{off} below the input waveform, the transistor, expected to be off, will leak. (c) A stage can amplify if the clock is gated by full pass gates, with the output waveform permitted to exceed the input by V^N_{off} in one direction and V^P_{off} in the other, before the gate begins to leak in the reverse direction.

Fig. 15c shows it is possible for a phase with full T-gates to increase the range of the data signal. If $d_1 = 1$, ϕ_2 may rise as much as $-V^P_{off}$ above D_1^H before the pFET is no longer fully turned off. Likewise, for $d_1 = 0$, ϕ_2 may drop to as much as V^N_{off} below D_1^L. Thus the stage allows the range to increase by $V^N_{off} - V^P_{off}$.

Thus, the condition for precisely restoring signal swing is $V^P_{off} = -V^N_{on} - 2V^N_{off}$, but V^P_{off} is negative, so a more convenient expression is $|V^P_{off}| > V^N_{on} + 2V^N_{off}$, supporting the point made previously that the pFETs need a high (magnitude) threshold compared to the nFETs. The roles of nFETs and pFETs can be reversed in principle, but nFETs tend to perform better than pFETs, making the description above more apt.

Cryogenic operation

Fig. 15d shows how the attenuation due to removing pFETs in stage 1 can be precisely offset by positive gain in the next stage. Theoretically, in the subthreshold range, $I_{on}/I_{off} = \exp(q(V_{on} - V_{off})/(kT))$, which increases exponentially as temperature drops. In a recent study,[6] subthreshold slope of a 40 nm CMOS process steepened from 88.2 to 27.7 mV/decade as temperature dropped from 300 K to 4 K, allowing a 3.2× reduction in $V^N_{on} - V^N_{off}$ for the same I_{on}/I_{off} ratio. The previous figures were for a device with a V_t of about 0.5 V. The exact relationship between V_t and V^N_{on} and V^N_{off} is beyond the scope of this report, but a 10^7 I_{on}/I_{off} ratio would correspond to about a 200 mV change in gate voltage, so plausible values are V_{on} = 0.5 V and V_{off} = 0 V. Applying the example in this report to this situation yields the example values in fig. 15d, showing a net gain of about 1.07×.

The same study[6] reported the subthreshold slope of a 160 nm CMOS process steepened from 87 to 9.9 mV/decade from 300 K to 100 mK, an 8.7× reduction. V_t was measured at 0.55 V. The 10^7 I_{on}/I_{off} ratio would correspond to about a 70 mV change in gate voltage, so plausible values are V_{on} = 0.5 V. Applying the example in this report to this situation yields a net gain of about 1.76×. (Alternate: V_{on} = 0.45 V and V_{off} = 0.38 V. Applying the example in this report to this situation yields a net gain of about 1.71×.)

Conclusions

This report supplements ref. 5 by providing additional detail and ideas on the use of cryogenic adiabatic transistor circuits for quantum computer control systems.

The ideas in this report extend both T-Gate shift registers[1]/2LAL[2, 3] and SCRL. The advantage of 2LAL is simplicity, but 2LAL circuits are not statically stable. This means that some data is defined by the voltage on a capacitor, which can leak—especially at the low frequencies characteristic of quantum computer control electronics. While 3-phase SCRL has more gates and clocks, all data is part of an active dual-inverter feedback loop at all times.

To build a quantum computer control system, the designer would first choose either 2LAL or SCRL. The methods in ref. 5 and this report explain how to implement a system that intelligently places some components at room temperature and other at a cryogenic temperature in order to maximize energy efficiency and minimize the amount of connection wire across the temperature gradient.

Acknowledgement

Mike Frank’s thesis, papers, and presentations provide an unusually broad perspective on adiabatic and reversible logic and its context in computing. While most authors have a favorite circuit, Mike’s thesis used SCRL, his attention shifted to 2LAL after he graduated, and (while not referenced in this report) he is now working on reversible superconductive electronics. Thus, if you hang around Mike long enough, you end up seeing these technologies as a continuum of circuits, materials, and operating temperatures. Mike’s broader view inspired some of the ideas in this document.

References

[1] Athas, William C. "Energy-recovery CMOS." Low Power Design Methodologies. Springer, Boston, MA, 1996. 65-100.

[2] V. Anantharam, M. He, K. Natarajan, H. Xie, and M. P. Frank. “Driving fully-adiabatic logic circuits using custom high-Q MEMS resonators,” in *Proc. Int. Conf. Embedded Systems and Applications and Proc. Int. Conf VLSI (ESA/VLSI)*. Las Vegas, NV, pp. 5-11.

[3] Zulehner, Alwin, Michael P. Frank, and Robert Wille. "Design automation for adiabatic circuits." *Proceedings of the 24th Asia and South Pacific Design Automation Conference*. ACM, 2019.

[4] Saed G. Younis. *Asymptotically Zero Energy Computing Using Split-Level Charge Recovery Logic*. No. AI-TR-1500. Massachusetts Institute of Technology Artificial Intelligence Laboratory, 1994.

[5] E. DeBenedictis, *New Design Principles for Cold, Scalable Electronics*. Technical report EPD001, http://www.zettaflops.org/CATC/.

[6] Incandela, Rosario M., et al. "Characterization and compact modeling of nanometer CMOS transistors at deep-cryogenic temperatures." *IEEE Journal of the Electron Devices Society* 6 (2018): 996-1006

[7] Frank, Michael Patrick, and Thomas F. Knight Jr. *Reversibility for efficient computing*. Diss. Massachusetts Institute of Technology, Dept. of Electrical Engineering and Computer Science, 1999..

Appendix: ngspice simulation of barrel shifter

While this report is not intended to represent the results of a research project, to assist reproducibility, this appendix includes code for ngspice simulation code that demonstrates the 4-bit barrel shifter in fig. 8 with the test harness and with 2LAL enhanced clocking from fig. 2b.

Fig. 16 shows the output from running the included code. The circuit is a five-stage, six-bit wide shift register with a barrel shifter in the last stage. The six-bit word width comprises a two-bit rotation count and a four-bit data word, as illustrated in fig. 8. Four of the five stages are loaded with the binary codes 0001, 0010, 0100, and 1000. The fifth state is all 1s, and corresponds to the red overlay markings in fig. 16. The binary codes rotate one position every five cycles. This is apparent by the black overlay arrows in fig. 16: the first bit after the red line moves up one position, and then jumps to the bottom.

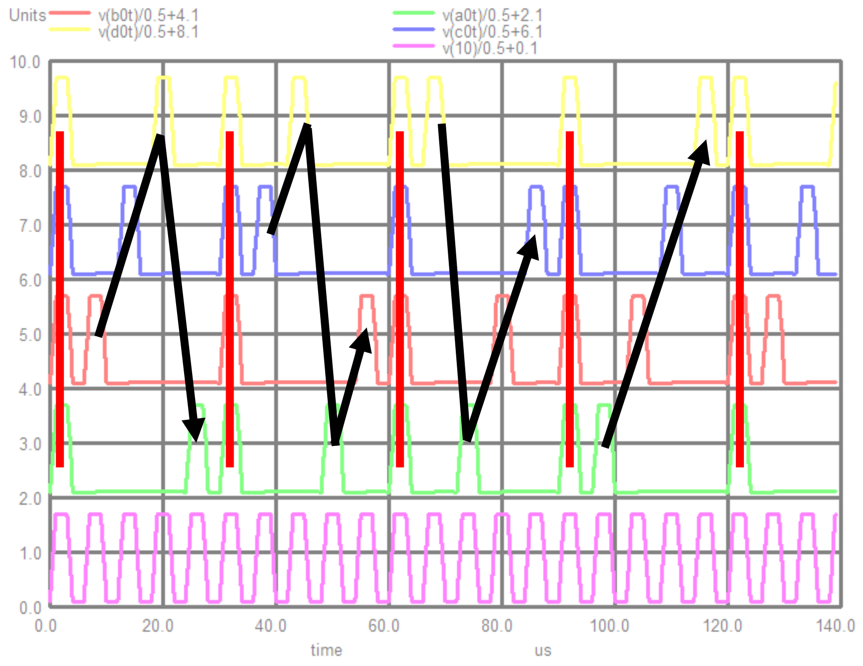


Fig. 16. Spice simulation of barrel rotator. Each gray line represents the start of a cycle. The pulse that starts each cycle moves down three traces (mod 4), which is actually up one trace. Bottom trace is a clock.

The source code appears below for reference. ngspice can run from single control file, so readers should be able to cut and paste the text below and run it under ngspice v. 30.

```
  2LALINVERTER

*** SUBCIRCUIT DEFINITIONS
.SUBCKT PASS D GT GC S psub nsub                          // Pass gate. Args: Drain GateT/C Source psub nsub
M1 D GT S nsub n1 W=.18u L=0.09u
M2 D GC S psub p1 W=.18u L=0.09u
.ENDS PASS

.SUBCKT PHASE iiT iiC ooT ooC p0T p0C p1T p1C psub nsub     // One phase of the 2LAL shift register. Args: iiT/C ooT/C clock0T/C clock1T/C
X1 iiT ooT ooC p0T psub nsub PASS
X2 iiC ooT ooC p0C psub nsub PASS
X3 ooT iiT iiC p1T psub nsub PASS
X4 ooC iiT iiC p1C psub nsub PASS
C1 iiT 0 5e-12
C2 iiC 0 5e-12
.ENDS PHASE

.SUBCKT DELAY d0T d0C d4T d4C                             // Four phases that just delay. Args: 2*{ data<n>T/C }
+ p0T p0C p1T p1C p1Tf p1Cf p2T p2C p3T p3C psub nsub       // clocks/power supplies
+ ini=0
.ic V(d0T)={gg} V(d1T) = {gg} V(d2T) = {   ini} V(d3T) = {   ini}
```

```
.ic V(d0C)={vv} V(d1C) = {vv} V(d2C) = {vv-ini} V(d3C) = {vv-ini}
X1 d0T d0C d1T d1C P0T P0C P1Tf P1Cf psub nsub PHASE
X2 d1T d1C d2T d2C P1Tf P1Cf p2T p2C psub nsub PHASE
X3 d2T d2C d3T d3C p2T p2C p3T p3C psub nsub PHASE
X4 d3T d3C d4T d4C p3T p3C P0T P0C psub nsub PHASE
.ENDS DELAY

.SUBCKT DELAY1 d0T d0C d1T d1C x1T x1C d4T d4C              // Four phases that delay, but tapping phase 1. Args: 4*{ data<n>T/C }
+ p0T p0C p1T p1C p2T p2C p3T p3C psub nsub                 // clocks/power supplies. Includes either phase 1 slow or fast, but not both
+ ini=0
.ic V(d0T)={gg} V(x1T) = {gg} V(d2T) = {   ini} V(d3T) = {   ini}
.ic V(d0C)={vv} V(x1C) = {vv} V(d2C) = {vv-ini} V(d3C) = {vv-ini}
X1 d0T d0C d1T d1C P0T P0C P1T P1C psub nsub PHASE
X2 x1T x1C d2T d2C P1T P1C p2T p2C psub nsub PHASE
X3 d2T d2C d3T d3C p2T p2C p3T p3C psub nsub PHASE
X4 d3T d3C d4T d4C p3T p3C P0T P0C psub nsub PHASE
.ENDS DELAY1

.SUBCKT MUX2 a1T a1C b1T b1C c1T c1C x1T x1C psub nsub      // One input, three bidirectional, T/C. Args: addrT/C in<0..1>T/C outT/C
X1  c1T a1T a1C x1T psub nsub PASS
X2  c1C a1T a1C x1C psub nsub PASS
X3  b1T a1C a1T x1T psub nsub PASS
X4  b1C a1C a1T x1C psub nsub PASS
.ENDS

.SUBCKT FREDKIN a0T a0C b0T b0C c0T c0C                     // Three inputs, T/C. Args: 5*{ data<n>T/C }
+ a4T a4C b4T b4C c4T c4C           // Three outputs, T/C; first two delayed copies of arguments, third is a specified function of the first two
+ p0T p0C p1T p1C p1Tf p1Cf p2T p2C p3T p3C psub nsub       // clocks/power supplies
+ inia=0 inib=0 inic=0
X1 a0T a0C a1T a1C a1T a1C a4T a4C p0T p0C p1T  p1C  p2T p2C p3T p3C psub nsub DELAY1 ini=inia
X2 b0T b0C b1T b1C x1T x1C b4T b4C p0T p0C p1Tf p1Cf p2T p2C p3T p3C psub nsub DELAY1 ini=inib
X3 c0T c0C c1T c1C y1T y1C c4T c4C p0T p0C p1Tf p1Cf p2T p2C p3T p3C psub nsub DELAY1 ini=inic
X4 a1T a1C b1T b1C c1T c1C x1T x1C psub nsub MUX2
X5 a1T a1C c1T c1C b1T b1C y1T y1C psub nsub MUX2
.ENDS FREDKIN

.SUBCKT ROT3 a0T a0C b0T b0C c0T c0C d0T d0C                // Three inputs, T/C. Args: 5*{ data<n>T/C }
+ a4T a4C b4T b4C c4T c4C d4T d4C   // Three outputs, T/C; first two delayed copies of arguments, third is a specified function of the first two
+ p0T p0C p1T p1C p1Tf p1Cf p2T p2C p3T p3C psub nsub       // clocks/power supplies
+ inia=0 inib=0 inic=0 inid=0
X1 a0T a0C a1T a1C a1T a1C a4T a4C p0T p0C p1T  p1C  p2T p2C p3T p3C psub nsub DELAY1 ini=inia
X2 b0T b0C b1T b1C x1T x1C b4T b4C p0T p0C p1Tf p1Cf p2T p2C p3T p3C psub nsub DELAY1 ini=inib
X3 c0T c0C c1T c1C y1T y1C c4T c4C p0T p0C p1Tf p1Cf p2T p2C p3T p3C psub nsub DELAY1 ini=inic
X4 d0T d0C d1T d1C z1T z1C d4T d4C p0T p0C p1Tf p1Cf p2T p2C p3T p3C psub nsub DELAY1 ini=inid
X5 a1T a1C b1T b1C c1T c1C x1T x1C psub nsub MUX2
X6 a1T a1C c1T c1C d1T d1C y1T y1C psub nsub MUX2
X7 a1T a1C d1T d1C b1T b1C z1T z1C psub nsub MUX2
.ENDS ROT3

.SUBCKT ROT4 a0T a0C b0T b0C c0T c0C d0T d0C e0T e0C        // Three inputs, T/C. Args: 5*{ data<n>T/C }
+ a4T a4C b4T b4C c4T c4C d4T d4C e4T e4C// Three outputs, T/C; first two delayed copies of arguments, third is a specified function of the first two
+ p0T p0C p1T p1C p1Tf p1Cf p2T p2C p3T p3C psub nsub       // clocks/power supplies
+ inia=0 inib=0 inic=0 inid=0 inie=0
X1 a0T a0C a1T a1C a1T a1C a4T a4C p0T p0C p1T  p1C  p2T p2C p3T p3C psub nsub DELAY1 ini=inia
X2 b0T b0C b1T b1C w1T w1C b4T b4C p0T p0C p1Tf p1Cf p2T p2C p3T p3C psub nsub DELAY1 ini=inib
X3 c0T c0C c1T c1C x1T x1C c4T c4C p0T p0C p1Tf p1Cf p2T p2C p3T p3C psub nsub DELAY1 ini=inic
X4 d0T d0C d1T d1C y1T y1C d4T d4C p0T p0C p1Tf p1Cf p2T p2C p3T p3C psub nsub DELAY1 ini=inid
X5 e0T e0C e1T e1C z1T z1C e4T e4C p0T p0C p1Tf p1Cf p2T p2C p3T p3C psub nsub DELAY1 ini=inie
X6 a1T a1C b1T b1C c1T c1C w1T w1C psub nsub MUX2
X7 a1T a1C c1T c1C d1T d1C x1T x1C psub nsub MUX2
X8 a1T a1C d1T d1C e1T e1C y1T y1C psub nsub MUX2
X9 a1T a1C e1T e1C b1T b1C z1T z1C psub nsub MUX2
.ENDS ROT4

.SUBCKT ROT2x4 i0T i0C j0T j0C                              // Args: Two address inputs, i0, j0, T/C.
+ a0T a0C b0T b0C c0T c0C d0T d0C                           // Four inputs, a0, b0, c0, d0, T/C.
+ i4T i4C j4T j4C                                           // Two address outputs, i4, j4, T/C, copies of inputs
+ a4T a4C b4T b4C c4T c4C d4T d4C                           // Four outputs, a4, b4, c4, d4, T/C; circular rotation of inputs
+ p0T p0C p1T p1C p1Tf p1Cf p2T p2C p3T p3C psub nsub       // clocks/power supplies
+ inii=0 inij=0 inia=0 inib=0 inic=0 inid=0
X1 i0T i0C i1T i1C i1T i1C i4T i4C p0T p0C p1T  p1C  p2T p2C p3T p3C psub nsub DELAY1 ini=inii
X2 j0T j0C j1T j1C j1T j1C j4T j4C p0T p0C p1T  p1C  p2T p2C p3T p3C psub nsub DELAY1 ini=inij
X3 a0T a0C a1T a1C w1T w1C a4T a4C p0T p0C p1Tf p1Cf p2T p2C p3T p3C psub nsub DELAY1 ini=inia
X4 b0T b0C b1T b1C x1T x1C b4T b4C p0T p0C p1Tf p1Cf p2T p2C p3T p3C psub nsub DELAY1 ini=inib
X5 c0T c0C c1T c1C y1T y1C c4T c4C p0T p0C p1Tf p1Cf p2T p2C p3T p3C psub nsub DELAY1 ini=inic
X6 d0T d0C d1T d1C z1T z1C d4T d4C p0T p0C p1Tf p1Cf p2T p2C p3T p3C psub nsub DELAY1 ini=inid
X7 i1T i1C a1T a1C b1T b1C q1T q1C psub nsub MUX2
X8 i1T i1C b1T b1C c1T c1C r1T r1C psub nsub MUX2
X9 i1T i1C c1T c1C d1T d1C s1T s1C psub nsub MUX2
X10 i1T i1C d1T d1C a1T a1C t1T t1C psub nsub MUX2
X11 j1T j1C q1T q1C s1T s1C w1T w1C psub nsub MUX2
X12 j1T j1C r1T r1C t1T t1C x1T x1C psub nsub MUX2
X13 j1T j1C s1T s1C q1T q1C y1T y1C psub nsub MUX2
X14 j1T j1C t1T t1C r1T r1C z1T z1C psub nsub MUX2
.ENDS ROT2x4

.SUBCKT f_XOR a1T a1C b1T b1C O1T O1C p1Tf p1Cf             // Core XOR circuit. Args: 4*{ data<n>T/C }
+ psub nsub
.ic V(O1T)={gg}                                             // f_ functions are deasserted at initialization
.ic V(O1C)={vv}
X1 q1T a1T a1C p1Tf psub nsub PASS
X2 q1C a1T a1C p1Cf psub nsub PASS
X3 O1T b1C b1T q1T psub nsub PASS
X4 O1C b1C b1T q1C psub nsub PASS
X5 z1T a1C a1T p1Tf psub nsub PASS
X6 z1C a1C a1T p1Cf psub nsub PASS
X7 O1T b1T b1C z1T psub nsub PASS
X8 O1C b1T b1C z1C psub nsub PASS
C1 O1T 0 5e-12
C2 O1C 0 5e-12
.ENDS f_XOR

.SUBCKT f_XNOR a1T a1C b1T b1C O1T O1C p1Tf p1Cf            // Core XNOR circuit. Args: 4*{ data<n>T/C }
+ psub nsub
.ic V(O1T)={gg}                                             // f_ functions are deasserted at initialization
.ic V(O1C)={vv}
X1 q1T a1C a1T p1Tf psub nsub PASS
X2 q1C a1C a1T p1Cf psub nsub PASS
X3 O1T b1C b1T q1T psub nsub PASS
X4 O1C b1C b1T q1C psub nsub PASS
X5 z1T a1T a1C p1Tf psub nsub PASS
X6 z1C a1T a1C p1Cf psub nsub PASS
X7 O1T b1T b1C z1T psub nsub PASS
X8 O1C b1T b1C z1C psub nsub PASS
```

```
C1 O1T 0 5e-12
C2 O1C 0 5e-12
.ENDS f_XNOR

.SUBCKT f_AND a1T a1C b1T b1C O1T O1C p1Tf p1Cf             // Core AND circuit. Args: 4*{ data<n>T/C }
+ psub nsub
.ic V(O1T)={gg}                                             // f_ functions are deasserted at initialization
.ic V(O1C)={vv}
X1 q1T a1T a1C p1Tf psub nsub PASS
X2 q1C a1T a1C p1Cf psub nsub PASS
X3 O1T b1T b1C q1T psub nsub PASS
X4 O1C b1T b1C q1C psub nsub PASS
C1 O1T 0 5e-12
C2 O1C 0 5e-12
.ENDS f_AND

.SUBCKT f_OR a1T a1C b1T b1C O1T O1C p1Tf p1Cf              // Core OR circuit. Args: 4*{ data<n>T/C }
+ psub nsub
.ic V(O1T)={gg}                                             // f_ functions are deasserted at initialization
.ic V(O1C)={vv}
X1 O1T a1T a1C p1Tf psub nsub PASS
X2 O1C a1T a1C p1Cf psub nsub PASS
X3 O1T b1T b1C p1Tf psub nsub PASS
X4 O1C b1T b1C p1Cf psub nsub PASS
C1 O1T 0 5e-12
C2 O1C 0 5e-12
.ENDS f_OR

.SUBCKT FUNC a0T a0C b0T b0C                                // Two inputs, T/C. Args: 5*{ data<n>T/C }
+ a4T a4C b4T b4C FnT FnC           // Three outputs, T/C; first two delayed copies of arguments, third is a specified function of the first two
+ p0T p0C p1T p1C p1Tf p1Cf p2T p2C p3T p3C psub nsub       // clocks/power supplies
+ ini=0
.ic             V(i1T) = {gg} V(x2T) = {   ini} V(x3T) = {   ini}
.ic             V(i1C) = {vv} V(x2C) = {vv-ini} V(x3C) = {vv-ini}
X1 a0T a0C d1T d1C d1T d1C a4T a4C p0T p0C p1T p1C p2T p2C p3T p3C psub nsub DELAY1 ini=ini
X2 b0T b0C b1T b1C b1T b1C b4T b4C p0T p0C p1T p1C p2T p2C p3T p3C psub nsub DELAY1 ini=ini
X3 d1T d1C b1T b1C i1T i1C p1Tf p1Cf psub nsub f_XNOR       // *** Enter circuit for desired function
X4 i1T i1C x2T x2C p1Tf p1Cf p2T p2C psub nsub PHASE
X5 x2T x2C x3T x3C p2T p2C p3T p3C psub nsub PHASE
X6 x3T x3C FnT FnC p3T p3C p0T p0C psub nsub PHASE
.ENDS FUNC

.SUBCKT CNUF a0T a0C b0T b0C FnT FnC                        // Three inputs, T/C; third is a specified function of the first two. Args: 5*{ data<n>T/C }
+ a4T a4C b4T b4C                                           // Two outputs, T/C; copies of first two inputs
+ p0T p0C p1T p1C p1Tf p1Cf p2T p2C p3T p3C psub nsub       // clocks/power supplies
+ ini=0
.ic V(a0T)={gg}
.ic V(a0C)={vv}
X1 a0T a0C d1T d1C d1T d1C a4T a4C p0T p0C p1T p1C p2T p2C p3T p3C psub nsub DELAY1
X2 b0T b0C b1T b1C b1T b1C b4T b4C p0T p0C p1T p1C p2T p2C p3T p3C psub nsub DELAY1
X3 d1T d1C b1T b1C i1T i1C p1Tf p1Cf psub nsub f_XNOR       // *** Enter circuit for desired function
X4 FnT FnC i1T i1C p0T p0C p1T p1C psub nsub PHASE
.ENDS CNUF

.SUBCKT LINEARSHIFT d0T d0C                                 // Four full cycle delay. Args: data in/out T/C
+ e0T e0C                                                   // stuff
+ p0T p0C p1T p1C p1Tf p1Cf p2T p2C p3T p3C psub nsub       // clocks/power supplies
+ ini0=0 ini1=0 ini2=0 ini3=0
X1 d0T d0C d4T d4C     p0T p0C p1T p1C p1Tf p1Cf p2T p2C p3T p3C     psub nsub DELAY ini=ini3
X2 d4T d4C d8T d8C     p0T p0C p1T p1C p1Tf p1Cf p2T p2C p3T p3C     psub nsub DELAY ini=ini2
X3 d8T d8C dCT dCC     p0T p0C p1T p1C p1Tf p1Cf p2T p2C p3T p3C     psub nsub DELAY ini=ini1
X4 dCT dCC e0T e0C     p0T p0C p1T p1C p1Tf p1Cf p2T p2C p3T p3C     psub nsub DELAY ini=ini0
.ENDS LINEARSHIFT

*** POWER
VCC   99  0   DC {vv}
VGG   98  0   DC {gg}

*** ALL INPUTS
.param gg=0V
.param vv=.8V

.param ticks=139                                            // number of ticks in the simulation
.param tick=1000NS                                          // time of a tick
.param tstep=50NS                                           // time of a simulation step, so number of steps is tick*ticks/tstep
.param ttn=18000ns                                          // integration time for energy

*** CLOCKS -- Original 4 clock phases and inverses (total four unique signal), but with slow and fast phase 1's (total six unique signals)
vph0  10  0   DC {gg} PWL({0*tick} {gg} {1*tick} {vv} {2*tick} {vv} {3*tick} {vv} {4*tick} {gg} {5*tick} {gg} {6*tick} {gg}) r={0*tick}
vp1s  11  0   DC {gg} PWL({0*tick} {gg} {1*tick} {gg} {2*tick} {vv} {3*tick} {vv} {4*tick} {vv} {5*tick} {vv} {6*tick} {gg}) r={0*tick}
vp1f  12  0   DC {gg} PWL({0*tick} {gg} {1*tick} {gg} {2*tick} {gg} {3*tick} {vv} {4*tick} {vv} {5*tick} {gg} {6*tick} {gg}) r={0*tick}
vph2  13  0   DC {vv} PWL({0*tick} {vv} {1*tick} {gg} {2*tick} {gg} {3*tick} {gg} {4*tick} {vv} {5*tick} {vv} {6*tick} {vv}) r={0*tick}
vph3  14  0   DC {vv} PWL({0*tick} {vv} {1*tick} {vv} {2*tick} {gg} {3*tick} {gg} {4*tick} {gg} {5*tick} {gg} {6*tick} {vv}) r={0*tick}

vqh0  20  0   DC {vv} PWL({0*tick} {vv} {1*tick} {gg} {2*tick} {gg} {3*tick} {gg} {4*tick} {vv} {5*tick} {vv} {6*tick} {vv}) r={0*tick}
vq1s  21  0   DC {vv} PWL({0*tick} {vv} {1*tick} {vv} {2*tick} {gg} {3*tick} {gg} {4*tick} {gg} {5*tick} {gg} {6*tick} {vv}) r={0*tick}
vq1f  22  0   DC {vv} PWL({0*tick} {vv} {1*tick} {vv} {2*tick} {vv} {3*tick} {gg} {4*tick} {gg} {5*tick} {vv} {6*tick} {vv}) r={0*tick}
vqh2  23  0   DC {gg} PWL({0*tick} {gg} {1*tick} {vv} {2*tick} {vv} {3*tick} {vv} {4*tick} {gg} {5*tick} {gg} {6*tick} {gg}) r={0*tick}
vqh3  24  0   DC {gg} PWL({0*tick} {gg} {1*tick} {gg} {2*tick} {vv} {3*tick} {vv} {4*tick} {vv} {5*tick} {vv} {6*tick} {gg}) r={0*tick}

*** TOP-LEVEL CIRCUIT
X1 i0T i0C i1T i1C                              10 20 11 21 12 22 13 23 14 24 vv gg LINEARSHIFT ini0=vv ini1=vv ini2=vv ini3=vv     // 1's bit
X2 j0T j0C j1T j1C                              10 20 11 21 12 22 13 23 14 24 vv gg LINEARSHIFT ini0=vv ini1=vv ini2=vv ini3=vv     // 2's bit
X3 a0T a0C a1T a1C                              10 20 11 21 12 22 13 23 14 24 vv gg LINEARSHIFT ini0=vv ini1=gg ini2=gg ini3=gg
X4 b0T b0C b1T b1C                              10 20 11 21 12 22 13 23 14 24 vv gg LINEARSHIFT ini0=gg ini1=vv ini2=gg ini3=gg
X5 c0T c0C c1T c1C                              10 20 11 21 12 22 13 23 14 24 vv gg LINEARSHIFT ini0=gg ini1=gg ini2=vv ini3=gg
X6 d0T d0C d1T d1C                              10 20 11 21 12 22 13 23 14 24 vv gg LINEARSHIFT ini0=gg ini1=gg ini2=gg ini3=vv
X7 i1T i1C j1T j1C a1T a1C b1T b1C c1T c1C d1T d1C
+ i0T i0C j0T j0C a0T a0C b0T b0C c0T c0C d0T d0C
+ 10 20 11 21 12 22 13 23 14 24 vv gg ROT2x4 inii=gg inij=gg inia=vv inib=vv inic=vv inid=vv

* power and energy calculation
B4 0 16 V=I(vph0)*v(10)+I(vp1s)*v(11)+I(vp1f)*v(12)+I(vph2)*v(13)+I(vph3)*v(14)+I(vqh0)*v(20)+I(vq1s)*v(21)+I(vq1f)*v(22)+I(vqh2)*v(23)+I(vqh3)*v(24)
A1 16 17 power_tally
.model power_tally int(in_offset=0.0 gain=1.0 out_lower_limit=-1e12 out_upper_limit=1e12 limit_range=1e-9 out_ic=0.0)

.option noinit acct
.TRAN {tstep} {ticks*tick}

* use BSIM3 model with default parameters
.include ./modelcard.nmos
.include ./modelcard.pmos

.control
```

```
pre_set strict_errorhandling
unset ngdebug
run

* measure power consumption
meas tran Energy1us INTEG v(16) from=0 to=5us
meas tran EnergyLev INTEG v(16) 'from=5us to=ttn'

**********************************************
plot v(16)                                              // plot instantaneous energy consumption
plot v(17)                                              // plot accumulated energy dissipation

* white background
set color0=white
* black grid and text (only needed with X11, automatic with MS Win)
set color1=black
* wider grid and plot lines
set xbrushwidth=3

plot
+ v(a0T)/0.5+2.1 v(b0T)/0.5+4.1 v(c0T)/0.5+6.1 v(d0T)/0.5+8.1           // non-inverted output
+ v(10)/0.5+0.1                                         // phase 0 clock

.endc

.END


.
```