Logarithmic Dissipation Shift Register

Technical note (report) ZF006, October 10, 2020
Erik P. DeBenedictis
Zettaflops, LLC, Albuquerque, NM 87112
erikdebenedictis@zettaflops.org

Abstract

This note proposes a new use case for adiabatic transistor circuits, specifically low energy cryogenic memory. Adiabatic transistor circuits, such as SCRL,[1] 2LAL,[2] and S2LAL[3] have been touted for low-energy logic circuits.[4] They have also been proposed for cryogenic logic, such as the classical control systems of a quantum computer.[5] This note elaborates on novel design concept used in cryogenic waveform storage. The concept includes using a ladder of multiple clock rates that exploits the property of increasing energy efficiency as function of clock rate.

Introduction

Fig. 1 shows a storage subsystem for a digitized waveform,[5] or an instance of a sequential access memory. The example shows the bits stored in a series of large but slow shift registers whose outputs are combined into a smaller number of faster streams.

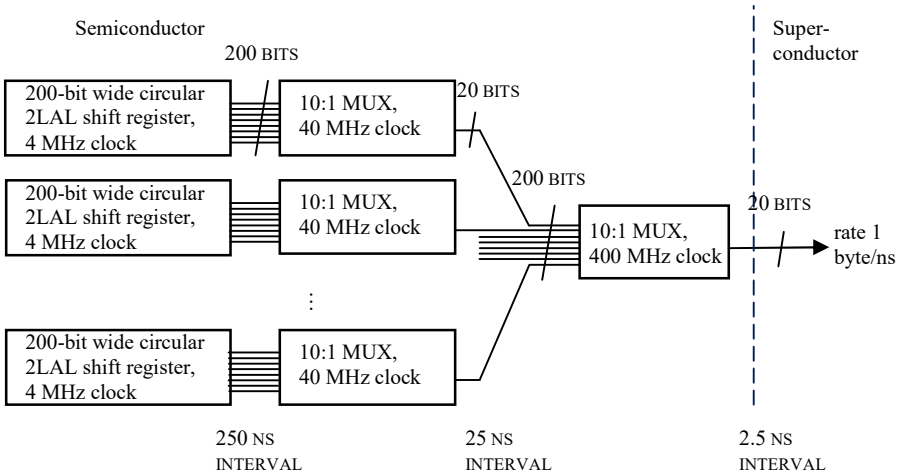


Fig. 1. Hybrid subsystem for sequential storage. The adiabatic transistors are physically small but muse be slowed down to increase energy efficiency. However, the multiplexers speed up the data rate. The Josephson junctions on the right are physically large and hence a poor choice for storage, but they are fast and energy efficiency, making them suitable for the final multiplexing step.

The ladder of clock rates would not have much value for a CMOS circuit, but adiabatic circuits become more energy efficient as the clock rate slows down, as illustrated in fig. 2. The figure shows the energy per transistor drops quadratically with the clock period, i. e. inverse clock rate. The large-capacity shift registers in fig. 1 are clocked at 4 MHz, where fig. 2 shows they dissipate about 10^{-7} as much as CMOS at full speed. The multiplexers in fig. 1 that raise the clock rate will have more dissipation per device in accordance with the curve in fig. 2. However, there are fewer transistors operating at the higher dissipation levels. As will be detailed later, the left of fig. 1 has a many bits while the right has high speed, giving the effect of many bits at high speed.

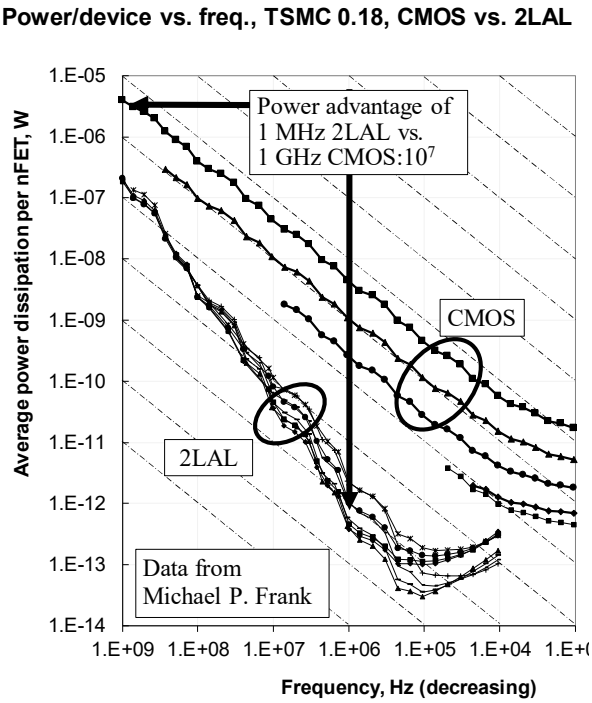


Fig. 2. Comparison of circuit efficiency for standard CMOS (top) and an adiabatic circuit 2LAL (bottom), showing a maximum $10^7\times$ power advantage about 1 MHz. The relative positions of the curves are due to the circuit, but the absolute positions will vary by the specific transistor parameters. The upward slope of the curves is due to a combination of transistor parameters and instability in Spice. If a transistor were optimized for 2LAL, the downward sloping section could continue.

The logarithmic dissipation shift register that is the topic of this document has advantages, but the complete example in ref. 5 also includes a second, Josephson junction-based technology for even higher speed. The hybrid technology is out of the scope of this document and the reader is referred to ref. 5.

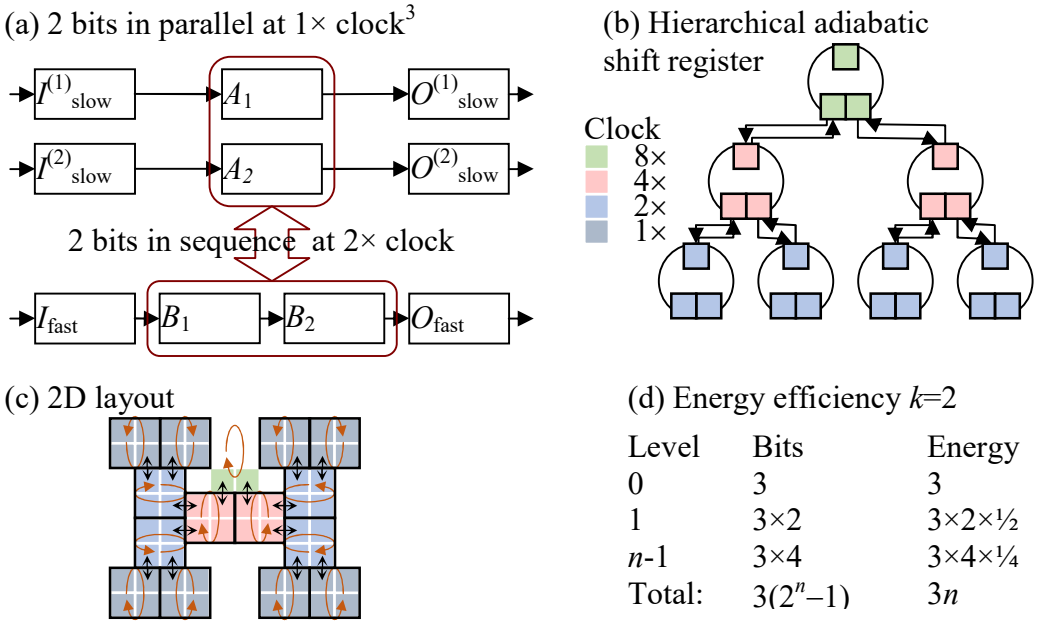
This document introduces and demonstrates several new adiabatic design principles:

- To the best of the author's knowledge, there are no adiabatic circuits in the literature that include multiple clocks at different speeds.
- This note further describes how to use a ladder of different speed vs. energy efficiency tradeoffs as a design technique for adiabatic circuits.
- The logarithmic dissipation shift register illustrates how to use adiabatic transistor circuits designed and demonstrated for logic to create a memory that should be superior in certain applications.
- This document includes an appendix with ngspice simulation code.

Logarithmic Shift Register

In lieu of the clock period steps of 250 ns, 25 ns, and 2.5 ns in fig. 1, let us consider a homogeneous *n*-level structure. For simplicity, let us assume the clock rate change is 2× between each pair of levels.

Fig. 3a illustrates a 2× clock rate change between shift registers. Its operation is based on the fact that there is a point in the clocking of all adiabatic transistorized shift registers (that the author is aware of) where the data is entirely contained in a stage. At this point, all the inputs, outputs, and clocks are at a voltage that independent of the stored data, and sometimes in a high-impedance state. It is possible to logically relocate the shift register stage and the information it contains using pass gates. Since the voltages are independent of stored data, the relocation can be performed without any voltages changing and therefore without dissipation. If the relocation is implemented as the swapping of two shift register stages, including their data, the resulting system will naturally obey higher level properties required for reversible and adiabatic logic design. The diagram illustrates a binary tree with a 2× clock rate change per stage, but it should be clear how to extend the fanout to other values.



Level	Bits	Energy
0	3	3
1	3×2	3×2×½
n-1	3×4	3×4×¼
Total:	$3(2^n-1)$	$3n$

Fig. 3. The utility of clock speed shifting. (a) The clock-rate/parallelism changing circuit. (b) Imagine the green blocks comprise a cyclic shift register. The shift register would become longer if in every other clock cycle two adjacent bits were swapped with a red block in one of two additional cyclic shift registers, which are clocked at ½ the speed. This could repeat for blue blocks that are part of four cyclic shift registers, which are clocked at ¼ the speed, etc. (c) A 2D layout of the structure with four levels. (d) The energy efficiency will be better than CMOS in some cases. Each level of the hierarchy has more bits than the previous but higher energy efficiency, so dissipation per level is the same. This would lead to energy per shift being logarithmic in the total number of bits.

Fig. 3b uses the clock rate changer to create a hierarchical shift register. Each of the circles represents a 3-bit cyclic shift register, with each colored square representing one adiabatic shift register stage, holding one bit, clocked at the rate shown in the legend.

In the absence of any bit exchanges by the circuit in fig. 3a, the entire system in fig. 3b would be a series of independent 3-bit cyclic shift registers clocked at the rates shown in the legend.

Now consider each cyclic shift register, between each shift, exchanging one bit with the register above it. Furthermore, after every second shift, two adjacent bits are exchanged with the two registers below. Thus, the bits originally in the green squares will no longer complete a cycle in three shifts, but each bit will go down either the left or right subtree until reaching a leaf node and then climb back up. Since both the left and right branches have the same structure, each pair of bits shifted from green to red will return at the same time—thus preserving the order of bits.

From the perspective of the green register at the top of the tree, the entire structure below it simply serves to make the register longer. If the tree has n levels of fanout k, meaning the cyclic registers were of length k+1, the effective length of the register would be $(k+1)(k^n - 1)$.

Fig. 3c shows a more intelligent 2D layout. It should be noted that the number of bits is exponential in the number of levels, so beyond a point, there will not be enough room in the 2D plane to accommodate the tree without long wires to connect shift register stages.

Fig. 3d computes the number of bits stored in an n-level structure and the energy per shift. The key point is that the number of cyclic registers at each level is the same as the energy efficiency increase due to the slower clock, so the total energy of each level is the same. Thus, for a register of capacity N, the dissipation per shift is O(log N). By comparison, a standard CMOS register has dissipation O(N) and a 2D array structure such as a DRAM or SRAM would be O($\sqrt{N}$).

Test Circuit

The circuit in fig. 3a has been coded in ngspice based on S2LAL with the basic operation illustrated in fig. 4. The code appears in the appendix.

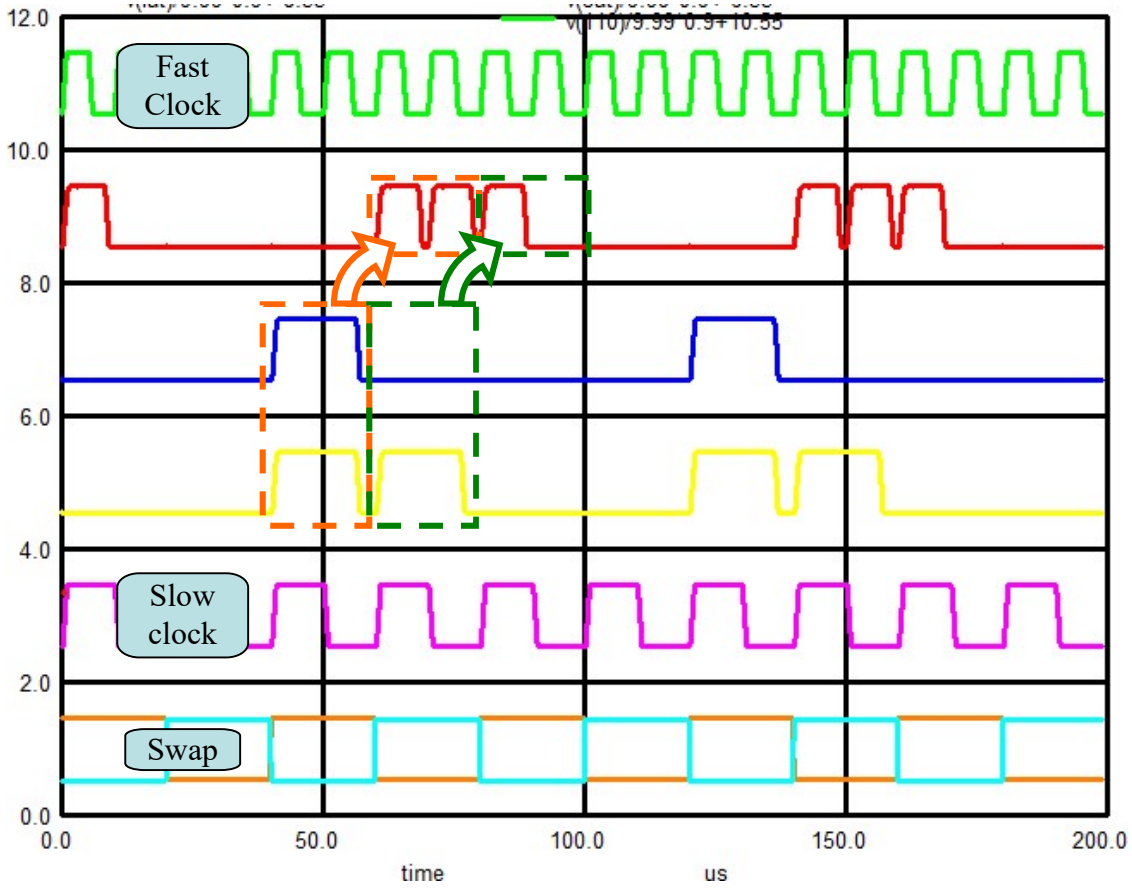


Fig. 4. Simulation of clock rate change circuit. Fast clock, slow clock, and swap lines are labeled. The red top data trace at rate 2× is synthesized from blue and yellow data traces at rate 1×. The diagram shows two parallel bits of data at rate 1× swapping into two serial bits of data at rate 2×, demonstrating th key step in this note.

The S2LAL circuits are powered by a 2× clock in green and a 1× clock in purple, with the lower traces labeled swap causing an interchange of the bit values. In this circuit, bits in the blue and yellow traces are swapped so they are serialized in the red trace. The circuit is in fact a cycle similar to fig. 3b (but just two levels), so the data pattern repeats.

Conclusions

This note shows how adiabatic transistors circuits occupying different rungs on a ladder of different speed vs. energy efficiency tradeoffs can achieve better results than is possible with CMOS or adiabatic circuits of a single speed.

The logarithmic shift register shows how adiabatic transistor circuits that are in the literature as an energy efficient logic family can also be used to create a memory with certain benefits.

Ref. 5 shows the logarithmic shift register with a final stage built using Josephson junction technology. Transistors are physically smaller than Josephson junctions, leading memories constructed from the latter to have lower density. Therefore, this note shows how to construct an adiabatic memory with a storage density similar to that of other transistor circuits, as fast as CMOS, but with higher energy efficiency.

References

[1] 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.

[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] Frank, Michael P., et al. "Reversible Computing with Fast, Fully Static, Fully Adiabatic CMOS." *arXiv preprint arXiv:2009.00448* (2020).

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

[5] DeBenedictis, Erik P. "*Quantum Computer Control using Novel, Hybrid Semiconductor-Superconductor Electronics*." arXiv preprint arXiv:1912.11532 (2019).

Appendix: ngspice files

s2lal.cir

```
    Ratex
* S2LAL initial test setup. Demonstrates a 2x rate change.
* S2LAL circuit from:
* Frank, Michael P., et al. "Reversible Computing with Fast, Fully Static, Fully Adiabatic CMOS." arXiv preprint arXiv:2009.00448 (2020).
* Contains Athas's adiabatic amplifier from:
* Athas, W. C., et al. "Low-power digital systems based on adiabatic-switching principles." IEEE Transactions on VLSI Systems 2.4 (1994): 398-407
* Tested with ngspice-30 (creation date Dec 28, 2018, from ngspice-30_64.zip 8,687,648 bytes)
* (NOT TESTED RECENTLY) Also works with WRSPICE, except that the .control block is different for the two and has to be switched back and forth
* For tutorial docs: no tabs; comments start column 61; 169 character maximum line length

.param WRSPICE_PROGRAM=0                                $ From WRspice manual: This enables users to include WRspice-specific input in SPICE files...
.if (WRSPICE_PROGRAM=1)                                 $ WRspice builtin
.MODEL p1 pmos(LEVEL=49 version=3.3.0)
.MODEL n1 nmos(LEVEL=49 version=3.3.0)
.endif
.if (WRSPICE_PROGRAM=0)                                 $ ngspice builtin
.MODEL p1 pmos(LEVEL=49 version=3.3.0)
.MODEL n1 nmos(LEVEL=49 version=3.3.0)
.endif

.param CLAMP=1                                          $ clamp transistor of Athas's adiabatic amplifier, set to 0 to disable
.param FULLPASS=0                                       $ other transistor to make the clamp a full pass gate
.param ACAP=2e-12                                       $ capacitive load on the data line
.param QQCAP=0e-12                                      $ capacitive load on the internal QQ node
.param MUXCAP=1e-12                                     $ capacitive load on the MUX output

*** SUBCIRCUIT DEFINITIONS
* Figure 4 in arXiv:2009.00448, Athas's adiabatic amplifier but with complementary voltages on the two halves
.SUBCKT AAMP AT AC T C piT piC GND PWR nsub psub ini='gg'   $ Athas's adiabatic amplifier. Args: AT/C T/C clockT/C substrate supplies
.ic V(T)='ini' V(C)='vv-ini'                            $ .ic V(a)={gg} V(a2)=ini
M0 piT AT T nsub n1                                     $ pass gate
M1 piT AC T psub p1
M2 piC AT C nsub n1                                     $ pass gate
M3 piC AC C psub p1
.if (CLAMP=1)
M4 GND AC T nsub n1                                     $ clamp
M5 PWR AT C psub p1
.endif
.if (FULLPASS=1)
M6 GND AT T psub p1
M7 PWR AC C nsub n1
.endif
.ENDS AAMP

* Figure 5 in arXiv:2009.00448
.SUBCKT LATCH AT AC QT QC piT piC pjT pjC GND PWR           $ One phase of the 2LAL shift register. Args: AT/C QT/C clock0T/C clock1T/C
+ nsub psub tap0 tap1 tap2 tap3 ini='gg'                 $ substrate supplies
R0 tap5 QT 1                                            $ circuit taps for debugging
X1 AT AC T C piT piC GND PWR nsub psub AAMP ini='ini'
M1 T pjT QT nsub n1                                     $ Frank's latch
M2 T pjC QT psub p1
M3 C pjT QC nsub n1                                     $ Frank's latch
M4 C pjC QC psub p1
C1 AT 0 ACAP
C2 AC 0 ACAP
C3 T 0 QQCAP
C4 C 0 QQCAP
.ENDS LATCH

* Figure 6 in arXiv:2009.00448, except this is just the first stage; shift clocks for subsequent stages
.SUBCKT PHASE S0T S0C S1T S1C                           $ One stage of the 2LAL shift register. Args: AT/C QT/C
+ p0T p0C p1T p1C p2T p2C p3T p3C GND PWR nsub psub      $ 4x{ phi<n>T/C } DC Supply substrate supplies
+ tap0 tap1 tap2 tap3 tap4 tap5 tap6 tap7 ini='gg'
X0  S0T S0C S1T S1C p1T p1C p0T p0C GND PWR nsub psub tap0 tap1 tap2 tap3 LATCH ini=ini
X10 S1T S1C S0T S0C p2T p2C p3T p3C GND PWR nsub psub tap4 tap5 tap6 tap7 LATCH ini=ini
.ends PHASE

* Figure 6 in arXiv:2009.00448, except this is all 8 stages
.SUBCKT SDELAY S0T S0C S8T S8C                          $ Four phases that just delay. Args: 2*{ data<n>T/C }
+ p0T p0C p1T p1C p2T p2C p3T p3C                       $ clocks/power supplies
+ p4T p4C p5T p5C p6T p6C p7T p7C
+ GND PWR nsub psub                                    $ DC Supply substrate supplies
+ tap0 tap1 tap2 tap3 tap4 tap5 tap6 tap7 tap8 tap9 tapA tapB tapC tapD tapE tapF ini='gg'
R0 tap0 S0T 1                                           $ circuit taps for debugging
R1 tap1 S0C 1
R2 tap2 S1T 1
R3 tap3 S1C 1
R4 tap4 S2T 1
R5 tap5 S2C 1
R6 tap6 S3T 1
R7 tap7 S3C 1
R8 tap8 S4T 1
R9 tap9 S4C 1
RA tapA S5T 1
RB tapB S5C 1
RC tapC S6T 1
RD tapD S6C 1
RE tapE S7T 1
RF tapF S7C 1
X0  S0T S0C S1T S1C p0T p0C p1T p1C p2T p2C p3T p3C GND PWR nsub psub t100 t101 t102 t103 t200 t201 t202 t203 PHASE ini=gg
X1  S1T S1C S2T S2C p1T p1C p2T p2C p3T p3C P4T P4C GND PWR nsub psub t110 t111 t112 t113 t210 t211 t212 t213 PHASE ini=ini
X2  S2T S2C S3T S3C p2T p2C p3T p3C P4T P4C P5T P5C GND PWR nsub psub t120 t121 t122 t123 t220 t221 t222 t223 PHASE ini=ini
X3  S3T S3C S4T S4C p3T p3C P4T P4C P5T P5C P6T P6C GND PWR nsub psub t130 t131 t132 t133 t230 t231 t232 t233 PHASE ini=ini
X4  S4T S4C S5T S5C P4T P4C P5T P5C P6T P6C P7T P7C GND PWR nsub psub t140 t141 t142 t143 t240 t241 t242 t243 PHASE ini=ini
X5  S5T S5C S6T S6C P5T P5C P6T P6C P7T P7C P0T P0C GND PWR nsub psub t150 t151 t152 t153 t250 t251 t252 t253 PHASE ini=ini
X6  S6T S6C S7T S7C P6T P6C P7T P7C P0T P0C P1T P1C GND PWR nsub psub t160 t161 t162 t163 t260 t261 t262 t263 PHASE ini=gg
X7  S7T S7C S8T S8C P7T P7C P0T P0C P1T P1C P2T P2C GND PWR nsub psub t170 t171 t172 t173 t270 t271 t272 t273 PHASE ini=gg
.ENDS SDELAY

$ 2-input bi-directional MUX built with 2-rail address and pass gates
.SUBCKT STR in0 in1 adrT adrC out0 out1 nsub psub       $ inputs in0 in1 adrT/C out; connect in[adr] to out
M1 in0 adrT out0 psub p1                                $ adr = 0 --> in0 connects to out
.if (0)
R1 in0 out0 1
R2 in1 out1 1
.else
M2 in0 adrC out0 nsub n1                                $ adr = 0 --> in0 connects to out
```

```
M3 in1 adrC out0 psub p1                                    $ adr = 1 --> in1 connects to out
M4 in1 adrT out0 nsub n1                                    $ adr = 1 --> in1 connects to out
M5 in1 adrT out1 psub p1                                    $ adr = 0 --> in0 connects to out
M6 in1 adrC out1 nsub n1                                    $ adr = 0 --> in0 connects to out
M7 in0 adrC out1 psub p1                                    $ adr = 1 --> in1 connects to out
M8 in0 adrT out1 nsub n1                                    $ adr = 1 --> in1 connects to out
.endif
C1 out0 0 MUXCAP
C2 out1 0 MUXCAP
.ENDS STR

* Two stages with clock rate swap. Actually, it's the data that swaps
.SUBCKT RATEX ATi ACi BTi BCi p0Ti p0Ci p1Ti p1Ci p2Ti p2Ci p3Ti p3Ci p4Ti p4Ci p5Ti p5Ci p6Ti p6Ci p7Ti p7Ci
+             CTi CCi DTi DCi q0Ti q0Ci q1Ti q1Ci q2Ti q2Ci q3Ti q3Ci q4Ti q4Ci q5Ti q5Ci q6Ti q6Ci q7Ti q7Ci
+ G1 G2 GND PWR nsub psub iniA=0 iniB=0                      $ DCi Supply substrate supplies
X1 ATo ACo BTo BCo p0To p0Co p1To p1Co p2To p2Co p3To p3Co p4To p4Co p5To p5Co p6To p6Co p7To p7Co GND PWR nsub psub t300 t301 t302 t303 t304 t305 t306 t307 t308 t309 t30A
t30B t30Ci t30D t30E t30F SDELAY ini=iniA
X5 CTo CCo DTo DCo q0To q0Co q1To q1Co q2To q2Co q3To q3Co q4To q4Co q5To q5Co q6To q6Co q7To q7Co GND PWR nsub psub u300 u301 u302 u303 u304 u305 u306 u307 u308 u309 u30A
u30B u30Ci u30D u30E u30F SDELAY ini=iniB
X10 ATi CTi G1 G2 ATo CTo nsub psum STR                     $ inputs in0 in1 adrT/Ci out0 out1; optionally swap ins and outs
X11 ACi CCi G1 G2 ACo CCo nsub psum STR
X12 BTi DTi G1 G2 BTo DTo nsub psum STR
X13 BCi DCi G1 G2 BCo DCo nsub psum STR
X14 p0Ti q0Ti G1 G2 p0To q0To nsub psum STR
X15 p0Ci q0Ci G1 G2 p0Co q0Co nsub psum STR
X16 p1Ti q1Ti G1 G2 p1To q1To nsub psum STR
X17 p1Ci q1Ci G1 G2 p1Co q1Co nsub psum STR
X18 p2Ti q2Ti G1 G2 p2To q2To nsub psum STR
X19 p2Ci q2Ci G1 G2 p2Co q2Co nsub psum STR
X20 p3Ti q3Ti G1 G2 p3To q3To nsub psum STR
X21 p3Ci q3Ci G1 G2 p3Co q3Co nsub psum STR
X22 p4Ti q4Ti G1 G2 p4To q4To nsub psum STR
X23 p4Ci q4Ci G1 G2 p4Co q4Co nsub psum STR
X24 p5Ti q5Ti G1 G2 p5To q5To nsub psum STR
X25 p5Ci q5Ci G1 G2 p5Co q5Co nsub psum STR
X26 p6Ti q6Ti G1 G2 p6To q6To nsub psum STR
X27 p6Ci q6Ci G1 G2 p6Co q6Co nsub psum STR
X28 p7Ti q7Ti G1 G2 p7To q7To nsub psum STR
X29 p7Ci q7Ci G1 G2 p7Co q7Co nsub psum STR
.ENDS RATEX

*** POWER-CLOCKS
.param gg= 0V
.param vv= 9.99V

.param ticks=199                                            $ number of ticks in the simulation
.param tick=1000NS                                          $ time of a tick
.param tstep=24NS                                           $ 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 Sw and fast phase 1's (total six unique signals)

.param Ramp=0.80*tick
.param PPT=0.10*tick                                        $ one PPT at beginning and end of sequence, two of these PPTs between ramps
$ Extra delay to split phi0 into a fast and slow clock; if Fast=0, the clocks become the same
$ See Saed G. Younis. Asymptotically Zero Energy Computing Using Split-Level Charge Recovery Logic. No. AI-TR-1500. MIT AI Laboratory, 1994.
.param Fast=PPT+Ramp+PPT

$ The clocks comprise a series transistions (separated by PPTs). Starting at the beginning of the three-phase cycle, the clock are computed by repeatedly
$ incrementing the time by the length of a transition and a PPT.
.param f0uS=PPT
.param f0uF=f0uS+Fast
.param f1up=f0uF+Ramp+2*PPT
.param f2up=f1up+Ramp+2*PPT
.param f3up=f2up+Ramp+2*PPT
.param f0dn=f3up+Ramp+2*PPT
.param f1dn=f0dn+Ramp+2*PPT
.param f2dF=f1dn+Ramp+2*PPT
.param f2dS=f2dF+Fast
.param f3dn=f2dS+Ramp+2*PPT
.param epoc=f3dn+Ramp+PPT

Vphi0P 110 0 PWL('0' 'gg'                           'f0uS' 'gg' 'f0uS+Ramp' 'vv'     'f0dn' 'vv' 'f0dn+Ramp' 'gg'     'epoc' 'gg' r='0')
Vphi0f 510 0 PWL('0' 'gg'                           'f0uF' 'gg' 'f0uF+Ramp' 'vv'     'f0dn' 'vv' 'f0dn+Ramp' 'gg'     'epoc' 'gg' r='0')
Vphi1P 111 0 PWL('0' 'gg'                           'f1up' 'gg' 'f1up+Ramp' 'vv'     'f1dn' 'vv' 'f1dn+Ramp' 'gg'     'epoc' 'gg' r='0')
Vphi2P 112 0 PWL('0' 'gg'                           'f2up' 'gg' 'f2up+Ramp' 'vv'     'f2dS' 'vv' 'f2dS+Ramp' 'gg'     'epoc' 'gg' r='0')
Vphi2f 512 0 PWL('0' 'gg'                           'f2up' 'gg' 'f2up+Ramp' 'vv'     'f2dF' 'vv' 'f2dF+Ramp' 'gg'     'epoc' 'gg' r='0')
Vphi3P 113 0 PWL('0' 'gg'                           'f3up' 'gg' 'f3up+Ramp' 'vv'     'f3dn' 'vv' 'f3dn+Ramp' 'gg'     'epoc' 'gg' r='0')
Vphi4f 514 0 PWL('0' 'vv'                           'f0uF' 'vv' 'f0uF+Ramp' 'gg'     'f0dn' 'gg' 'f0dn+Ramp' 'vv'     'epoc' 'vv' r='0')
Vphi4P 114 0 PWL('0' 'vv'                           'f0uS' 'vv' 'f0uS+Ramp' 'gg'     'f0dn' 'gg' 'f0dn+Ramp' 'vv'     'epoc' 'vv' r='0')
Vphi5P 115 0 PWL('0' 'vv'                           'f1up' 'vv' 'f1up+Ramp' 'gg'     'f1dn' 'gg' 'f1dn+Ramp' 'vv'     'epoc' 'vv' r='0')
Vphi6f 516 0 PWL('0' 'vv'                           'f2up' 'vv' 'f2up+Ramp' 'gg'     'f2dF' 'gg' 'f2dF+Ramp' 'vv'     'epoc' 'vv' r='0')
Vphi6P 116 0 PWL('0' 'vv'                           'f2up' 'vv' 'f2up+Ramp' 'gg'     'f2dS' 'gg' 'f2dS+Ramp' 'vv'     'epoc' 'vv' r='0')
Vphi7P 117 0 PWL('0' 'vv'                           'f3up' 'vv' 'f3up+Ramp' 'gg'     'f3dn' 'gg' 'f3dn+Ramp' 'vv'     'epoc' 'vv' r='0')

ViiP   118 0 PWL('0' 'gg'                           'f0uS' 'gg' 'f0uS+Ramp' 'vv'     'f2dS' 'vv' 'f2dS+Ramp' 'gg'     'epoc' 'gg' r='0')
ViiN   119 0 PWL('0' 'vv'                           'f0uS' 'vv' 'f0uS+Ramp' 'gg'     'f2dS' 'gg' 'f2dS+Ramp' 'vv'     'epoc' 'vv' r='0')

.param g0uS=2*PPT
.param g0uF=g0uS+2*Fast
.param g1up=g0uF+2*Ramp+4*PPT
.param g2up=g1up+2*Ramp+4*PPT
.param g3up=g2up+2*Ramp+4*PPT
.param g0dn=g3up+2*Ramp+4*PPT
.param g1dn=g0dn+2*Ramp+4*PPT
.param g2dF=g1dn+2*Ramp+4*PPT
.param g2dS=g2dF+2*Fast
.param g3dn=g2dS+2*Ramp+4*PPT
.param gpoc=g3dn+2*Ramp+2*PPT

Vphj0P 810 0 PWL('0' 'gg'                           'g0uS' 'gg' 'g0uS+Ramp' 'vv'     'g0dn' 'vv' 'g0dn+Ramp' 'gg'     'gpoc' 'gg' r='0')
Vphj0f 910 0 PWL('0' 'gg'                           'g0uF' 'gg' 'g0uF+Ramp' 'vv'     'g0dn' 'vv' 'g0dn+Ramp' 'gg'     'gpoc' 'gg' r='0')
Vphj1P 811 0 PWL('0' 'gg'                           'g1up' 'gg' 'g1up+Ramp' 'vv'     'g1dn' 'vv' 'g1dn+Ramp' 'gg'     'gpoc' 'gg' r='0')
Vphj2P 812 0 PWL('0' 'gg'                           'g2up' 'gg' 'g2up+Ramp' 'vv'     'g2dS' 'vv' 'g2dS+Ramp' 'gg'     'gpoc' 'gg' r='0')
Vphj2f 912 0 PWL('0' 'gg'                           'g2up' 'gg' 'g2up+Ramp' 'vv'     'g2dF' 'vv' 'g2dF+Ramp' 'gg'     'gpoc' 'gg' r='0')
Vphj3P 813 0 PWL('0' 'gg'                           'g3up' 'gg' 'g3up+Ramp' 'vv'     'g3dn' 'vv' 'g3dn+Ramp' 'gg'     'gpoc' 'gg' r='0')
Vphj4f 914 0 PWL('0' 'vv'                           'g0uF' 'vv' 'g0uF+Ramp' 'gg'     'g0dn' 'gg' 'g0dn+Ramp' 'vv'     'gpoc' 'vv' r='0')
Vphj4P 814 0 PWL('0' 'vv'                           'g0uS' 'vv' 'g0uS+Ramp' 'gg'     'g0dn' 'gg' 'g0dn+Ramp' 'vv'     'gpoc' 'vv' r='0')
Vphj5P 815 0 PWL('0' 'vv'                           'g1up' 'vv' 'g1up+Ramp' 'gg'     'g1dn' 'gg' 'g1dn+Ramp' 'vv'     'gpoc' 'vv' r='0')
Vphj6f 916 0 PWL('0' 'vv'                           'g2up' 'vv' 'g2up+Ramp' 'gg'     'g2dF' 'gg' 'g2dF+Ramp' 'vv'     'gpoc' 'vv' r='0')
Vphj6P 816 0 PWL('0' 'vv'                           'g2up' 'vv' 'g2up+Ramp' 'gg'     'g2dS' 'gg' 'g2dS+Ramp' 'vv'     'gpoc' 'vv' r='0')
Vphj7P 817 0 PWL('0' 'vv'                           'g3up' 'vv' 'g3up+Ramp' 'gg'     'g3dn' 'gg' 'g3dn+Ramp' 'vv'     'gpoc' 'vv' r='0')

Vg1    PPC 0 PWL('0' 'vv'                       'gpoc-PPT' 'vv' 'gpoc'      'gg'              '2*gpoc-PPT' 'gg'   '2*gpoc' 'vv' r='0')
Vg2    PPT 0 PWL('0' 'gg'                       'gpoc-PPT' 'gg' 'gpoc'      'vv'              '2*gpoc-PPT' 'vv'   '2*gpoc' 'gg' r='0')

VGND   200 0 DC 'gg'
```

```
VPWR   201 0 DC 'vv'

*** TOP-LEVEL CIRCUIT
X0 FAT FAC BAT BAC 110 114 111 115 112 116 113 117 114 110 115 111 116 112 117 113
+  SAT SAC SBT SBC 810 814 811 815 812 816 813 817 814 810 815 811 816 812 817 813 PPC PPT 200 201 200 201 RATEX iniA=vv iniB=gg
X1 SBT SBC SCT SCC 810 814 811 815 812 816 813 817 814 810 815 811 816 812 817 813 200 201 200 201 u320 u321 u322 u323 u324 u325 u326 u327 u328 u329 u32A u32B u32C u32D u32E
u32F SDELAY ini=gg
X5 SCT SCC SAT SAC 810 814 811 815 812 816 813 817 814 810 815 811 816 812 817 813 200 201 200 201 x320 x321 x322 x323 x324 x325 x326 x327 x328 x329 x32A x32B x32C x32D x32E
x32F SDELAY ini=gg

X2 BAT BAC FAT FAC 110 114 111 115 112 116 113 117 114 110 115 111 116 112 117 113
+  SXT SXC SYT SYC 810 814 811 815 812 816 813 817 814 810 815 811 816 812 817 813 PPC PPT 200 201 200 201 RATEX iniA=vv iniB=vv
X3 SYT SYC SZT SZC 810 814 811 815 812 816 813 817 814 810 815 811 816 812 817 813 200 201 200 201 v320 v321 v322 v323 v324 v325 v326 v327 v328 v329 v32A v32B v32C v32D v32E
v32F SDELAY ini=gg
X4 SZT SZC SXT SXC 810 814 811 815 812 816 813 817 814 810 815 811 816 812 817 813 200 201 200 201 w320 w321 w322 w323 w324 w325 w326 w327 w328 w329 w32A w32B w32C w32D w32E
w32F SDELAY ini=gg

* power and energy calculation
.if (WRSPICE_PROGRAM=0)                                 $ ngspice builtin
B4 0 16 V=0
+ +I(vphi0P)*v(110)+I(vphi1P)*v(111)+I(vphi2P)*v(112)+I(vphi3P)*v(113)+I(vphi4P)*v(114)+I(vphi5P)*v(115)+I(vphi6P)*v(116)+I(vphi7P)*v(117)
+ +I(ViiP)*v(118)+I(viiN)*v(119)
+ +I(VGND)*v(200)+I(VPWR)*v(201)
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)
.endif

.option noinit acct

*********************************************************************************************
$ NGSPICE CONTROL AREA
.TRAN 'tstep' 'ticks*tick'
.control
pre_set strict_errorhandling
unset ngdebug
run

set color0=white
set xbrushwidth=3
set xgridwidth=1

* measure power consumption
meas tran Energylus INTEG v(16) from=0 to=5us
meas tran EnergyLev INTEG v(16) 'from=5us to=ttn'
echo --------------------Results $&Energylus , $&EnergyLev
echo Results , $&Energylus , $&EnergyLev >>scrl_s.csv

plot v(16)                                              $ plot instantaneous energy consumption
+ ylimit -25m 25m
plot v(17)                                              $ plot accumulated energy dissipation
+ ylimit 0 350n

*********************************************************************************************
$ WRSPICE CONTROL AREA
$ .control
$ tran 'tstep' 'ticks*tick'

*********************************************************************************************
$ END CONTROL AREA

plot title "S2LAL clock and gated clock" ylimit 0 12 xlimit 0 200u
$ gnuplot ylimit 0 12 xlimit 0 300u
+ v(110)/9.99*0.9+10.55
+ v(FAT)/9.99*0.9+ 8.55
+ v(SAT)/9.99*0.9+ 6.55
+ v(SXT)/9.99*0.9+ 4.55
+ v(810)/9.99*0.9+ 2.55
+ v(PPT)/9.99*0.9+ 0.525
+ v(PPC)/9.99*0.9+ 0.55

$ set fn=file$&loop+x.png
$ gnuplot gp/$fn v(a)-1 v(24)/2 v(25)/2 v(26)/2+.5 v(27)/2+.5 v(b2)+1.5 v(22)/2+2.5 v(23)/2+2.5 v(18)/2+3 v(19)/2+3 v(b1)+4 v(20)/2+5 v(21)/2+5 v(16)/2+5.5 v(17)/2+5.5
v(b)+6.5 v(24)/2+7.5 v(25)/2+7.5 v(14)/2+8 v(15)/2+8 v(a2)+9 v(22)/2+10 v(23)/2+10 v(12)/2+10.5 v(13)/2+10.5 v(a1)+11.5 v(20)/2+12.5 v(21)/2+12.5 v(10)/2+13 v(11)/2+13
v(a)+14
$ * + v(i1)-2 v(i2)-3 v(j1)-4 v(j2)-5 10000000*v(42)-6 10000000*v(40)-7 10000000*v(41)-8
$ + title "Curves: $&Stick s tick, $&Sticks ticks, $&Sttn s total, $&SloadC F ld, wid x $&Swidx, $&Svv V/2" ylimit -9 15
$ *gnuplot gp/$fn v(a)-1 v(24)/2 v(25)/2 v(26)/2+.5 v(27)/2+.5 v(b2)+1.5 v(22)/2+2.5 v(23)/2+2.5 v(18)/2+3 v(19)/2+3 v(b1)+4 v(20)/2+5 v(21)/2+5 v(16)/2+5.5 v(17)/2+5.5
v(b)+6.5 v(24)/2+7.5 v(25)/2+7.5 v(14)/2+8 v(15)/2+8 v(a2)+9 v(22)/2+10 v(23)/2+10 v(12)/2+10.5 v(13)/2+10.5 v(a1)+11.5 v(20)/2+12.5 v(21)/2+12.5 v(10)/2+13 v(11)/2+13
v(a)+14 v(i1)-2 v(i2)-3 v(j1)-4 v(j2)-5 10000000*v(42)-6 10000000*v(40)-7 10000000*v(41)-8 title "step=$&Sstep s Ptick=$&Stick s time=$&Sttn s Pticks=$&Sticks split v=$&Svv
V" ylimit -9 15


.endc

.END
```