Quiet 2-Level Adiabatic Logic

Zettaflops, LLC Technical Report ZF009

Erik P. DeBenedictis
Zettaflops, LLC
Albuquerque, New Mexico, USA
April 9, 2021

***Abstract*—This document defines Quiet 2-Level Adiabatic Logic (Q2LAL), a variant of Static 2-Level Adiabatic Logic (S2LAL). Q2LAL yields universal logic with two rails and places a constant load on the power supply. In comparison, S2LAL requires four rails for universal logic and four rails to achieve constant load. The constant load is relevant to resistance to Differential Power Analysis (DPA), a cyber security issue, and also makes the power-clock generators simpler.**

Keywords—adiabatic computing; reversible computing; quantum computer; supercomputer; CMOS; cryo CMOS; Static 2-Level Adiabatic Logic; S2LAL; Quiet 2-Level Adiabatic Logic; Q2LAL

I. Quiet 2-Level Adiabatic Logic

This document formally defines Quiet 2-Level Adiabatic logic (Q2LAL) [1].

At least four logic families, SCRL [2], 2LAL [3], S2LAL [4] and now Q2LAL use a common circuit framework and the same notation. Q2LAL can be most effectively explained by symbolic manipulation of S2LAL's circuit equations, which requires knowledge of both S2LAL and the common notation.

This document only attempts to describe the basic shift register, universal Boolean logic, and shows the even load property via simulation.

II. Data Waveforms

Fig. 1 illustrates the signal waveforms for S2LAL and Q2LAL. The primary signal waveforms are in the upper left of Fig. 1a and b and called $\hat{S}$ and $\hat{Q}$ depending on the logic family. The primary waveform in both cases represents a 1 as a positive-going pulse. The circumflex (hat) diacritical mark was chosen because it looks like the waveform of a positive-going pulse.

S2LAL's second waveform in Fig. 1a is carried on a second wire, or rail, that always carries the electrical complement of the first waveform. This means the second wire rests at supply voltage V_{dd} and has negative-going pulses denoted $\check{S}$. The caron (cup) diacritical mark was chosen because it looks like the waveform of a negative going pulse. The absence of a pulse represents a 0 in S2LAL.

S2LAL can store data in a shift register with the dual rail signaling just described, but it requires two more rails to implement universal logic. The other waveforms are found below the first pair in Fig. 1a. These rails hold the logical complement of the data on the first pair of rails.

Each rail requires a copy of the circuit. S2LAL requires four rails to create universal adiabatic logic, but it would be preferable to have fewer rails.

Q2LAL is dual rail as illustrated in Fig. 1b. Q2LAL signal has two rails and two wires, both of which have a resting state of 0. A positive-going pulse on one wire is called $\hat{Q}^1$ and indicates the transmission of 1, and likewise $\hat{Q}^0$ on the other wire represents a 0.

S2LAL and Q2LAL use the same 8-phase power-clocks illustrated in Fig. 2a, where the 8 phases are called ticks. If the waveforms are considered positive pulses, they are denoted ϕ_i, i=0…7, but they can also be considered negative pulses due to symmetry. So, $\phi_i = \check{\phi}_{i+4 \bmod 8}$, i=0…7. A waveform should follow a linear ramp for the entire duration of the tick. The consistency of the ramp's slope is critical to energy efficiency in all adiabatic circuits, so the unfamiliar reader should not see the ramps as just an artistic convenience.

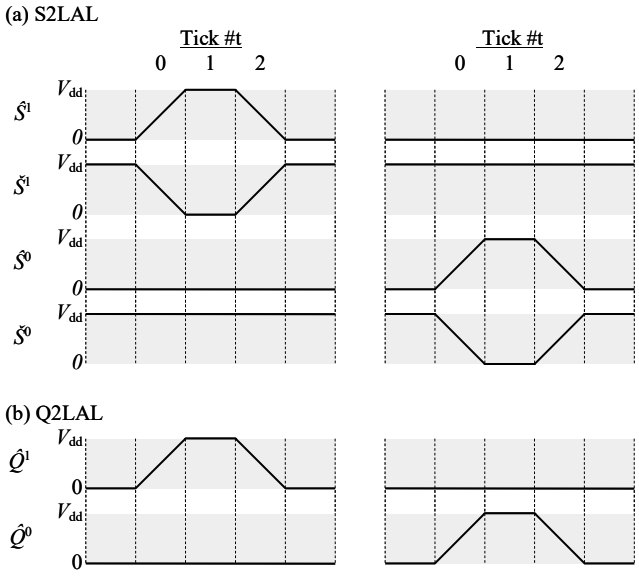

Fig. 1. Signal waveforms. (a) Predecessor S2LAL is dual or quad rail. (b) Q2LAL is dual rail. Notation from [4].

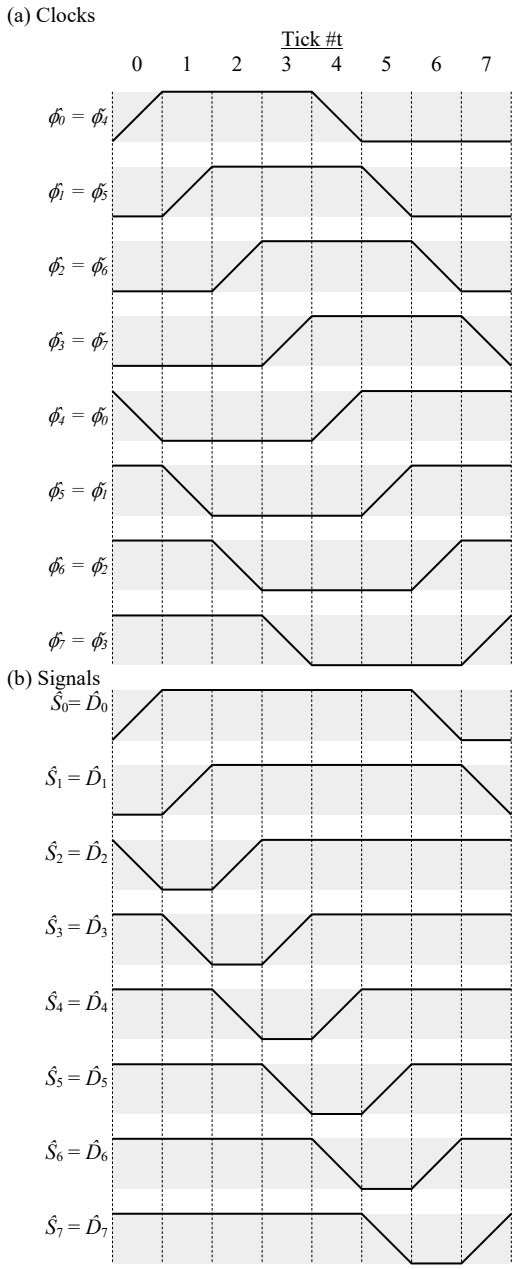

Fig 2. (a) Clocks and (b) data signaling formats are the same between S2LAL and Q2LAL. $\hat{\phi}_i = \check{\phi}_{i+4 \bmod 8}$, but this is not true for the S's. Notation from [4].

Both S2LAL and Q2LAL have the same data timing. The data waveforms in Fig. 2b are pulses that are stable for 5 of 8 ticks. The other three ticks include one tick in the resting state and two transition ticks.

All the logic families involve transmission gates. A pair of back-to-back p- and n-channel field effect transistors (FETs) appear as a rectangle with a line connected to the long side, as shown in Fig. 3a. The line represents two complementary electrical signals that go to the transistors' gates. If the lines on the short side of the rectangle are a quad-rail signal, the rectangle implicitly represents two transmission gates or four transistors, also illustrated in Fig. 3a.

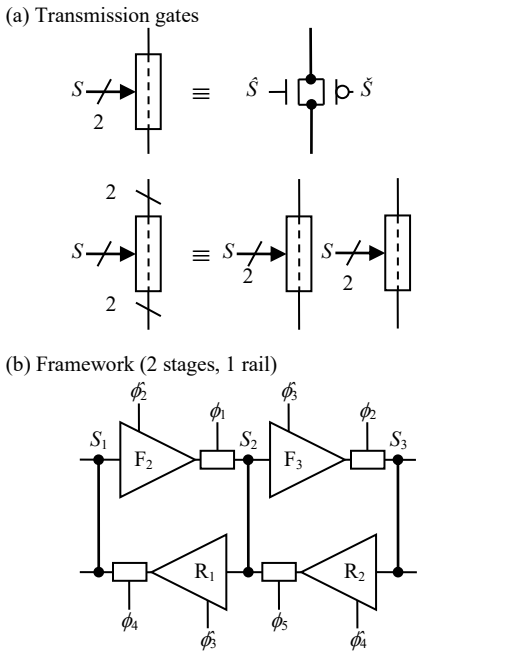

Fig 3. (a) Transmission gate notation, notably including multi-rail. (b) Emerging framework for a SCRL, 2LAL, S2LAL, and Q2LAL. Notation from [4].

The framework involves the coupled cycles in Fig. 3b, where 8 cycles form a complete logic stage in both S2LAL and Q2LAL.

The various families differ in the data representation on the lines, the clock sequencing, and the circuitry in the functional units. The lines in SCRL are trits with three signal levels of $V_{dd}/2$, 0, and $-V_{dd}/2$, as opposed to more familiar bits with two signal levels of V_{dd} and 0 in the other families.

Signals are defined by the clock phase or tick where the level is valid, so a signal A could be denoted by $\hat{A}_i$, where i identifies the tick.

Prior to Q2LAL, circuits using the framework could perform AND and OR logic functions but could not invert a signal without doubling the number of rails. While there are useful circuits that do not need inversion, such as memory, inversion is needed for universal logic.

The effect of inversion in non-Q2LAL families can be created by duplicating a circuit, except all ANDs are replaced by ORs and vice-versa. If all input data provided to the original circuit is also provided to the copy in a logically inverted form, the two circuits will proceed in lockstep with corresponding signals in the two circuits being logical inverses of each other. With the setup just described, a signal can be inverted by swapping it with the corresponding signal in the other circuit. SCRL stages unavoidably invert data, leading to a similar problem whose solution also requires a copy of the circuit.

However, a Q2LAL signal can be logically inverted by swapping the two wires. There is no need to copy the circuit.

Let us derive Q2LAL by symbolic manipulation of S2LAL's circuit diagram. By replacing negative-going pulses (cups) with functionally equivalent positive-going pulses (hats)

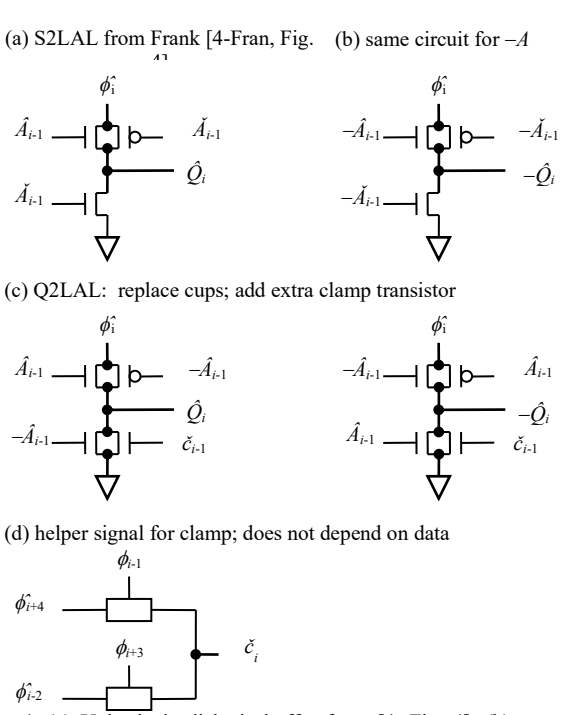

Fig. 4. (a) Unlatched adiabatic buffer from [4, Fig. 4], (b) same buffer for the negated signal, (c) however, the incoming cup signals can be generated from the negated signals in the previous stage, provided that a helper signal $\check{c}_i$ is available. (d) The helper signal can be generated once in an entire circuit from available clocks.

representing logically inverted data, we will create circuitry that does not depend on negative-going pulses. This allows us to delete the electrically complemented rail altogether, simplifying the circuit.

Fig. 4 illustrates the circuitry within the triangular structures of the framework called adiabatic amplifiers [5]. Fig. 4a is from S2LAL [4, Fig. 1], but expanding the transmission gate into its two transistors and labeling the input with the applicable phase. Fig. 4b is the same circuitry processing the logically inverted signal $-A$.

The upper symbol $\check{A}_{i-1}$ in Fig. 4a enables one transistor of the transmission gate that connects clock $\phi\hat{}_i$ to the output. In Fig. 4c, we can replace this signal with $-\hat{A}_{i-1}$ because the alternative signal is stable at the correct level when needed to gate the clock and is simply creating a redundant path to ground at other times.

Likewise, the lower symbol $\check{A}_{i-1}$ in Fig. 4a enables the transistor that clamps the output to ground. Replacing that signal with $-\hat{A}_{i-1}$ in Fig. 4c helps if the desired output is a 0 but will leave the output floating between output pulses. This leads us add a transistor gated by the signal $\check{c}_{i-1}$. Waveform $\check{c}_k$ is the electrical inverse of $\hat{D}_k$ in Fig. 2b. Thus, the signal $\check{c}_{i-1}$ goes high during the period where the output needs to be clamped to ground, irrespective of whether the output is a 0 or 1.

Fig. 4d shows how to create the $\check{c}_k$ signal for stage k from four available clocks and four transistors. There would need to be 8 variants of this circuit to create $\check{c}_k$ for $k = 0\ldots7$. However,

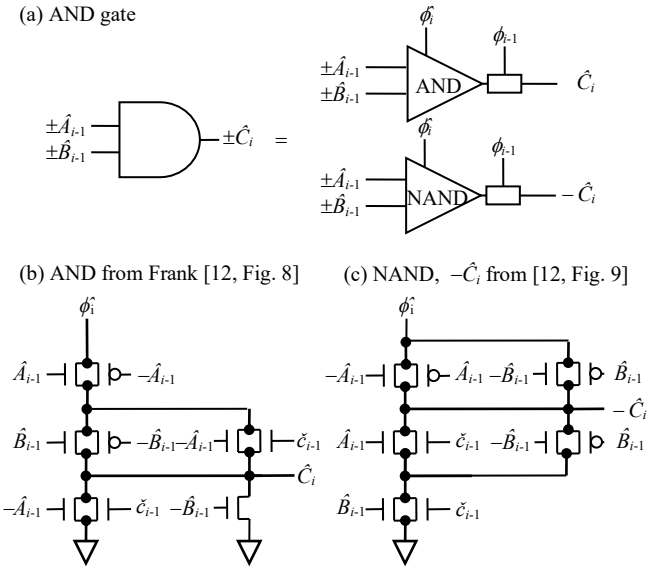

Fig. 5. (a) Definition of AND gate. (b) AND circuit based on S2LAL [4, Fig. 8], but modified for Q2LAL. (c) NAND circuit based on the S2LAL similar to OR gate [4, Fig. 9]. Becomes NAND, OR, NOR with input/output inversions.

the $\check{c}_k$'s are independent of data, so each signal can be shared across multiple gates.

Now notice that Fig. 4c and d, the three circuits that become the Q2LAL implementation, contain only $\hat{A}$ and $-\hat{A}$, there is no $\check{A}$, so we define Q2LAL as S2LAL with the second rail logically instead of electrically inverted.

Since Q2LAL signals can be inverted by swapping their wires, AND, OR, NAND and NOR are equivalent up to the labeling of inputs and outputs. Fig. 5 describes a 2-input AND gate and hence demonstrates universality.

The AND gate symbol shown in Fig. 5a defines inputs $\pm\hat{A}$ and $\pm\hat{B}$ and output $\pm\hat{C}$, all as dual rail signals with positive-going pulses. The $+\hat{C}$ pulse will appear when there are pulses on both inputs, which corresponds to a logical AND function. The $-\hat{C}$ pulse would appear in other circumstances, which are readily identified as the result of a logical NAND. Q2LAL uses separate circuitry for AND and NAND.

Q2LAL's AND-gate circuitry is the result of the same type of symbolic manipulation used in Fig. 4 to create the Q2LAL buffer. The AND circuit is the result of applying the symbolic manipulation to the S2LAL AND circuit. However, the NAND circuit starts out as a S2LAL OR gate with both inputs having their wires swapped and hence inverted.

The AND circuitry makes use of the clamp signal $\check{c}_k$ described previously.

A. Circuit complexity

There are more transistors in Q2LAL's circuit than S2LAL's, but the two families are closer in complexity than one might think—and Q2LAL pulls ahead when one considers S2LAL's need for a second copy of a circuit for inversion. As described here, Q2LAL adds clamp transistors to what had

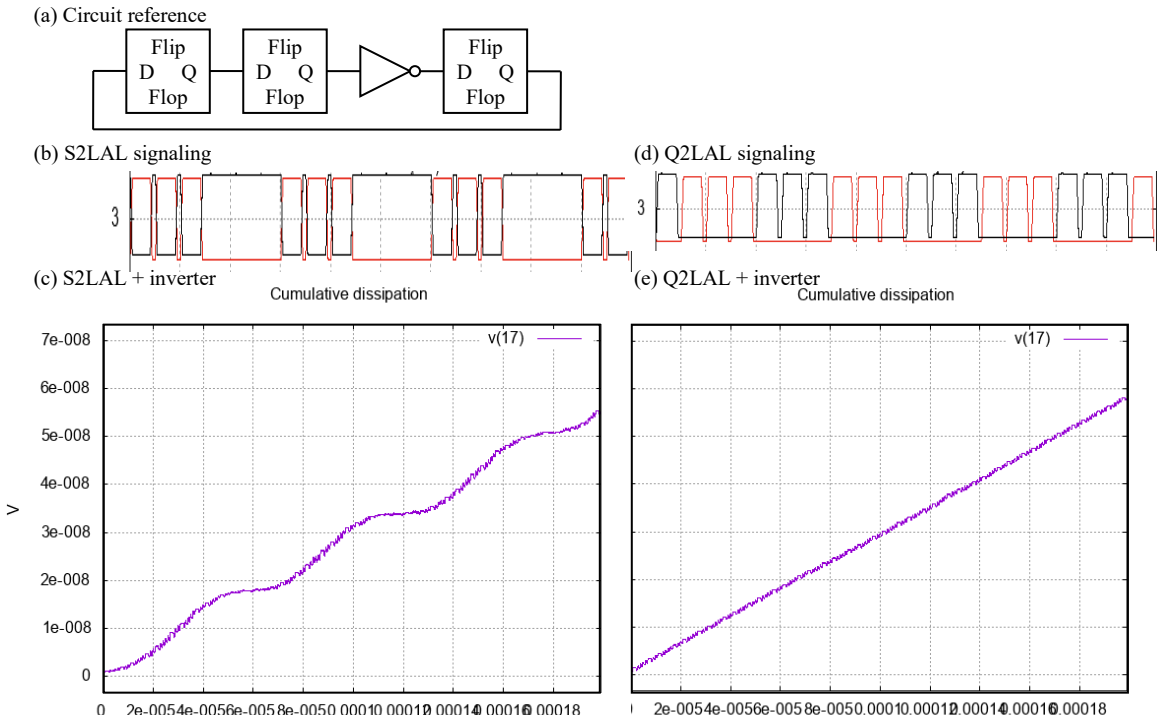

Fig. 6. (a) Circuit reference, generating repeating sequence 000 100 110 111 011 001. (b) S2LAL output $\hat{Q}$ and $\check{Q}$ (red and black) showing one bit position in the circuit (c) S2LAL cumulative dissipation, showing variance as the number of 1s changes. (d) Q2LAL signaling, where either $\hat{Q}$ or $-\hat{Q}$ is a 1 on each clock (e) Q2LAL dissipation, where the total number of 0s and 1s does not change and so the dissipation is constant.

been an adiabatic amplifier and a circuit to compute $\check{c}_k$. However, these extra costs are offset by some simplifications:

Signal $\check{c}_k$ does not depend on data and can be generated once and serve up to, say, 10 gates before the electrical loading becomes excessive. Fig. 4 shows $\check{c}_k$ generated by four transistors, but this could be taken as a 0.4 transistor share of a circuit that generates a standard signal.

Only waveform $\hat{\phi}_k$ is used in Fig. 4, i. e. $\check{\phi}_k$ does not appear. However, both $\hat{\phi}_k$ and $\check{\phi}_k$ are required for the equivalent transmission gate latch in S2LAL [4, Fig. 6]. This permits a circuit simplification called "nFET-only stages" [6]. A person familiar with the literature will realize that an nFET-only stage results from deleting the pFETs from transmission gates. This cuts transistor count but means that voltage swings will decrease slightly. However, a stage with full transmission gates can restore the voltage swings. Thus, the literature shows how to delete the pFETs in even-numbered stages, restoring full signal swing in odd-numbered stages.

B. Even load

Adiabatic circuits have been developed for computer security purposes that place a very even load on the power supply, such as EE-SPFAL [7]. Q2LAL has this even-load property, with this document showing how the even-load property can facilitate energy management.

For background, a differential power analysis (DPA) attack attempts to figure out secret information in a chip by measuring changes in power supply current. Fig. 6a is an exemplary circuit that cycles back and forth between being filled with 0s and 1s. If processing a 0 consumes a different amount of power than a 1, measuring the power supply current at a particular point in time may reveal the value of a certain data bit. While the analysis requires knowledge of the circuit and many trials, attackers find it worthwhile for obtaining high-value information such as passwords. There is literature on DPA, but further discussion of computer security is beyond the scope of this document. See ref. [8].

Fig. 6b and c show an ngspice simulation of cumulative energy dissipation of an S2LAL implementation of Fig. 6a and its signaling pattern in Fig. 6b. The curve is horizontal when the circuit is filled with 0s, indicating low dissipation, but rises steeply when filled with 1s, leading to the wavy appearance.

The two circuits in Fig. 4c differ only by swapping A's with $-A$'s. If the circuits are laid out near each other and have similar geometry, the combined electrical characteristics will be the same irrespective of the data.

Fig. 6d is the signaling pattern for the same circuit implemented in Q2LAL, with its dissipation in Fig. 6e. One would expect a linear increase in dissipation over time, which is true to the resolution of the eye.

Q2LAL would thus be suitable for computer security applications, but its even-load feature can simplify most approaches to energy management.

If power-clocks are delivered to the adiabatic circuit via a transmission line, the load will distort the waveform. If the load does not change over time, the power-clock generator can apply a fixed predistortion such that that the predistorted signal plus the distortion caused by the line and the load will result in

the desired waveform. This will cause everything to work as expected.

If a resonant power supply is used for power-clocks, an uneven load will cause the energy consumption by the various harmonics—and their phase—to vary over time, creating the added task of regulating both the energy to each harmonic and adjusting its phase over time.

III. Conclusions

The new Q2LAL circuit combines ideas from traditional adiabatic logic and a branch developing around computer security. People investigating adiabatic logic for computer security have found circuit families that place a very even load on the power supply, yet computer security does not specifically require high energy efficiency. Q2LAL is fully adiabatic and has an even load, meaning that the purple arrow in Fig. 3 would extend forever if transistors had zero gate and source-drain leakage.

Acknowledgment

Michael P. Frank has made many contributions to reversible computing over the years. Mike championed the framework in Fig. 3b that contains at least four circuit families so far—plus versions within each family containing different numbers of cycles. Mike also developed S2LAL and a consistent terminology [4], both of which became a starting point for this work. This document uses Mike's terminology, including diagrams, with his permission.

References

[1] Q2LAL has not been officially published previously, but the following document posted on the internet includes the Q2LAL circuit and attached ngspice code that generates figures in this document: DeBenedictis, Erik P., "Inversion for S2LAL." Zettaflops LLC Technical report ZF004, online at http://www.zettaflops.org/CATC/S2LAL_Inv_1.02.pdf

[2] 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.

[3] 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.

[4] Frank, Michael P., et al. "Reversible Computing with Fast, Fully Static, Fully Adiabatic CMOS," *2020 IEEE International Conference on Rebooting Computing (ICRC)*, Atlanta, GA, USA, 2020, pp. 1-8, doi: 10.1109/ICRC2020.2020.00014.

[5] W. C. Athas, L. "J." Svensson, J. G. Koller, N. Tzartzanis, and E. Y.-C. Chou, "Low-Power Digital Systems Based on Adiabatic-Switching Principles," *IEEE Trans. VLSI Sys.*, vol. 2, no. 4, pp. 398–407, Dec. 1994.

[6] E. DeBenedictis, *Enhancements to Adiabatic Logic for Quantum Computer Control Electronics*, technical report ZF002, http://www.zettaflops.org/CATC.

[7] Kumar, S. Dinesh, Himanshu Thapliyal, and Azhar Mohammad. "EE-SPFAL: A Novel Energy-Efficient Secure Positive Feedback Adiabatic Logic for DPA Resistant RFID and Smart Card," in *IEEE Transactions on Emerging Topics in Computing*, vol. 7, no. 2, pp. 281-293, 1 April-June 2019, doi: 10.1109/TETC.2016.2645128.

[8] Moradi, Amir, and Axel Poschmann. "Lightweight cryptography and DPA countermeasures: A survey." *International Conference on Financial Cryptography and Data Security*. Springer, Berlin, Heidelberg, 2010.

APPENDIX: NGSPICE FILE

The file below includes:

- S2LAL basic circuits
- Q2LAL basic circuits
- The even load comparison between the two, generating the graphs in Fig. 6.
- Testing for the AND gate circuits in Fig. 5.
- There is also code for testing an extended clock phase and modulating the onset of the ramp in the Q/S2LAL clocks.

The code uses built-in transistor models, which are based on obsolete transistors. Therefore, no absolute performance is revealed.

A. Q2LAL.cir

```
  Q2LAL
* Proprietary information of Zettaflops LLC. Not for public distribution.
* Q2LAL initial test setup. Q2LAL is "quiet 2LAL" derived from Static 2-Level Adiabatic Logic (S2LAL). More information at the end of the file.
*
* Instructions for duplicate the figures in [ZF008] and several PowerPoints:
* Fig q2la1 periods period FastSlow Porch GentleT GentleV  ylimit   dcc cwf
* 12c   0     20      10u      1      .1    .15     .15  -200u 200u  0   0
* 12e   1     20      10u      1      .1    .15     .15  -200u 200u  0   0
* 13a   1      1     .67u      0     .01    .15     .15    -5m 5m    0   0
* 13b   1      1       1u      0     .01    .15     .15    -5m 5m    0   0
* 13c   1      1      10u      0     .01    .15     .15    -5m 5m    0   0
* 13d   1      1     100u      0     .01    .15     .15    -5m 5m    0   0
* 13+   1      1     100u      0     .01    .15     .15  -400m 400m  0   0
* xxx   1      1      10u      0     .01    .05     .15  -400m 400m  0   0
* xxx   1      1     100u      0     .01    .05     .15   -50m 50m   0   0

* To duplicate slides in [Q2LALv2.ppt]. Slide number:
*  5    1      1     .67u      0     .01    .15     .15    -5m 5m    0   0
*  6    1      1       1u      0     .01    .15     .15    -5m 5m    0   0
*  7    1      1      10u      0     .01    .15     .15    -5m 5m    0   0
*  8    1      1     100u      0     .01    .15     .15    -5m 5m    0   0
* 10    1      1     .67u      0     .01    .15     .15  -200u 200u  0   0
* 11    1      1       1u      0     .01    .15     .15  -200u 200u  0   0
* 12    1      1      10u      0     .01    .15     .15  -200u 200u  0   0
* 13    1      1     100u      0     .01    .15     .15  -200u 200u  0   0

* Slide deck [ZF007] is the same as above but Porch is .1

* ???   1     20      10u      0     .01    .15     .15    -5m 5m    1   1

.param q2la1=1                                          $ nonzero for q2la1; otherwise s2la1
.param periods=20                                       $ number of repetitions of the basic waveform
.param period= 10u                                      $ period of the clock waveform, which comprises a number of ticks
.param FastSlow=1                                       $ 0 for regular clock 1 for several waveforms having fast and slow versions
.param Porch=.1                                         $ A tick as this proportion of 0 V gap at the start and end, so the spacing is twice this
.param GenT=.15                                         $ Gentle rise time as a proportion of the period
.param GenV=.15                                         $ Gentle rise voltage as a proportion of the voltage
* vertical scale must be set manually on lines identified BOOKMARK1
.param dcc=0 $ manage comments on lines identified BOOKMARK2$ demonstrate data controlled clock [ZF005 Fig. 10] (consumes power when on)
.param cwf=0                                            $ generate clamp waveform from (0) ngspice waveform generator or (1) [ZF008 Fig 8d] (consumes power when on)
.param ats=0                                            $ include code to test AND gates. Wire swap inversions turn AND into NAND, OR, and NOR
* There are three sets of plot commands at the end. Comment out either "plot" or "gnuplot"

.MODEL p1 pmos(LEVEL=49 version=3.3.0)
.MODEL n1 nmos(LEVEL=49 version=3.3.0)

.param CLAMP=1                                          $ clamp transistor of Athas's adiabatic amplifier [Athas], set to 0 to disable
.param ACAP=2e-12                                       $ capacitive load on the data line
.param QQCAP=0e-12                                      $ capacitive load on the internal QQ node

*** SUBCIRCUIT DEFINITIONS
* [S2LAL Fig. 4], Athas's adiabatic amplifier but with complementary voltages on the two halves [Athas]
.SUBCKT AAMP AT AC T C piT piC GND PWR nsub psub ini='gg'   $ [Athas] 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
.ENDS AAMP

* [S2LAL Fig. 5]
.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

* [S2LAL Fig. 6], 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
```

```
* [S2LAL Fig. 6], except this is all 8 stages
.SUBCKT SDELAY S0T S0C S8T S8C                          $ Four phases that just delay. Args: 2*{ data<n>T/C }
+ p0T p1T p2T p3T p4T p5T p6T p7T                       $ clocks/power supplies
+ GND PWR nsub psub                                    $ DC Supply substrate supplies
+ tap0 tap1 tap2 tap3 tap4 tap5 tap6 tap7 tap8 tap9 tapA tapB 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 p4T p1T p5T p2T p6T p3T p7T GND PWR nsub psub t100 t101 t102 t103 t200 t201 t202 t203 PHASE ini=gg
X1  S1T S1C S2T S2C p1T p5T p2T p6T p3T p7T P4T p0T GND PWR nsub psub t110 t111 t112 t113 t210 t211 t212 t213 PHASE ini=ini
X2  S2T S2C S3T S3C p2T p6T p3T p7T P4T p0T P5T p1T GND PWR nsub psub t120 t121 t122 t123 t220 t221 t222 t223 PHASE ini=ini
X3  S3T S3C S4T S4C p3T p7T P4T p0T P5T p1T P6T p2T GND PWR nsub psub t130 t131 t132 t133 t230 t231 t232 t233 PHASE ini=ini
X4  S4T S4C S5T S5C P4T p0T P5T p1T P6T p2T P7T p3T GND PWR nsub psub t140 t141 t142 t143 t240 t241 t242 t243 PHASE ini=ini
X5  S5T S5C S6T S6C P5T p1T P6T p2T P7T p3T P0T p4T GND PWR nsub psub t150 t151 t152 t153 t250 t251 t252 t253 PHASE ini=ini
X6  S6T S6C S7T S7C P6T p2T P7T p3T P0T p4T P1T p5T GND PWR nsub psub t160 t161 t162 t163 t260 t261 t262 t263 PHASE ini=gg
X7  S7T S7C S8T S8C P7T p3T P0T p4T P1T p5T P2T p6T GND PWR nsub psub t170 t171 t172 t173 t270 t271 t272 t273 PHASE ini=gg
.ENDS SDELAY

* This is an inverting version of the phase circuit. It simply reverses the input wires.
.SUBCKT PHASEv 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  S0C S0T S1T S1C p1T p1C p0T p0C GND PWR nsub psub tap0 tap1 tap2 tap3 LATCH ini=ini
X10 S1C S1T S0T S0C p2T p2C p3T p3C GND PWR nsub psub tap4 tap5 tap6 tap7 LATCH ini=ini
.ends PHASEv

* This is an inverting version of the delay circuit. It simply calls PHASEv at a point that doesn't interfere with initialization.
.SUBCKT SDELAYv S0T S0C S8T S8C                         $ Four phases that just delay. Args: 2*{ data<n>T/C }
+ p0T p1T p2T p3T p4T p5T p6T p7T                       $ clocks/power supplies
+ GND PWR nsub psub                                    $ DC Supply substrate supplies
+ tap0 tap1 tap2 tap3 tap4 tap5 tap6 tap7 tap8 tap9 tapA tapB 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 p4T p1T p5T p2T p6T p3T p7T GND PWR nsub psub t100 t101 t102 t103 t200 t201 t202 t203 PHASE ini=gg
X1  S1T S1C S2T S2C p1T p5T p2T p6T p3T p7T P4T p0T GND PWR nsub psub t110 t111 t112 t113 t210 t211 t212 t213 PHASE ini=ini
X2  S2T S2C S3T S3C p2T p6T p3T p7T P4T p0T P5T p1T GND PWR nsub psub t120 t121 t122 t123 t220 t221 t222 t223 PHASE ini=ini
X3  S3T S3C S4T S4C p3T p7T P4T p0T P5T p1T P6T p2T GND PWR nsub psub t130 t131 t132 t133 t230 t231 t232 t233 PHASE ini=ini
X4  S4T S4C S5T S5C P4T p0T P5T p1T P6T p2T P7T p3T GND PWR nsub psub t140 t141 t142 t143 t240 t241 t242 t243 PHASE ini=ini
X5  S5T S5C S6T S6C P5T p1T P6T p2T P7T p3T P0T p4T GND PWR nsub psub t150 t151 t152 t153 t250 t251 t252 t253 PHASE ini=ini
X6  S6T S6C S7T S7C P6T p2T P7T p3T P0T p4T P1T p5T GND PWR nsub psub t160 t161 t162 t163 t260 t261 t262 t263 PHASEv ini=gg
X7  S7T S7C S8T S8C P7T p3T P0T p4T P1T p5T P2T p6T GND PWR nsub psub t170 t171 t172 t173 t270 t271 t272 t273 PHASE ini=gg
.ENDS SDELAYv

* Erik's "two hat" adiabatic amplifier. In S2LAL notation, it expects data input as A^ and -A^. Given this, it produces the correct output [ZF008 Fig. 8c (both sides)].
* Same role in framework as [S2LAL Fig. 4], Athas's adiabatic amplifier but with complementary voltages on the two halves [Athas]
*  :           :           :           :           :           :           :           :
* 1: A(i-1)^   2: -A(i-1)^
* 3: Q(i)^     4: Q(i)^
* 5: phi(i)^   6: Clmp(i-1)v
* 7: GND
* 8: nsub      9: psub
.SUBCKT QAAmp AT AC T C pT C1 GND nsub psub ini='gg'    $ Erik's adiabatic amplifier. Args: AT/C T/C clock&clamp substrate supplies
.ic V(T)='ini' V(C)='vv-ini'                           $ .ic V(a)={gg} V(a2)=ini
M0 pT AT T nsub n1                                     $ pass gate
M1 pT AC T psub p1
M2 pT AC C nsub n1                                     $ pass gate
M3 pT AT C psub p1
.if (CLAMP=1)
M4 GND AC T nsub n1                                    $ clamp
M5 GND AT C nsub n1
M6 GND C1 T nsub n1                                    $ clamp
M7 GND C1 C nsub n1
.endif
.ENDS QAAmp

* This is the latched version; it is just a QAAmp followed by a pass gate.
* Erik's "two hat" adiabatic amplifier plus pass gate. In S2LAL notation, it expects data input as A^ and -A^. Given this, it produces the correct output
* re. (a) [ZF008 Fig. 8c] followed by two pass gates or (b) [ZF008 Fig. 9a, right side] but a non-inverting buffer instead of an AND gate.
* Same role in framework as [S2LAL Fig. 5 (left)].
*  :           :           :           :           :           :           :           :           :
* 1: A(i-1)^   2: -A(i-1)^
* 3: C(i)^     4: -C(i)^
* 5: phi(i)^   6: Clmp(i-1)v
* 7: phi(j)^   8: phi(j)v
* 9: GND      10: PWR
*11: nsub     12: psub
*13: Q(i)^    14: -Q(i)^   15: tap      16: tap
.SUBCKT qLatch AT AC QT QC piT C1i pjT pjC GND PWR      $ One phase of the 2LAL shift register. Args: AT/C QT/C clockiT&clamp clockjT/C
+ nsub psub tap0 tap1 tap2 tap3 ini='gg'                $ substrate supplies
r0 tap0 T 1                                            $ green
r1 tap1 C 1                                            $ red
r2 tap2 piT 1e9                                        $ blue
r3 tap3 C1i 1e9                                        $ yellow
X1 AT AC T C piT C1i GND nsub psub QAAmp 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 qLatch

* One phase of a Q2LAL shift register [ZF008 Fig. 7b].
* Same role in framework as one loop of [S2LAL Fig. 6].
*  :           :           :           :           :           :           :           :
* 1: S0        2: -S0
* 3: S1        4: -S1
* 5: phi(0)    6: -phi(0)   7: phi(1)    8: -phi(1)   9: phi(2)   10: -phi(2)
*11: phi(3)   12: -phi(3)
*13: GND      14: PWR
*15: nsub     16: psub
*17: Q(i)^    18: -Q(i)^   19: q(i)^    20: -q(i)^   21: tap      22: tap      23: tap      24: tap
.SUBCKT qPhase S0T S0C S1T S1C                          $ One stage of the 2LAL shift register. Args: AT/C QT/C
+ p0T p0C p1T C11 p2T C12 p3T p3C GND PWR nsub psub      $ two clocks T/C and two clocks T&clamp DC Supply substrate supplies
+ tap0 tap1 tap2 tap3 tap4 tap5 tap6 tap7 ini='gg'
r0 tap0 t0 1
r1 tap1 t1 1
r2 tap2 t2 1
r3 tap3 t3 1
X0  S0T S0C S1T S1C p1T C11 p0T p0C GND PWR nsub psub t0 t1 tap4 tap5 qLatch ini=ini
X10 S1T S1C S0T S0C p2T C12 p3T p3C GND PWR nsub psub t2 t3 tap6 tap7 qLatch ini=ini
.ends qPhase

* 8 phases of a Q2LAL shift register [ZF008 Fig. 7b].
* Same role in framework as one loop of [S2LAL Fig. 6].
*  :           :           :           :           :           :           :           :
* 1: S0        2: -S0
* 3: S8        4: -S8
* 5: phi(0)    6: -phi(0)   7: phi(1)    8: -phi(1)   9: phi(2)   10: -phi(2)  11: phi(3)   12: -phi(3)
*13: phi(4)   14: -phi(4)  15: phi(5)   16: -phi(5)  17: phi(6)   18: -phi(6)  19: phi(7)   20: -phi(7)
*21: Clmp(0)v 22: Clmp(1)v 23: Clmp(2)v 24: Clmp(3)v 25: Clmp(4)v 26: Clmp(5)v 27: Clmp(6)v 28: Clmp(7)
*29: GND      30: PWR      31: nsub     32: psub
*33: tap      34: tap      35: tap      36: tap
.SUBCKT qDelay SiT SiC S7T S7C                          $ Four phases that just delay. Args: 2*{ data<n>T/C }
+ p0T p1T p2T p3T p4T p5T p6T p7T                       $ clocks/power supplies
+ C10 C11 C12 C13 C14 C15 C16 C17                       $ clamps
+ tap8 tap9 tapA tapB                                   $ debugging taps
+ tapC tapD tapE tapF                                   $ debugging taps and initialization
+ GND PWR nsub psub ini='gg'                            $ DC Supply substrate supplies
R8 tap8 t100 1
R9 tap9 t110 1
RA tapA t120 1
RB tapB t130 1
RC tapC t140 1
RD tapD t150 1
RE tapE t160 1
RF tapF t170 1
X0  S0T S0C S1T S1C p0T p4T p1T C10 p2T C11 p3T p7T GND PWR nsub psub t100 t101 t102 t103 t200 t201 t202 t203 qPhase ini=gg
X1  S1T S1C S2T S2C p1T p5T p2T C11 p3T C12 p4T p0T GND PWR nsub psub t110 t111 t112 t113 t210 t211 t212 t213 qPhase ini=ini
X2  S2T S2C S3T S3C p2T p6T p3T C12 p4T C13 p5T p1T GND PWR nsub psub t120 t121 t122 t123 t220 t221 t222 t223 qPhase ini=ini
X3  S3T S3C S4T S4C p3T p7T p4T C13 p5T C14 p6T p2T GND PWR nsub psub t130 t131 t132 t133 t230 t231 t232 t233 qPhase ini=ini
X4  S4T S4C S5T S5C p4T p0T p5T C14 p6T C15 p7T p3T GND PWR nsub psub t140 t141 t142 t143 t240 t241 t242 t243 qPhase ini=ini
X5  S5T S5C S6T S6C p5T p1T p6T C15 p7T C16 p0T p4T GND PWR nsub psub t150 t151 t152 t153 t250 t251 t252 t253 qPhase ini=ini
X6  S6T S6C S7T S7C p6T p2T p7T C16 p0T C17 p1T p5T GND PWR nsub psub t160 t161 t162 t163 t260 t261 t262 t263 qPhase ini=gg
X7  SiT SiC S0T S0C p7T p3T p0T C17 p1T C10 p2T p6T GND PWR nsub psub t170 t171 t172 t173 t270 t271 t272 t273 qPhase ini=gg
.ENDS qDelay

* Clamp waveform [ZF008 Fig. 8d]. Operates in two modes:
* Production, where the wave is generated from four clocks [ZF008 Fig. 8d].
* Testing, where the function is created from a hardcoded ngspice clock. This clock is included in the power computation.
* Choose by switch the condition between .if (1) and .if (0)
*  :           :           :           :           :           :           :           :
* 1: Phi(i+4)^ 2: Phi(i-2)^
* 3: Phi(i-1)^ 4: Phi(i-1)v
* 5: Test      6: Clmp(i)v
.SUBCKT Clp Pip4 Pim2 Pmlhat Pmlcup Test Clmp           $ Phi(i+4) Phi(i-2) Phi(i-1)hat Phi(i-1)cup Clamp(i)
+ nsub psub                                            $ Substrate supplies
.if (cwf!=0)
M0 Pip4  Pmlhat Clmp nsub n1                            $ pass gate
M1 Pip4  Pmlcup Clmp psub p1
M2 Pim2 Pmlcup Clmp nsub n1                             $ pass gate
M3 Pim2 Pmlhat Clmp psub p1
* C1 Clmp 0 5p
.else
R1 Test Clmp 0 $1000                                       $ basically a direct connection
* C1 Clmp 0 10p
.endif
.ENDS Clp

* Special circuit waveform [ZF008 Fig. 10b].
*  :           :           :           :           :           :           :           :
* 1: Phi(i-1)v 2: Phi(i+1)v
* 3: Phi(i+2)^ 4: Phi(i+2)v
* 5: A(i-5)^   6: -A(i-5)^
* 7: Phi(i)^   8: J(i)^
.SUBCKT Spec Pip4 Pim2 Pmlcup Pmlhat AT AC Picup J      $ Phi(i+4) Phi(i-2) Phi(i-1)hat Phi(i-1)cup Clamp(i)
+ VDD nsub psub                                        $ Substrate supplies
M0 Pip4 Pmlcup c nsub n1                                $ pass gate
M1 Pip4 Pmlhat c psub p1
M2 Pim2 Pmlhat c nsub n1                                $ pass gate
M3 Pim2 Pmlcup c psub p1
M4 VDD c J psub p1                                      $ c is c(i+3)hat
M5 VDD AT J psub p1
M6 Picup AT J nsub n1
M7 Picup AC J psub p1
.ENDS Spec

* 8 phases of a Q2LAL shift register [ZF008 Fig. 7b].
* Same role in framework as one loop of [S2LAL Fig. 6].
*  :           :           :           :           :           :           :           :
* 1: S0        2: -S0
* 3: S8        4: -S8
* 5: phi(0)    6: -phi(0)   7: phi(1)    8: -phi(1)   9: phi(2)   10: -phi(2)  11: phi(3)   12: -phi(3)
*13: phi(4)   14: -phi(4)  15: phi(5)   16: -phi(5)  17: phi(6)   18: -phi(6)  19: phi(7)   20: -phi(7)
*21: Clmp(0)v 22: Clmp(1)v 23: Clmp(2)v 24: Clmp(3)v 25: Clmp(4)v 26: Clmp(5)v 27: Clmp(6)v 28: Clmp(7)v
*29: GND      30: PWR      31: nsub     32: psub
*33: tap      34: tap      35: tap      36: tap
.SUBCKT qDataClock SiT SiC S7T S7C                      $ Four phases that just delay. Args: 2*{ data<n>T/C }
+ p0T p1T p2T p3T p4T p5T p6T p7T                       $ clocks/power supplies
+ C10 C11 C12 C13 C14 C15 C16 C17                       $ clamps
+ J0 J1 J2 J3 J4 J5 J6 J7                               $ Generated clocks. These are the "hat" clocks; J(n)v = J(n + 4 mod 8)^
+ GND PWR nsub psub ini='gg'                            $ DC Supply substrate supplies

X0  S0T S0C S1T S1C p0T p4T p1T C10 p2T C11 p3T p7T GND PWR nsub psub J1   t101 t102 t103 t200 t201 t202 t203 qPhase ini=gg
```

```
X1  S1T S1C S2T S2C p1T p5T p2T C11 p3T C12 P4T p0T GND PWR nsub psub J2   t111 t112 t113 t210 t211 t212 t213 qPhase ini=ini
X2  S2T S2C S3T S3C p2T p6T p3T C12 P4T C13 P5T p1T GND PWR nsub psub J3   t121 t122 t123 t220 t221 t222 t223 qPhase ini=ini
X3  S3T S3C S4T S4C p3T p7T P4T C13 P5T C14 P6T p2T GND PWR nsub psub t130 t131 t132 t133 t230 t231 t232 t233 qPhase ini=ini
X4  S4T S4C S5T S5C P4T p0T P5T C14 P6T C15 P7T p3T GND PWR nsub psub t140 t141 t142 t143 t240 t241 t242 t243 qPhase ini=ini
X5  S5T S5C S6T S6C P5T p1T P6T C15 P7T C16 P0T p4T GND PWR nsub psub t150 t151 t152 t153 t250 t251 t252 t253 qPhase ini=ini
X6  S6T S6C S7T S7C P6T p2T P7T C16 P0T C17 P1T p5T GND PWR nsub psub t160 t161 t162 t163 t260 t261 t262 t263 qPhase ini=gg
X7  SiT SiC S0T S0C P7T p3T P0T C17 P1T C10 P2T p6T GND PWR nsub psub J0   t171 t172 t173 t270 t271 t272 t273 qPhase ini=gg
* Selectively turn on/off the data-controlled clock. Connect clocks to PWR when off to avoid an error when trying to plot a disconnected node.
.if (dcc!=0)
X8  p7T p1T p6T p2T SiT SiC p4T J4 PWR nsub psub Spec
X9  p0T p2T p7T p3T S0T S0C p5T J5 PWR nsub psub Spec
X10 p1T p3T p0T p4T S1T S1C p6T J6 PWR nsub psub Spec
X11 p2T p4T p1T p5T S2T S2C p7T J7 PWR nsub psub Spec
.else
R0 J4 PWR 1e6
R1 J5 PWR 1e6
R2 J6 PWR 1e6
R3 J7 PWR 1e6
.endif
.ENDS qDataClock

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

*** CLOCKS -- Original 8 clock phases and inverses (total eight unique signals), but with slow and fast phase 1's (total 12 unique signals)
.param simlen=periods*period                                $ length of the plot in time

$ 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 tick=period/(8+FastSlow*2)                           $ regular: period/8; FastSlow: period/10
.param Fast=FastSlow*tick                                   $ regular: tick; FastSlow: tick

.param Ramp=(1-2*Porch)*tick                                $ waveform is parameterized so there is a "porch" on either side of a ramp
.param PPT=Porch*tick                                       $ one PPT at beginning and end of sequence, two of these PPTs between ramps

$ Parameters for three-segment ramp
.param Rx=GenT*Ramp                                         $ end time of initial gentle rise
.param v2=GenV*vv                                           $ end height of initial gentle rise

.param Ry=(1-GenT)*Ramp                                     $ start time of final gentle rise
.param v3=(1-GenV)*vv                                       $ start height of final gentle rise

.param Rp=Ramp                                              $ total length of ramp

.param ticks=simlen/tick                                    $ number of ticks in the simulation
.param ttn=18000ns                                          $ integration time for energy

.param tstep=25NS*period/10u*periods/20                             $ time of a simulation step, so number of steps is tick*ticks/tstep

$ The clocks comprise a series of transitions (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

* Clamp waveforms that are high for one tick to clamp signals to ground. VCi is high on tick i-1. These are for testing only.
* Each can be generated with four transistors from existing clocks. They only connect to transistor gates, so they do not need a lot of drive capability.
Vc0    720 0 DC 'vv' PWL('0' 'vv' 'f0uS' 'vv' 'f0uS+Rx' 'v3' 'f0uS+Ry' 'v2' 'f0uS+Rp' 'gg' 'f2dS' 'gg' 'f2dS+Rx' 'v2' 'f2dS+Ry' 'v3' 'f2dS+Rp' 'vv' 'epoc' 'vv' r='0')
Vc1    721 0 DC 'vv' PWL('0' 'vv' 'f1up' 'vv' 'f1up+Rx' 'v3' 'f1up+Ry' 'v2' 'f1up+Rp' 'gg' 'f3dn' 'gg' 'f3dn+Rx' 'v2' 'f3dn+Ry' 'v3' 'f3dn+Rp' 'vv' 'epoc' 'vv' r='0')
Vc2    722 0 DC 'gg' PWL('0' 'gg' 'f0uS' 'gg' 'f0uS+Rx' 'v2' 'f0uS+Ry' 'v3' 'f0uS+Rp' 'vv' 'f2up' 'vv' 'f2up+Rx' 'v3' 'f2up+Ry' 'v2' 'f2up+Rp' 'gg' 'epoc' 'gg' r='0')
Vc3    723 0 DC 'gg' PWL('0' 'gg' 'f1up' 'gg' 'f1up+Rx' 'v2' 'f1up+Ry' 'v3' 'f1up+Rp' 'vv' 'f3up' 'vv' 'f3up+Rx' 'v3' 'f3up+Ry' 'v2' 'f3up+Rp' 'gg' 'epoc' 'gg' r='0')
Vc4    724 0 DC 'gg' PWL('0' 'gg' 'f2up' 'gg' 'f2up+Rx' 'v2' 'f2up+Ry' 'v3' 'f2up+Rp' 'vv' 'f0dn' 'vv' 'f0dn+Rx' 'v3' 'f0dn+Ry' 'v2' 'f0dn+Rp' 'gg' 'epoc' 'gg' r='0')
Vc5    725 0 DC 'gg' PWL('0' 'gg' 'f3up' 'gg' 'f3up+Rx' 'v2' 'f3up+Ry' 'v3' 'f3up+Rp' 'vv' 'f1dn' 'vv' 'f1dn+Rx' 'v3' 'f1dn+Ry' 'v2' 'f1dn+Rp' 'gg' 'epoc' 'gg' r='0')
Vc6    726 0 DC 'gg' PWL('0' 'gg' 'f0dn' 'gg' 'f0dn+Rx' 'v2' 'f0dn+Ry' 'v3' 'f0dn+Rp' 'vv' 'f2dS' 'vv' 'f2dS+Rx' 'v3' 'f2dS+Ry' 'v2' 'f2dS+Rp' 'gg' 'epoc' 'gg' r='0')
Vc7    727 0 DC 'gg' PWL('0' 'gg' 'f1dn' 'gg' 'f1dn+Rx' 'v2' 'f1dn+Ry' 'v3' 'f1dn+Rp' 'vv' 'f3dn' 'vv' 'f3dn+Rx' 'v3' 'f3dn+Ry' 'v2' 'f3dn+Rp' 'gg' 'epoc' 'gg' r='0')

* These are the power clocks, including separate fast and slow clocks
Vphi0P 110 0 DC 'gg' PWL('0' 'gg' 'f0uS' 'gg' 'f0uS+Rx' 'v2' 'f0uS+Ry' 'v3' 'f0uS+Rp' 'vv' 'f0dn' 'vv' 'f0dn+Rx' 'v3' 'f0dn+Ry' 'v2' 'f0dn+Rp' 'gg' 'epoc' 'gg' r='0')
Vphi0f 510 0 DC 'gg' PWL('0' 'gg' 'f0uF' 'gg' 'f0uF+Rx' 'v2' 'f0uF+Ry' 'v3' 'f0uF+Rp' 'vv' 'f0dn' 'vv' 'f0dn+Rx' 'v3' 'f0dn+Ry' 'v2' 'f0dn+Rp' 'gg' 'epoc' 'gg' r='0')
Vphi1P 111 0 DC 'gg' PWL('0' 'gg' 'f1up' 'gg' 'f1up+Rx' 'v2' 'f1up+Ry' 'v3' 'f1up+Rp' 'vv' 'f1dn' 'vv' 'f1dn+Rx' 'v3' 'f1dn+Ry' 'v2' 'f1dn+Rp' 'gg' 'epoc' 'gg' r='0')
Vphi2P 112 0 DC 'gg' PWL('0' 'gg' 'f2up' 'gg' 'f2up+Rx' 'v2' 'f2up+Ry' 'v3' 'f2up+Rp' 'vv' 'f2dS' 'vv' 'f2dS+Rx' 'v3' 'f2dS+Ry' 'v2' 'f2dS+Rp' 'gg' 'epoc' 'gg' r='0')
Vphi2f 512 0 DC 'gg' PWL('0' 'gg' 'f2up' 'gg' 'f2up+Rx' 'v2' 'f2up+Ry' 'v3' 'f2up+Rp' 'vv' 'f2dF' 'vv' 'f2dF+Rx' 'v3' 'f2dF+Ry' 'v2' 'f2dF+Rp' 'gg' 'epoc' 'gg' r='0')
Vphi3P 113 0 DC 'gg' PWL('0' 'gg' 'f3up' 'gg' 'f3up+Rx' 'v2' 'f3up+Ry' 'v3' 'f3up+Rp' 'vv' 'f3dn' 'vv' 'f3dn+Rx' 'v3' 'f3dn+Ry' 'v2' 'f3dn+Rp' 'gg' 'epoc' 'gg' r='0')
Vphi4f 514 0 DC 'vv' PWL('0' 'vv' 'f0uF' 'vv' 'f0uF+Rx' 'v3' 'f0uF+Ry' 'v2' 'f0uF+Rp' 'gg' 'f0dn' 'gg' 'f0dn+Rx' 'v2' 'f0dn+Ry' 'v3' 'f0dn+Rp' 'vv' 'epoc' 'vv' r='0')
Vphi4P 114 0 DC 'vv' PWL('0' 'vv' 'f0uS' 'vv' 'f0uS+Rx' 'v3' 'f0uS+Ry' 'v2' 'f0uS+Rp' 'gg' 'f0dn' 'gg' 'f0dn+Rx' 'v2' 'f0dn+Ry' 'v3' 'f0dn+Rp' 'vv' 'epoc' 'vv' r='0')
Vphi5P 115 0 DC 'vv' PWL('0' 'vv' 'f1up' 'vv' 'f1up+Rx' 'v3' 'f1up+Ry' 'v2' 'f1up+Rp' 'gg' 'f1dn' 'gg' 'f1dn+Rx' 'v2' 'f1dn+Ry' 'v3' 'f1dn+Rp' 'vv' 'epoc' 'vv' r='0')
Vphi6f 516 0 DC 'vv' PWL('0' 'vv' 'f2up' 'vv' 'f2up+Rx' 'v3' 'f2up+Ry' 'v2' 'f2up+Rp' 'gg' 'f2dF' 'gg' 'f2dF+Rx' 'v2' 'f2dF+Ry' 'v3' 'f2dF+Rp' 'vv' 'epoc' 'vv' r='0')
Vphi6P 116 0 DC 'vv' PWL('0' 'vv' 'f2up' 'vv' 'f2up+Rx' 'v3' 'f2up+Ry' 'v2' 'f2up+Rp' 'gg' 'f2dS' 'gg' 'f2dS+Rx' 'v2' 'f2dS+Ry' 'v3' 'f2dS+Rp' 'vv' 'epoc' 'vv' r='0')
Vphi7P 117 0 DC 'vv' PWL('0' 'vv' 'f3up' 'vv' 'f3up+Rx' 'v3' 'f3up+Ry' 'v2' 'f3up+Rp' 'gg' 'f3dn' 'gg' 'f3dn+Rx' 'v2' 'f3dn+Ry' 'v3' 'f3dn+Rp' 'vv' 'epoc' 'vv' r='0')

VGND   200 0 DC 'gg'
VPWR   201 0 DC 'vv'

*** TOP-LEVEL CIRCUIT
* Initialization pattern gg gg vv results in 6-cycle 001 000 100 110 111 011; pattern vv gg vv results in 2-cycle 101 010
* Set the q2la1 variable to 0 for a test of the quiet circuit and 1 for standard 2LAL
.if (q2la1!=0)
X0 SAT SAC SBT SBC 110 111 112 113 114 115 116 117 710 711 712 713 714 715 716 717 pp8 pp9 ppA ppB ppC ppD ppE ppF 200 201 200 201 qDataClock ini=gg $ flip for cycle...
X1 SBT SBC SCT SCC 110 111 112 113 114 115 116 117 710 711 712 713 714 715 716 717 uu8 uu9 uuA uuB uuC uuD uuE uuF 200 201 200 201 qDelay ini=gg
X5 SCT SCC SAC SAT 110 111 112 113 114 115 116 117 710 711 712 713 714 715 716 717 xx8 xx9 xxA xxB xxC xxD xxE xxF 200 201 200 201 qDelay ini=vv

X2 SXT SXC SYT SYC 110 111 112 113 114 115 116 117 710 711 712 713 714 715 716 717 qq8 qq9 qqA qqB qqC qqD qqE qqF 200 201 200 201 qDelay ini=gg
X3 SYT SYC SZT SZC 110 111 112 113 114 115 116 117 710 711 712 713 714 715 716 717 vv8 vv9 vvA vvB vvC vvD vvE vvF 200 201 200 201 qDelay ini=gg
X4 SZT SZC SXC SXT 110 111 112 113 114 115 116 117 710 711 712 713 714 715 716 717 ww8 ww9 wwA wwB wwC wwD wwE wwF 200 201 200 201 qDelay ini=vv

X5  114 116 117 113 720 710 nsub psub C1p                   $ VPhi4P VPhi6P VPhi7P VPhi3P C0
X6  115 117 110 114 721 711 nsub psub C1p                   $ VPhi5P VPhi7P VPhi0P VPhi4P C1
X7  116 110 111 115 722 712 nsub psub C1p                   $ VPhi6P VPhi0P VPhi1P VPhi5P C2
X8  117 111 112 116 723 713 nsub psub C1p                   $ VPhi7P VPhi1P VPhi2P VPhi6P C3
X9  110 112 113 117 724 714 nsub psub C1p                   $ VPhi0P VPhi2P VPhi3P VPhi7P C4
X10 111 113 114 110 725 715 nsub psub C1p                   $ VPhi1P VPhi3P VPhi4P VPhi0P C5
X11 112 114 115 111 726 716 nsub psub C1p                   $ VPhi2P VPhi4P VPhi5P VPhi1P C6
X12 113 115 116 112 727 717 nsub psub C1p                   $ VPhi3P VPhi5P VPhi6P VPhi2P C7

.if (ats)
* AND gate [ZF007 Fig. 9b and c] test process.
* First, create test inputs. Manually change the two q2la1 registers so the initialization patterns are gg gg vv and vv gg vv. This will create period 6 and 2
* repetition patterns. These pattern will naturally creates all four binary combinations of two bits in (for example) SAT/SAC and SXT/SXC. Enable the code below and the
```

```
* "AND test code" plotting. This code below will then compute the AND and NAND function to wires aout and oout. Set FastSlow to 0 to avoid going bonkers.
M1   110 SAT t1 200 n1                                  $ 1. AND function. Two series transmission gates pass the clock when both inputs are asserted
M2   110 SAC t1 201 p1                                  $ 2.
M3    t1 SXT aout 200 n1                                $ 3. Second transmission gate
M4    t1 SXC aout 201 p1                                $ 4.
M5    t1 SAC aout 200 n1                                $ 5. Internal node clamp
M6    t1 727 aout 200 n1                                $ 6. Idle internal node clamp
M7     0 SAC aout 200 n1                                $ 7. Output pull down
M8     0 727 aout 200 n1                                $ 8. Idle output clamp
M9     0 SXC aout 200 n1                                $ 9. Output pull down

M10  110 SAC oout 200 n1                                $ 1. NAND function. Two parallel transmission gates pass the clock when both inputs are asserted
M11  110 SAT oout 201 p1                                $ 2.
M12  110 SXC oout 200 n1                                $ 3. Second transmission gate
M13  110 SXT oout 201 p1                                $ 4
M14   t2 SAT oout 200 n1                                $ 5. Output pull down
M15    0 727 oout 200 n1                                $ 6. Idle clamp
M16   t2 SXC oout 200 n1                                $ 7. Internal node clamp
M17   t2 SXT oout 201 p1                                $ 8.
M18    0 SXT   t2 200 n1                                $ 9. Output pull down
M19   t2 727 oout 200 n1                                $ 10. Idle clamp
.endif

.else
X0 SAT SAC SBT SBC 110 111 112 113 114 115 116 117 200 201 200 201 pp4 pp5 pp6 pp7 pp8 pp9 ppA ppB ppC ppD ppE ppF SDELAY ini=gg $ flip for cycle...
X1 SBT SBC SCT SCC 110 111 112 113 114 115 116 117 200 201 200 201 uu0 uu1 uu2 uu3 uu4 uu5 uu6 uu7 uu8 uu9 uuA uuB SDELAY ini=gg
X5 SCT SCC SAT SAC 110 111 112 113 114 115 116 117 200 201 200 201 xx0 xx1 xx2 xx3 xx4 xx5 xx6 xx7 xx8 xx9 xxA xxB SDELAYv ini=vv

X2 SXT SXC SYT SYC 110 111 112 113 114 115 116 117 200 201 200 201 qq0 qq1 qq2 qq3 qq4 qq5 qq6 qq7 qq8 qq9 qqA qqB SDELAY ini=gg
X3 SYT SYC SZT SZC 110 111 112 113 114 115 116 117 200 201 200 201 vv0 vv1 vv2 vv3 vv4 vv5 vv6 vv7 vv8 vv9 vvA vvB SDELAY ini=gg
X4 SZT SZC SXT SXC 110 111 112 113 114 115 116 117 200 201 200 201 ww0 ww1 ww2 ww3 ww4 ww5 ww6 ww7 ww8 ww9 wwA wwB SDELAYv ini=vv
.endif

* power and energy calculation
B4 0 16 V=0
+ +I(Vc0)*v(720)+I(Vc1)*v(721)+I(Vc2)*v(722)+I(Vc3)*v(723)+I(Vc4)*v(724)+I(Vc5)*v(725)+I(Vc6)*v(726)+I(Vc7)*v(727)
+ +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(vphi0f)*v(510)+I(vphi2f)*v(512)+I(vphi4f)*v(514)+I(vphi6f)*v(116)
+ +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)

.option noinit acct

*********************************************************************************************
$ NGSPICE CONTROL AREA
.TRAN 'tstep' 'ticks*tick'
.csparam slen = 'simlen*1e6'
.csparam prds = 'periods'
.csparam epch = 'epoc*1e6'
.csparam ticu = 'tick*1e6'
.csparam ntks = 'ticks'
.csparam tste = 'tstep*1e9'
.csparam fstp = 'Fast*1e6'
.csparam rmpu = Ramp*1e6
.control
pre_set strict_errorhandling
unset ngdebug
echo "********************Sim: $&slen us=$&prds*$&epch us. Tick: $&ticu us=$&ntks*$&tste ns. Rmp: $&rmpu us. Fast $&fstp us."
run

* 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 >>Q2LAL.csv

* 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=1
set xgridwidth=1

set hcopypscolor=1
set hcopyscale=4
set hcopypstxcolor=2
set hcopyfontsize=3
set gnuplot_terminal=png

$ plot                                                  $ plot clock current
gnuplot gp/clkcur
+ title "Clock current. Sim: $&slen us=$&prds*$&epch us. Tick: $&ticu us=$&ntks*$&tste ns. Rmp: $&rmpu us. Fast $&fstp us."
$ + ylimit -5m 5m                                       $ BOOKMARK1
+ ylimit -200u 200u                                    $ BOOKMARK1
+ I(Vphi0P) I(Vphi0f) I(Vphi1P) I(Vphi2P) I(Vphi2f) I(Vphi3P) I(Vphi4f) I(Vphi4P) I(Vphi5P) I(Vphi6f) I(Vphi6P) I(Vphi7P)

plot                                                    $ plot instantaneous energy consumption
$ gnuplot gp/power
+ title "Dissipation. Sim: $&slen us=$&prds*$&epch us. Tick: $&ticu us=$&ntks*$&tste ns. Rmp: $&rmpu us. Fast $&fstp us."
+ ylimit -25m 25m
+ v(16)

plot                                                    $ plot accumulated energy dissipation
$ gnuplot gp/energy
+ title "Cum. dissipation. Sim: $&slen us=$&prds*$&epch us. Tick: $&ticu us=$&ntks*$&tste ns. Rmp: $&rmpu us. Fast $&fstp us."
+ ylimit 0 70n
+ v(17)

plot ylimit 0 7 xlimit 0 50u
$ gnuplot gp/traces ylimit 0 7 xlimit 0 200u
+ title "3-stage Q2LAL/S2LAL inverting shift register"
+ v(ppF)/49.99*0.9+ 4.55+.000*10
+ v(ppE)/49.99*0.9+ 4.55+.025*10
+ v(ppD)/49.99*0.9+ 4.55+.050*10
+ v(ppC)/49.99*0.9+ 4.55+.075*10
+ v(ppB)/49.99*0.9+ 4.55+.100*10
+ v(ppA)/49.99*0.9+ 4.55+.125*10
+ v(pp9)/49.99*0.9+ 4.55+.150*10
+ v(pp8)/49.99*0.9+ 4.55+.175*10
+
* These lines create a non-controlled set of waveforms to make [ZF008 Fig. 10a] more understandable BOOKMARK2
* + v(117)/49.99*0.9+ 1.55+.000*10
* + v(116)/49.99*0.9+ 1.55+.025*10
* + v(115)/49.99*0.9+ 1.55+.050*10
* + v(114)/49.99*0.9+ 1.55+.075*10
```

```
* + v(113)/49.99*0.9+ 1.55+.100*10
* + v(112)/49.99*0.9+ 1.55+.125*10
* + v(111)/49.99*0.9+ 1.55+.150*10
* + v(110)/49.99*0.9+ 1.55+.175*10
* end BOOKMARK2
+
* AND test code
* + v(oout)/49.99*0.9+2.55+.175*10                          $ NAND output, allows AND and NAND to be a two-rail signal (green)
* + v(aout)/49.99*0.9+2.55+.150*10                          $ AND output (red)
* + v(727)/49.99*0.9+2.55+.125*10                           $ clamp c(i-1)v in [ZF008 Fig. 9b and c] (blue)
* + v(117)/49.99*0.9+2.55+.100*10                           $ clock for the next phase, phi(i)^ in [ZF008 Fig. 9b and c] (yellow)
* + v(SXC)/49.99*0.9+2.55+.075*10                           $ B input complementary value, asserted when B is 0 (magenta)
* + v(SAC)/49.99*0.9+2.55+.050*10                           $ A input complementary value, asserted when A is 0 (turquoise)
* + v(SXT)/49.99*0.9+2.55+.025*10                           $ B input (orange)
* + v(SAT)/49.99*0.9+2.55+.000*10                           $ A input (brown)
+
+ v(uu8)/9.99*0.9+2.55
+
* These lines are the source of [ZF008 Fig. 12b and d]
+ v(SAT)/9.99*0.9+ 0.55
+ v(SAC)/9.99*0.9+ 0.55+.05

.endc

.END
* Notes:
* Q2LAL is a significant conceptual modification to S2LAL, albeit one that differs only in one transistor.
* Q2LAL transmits bits in straightforward dual-rail, which means a 1 is a pulse from 0 V to Vdd. Using S2LAL terminology, this is a "hat" pulse, meaning it has
* the most positive voltage in the middle. A Q2LAL 0 is a "hat" pulse on a second wire. In contrast, S2LAL sends a 1 on two wires, a hat pulse like Q2LAL but also
* an electrically inverted pulse on a different wire, i. e. a pulse from the idle Vdd state to 0 V. S2LAL sends a 0 by leaving both wires in the idle state.
*
* Tested with ngspice-30 (creation date Dec 28, 2018, from ngspice-30_64.zip 8,687,648 bytes)
*
* For tutorial docs: no tabs; comments start column 61; 169 character maximum line length
*
* Notation: A postive pulse A is designated in print with a circumflex (^) diacritical mark. It may be designated here as "A-hat" or "A^"; a negative pulse is
* designated in print with a caron (v) diacritical mark. It may be designated here as "A-cup" or Av. In this notation, -A^ does not mean -(A^) = Av but rather (-A)^,
* a positive-going pulse when A is 0
*
* References:
* [ZF008] DeBenedictis, Erik. "Energy Management with Adiabatic Circuits." Technical report ZF008 (publication pending)
* [ZF007] http://zettaflops.org/CATC/DPA-Q2LAL.pdf December 19, 2020 Document ZF007
* [Q2LALv2.ppt] Slide deck DPA-Q2LALv2.ppt January 2, 2021 Document ZF007, a non-public PowerPoint on Erik's computer
* [S2LAL] Frank, Michael P., et al. "Reversible Computing with Fast, Fully Static, Fully Adiabatic CMOS." arXiv preprint arXiv:2009.00448 (2020).
* [Athas] Athas, W. C., et al. "Low-power digital systems based on adiabatic-switching principles." IEEE Transactions on VLSI Systems 2.4 (1994): 398-407
```