# Discrete Semantics for Reversible Transistor Network Verification

Hannah Blyton and Hugh Potter · RC 2026

# Two kinds of chip bug


## Pentium FDIV: logical bug

· The implemented arithmetic did not match the intended digital behaviour.

## Z80 ED-71: floating bus

· An undefined instruction samples an electrically floating internal bus.

# How we prevent them

- **Formal verification:** assertions, model checking, equivalence checking, STE, SMT, and theorem proving for RTL and above
- **Electrical rule checking:** foundry and PDK rules over extracted netlists, including shorts, floating nodes, drive strength, and conduction cases
- **Simulation:** RTL, gate-level, switch-level, and analogue simulation, each at a different abstraction boundary
- **Silicon testing:** process corners, voltage and temperature corners, burn-in, ageing, noise, power integrity, and long-run stress

# DSRT

Adiabatic switching rules are electrical rules, not logical rules.

We want formal verification at the switch level
for adiabatic transistor networks.

# DSRT: Machine-checked semantics for adiabatic switching

- Electrical adiabaticity constraints formalised in Lean.
- Logical reversibility comes as a corollary of constraints.
- Definitions support machine-checked proofs on circuit behaviour.
- Enables easier switch-level reasoning.

# DSRT: Applications

- `dsrt-sim`: a verified timestep simulator
  - Simple timestep simulation relation: adiabaticity forbids switching cascades, avoiding the cycle resolution required for traditional switch-level simulation. $O(t + n)$
  - Verified correct in forward and reverse directions.
  - Python + Rust simulator implementations.
- Verification of new logic families
- Verification of compiler output

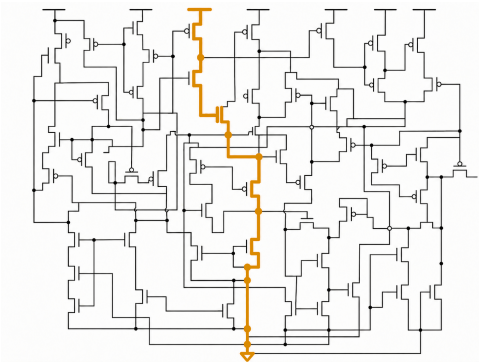
# Adiabatic switching rules

(0) No active shorts.

1. Avoid the use of diodes.
2. Allow voltages to change only via slow, linear ramps.
3. Avoid turning a transistor on when its source and drain are at different voltages (no sparks).
4. Avoid turning a transistor off while current is passing through it, unless there is an alternate low-impedance path (no squelches).

Frank et al. [1]

# (0) No active shorts

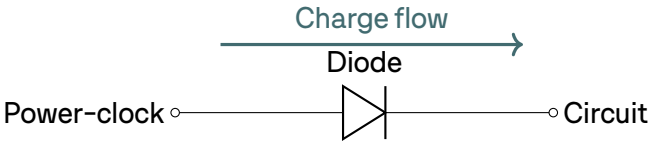


$$CON(S) \iff \nexists\, 0 \nrightarrow_S 1$$

No conductive path may join a node driven to $1$ with one driven to $0$.

$CON$ is the baseline well-formedness condition before any adiabatic handoff is considered.

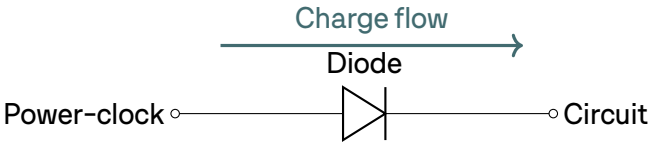
# 1. Avoid the use of diodes




$$P_{\text{dissipated}} = V_D I$$

A conducting diode's built-in voltage drop prevents fully adiabatic charge recovery.

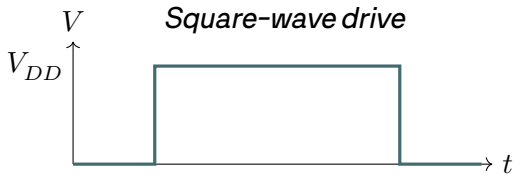
# 1. Avoid the use of diodes

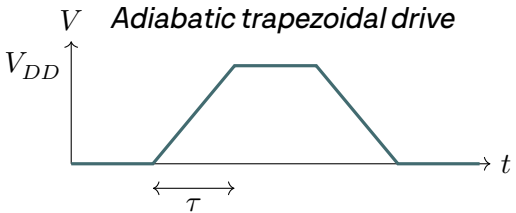



$$P_{\text{dissipated}} = V_D I$$

**DSRT networks do not contain diodes**; they are excluded by construction.

# 2. Allow voltages to change only via slow, linear ramps




$$E_{\text{loss}} = \frac{1}{2} C V_{DD}^2$$

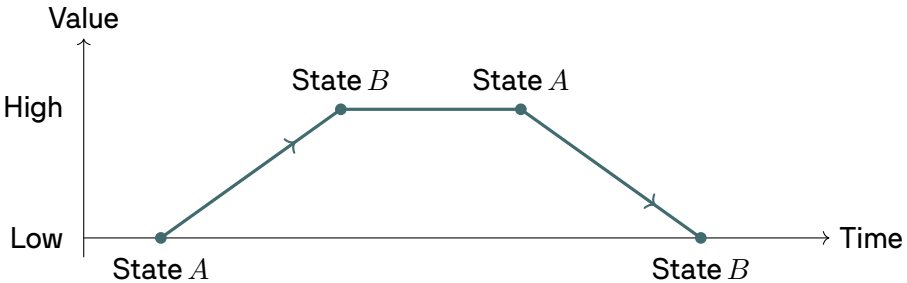
Per charging event

$$E_{\text{loss}} \approx \frac{RC}{\tau} C V_{DD}^2$$

Vanishes as $\tau / RC \to \infty$

Athas et al. [2]

# 2. Allow voltages to change only via slow, linear ramps

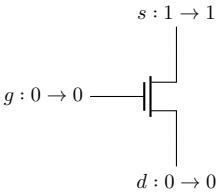



The low/rising/high/falling four-phase waveform is implicit, while the transition relation checks one two-state window $(A, B)$ at a time.

# 3. Avoid turning a transistor on when its source and drain are at different voltages (no sparks)

**Allowed**

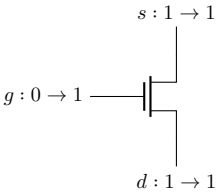
*Gate unchanged*




The transistor stays off; no conducting path appears.

**Allowed**

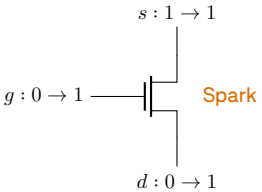
*Gate changes; endpoints agree*




The transistor turns on across zero voltage difference.

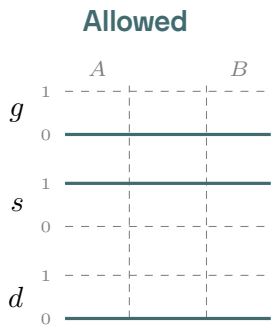
**Not allowed**

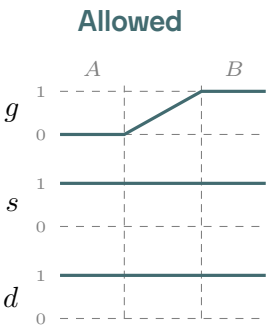
*Gate changes; endpoints disagree*

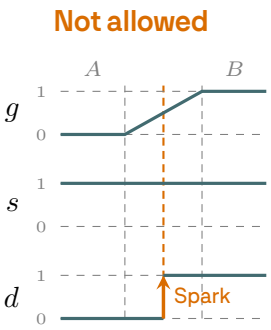



Turning the transistor on forces unequal voltages to equalise.

# 3. Avoid turning a transistor on when its source and drain are at different voltages (no sparks)



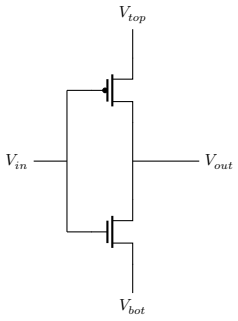
# 3. Avoid turning a transistor on when its source and drain are at different voltages (no sparks)

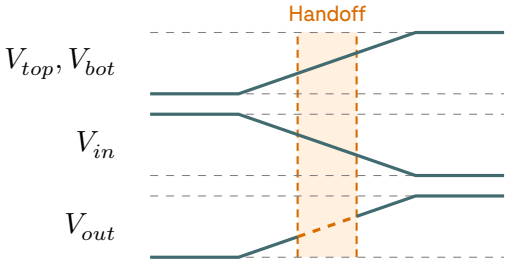
For every transistor $t$, with gate $g_t$, source $s_t$, and drain $d_t$:

$$ADB(A, B) \iff \forall t,\ A(g_t) \neq B(g_t) \implies (A(s_t) = A(d_t)) \wedge (B(s_t) = B(d_t)).$$

"For every transistor, if its gate changes between state $A$ and state $B$, then its source and drain must already agree in state $A$, and they must also agree in state $B$."

# 4. Avoid turning a transistor off while current is passing through it, unless there is an alternate low-impedance path (no squelches)




Both rails ride one rising ramp and the gates share $V_{in}$. As $V_{in}$ falls, the output hands off from the NMOS to the PMOS path; $V_{out}$ tracks the ramp, so source and drain agree – no spark, $ADB$ holds. But if the NMOS opens before the PMOS closes, $V_{out}$ loses all power mid-current: a squelch, $PTC$ fails.

# 4. Avoid turning a transistor off while current is passing through it, unless there is an alternate low-impedance path (no squelches)

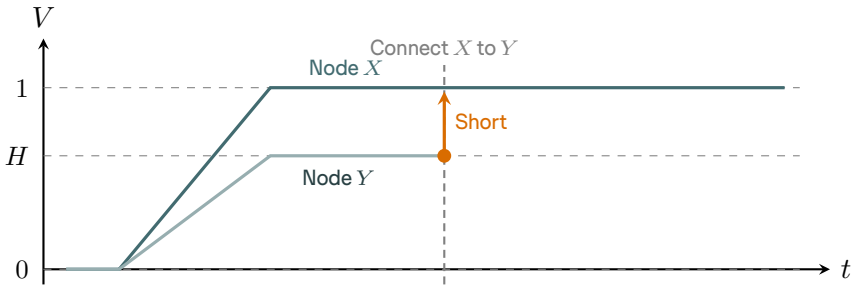
For every node $v$:

$$PTC(A,B) \iff \forall v,\ A(v) \neq B(v) \implies (\exists p,\ \text{powered}_A(p) \wedge v \leftrightarrow_A p \wedge v \leftrightarrow_B p)$$
$$\wedge\ (\exists q,\ \text{powered}_B(q) \wedge v \leftrightarrow_A q \wedge v \leftrightarrow_B q)\,.$$

"If a node changes between state $A$ and state $B$, then it must have a persistent conductive path to an $A$-powered node, and also to a $B$-powered node."

# CLEAN: Don’t charge to weak values

*optional*




Node $X$ is charged by a PMOS; node $Y$ is charged by an NMOS.

# CLEAN: Don’t charge to weak values

*optional*

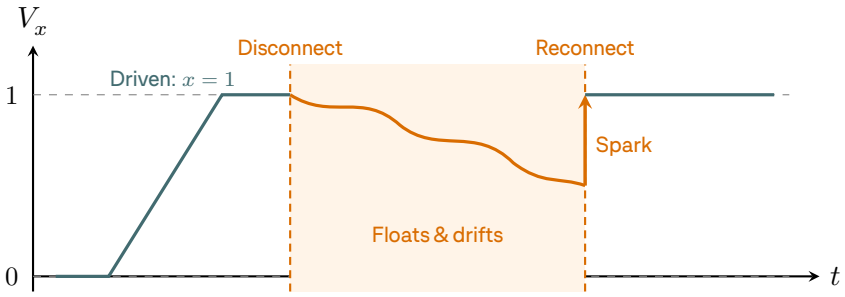
For every node $v$:

$$CLEAN(A, B) \iff \forall v,\ B(v) = 0 \ \vee \ B(v) = 1.$$

“Every sampled node in state $B$ must have a strong value: either $0$ or $1$, not a degraded low or degraded high.”

# STAT: Do not rely on a floating node staying put

*optional*




The discrete state records $x = 1$ throughout, but once the driver disconnects, the real node floats and drifts. Reconnecting to the $1$ driver snaps it back through an uncontrolled current spike – a spark. $STAT$ rules this out by requiring every node to keep a strong path to a powered node, so reconnection can never conceal a short.

# STAT: Do not rely on a floating node staying put

*optional*

For every node $v$:

$$STAT(A, B) \iff \forall v,\ \exists p,\ \text{powered}_B(p) \wedge v \leftrightarrow_B p.$$

"Every node in state $B$ must be connected, through transistors that are on in $B$, to some node powered in state $B$."

# The five constraints at a glance

## Constraints on state $B$

$CON$ Connected nodes have the same value.

$CLEAN$ Every sampled value is strong: $0$ or $1$.
OPTIONAL

$STAT$ Every node has a strong path to a node powered in $B$.
OPTIONAL

## Constraints on transition $(A, B)$

$ADB$ If a transistor gate changes, its source and drain agree in both $A$ and $B$.

$PTC$ Every changed node has a strong, persistent path to powered nodes in both $A$ and $B$.

A valid DSRT step satisfies $CON \wedge ADB \wedge PTC$, with $CLEAN \wedge STAT$ optionally enforced.

# Two useful consequences

$$CON(B) \wedge ADB(A,B) \wedge PTC(A,B) \implies DDC(A,B) \wedge CAP(A,B)$$

*DDC*

Nodes cut off from the $B$-powered components have the same values in $A$ and $B$.

*CAP*

Nodes $A$-connected to a $B$-powered component have its logical value in $B$.

This way of classifying nodes helps us reason about both simulation and logical reversibility.

# DDC: Disconnected nodes do not change

$$DDC(A, B) \iff \forall v,\ \left(\forall p,\ \text{powered}_B(p) \Rightarrow v \nleftrightarrow_A^{\text{strong}} p\right) \Rightarrow A(v) = B(v).$$

*If a node cannot be reached from any $B$-powered node, using strong connections in $A$, then it has the same value in $A$ and $B$.*

Intuition: $PTC$ says every changed node must have a persistent path to powered nodes. So if there is no path to a $B$-powered node, the node cannot have changed.

# CAP: Connected nodes take the powered value

$$CAP(A,B) \iff \forall v,p,\ (v \leftrightarrow_A p \wedge \text{powered}_B(p)) \Rightarrow \text{logic}(B(v)) = \text{logic}(B(p)).$$

*If a node is connected in $A$ to a node powered in $B$, then they have the same logical value in $B$.*

Intuition: Walk along the connection path in $A$. Each transistor either stays on, so $CON(B)$ keeps the endpoints equal, or switches off, so $ADB$ says the endpoints already agree in $B$.

# Logical reversibility

For a transition relation $R$, logical reversibility means uniqueness in both directions:

$$R(A, B) \wedge R(A, C) \Rightarrow B = C$$

$$R(B, A) \wedge R(C, A) \Rightarrow B = C$$

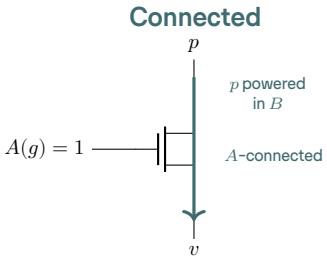
With the same powered assignment fixed, can two different electrical assignments both satisfy

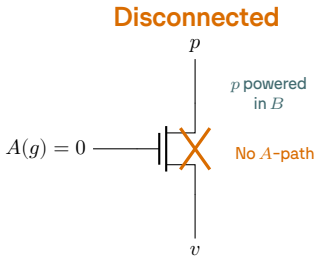
$$CON \wedge ADB \wedge PTC \wedge CLEAN$$

for the same tick?

# Every node is forced

Pick any node $v$. Look at the $A$-connectivity between $v$ and the nodes powered in the next state.




$CAP(A, B)$ :
$\text{logic}(B(v)) = \text{logic}(B(p))$.

$DDC(A, B)$ :
$B(v) = A(v)$.

# Every node is forced

$$CON \wedge ADB \wedge PTC \implies DDC \wedge CAP$$

**Connected in $A$**

$$CAP(A, B)$$

$\implies$ $v$ takes the powered value
therefore uniquely determined

**Disconnected in $A$**

$$DDC(A, B)$$

$\implies$ $v$ keeps its old value
therefore uniquely determined

# Backward determinism

**Forward view**

$$A + \text{powered}_B \Longrightarrow B$$

$$\begin{matrix} CON(A) \\ ADB(A,B) \\ PTC(A,B) \end{matrix} \quad \Longrightarrow \quad \begin{matrix} DDC(A,B) \\ CAP(A,B) \end{matrix}$$

**Backward view**

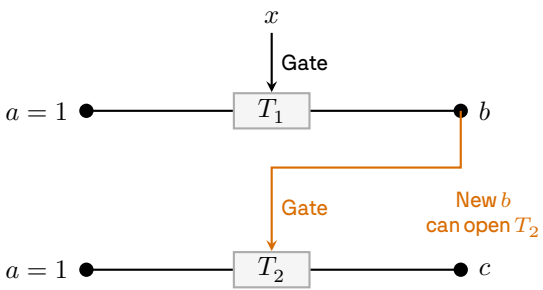
$$B + \text{powered}_A \Longrightarrow A$$

$$\begin{matrix} CON(B) \\ ADB(B,A) \\ PTC(B,A) \end{matrix} \quad \Longrightarrow \quad \begin{matrix} DDC(B,A) \\ CAP(B,A) \end{matrix}$$

$$ADB(A,B) \Longleftrightarrow ADB(B,A), \qquad PTC(A,B) \Longleftrightarrow PTC(B,A).$$

Backward uniqueness is the same local forcing argument, with the two endpoints renamed.

# Aside: Transistors and if statements




**Same cascade as code**

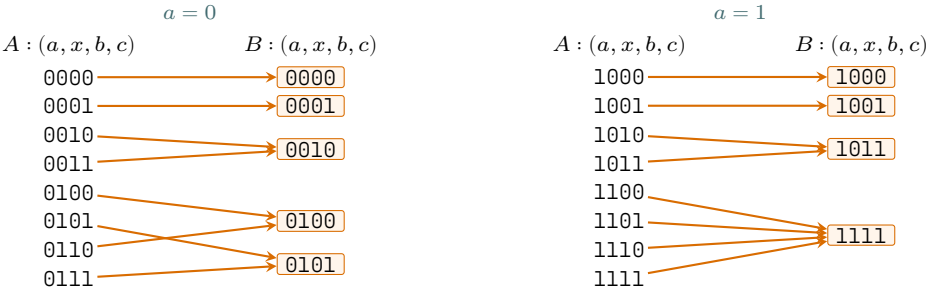
```
if x:
  b := a
if b:
  c := a
```

The second guard reads $b$ after the first assignment.

Ordinary switch-level evaluation may cascade inside one tick.

# Aside: Where reversibility breaks

Instantaneous cascade for `if x: b := a; if b: c := a`



# Why simulation becomes simple

DDC and CAP turn the semantic question into reachability.

| Reached from power in $B$ | Not reached |
| --- | --- |
| $A$-connected to a powered node | no $A$-path to power in $B$ |
| $\Longrightarrow$ take that powered value | $\Longrightarrow$ keep the old value |

This collapse fails for ordinary irreversible gates: the output can be driven by a path whose controlling gate changes in the same tick.

# The simulator

```
open := powered nodes;  all others := unknown
# connected nodes share a value  (CAP)
while open not empty:
    near := pop(open)
    for each on-transistor near -> far:
        drive far;  conflict => short circuit
# unknowns keep their previous value  (DDC)
for each node still unknown:
    node := its value in A
# validate, else: no adiabatic successor
check CON, ADB, PTC, CLEAN
```

# The simulator – full algorithm for reference

▷ Initialise: set powered nodes, mark rest unknown

1: $open \leftarrow \emptyset$; $changed \leftarrow \emptyset$
2: **for** each node $v$ **do**
3: **if** $v$ is powered in $B$ **then**
4: $B(v) \leftarrow$ powered value of $v$; add $v$ to $open$
5: **else**
6: $B(v) \leftarrow$ unknown

▷ Flood-fill from powered nodes using $A$-connectivity (CAP)

7: **while** $open \neq \emptyset$ **do**
8: remove some node $near$ from $open$
9: **for** each transistor $T$ with endpoint $near$ and other endpoint $far$ **do**
10: **if** $T$ is off in $A$ **then continue**
11: $drive \leftarrow$ PASS(type($T$), $B(near)$)
12: **if** $B(far) =$ unknown **then**
13: $B(far) \leftarrow drive$; add $far$ to $open$ and $changed$
14: **else if** LOGIC($B(far)$) $\neq$ LOGIC($drive$) **then**
15: **error:** short circuit at $far$
16: **else**
17: $B(far) \leftarrow$ STRONGER($B(far)$, $drive$)

▷ Set unknowns to previous value (DDC)

18: **for** each node $v$ with $B(v) =$ unknown **do**
19: $B(v) \leftarrow A(v)$

▷ Validation: check $CON$, $ADB$, $PTC$, $CLEAN$

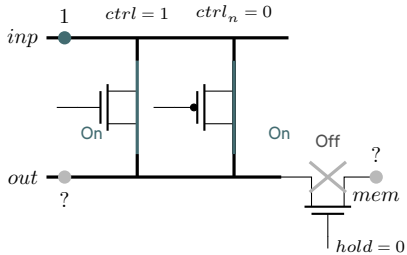
# Conclusions

DSRT: machine-checked semantics for adiabatic switching.

- Electrical adiabaticity constraints formalised in Lean; logical reversibility comes as a corollary.
- Definitions support machine-checked proofs on circuit behaviour.
- Simple timestep simulation relation – adiabaticity forbids switching cascades, so no cycle resolution needed. $O(t + n)$
- Verified simulator, correct in forward and reverse directions; Python + Rust implementations.

Questions?

# Flood fill by hand: A ramp and a stored bit




State $A$: $inp = 0, out = 0$, and a stored bit $mem = 1$.
Powered for $B$: $inp$ ramps to $1$; all gates held.

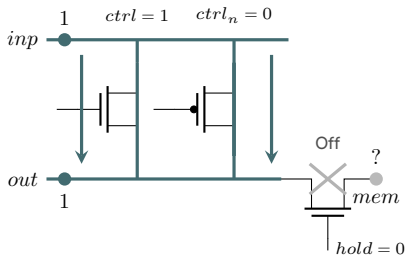
**Initial worklist**

$open = \{inp, ctrl, ctrl_n, hold\}$
$seen = \{inp, ctrl, ctrl_n, hold\}$

**Known assignments**

$B(inp) = 1, \ B(hold) = 0$
$B(ctrl) = 1, \ B(ctrl_n) = 0$
$B(out) = ?, \ B(mem) = ?$

# Flood fill by hand: CAP – the fill reaches $out$

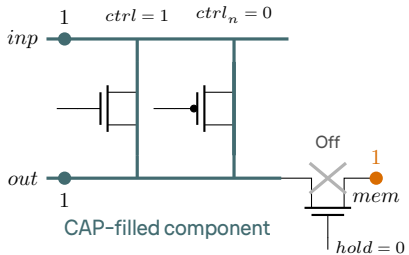



Both branches on in $A$: the ramp reaches $out$.
The off transistor blocks the fill at $mem$.

**Pop** $inp$
NMOS: $pass_N(1) = H$
PMOS: $pass_P(1) = 1$
$B(out) : ? \mapsto H \mapsto 1$

**Pop** $out$
$hold$ is off in $A \Rightarrow$ skip
$open = \emptyset, \quad B(mem) = ?$

# Flood fill by hand: DDC – $mem$ keeps its bit

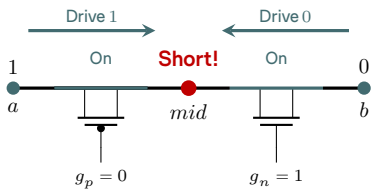



Unreached ⇒ disconnected in $A$ ⇒ the old value stands: $mem$ is remembered, not recomputed.

**DDC**
$B(mem)$ is still unknown
$B(mem) := A(mem) = 1$

**Validate**
$CON$, $ADB$, $PTC$, $CLEAN$
all hold
⇒ unique adiabatic successor

# Flood fill by hand: Colliding fills are shorts




State $A$: $a = mid = b = 0$; both gates on, held in $B$.
Powered for $B$: $a$ ramps to $1$ while $b$ is held at $0$.

**Flood fill**
pop $a$: $pass_P(1) = 1 \Rightarrow B(mid) = 1$
pop $b$: $pass_N(0) = 0$: conflict!

**Short circuit**
logic $1$ meets logic $0$ at $mid$
no state $B$ can satisfy $CON$
halt and report

Rule (0) – no active shorts – enforced mechanically, for free.

# References I

[1] Michael P. Frank et al. “Reversible Computing with Fast, Fully Static, Fully Adiabatic CMOS”. In: *2020 International Conference on Rebooting Computing (ICRC)*. IEEE Computer Society, 2020, pp. 1–8. DOI: 10.1109/ICRC2020.2020.00014.

[2] W. C. Athas et al. “Low-Power Digital Systems Based on Adiabatic-Switching Principles”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 2.4 (1994), pp. 398–407. DOI: 10.1109/92.335009.