# Thermal Logic Circuits

J.G. Koller, W.C. Athas, and L."J." Svensson
{koller,athas,svensson}@isi.edu

USC Information Sciences Institute
4676 Admiralty Way, Marina Del Rey, CA 90292

## Abstract

*Thermal Logic is a hypothetical device technology that allows one to analyze the energetics of computing machines in a simpler setting than real device technologies. The paper describes the rudiments of thermal logic, and uses it to analyze reversible logic pipelines. The similarity between thermal logic and electronic logic is explained, and thermal analogs of electronic devices and circuits are proposed. We show that adiabatically-reversible logic pipelines have a rich mathematical structure, including a local gauge symmetry, and suggest some directions for future research. Adiabatic power supplies are also addressed.*

## 1: Introduction

The field of reversible and adiabatic computing now ranges from the seminal theoretical work[4,7] to small working devices and circuits[2]. However, the theoretical work is typically couched in terms of idealized apparatus such as Turing machines, which bear little resemblance to modern digital computers. Conversely, the device and circuit studies are forced to work within the limitations of existing electronics technology such as bulk CMOS, and it is not always clear whether difficulties are fundamental or simply artifacts of the particular device technology. There is thus a need for a simple model with the following characteristics:

1) It must lead to computer systems structurally similar to existing computers.

2) The physics of the system must be simple, unambiguous and analyzable.

3) It must allow construction of complete self-contained computing systems, including power supplies and Input/Output (I/O) devices.

In this paper we present Thermal Logic, a hypothetical device technology that uses temperature differences and heat instead of voltages and charge to perform computation. The resulting circuits are directly analogous to electrical circuits, and the associated digital structures will certainly look familiar to logic designers. Conversely, the physics is straightforward, and involves only the familiar concepts of traditional elementary thermodynamics: heat, temperature, pressure, volume, energy and entropy.

The model should prove extremely useful in analyzing the energetics of computing machinery. Indeed, we developed it to allow us to study clocking schemes in adiabatic switching, without the complications of real CMOS. We present it here in the hope that others will be able to apply it to other issues.

This paper describes the rudiments of the Thermal Logic idea, and uses it to analyze reversible logic pipelines in a simpler setting than heretofore available in CMOS. In Section 2 we describe the relationship between electronics and thermodynamics that enables the thermal logic model. Seections 3 and 4 describe thermal logic components, and logic gates built from these components. Section 5 discusses reversible circuits, and Section 6 discusses the interesting properties of sequencing required when reversible circuits are concatenated. An Appendix presents some observations on the connection between adiabatic power supplies and phase transitions.

Two things should be emphasized: First, the computers built from thermal logic may or may not be thermodynamically reversible, depending on the circuits themselves and the way they are operated. Second, many of the results are not specific to the thermal logic model, and apply to other computational systems as well.

Our overall goal is to study the universal thermodynamics of self-contained computing systems, and the best understood such systems are modern electronic computers, powered by, for example, a battery. In studying the energetics of electronic logic, one typically computes an energy dissipation by following the flow of charge through voltage differences across switches. However, this approach is unsatisfactory for deducing universal thermodynamic theorems for several reasons. For instance, the "lost" energy is not truly dissipated in a thermodynamic sense: it has mere-



0-8186-6715-X/94 $4.00 

ly been converted to heat, which could conceivably be used as the source of a regeneration process to power another smaller computer. How do we take this into account? Similarly, to derive quantitative results, one needs to idealize the electronic devices in many ways simultaneously, and it is not immediately obvious that our idealizations are mutually consistent: we may have simply thrown away some fundamental relationship.

Thermal Logic (TL) solves both these problems, by exploiting an analogy between charge and heat, which is possible because the first law of thermodynamics relates energy ($E$), temperature ($T$) and entropy ($S$) in almost the same way it relates energy, voltage ($V$) and charge ($Q$): $dE = TdS + VdQ$. Table 1 shows how electronics quantities

| Description | Electronic analog | TL analog |
|---|---|---|
| Potential | Voltage (V) | Temperature (T) |
| Charge | Electric Charge (Q) | Heat (Q) |
| Current | I = dQ/dt | J = dQ/dt |
| Capacitance | C = dQ/dV | C = dQ/dT |
| Conductance | G = 1/R = I/ΔV | K = J/ΔT |
| Monotonic parameter | Energy: dE = VdQ | Entropy: dS = dQ/T |

**Table 1: TL and Electronic Analogs**

map into thermal quantities. Each system involves a potential, and a charge moved through this potential. Other analogies are electrical and thermal capacitance, and electrical and thermal conductance. There is also an empirical "Ohm's Law" relationship saying that in many circumstances, the rate of charge (heat) flow is roughly proportional to the voltage (temperature) difference.

The subtle difference between the two systems shows up in the last line of the table, the definition of "dissipation." In an electrical system, the total capacitive energy is non-increasing, whereas in a thermodynamic system the total entropy is non-decreasing.

The plan for mapping an entire computer over from the electrical domain to the thermal domain is therefore clear: find analogs of the basic electronic components, using well-understood idealizations like pistons and heat baths, and build switching circuits just like those found in a modern computer chip.

A rudimentary form of TL can be recognized as far back as Szilard's seminal 1929 thought experiment that tied information theory to thermodynamics[3]. However, since he was concerned with information storage rather than computing, his devices were limited to memory cells.

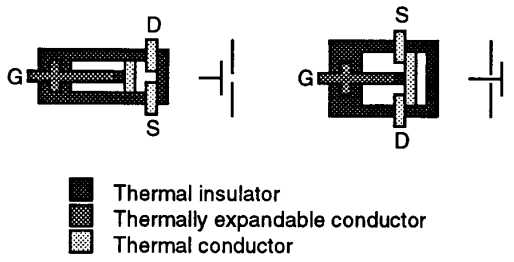


**Figure 1: Normally-off (left) and normally-on (right) thermal switches and their symbols.**

It is worth noting that the analogy between electronic and thermal systems can be extended to other types of systems as well. For instance, chemical systems, described in terms of particle number and chemical potential, could also be used to build reversible and non-reversible computers, provided a suitable switch is found. In fact, one might interpret the genes and proteins in a living cell as the switches and nodes of a chemical computational network.

## 2: Thermal logic components

The minimal component set for building a self-contained sequential computer consists of wires, insulators, a power source, switches, and an oscillator or delay element. In TL, these are as follows.

**Wires** are good thermal conductors, e.g. metal strips, characterized by a thermal capacitance $C$ and a thermal conductance $K$. A temperature difference across the conductor causes heat to flow: $J = K\Delta T$. The rate of entropy generation is analogous to the power dissipation in a resistor, apart from some extra temperature factors.

$$\frac{dS}{dt} = \frac{dQ}{dt}\left(\frac{1}{T_{out}} - \frac{1}{T_{in}}\right) = J\Delta T\left(\frac{1}{T_{out}T_{in}}\right)$$

**Insulators** are zero-conductance materials, usually left implicit in circuit diagrams.

**A DC power supply** is a component providing unlimited $Q$ at fixed potential $T_{DD}$, i.e., a heat bath. Similarly, **ground** is a heat bath at a lower temperature $T_{CC}$.

**Switches** are three-terminal devices. The simplest thermal switches are analogous to enhancement-mode Field Effect Transistors (FETs). By changing the temperature on a "gate" terminal, one can create or remove a conducting channel between the "source" and "drain" terminals. Figure 1 shows how to build such switches. The source and drain are two good conductors with a gap between them, and the gate is a conductor with a high thermal expansion coefficient. At the end of the gate is a layer of insulator, and a layer of conductor, the "channel."

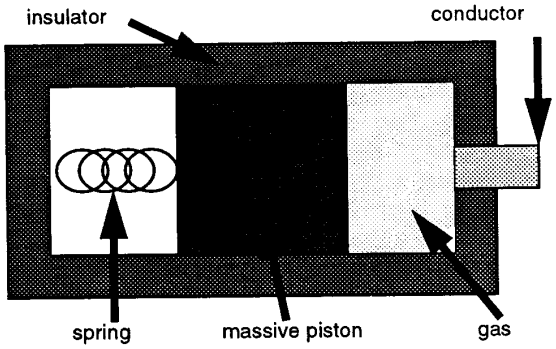


**Figure 2: Thermal oscillator**

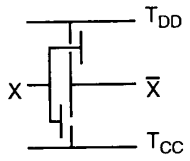


**Figure 3: Thermal inverter**

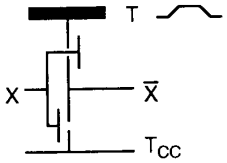


**Figure 4: Adiabatic implementation of inverter.**

Consider the "normally-off" switch on the left in Figure 1. When the gate G is at low temperature, there is a gap between the channel, source and drain. When the gate temperature is raised, the gate expands, and at some threshold temperature $T_{CC}+T_{th} < T_{DD}$ the channel is pushed against the source and drain, thus allowing heat to flow between S and D if they are at different temperatures. The switch is analogous to a MOSFET in that there is no heat transfer between G and S or G and D. A "normally-on" switch can be constructed as on the right of Figure 1. Also, the geometries of the devices can be chosen so that they switch at temperatures $T_{CC}+T_{th}$ and $T_{DD}-T_{th}$ respectively. Readers suspicious of expanding conductors can replace them with gas pistons and springs.

An **oscillator** or clock device, Figure 2, generates an oscillating temperature on the conductor by compressing and expanding a gas using a spring-mounted piston. The massive piston stores kinetic energy during part of the cycle in analogy to an inductor storing magnetic energy in an LC oscillator and a piezoelectric material storing elastic energy in a crystal oscillator. If the oscillations are sufficiently slow, the oscillator dissipates arbitrarily little energy.

## 3: Thermal logic circuits

Given the above components, we can immediately use the techniques of switching theory to build logic circuits, and arrive at a logic family analogous to CMOS (the class of circuits based on enhancement-mode FETs). Figure 3 shows the simplest combinational logic circuit, an inverter similar to a CMOS inverter. It consists of an input $X$, an output $\overline{X}$, and two switches, connected between two heat baths: a hot bath at temperature $T_{DD}$ and a cold bath at $T_{CC}$. A hot conductor represents logic value 1, and a cold one represents logic value 0. When input $X$ is high, the lower switch is closed, so there is a direct thermal path from the output to the cold bath. The output is therefore also at $T_{CC}$. On the other hand, when the input is low, the upper switch is closed, and the output is connected to the hot bath at $T_{DD}$. Note this inverter is clearly dissipative: when it changes state from $X$ low to $X$ high, the output, initially at $T_{DD}$, is connected directly to the cold bath at $T_{CC}$. Bringing together two objects at different temperatures leads to irreversible heat flow and entropy creation.

However, by replacing the hot rail $T_{DD}$ with a slowly time varying temperature source $T$, the circuit can be operated adiabatically (Figure 4). Initially, $T=T_{CC}$ and the input $X$ is free to change without causing heat to flow. Once $X$ is stable, $T$ slowly ramps up to $T_{DD}$, and the output either stays at $T_{CC}$ or follows $T$, depending on which switch is closed. $T$ now holds steady at $T_{DD}$ while any circuits to the left of this stage use the output value. $T$ can then be ramped down again to $T_{CC}$, after which the inputs are free to change. The whole process must occur "slowly," so that non-negligible temperature differences are not created. Our original introduction of the term "adiabatic" into computing was in fact motivated by this heat analogy.

In this paper, we will mainly deal with the "zeroth-order" approximation, that the energy dissipated in an adiabatic change is negligible. This approximation is only exact in the case of infinitely slow processes, and in practice the dissipation experienced is proportional to the relaxation time of the underlying thermodynamic system divided by the time scale of the adiabatic process. For most computations, it is likely that one can choose this time scale large enough that the dissipation is arbitrarily small. However, for computations with asymptotically infinite time or space complexity, one would need to be more careful about how the various limits are taken.

The rate of entropy generation in a thermal conductor referred to in section 2 is an example of a first order calculation of the adiabatic dissipation.

More complicated combinational logic is constructed similarly. Given any Boolean function $f(A_1,A_2,...,A_N)$ of $N$ Boolean variables, one can always construct a switch with

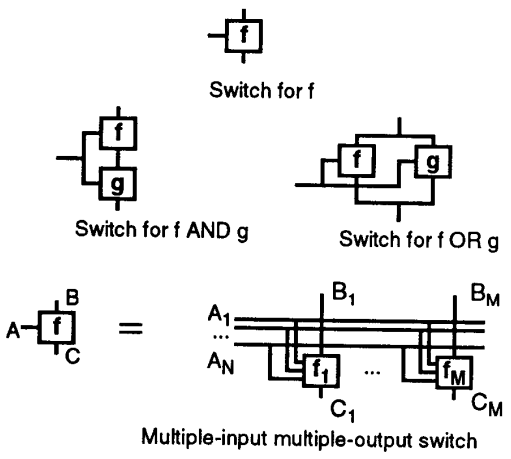


**Figure 5: Switching-logic notation and basics**

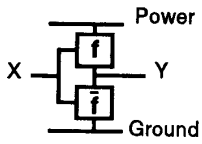


**Figure 6: Static logic gate for function f.**

a "source" terminal, a "drain" terminal, $N$ "gate" inputs labeled $A_1,A_2, \ldots,A_N$, and the property that the source and drain are connected when $f(A_1,A_2, \ldots,A_N) = 1$ and disconnected when $f(A_1,A_2, \ldots,A_N) = 0$. We represent the switch as in Figure 5. The generality of the construction follows from the well known fact that connecting two switches $f$ and $g$ in series or parallel produces switches representing ($f$ AND $g$) and ($f$ OR $g$) respectively, and that $\overline{A}_i$ (the complement of $A_i$) is effected by replacing normally-on switches with normally-off switches.

Some simple extensions of notation: First, any set of $M$ Boolean functions $(f_1,f_2,\ldots f_M)$ can be represented succinctly as a block with $M$ source-drain pairs. Second, any set of signals $A_1 \ldots A_N$ can be represented unambiguously as a single line with a single label $A$. Finally, $\bar{f}$ represents the Boolean complement of $f$; $I$ represents the identity function with $N$ inputs and $N$ source-drain pairs (for any $N$); $\bar{1}$ represents its negation; and an empty switch represents an "enable switch" with one input signal and $N$ source-drain pairs, such that the pairs are (dis)connected when the input is (0)1.

The generalized switches can now be used to construct general logic gates. Figure 6 shows how to configure an $f$ switch and an $\bar{f}$ switch to form a logic gate that computes $f$. When $f_i$ is true, the corresponding output $Y_i$ is connected directly to the power supply, which acts as a perpetual supply of 1's. Similarly, if $f_i$ is false, $Y_i$ is connected to ground, the perpetual supply of 0's.

This produces a complementary static logic family, which has the advantage that any valid signal is always directly connected to either power or ground. There are simpler "dynamic" circuits such as precharged logic, which replace $f$ or $\bar{f}$ with an enable switch, and rely on energy stored on internal wires to correctly represent signals. However, from our point of view, the greatest advantage of static logic is that it can equally well be operated adiabatically. Just as in the thermal inverter, if the inputs are valid, the power supply potential can be varied slowly between $T_{DD}$ and $T_{CC}$, moving heat adiabatically into and out of the output lines.

The heart of the adiabatic switching idea is thus the following: By slowly bringing together the potentials of the infinite supply of 1's and the infinite supply of 0's, we can remove the energy from signal lines without generating entropy. The complication is that this must be done locally, since the inputs signals must still have distinct 0 and 1 states while the output values are coalescing. Note that it is not critical whether we move the 1 to the 0 value, or the 0 to the 1 value, or both to some third value.

The local potential sources in an adiabatic circuit will be called "clocks," and their careful timing and interplay will be called "clocking". Design of a consistent adiabatic clocking scheme is one of the challenges addressed further on. Also, when a logic block has its 0 and 1 potentials coalesced, we will say the block is "de-energized," and when the 0 and 1 potentials have their normal values, we will say it is "energized."

## 4: Cascaded logic

In theory it is possible to perform any finite computation with one very complicated combinational logic gate, but in practice a computation is spread out over multiple simpler logic "stages" which compute consecutively. If the inputs are $A$, and the individual stages compute $f,g,\ldots,h$, then the whole circuit computes $h(\ldots g(f(A))\ldots)$.

In the case of normal dissipative logic, this is easy to implement by simply connecting the outputs of one stage to the inputs of the next. Then, whenever the inputs change, the computation ripples through consecutive stages to calculate the new answer.

The adiabatic version of this circuit[1,8,9] is slightly more subtle, since each stage must be de-energized before its inputs change, to prevent its outputs connecting a hot drain to a cold source, or vice versa. This can be achieved by providing a local source of 0's and 1's for each stage. Initially, all stages are de-energized, and the inputs to the computer are free to change. Once they stabilize, the stages

are successively energized, until the final stage is energized and outputs the answer. The stages are then de-energized in reverse order, so that no energized stage ever has its inputs change.

This "cascaded logic" is impractical as a general purpose computing scheme for several reasons. First, it requires a large and possibly indeterminate number of clocks. Second, these clocks all have different shapes, since different stages have their potentials coalesced for different lengths of time. And third, an $N$ stage cascade requires time proportional to $N$ to produce each result; more precisely, the latency (time to deliver) is proportional to $N$ and the throughput (number of results per unit time) is proportional to $1/N$.

The third issue applies equally to adiabatic and non-adiabatic logic, and has been addressed in the conventional case by using logic pipelines instead of logic cascades. In an $N$-stage pipeline, each stage computes a result, forwards it to the input of next stage, and begins computing the next result. In this way, the throughput is made independent of the number of stages.

In the non-adiabatic case, this is usually implemented by placing latches between the pipeline stages, to store the intermediate results while the new ones are computed. One usually introduces one or more clock signals to signal the latches to load the new result. However, latches are inherently dissipative, since they generate entropy when they are reloaded and their previous contents are erased[7]. The adiabatic case must therefore necessarily be different. On the plus side, in the adiabatic case the potential sources themselves can be periodic, so a pipeline clocking scheme can use the power supplies for timing.

The two key elements of an adiabatic pipeline design are the design of the individual stages, and the clocking scheme. We address these in turn.

## 5: Adiabatic pipeline stages

The reason the inputs were held valid during the entire operation of the adiabatic stage described above was to maintain the paths between the outputs and the potential sources which drive them. This allowed the circuit to return the energy from each output to the source from which it came, when the computation was complete.

The new idea in forming pipelines is that one could equally well use a different path to return the energy to the sources, as long as it is equivalent to the path originally used. For instance, consider any particular stage $S_n$ in a sequence: during the energizing of $S_n$, the paths between outputs and sources are determined by information provided by $S_{n-1}$, i.e., the inputs. By analogy, during the de-energize phase, we could try using information provided by $S_{n+1}$, and the simplest method is to have $S_{n+1}$ simply feed a regenerated copy of $S_n$'s outputs back to $S_n$. This allows $S_{n-1}$

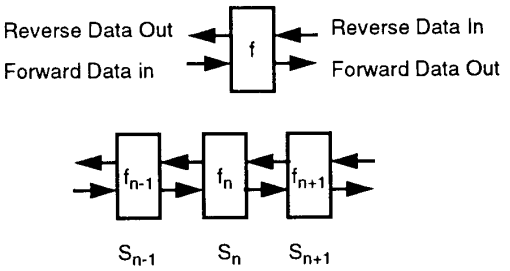


**Figure 7: Adiabatic pipeline stage symbol and notation**

to de-energize and begin working on a new computation, without impairing our ability to de-energize $S_n$.

For this scheme to work, certain restrictions must be placed on the function $f_n$ computed by $S_n$. To see this, note that in order for $S_{n-1}$ to begin discharging adiabatically, it must in turn be supplied with a copy of *its* outputs by $S_n$. In other words, in addition to stage $S_n$ being able to generate its outputs $O_n=f_n(I_n)$ from its inputs $I_n$, stage $S_n$ must also be able to regenerate a copy of its inputs $I_n=f_n^{-1}(O_n)$ from a copy of its outputs. This means that each $f_n$ must be an invertible function, at least on the set of values actually assumed by the inputs.

Since stage $S_n$ now has both a forward and a reverse set of inputs, a further complication arises: we need a way to specify which input to use to compute the outputs. This is necessary, because once $S_{n-1}$ begins computing a new result, the forward input is no longer related to the reverse input. Applying two distinct switch configurations between the outputs and the sources simultaneously would have a disastrous effect on the dissipation, since there will be at least one path directly from the high potential source to the low potential source. We thus need a way to disable each of the inputs as appropriate. The only times when both inputs can be enabled are when the forward and reverse inputs are correctly related, or when the high and low potentials are coalesced.

A circuit for $S_n$ which meets all these requirements is shown in Figure 8. It looks slightly different from other proposals in the literature[5,6], but in the next section we show that these are all special cases of a more general structure. Our stage consists of four separate complementary adiabatic drivers like the one in Figure 6. The lower left driver does the real computation by applying $f_n$ to the input from $S_{n-1}$ and outputting it to $S_{n+1}$. The upper left driver applies the identity function to the inputs, and outputs it back to $S_{n-1}$. The lower and upper right drivers apply the identity function and $f_n^{-1}$ respectively to the reverse input, and drive the same outputs as the left-side drivers. The left and right enable signals $E$ and $F$ allow either the left or right side drivers to drive the outputs.

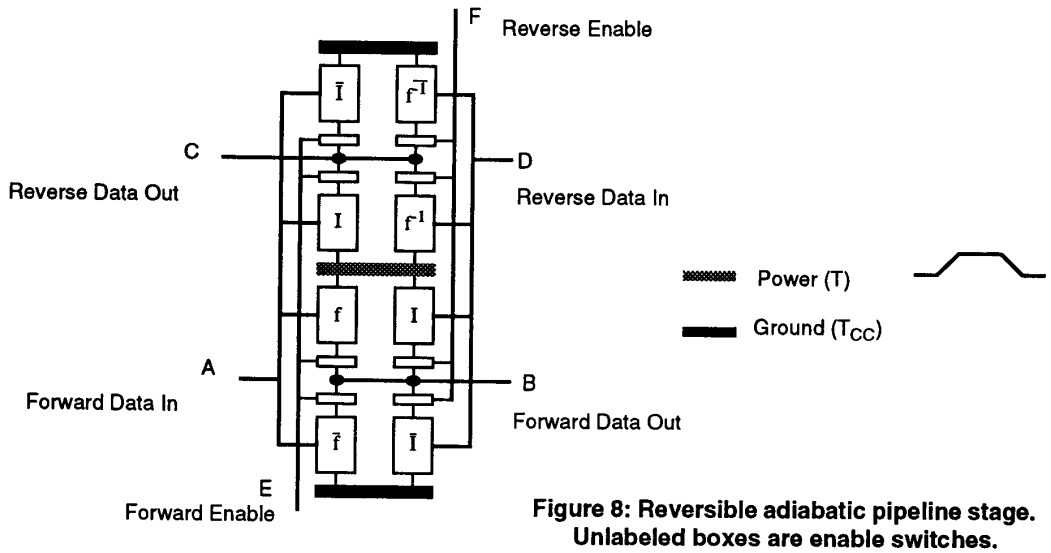


**Figure 8: Reversible adiabatic pipeline stage. Unlabeled boxes are enable switches.**

The stage operates as follows. Initially, the potential sources are coalesced, the inputs are zero, the outputs are zero, and all drivers are disabled. Then, by slowly (adiabatically) changing the inputs,

1. The forward input becomes valid.
2. The left drivers are enabled.
3. The stage is energized, which makes the forward and reverse outputs valid.
4. The reverse input becomes valid as $S_{n+1}$ regenerates the output from $S_n$.
5. The right drivers are also enabled, which is OK since the left and right inputs are correctly related.
6. The left drivers are disabled.
7. The forward input becomes invalid as $S_{n-1}$ de-energizes.
8. The stage is de-energized, making the outputs invalid.
9. The right drivers are disabled.
10. The reverse input becomes invalid.

Finally, notice that steps 1 and 2 can be interchanged, as can steps 9 and 10, since the stage is de-energized and the enable and input signals can be changed arbitrarily without causing dissipation. It is unusual in conventional circuits to enable an input before it is valid, but this property simplifies adiabatic clocking schemes, since it means that the enable pulses can have the same duration as input signals.

With this simplification, we can describe the behavior of the inputs to the stage quite cleanly. The stage has five sets of inputs: forward enable, forward data in, power, reverse data in, and reverse enable. The two outputs, forward data out and reverse data out have the same timing as the power input. Draw the five inputs as five square cells (which we call a state diagram), as in Figure 9, arranged from left to right according to the order in which they energize in the revised description above. Each input can be in one of four states: de-energized, energizing, energized (valid), or de-energizing, and we represent these by the shading patterns in Figure 9, as though energy flows into a cell through the left wall, and out through the right wall. Then the sequence of states above appear as in the figure, and is nicely summarized by saying that a pulse of energy five cells wide entered from the left of the state diagram, passed through, and exited to the right.

## 6: Adiabatic pipelines

An adiabatic pipeline is a chain of adiabatic pipeline stages, as shown in Figure 7. Initially, assume each stage has its own adiabatic potential source, to energize and de-energize the outputs appropriately.

To analyze this situation, temporarily represent each stage by a single cell, showing the state of its power input. Then the state of the pipeline is a pattern of empty and fully- or partially-shaded cells, showing which stages are energized.

The relationship between this "pipeline diagram" (Figure 10) and the five-cell state diagram is as follows: consider any cell in the pipeline diagram. The energization state of the power input, and hence of the stage outputs, is given by the shading of the cell. The states of the left and right data inputs are by construction equal to the states of the data outputs of the left and right neighboring stages. Thus the middle three cells in the state diagram of a stage corre-

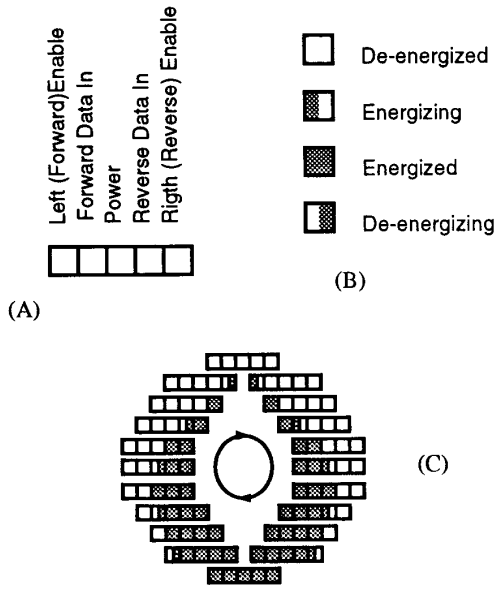


**Figure 9: Five-input state diagram for an adiabatic pipeline stage (A), possible input states (B), and adiabatic state sequence (C).**

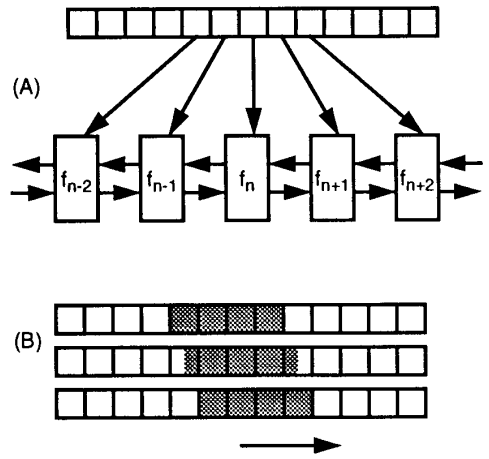


**Figure 10: Correspondence between pipeline stages and cells in pipeline diagram (A), and three snapshots showing motion of an energy pulse through an operating pipeline (B).**

spond to the three cells centered about the stage in the pipeline diagram. We now complete the correspondence by defining the left and right enable inputs to correspond to the two next-nearest neighboring cells. In fact, this tells us how to generate the enable signals: they are just the power inputs from the next-nearest stages. Thus the state diagram of a stage corresponds to the five cells centered about that stage in the pipeline diagram.

Next, consider a pipeline in the process of computing a result. At any moment, there is a contiguous subset of stages energized and involved in the computation, so the pipeline diagram contains at least one band of shaded cells. To correctly operate the pipeline, we simply need to sequence the power inputs so that the shaded band appears to travel to the right. We do this by simultaneously de-energizing the leftmost cell in the band and energizing the first empty cell to the right. If the band is five cells wide, then each pipeline stage goes through exactly the right sequence of states to compute and pass the result on down the pipeline. In this way, the computation travels down the pipe on a pulse of energy.

It is interesting to look at what happens to pulses of length other than five as they travel down the pipeline. (We are still assuming complete control over the power inputs, so we can sequence them as desired.) We find the following:

**Length 5 or more:** All pulses travel down the pipe successfully, carrying the computation with them.

**Length 4:** Here, the left enable turns off at the same time as the right enable turns on, but the pulse propagates as before. The pulse described in Figure 9 is thus not minimal, because some of the states can be overlapped.

**Length 3:** The left enable turns off just as the next stage is starting to use the output. The pulses are therefore dynamic, in the sense that the inputs to a stage must remain valid even if they are not driven. This is possible if the gate terminals of the switch devices can store some energy, so to zeroth order the pulse propagates. However, in the presence of noise, the computation can be corrupted. Random flipping of switches can cause some outputs to charge incompletely. Intuitively, it appears that the computation gradually picks up errors as it proceeds down the pipeline, but a proof of this would need more quantitative analysis than we have done here.

**Length 2:** Pulses of width two cause the drivers to disable during the middle of the charge or discharge cycle. Thus the output potential is not fully restored to its correct value, causing the signal to attenuate as it travels down the pipeline, even without ambient noise. The incomplete discharge also leads to non-adiabatic energy losses, so some of the energy in the pulse is dumped into the environment.

**Length 1:** Pulses of width one do not propagate the computation at all, since the drivers disable just as the stage

starts energizing, and no energy is transferred to the outputs.

Similarly, if we analyze what happens when we send multiple pulses down the pipeline, we find that pulses corrupt each other if there are fewer than four empty cells between them. Thus the maximum throughput of the pipeline occurs when we use 4-cell-wide pulses separated by 4 empty cells. Any higher density pulse train introduces errors into the computation, but any longer pulses propagate the computation essentially error free. Since the pipeline is an information channel, such behavior is necessary to avoid violating Shannon's Theorem that we necessarily introduce errors if we exceed the channel capacity.

Clearly, the adiabatic computational pipeline has an interesting mathematical structure suggesting many extensions and generalizations. For instance, by symmetry, pulses could propagate leftward to perform the reverse calculation.

A major property is the following "local gauge invariance." In the circuit of Figure 8, we have used a driver containing the identity function, to return a copy of the inputs to the preceding stage. However, we could equally well return an arbitrary invertible function $\theta$ of the inputs, as long as we compensate in the previous stage by including an additional transformation by $\theta^{-1}$ at the reverse inputs. In fact, a different $\theta$ can be chosen for each stage, without changing the overall computation of the pipeline.

The advantage of our choice is that consecutive stages are independent, so it is easy to concatenate stages. Another interesting choice of the "gauge" $\theta_n$ for stage $n$ is $\theta_n=f_n$. This means that the information fed backwards by any stage is identical to the information fed forwards. The block diagram simplifies, since we can use a single driver to drive both the forward and reverse outputs. However, it complicates pipeline composition, since each stage involves its own function $f_n$ as well as the function $f_{n+1}^{-1}$ from the next stage. Note that the adiabatic pipelines suggested till now for CMOS[5,6] have been formulated in an analog of this "minimal gauge," rather than our "local gauge."

Some other comments:

1. Many of the theorems and intuitions developed for gauge theories translate directly over to the adiabatic pipeline model.

2. The case of a finite pipeline provides insight into the I/O issue. At each end of a finite pipeline, we need to decide what to do with the enable and data inputs of the outermost stage, and the enable signal of the next-to-outermost stage, since the stages which would normally provide these signals are absent. There is a variety of choices for these boundary conditions, leading to different behaviors. We describe one choice: On the left input, permanently enable the first two forward enable signals, and provide fixed valid data to the data input. On the right end, permanently disable the last two reverse enable signals. Under these conditions, when a pulse of energy carrying a computation reaches the end of the pipeline, the final two stages energize as normal, but have no way to de-energize, since the de-energizing paths are permanently disabled. Thus the result is trapped at the end of the pipeline, where it is available for reading. Note, however, that no non-adiabatic energy dissipation actually occurs until the next pulse reaches the end and overwrites the trapped result, which is just what Landauer's work would predict[7].

3. By building a loop of pipeline stages, one forms a finite state machine, that can be used to generate control signals for datapaths. A pulse of energy travels round and round the loop, sequentially passing through the states of the machine. Clearly, the smallest static state machine must have 8 stages to be able to hold one pulse. Note also that the machine is gauge invariant, since all gauge variations cancel out round a loop.

## 7: Acknowledgments

We thank Jay Block, Eric Chou, Nestoras Tzartzanis and Josh Storrs-Hall for useful discussions. This work was supported by the Advanced Research Projects Agency, contract DABT63-92-C0052.

## Appendix: Energy supplies

Until now, we have assumed that arbitrary power potentials can be generated to order and applied to the power inputs of the stages in a pipeline. The general mechanism for generating an oscillating potential in thermal logic was sketched in Figure 2. A mechanical system (say) is coupled to a thermal system, and thermal energy is converted adiabatically to and from mechanical energy. However, this design in itself is insufficient, because we need a waveform with the specific property that it is "flat on the bottom." Thus, in Figure 4, it is important that the upper potential $T$ be brought down to and kept at exactly temperature $T_{CC}$ while the inputs change, to prevent non-adiabatic heat flow from $T$ to $T_{CC}$. This requires some non-analyticity in the temperature T as a function of time.

An easy way to obtain non-analytic motion in a mechanical system is through a device such as a cam with non-analytic shape. Thus, a large flywheel, several cams, and several frictionless pistons suffice to generate our waveform. However, it is rather ad hoc to burden the mechanical part of our system with generating this behavior, when the requirement itself is thermodynamic in origin. Let us therefore restrict the mechanical system to be essentially a harmonic oscillator, and attempt to generate the appropriate waveform using a thermodynamic device.

Nonanalytic behavior in thermodynamics is associated

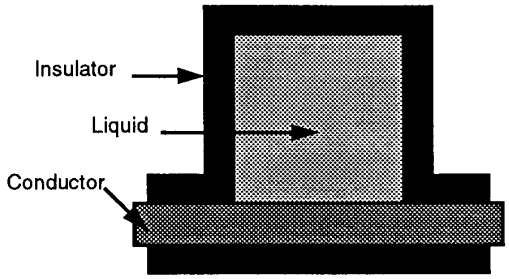


**Figure 11: Thermal clamp, using a liquid-solid phase transition to enforce a fixed minimum temperature. This simple design ignores expansion issues.**

with phase transitions: we can build a temperature clamp by choosing $T_{CC}$ to equal (say) the freezing point of a liquid (Figure 11). At temperatures above the melting point, extraction of heat lowers the temperature, but when the freezing temperature is reached, the temperature stays constant, and the energy is supplied by the latent heat of melting.

By putting a clamp in series with each thermal oscillator, one can now design power supplies that sequence adiabatic pipeline stages appropriately. For instance, a system of 8 clocks is sufficient to provide a steady train of energy pulses to an arbitrarily long pipeline, if stage $S_i$ is powered by clock $\phi_{i \bmod 8}$.

Another approach is to provide each stage with a separate clamped power supply, and to couple adjacent mechanical systems together. Then energy pulses traveling through the mechanical system are accompanied by computations traveling through the computational system.

Clearly, the energy supply system also has many interesting associated issues that are well worth studying.

## 8: References

[1] Koller, J.G., and Athas, W.C., *Adiabatic switching, low energy computing, and the physics of storing and erasing information*, in Proceedings of the Workshop on Physics and Computation, PhysComp '92, Dallas, Texas, Oct 2-4, 1992. IEEE Press, 1993.

[2] Athas, W.C., Koller, J.G., and Svensson, L."J.," *An energy-efficient CMOS line driver using adiabatic switching,* Accepted for publication at the IEEE Great lakes Symposium on VLSI, 1994.

[3] Szilard, L., *On the decrease of entropy in a thermodynamic system by the intervention of intelligent beings,* Behavioral Science 9, 301-10 (1964), translation by A. Rapoport and M. Knoller of Zeitschrift für Physik, 1929, 53, 840-856.

[4] Bennett, C.H., *Logical reversibility of computation*, IBM J. Res. Dev. **17**, 525-32 (1973).

[5] Younis, S.G. and Knight, T.F., *Practical implementation of charge recovery asymptotically zero power CMOS,* in Proceedings of the 1993 Symposium on Integrated Systems, MIT Press, pp. 234-250 (1993).

[6] Athas, W.C. and Svensson, L."J.", *Reversible logic issues in Adiabatic CMOS,* 1994 Workshop on Physics and Computing, PhysComp '94, November 1994.

[7] Landauer, R., Irreversibility and heat generation in the computing process, IBM. J. Res., **5**, pp183-91 (1961).

[8] Hall, J.S., *An electroid switching model for reversible computer architectures,* in Proc. ICCI'92, 4th International Conf. on Computing and Information, 1992.

[9] Merkle, R.C., *Reversible electronic logic using switches,* Nanotechnology, **4** pp. 21-40 (1993).