

# *An Electroid Switching Model for Reversible Computer Architectures*

J. Storrs Hall  
Laboratory for Computer Science Research  
Rutgers University

## *Keywords*

reversible computation, entropy, heat dissipation, switching models, finite automata, computer architecture, retractile cascade

## *Abstract*

The study of logical reversibility in computation has led to a proliferation of unconventional architectural proposals, from Brownian-motion clockwork Turing machines to billiard-ball computers. An architectural model for reversible computers is proposed herein, that is significantly closer to the switch-level models used in designing actual digital circuitry. This “electroid” (electronics-like) model facilitates the direct application of the substantial body of existing digital design techniques to reversible and dissipation-limited architectures.

## *Introduction*

Efficient computers, like efficient heat engines, must be as nearly reversible as possible. Landauer showed in a landmark paper [1961] that the characteristic irreversible operation in computation is the erasure of a bit of information; the other operations can be carried out in principle reversibly and without the dissipation of energy. In the years since 1961, various models (see below) have been developed which depict processes, some more realistic than others, by which reversible computation could be done.

Even though thirty years and a fairly substantial literature separate us from the beginnings of this field of investigation, reversible computation has received scant attention from the mainstream of computer architecture and logic design. This is primarily because the energy limitation implied by thermodynamics is still some orders of magnitude lower than the power dissipation per switching operation in current electronic technology; but it does not help that the models for reversible computing proposed to date have either relied on unconventional architectures or been mechanical models requiring frictionless parts, perfectly elastic collisions, etc, or both.

This paper introduces a new abstract model for reversible computers, and a new somewhat more concrete model which, it is hoped, will be much more congenial to the average logic designer than its predecessors.

## *Background*

A physical system consists of some number  $N$  of particles. Each particle can, at any given instant, be described by a vector of  $M$  numbers, including  $X$ ,  $Y$ , and  $Z$  components of its position and velocity, its angular momentum, etc. Thus the system as a whole can be described by a vector of  $MN$  numbers. A *phase space* for the system is the  $MN$ -dimensional space of all allowable such vectors. If the physical system is closed, i.e. occupies at most some finite volume of space and has some fixed energy, its phase space also has a fixed volume.

A *state*<sup>1</sup> is some subset of all the possible configurations a system may take on, i.e. it is some subvolume of the phase space. A system which is known to be in some state but about which no more is known, i.e. it may be at any point within that state with equal probability, is said to have an *entropy* equal to the log of the volume of the state, on an appropriate scale.

All known physical processes with applicability to the construction of computers can be described by Hamiltonian transformations of the phase space, i.e. transformations which preserve volume. (This includes quantum mechanics but does not include general relativity; gravitational singularities appear to be an exception to this rule.) Obviously such a transformation cannot map a state  $A$  into a state  $B$  which is smaller than  $A$ . On the other hand,  $A$  can be mapped into a state  $C$  whose volume (or a collection of states the sum of whose volumes) is greater than  $A$ . Thus the entropy of a closed system can

---

<sup>1</sup> An adequate exposition of the ontological basis of this section would more than consume the entire paper. The reader is therefore advised to consider this review as a *definition* of the terms, particularly state and entropy, as used herein.

increase but cannot decrease.

We wish to consider a computer as a closed physical system. There is clearly a mapping between the logical states of the computer (in the sense that it is a finite-state automaton) and the phase-space states of the physical mechanism. Furthermore, if the machinery of the computer operates correctly, the physical transformations between states will be a complete and accurate reflection of the transitions between logical states. Now if the states are of unequal volume, an ordering is imposed on them; the computer must always move from a smaller state to a larger one. In other words, it cannot loop. This severely constrains the applicability of such a computer.

An interesting computer must then have states of equal entropy. The transitions between such states are allowable in either direction; they are reversible. Hereinafter, then, we shall adopt the following simplified model: The state of a computer consists of a vector of  $N$  bits. The state space of the computer is then a set of  $2^N$  nodes in a directed graph. The edges of the graph represent the state transition function imposed by the physical mechanism of the computer. The computer is reversible if each connected component of this graph consists exactly of a directed Hamiltonian cycle.

There are four one-bit, two-node computers (Fig. 1). (The other two-node directed graphs represent non-deterministic automata.) Of the four, **Lazy** and **Flipper** are reversible, and **Setter** and **Eraser** are not. Consider any physical implementation of **Eraser**. The volume of state 0 must be the sum of the volume of state 1 and that of state 0 itself!

Of course, **Eraser** can be implemented in a physical system that is not closed. This typically involves some process which expands the entire phase space (e.g. adding energy) in such a way that both states are mapped into what is state 0 after the process. Then another process is performed (e.g. rejecting heat into the air) which compresses the phase space so that state 0 regains its original volume. If this process is in fact the rejection of heat, at least  $\ln 2 kT$  joules of heat must be rejected to halve the volume of the phase space, where  $k$  is Boltzmann's constant and  $T$  is the absolute temperature in Kelvins.

$\ln 2 kT$  joules is comparable to the thermal energy of a single atom at temperature  $T$ . For macroscopic logic devices, the energy involved is clearly trivial. As devices become ever more microscopic, approaching atomic size in the limit, the density of energy dissipation due to this effect increases dramatically, toward a limit of attempting to double the absolute temperature of the device with each operation.

### *History*

Since the earliest electronic computers in the 1940's, energy dissipation per device has been declining exponentially with time. Device size has undergone a similar decline, with the result that overall dissipation per unit volume has been relatively constant (see the horizontal bar in Fig. 2). Historically, the portion represented by the thermodynamic limit for irreversible operations was completely insignificant; it is still in the parts per million range. However, with increasing densities and speeds, this figure is increasing exponentially, as seen in Fig. 2. It is clear from this why attention to reversible computation will probably be increasing in coming years.

The earliest models for reversible computers drew on the example of DNA in the cell,

which engages in computation-like activities in a reversible way. Such a model is stochastic, in that there typically a non-zero probability for the previous step to be undone at any point, and a non-unit probability for the next step to be done. This was the basis for Bennett's Brownian motion clockwork Turing machine, which was first used to prove time and efficacy bounds on reversible computing. DNA computation continues to attract interest [Conrad 1985] as a pathway to molecular-scale devices.

In a more conventional framework, Fredkin and Toffoli [1981] showed that a deterministic computer could be reversible, with their ballistic (billiard ball) computational model. More recently, Likharev's parametric quantron, Drexler's [1988] molecular rod logic, and Merkle's [1991] clockable elastic wells, all deterministic models, have made it clear that a stochastic regime is unnecessary for reversible computation. Therefore we shall consider only deterministic models in what follows.

### *Reversible Computers*

As mentioned, a computer is reversible if its graph is such that each connected component consists of a Hamiltonian cycle. Equivalently, such a graph represents a permutation, and thus there are  $(2^N)!$  distinct reversible computers with  $N$ -bit states.

Among the 40320 reversible 3-bit computers is one that performs the function of a Fredkin gate (Fig. 3). The Fredkin gate, which exchanges its top two inputs if the bottom input is 1, can simulate AND and NOT, and thus any logic function can be built of Fredkin gates. Used in the billiard-ball computer, it is the prototypical formulation that considers each bit of information to be objects, not to be destroyed.

We can exhibit other reversible computers that manipulate bits in this fashion (Fig. 4). **Swapper** exchanges the bits of its state, for example. However, other operations are clearly reversible also: **Copier** alternately copies its first bit into its second, and erases the copy (but is stymied if the second bit is neither empty or a copy of the first). **Climber** increments its state interpreted as a 3-bit number (mod 8).

It is impossible in general to embed an arbitrary irreversible computer into a reversible one of fixed size. Consider **Eraser**. It could in principle be started in state 1 and run for  $N$  steps; to reverse it, a computer would have to run for  $N - 1$  steps with some bit, with which it was modelling the state of **Eraser**, 0; and then set it to 1. This would require at least an  $N$ -state machine, and  $N$  can be arbitrarily large. (Note, however, that provision for input/output or an infinite "tape" as for a Turing machine changes this result.)

### *The Electroid Model*

In a reversible heat engine, there is never any point at which heat flows directly from a hot reservoir to a cold one. Such a flow would be irreversible. More intuitively, one could augment the design of the engine by adding another engine which harnessed such a heat flow to do useful work. Similarly, in a reversible electronic computer, there must never be a point where a switch is closed between two wires at different voltages. The current flow across such a switch could theoretically be harnessed to do useful work, but is instead dissipated as heat.

Indeed, in a reversible electric machine, no current must ever be allowed to flow across

a resistance; such a flow is always dissipative and irreversible. To design a reversible logic that is electronic in character, then, we must assume wires and switches with no resistance at all. Remarkably, however, we need assume nothing else.

CMOS logic consists of a power bus, a ground bus, and an arrangement of signal wires and switches. A signal wire is placed in a “1” state by connecting it to power, and returned to “0” by connecting it to ground. Each such switching event (which changes the state of the wire) is dissipative. (Other logics than CMOS are more dissipative still, allowing current to flow between, as well as during, switching events.)

The reversible electroid model uses a conceptually similar framework of wires and switches, but replaces the power and ground with one or more clock busses which alternate from “0” to “1”.<sup>2</sup> The crucial point is that the clocks are to be driven in such a way that the voltage change is reversible. This means that behind the clock is some mechanism which is capable of storing the energy from a 1-0 transition and restoring it into a 0-1 transition. There are any number of conceptual formulations of such a device, from LC circuits to motor-generators with flywheels, which will serve identically well as far as the computational model is concerned.

Given such a clock, the design rules for the model are easily stated: Never close a switch between wires at different states, and never change the state of a switch while the state of the wires it connects are changing.

It is not immediately obvious that these rules prevent the erasure of a non-redundant bit, but they do. Obviously a wire can only change state by being connected to a clock that is changing state, or to some other wire that is connected to a clock, etc. Such a connection must have been made while the wire and clock were equal, e.g. both “1”. However, such a connection must not be made if the wire were “0” at the time; hence, the connection depends on the state of the wire. Furthermore, since the connecting switch cannot change while the wire is changing, the signal controlling the switch must outlive the state of the wire; but this signal reflects the bit that was to be “erased”, Q.E.D.

For compactness in design and simplicity of exposition, we will use both positive- and negative-sense switches (Fig. 5). They may be thought of as electrostatically operated relays. Higher-level design rules, e.g. fanout, will be largely ignored, since we are not proposing any actual physical model. However, it is appropriate to observe that the model is close enough to feasible mechanisms that it could serve to suggest techniques to reduce dissipation in a technology that could not eliminate it entirely.

### *Conservative Logic*

There are two distinct categories of design techniques within the broad class of deterministic reversible computers. The first is conservative logic; in this regime, information is thought of as a kind of “stuff” which can be transformed but not destroyed. The archetype for conservative logic is the billiard-ball machine, where each ball represents an indestructible bit. The use of conservative logic elements ensures that the values present at any

---

<sup>2</sup> These are the “hot clocks” of Seitz [1985]. Their use to implement reversible logic was independently discovered by this author and Merkle [1992].

stage contain enough information uniquely to determine the input (which can in fact be done simply by reversing the direction of information flow through the gates).

Conservative logic is exemplified in the electroid model by the “alpha-beta pass gate” (see Fig. 6). An incoming signal on X is “moved” to Y. In operation, the signal is present on X, controlling switch 1. Y is initially 0, and switch 2 is open. Clock alpha comes up first, separated from the signal by switch 2. Then clock beta comes up, setting Y to the same value as X. If this value is 1, switch 2 is closed, but since alpha is already 1, it connects equal values. (If the value is 0, both switches remain open throughout.) Next alpha goes down, draining X if it was 1. Now X is 0 and switch 1 is open in all cases. Thus beta can go down in isolation from the value now on Y.

The alpha-beta pass gate is obviously reversible, simply by reversing the order of clocks alpha and beta.

A series of alpha-beta pass gates can act as a shift register, with the proviso that only every third position is occupied by actual data. This is because no gate can be allowed to have active data on both sides at once, and briefly during switching both the input and output wires hold the data simultaneously. The third position separates the bits during this part of the switching cycle. However, the gate uses so few switches that three of them are comparable to any more complex gate capable of being used without buffer positions.

A conservative-logic Fredkin gate can easily be designed (Fig. 7). The buffer symbols represent alpha-beta pass gates. They are to be clocked in the sequence shown. Obviously the Fredkin gate can be operated in the reverse direction by clocking the pass gates in the reverse direction and in the reverse order. Thus a set of values could be introduced at the inputs of a Fredkin gate circuit, clocked across, and the outputs produced; or just as simply, the outputs introduced at the far end, and clocked backward to produce the inputs.

The Fredkin gate is computationally complete, in the sense that any piece of digital hardware can be built exclusively of Fredkin gates. However, the design is cumbersome and uses many more switches than its equivalent in, say, CMOS. In particular, all the extraneous values produced during the computation must be accounted for; if we use Fredkin gates to simulate a conventional design in NANDs or NORs we introduce two extraneous values for each useful one!

### *Retractable Cascades*

It is not necessary, however, to treat each bit as an object. It turns out that any combinational circuit can be implemented reversibly by the use of a very simple technique: simply remember the inputs.<sup>1</sup> A state-space diagram shows this clearly (Fig. 8)—the first bit(s) in the state vector are the input to the function, the final bit the output. Implementation of such functions in the electroid logic is obviously straightforward.

The only problem with such a technique is that the inputs must remain asserted until after the output clocks have been retracted. A circuit that performs a logic function several layers deep must have a sequence of rising clocks, and then the reverse sequence of falling ones. Such a circuit is called a *retractile cascade*.

---

<sup>1</sup> This technique appears first to have been used systematically by Drexler [1992].

A retractile cascade adder is seen in Fig. 9. First the two input numbers  $A[0-n]$  and  $B[0-n]$  are asserted. Then clock LL1 is asserted; this energizes the first stage of each full adder, consisting of an XOR and an AND. Then clock CC is asserted, energizing the carry chain; finally clock LL2 is asserted, producing the sum.

If the sum is needed more than momentarily, and the adder or its inputs need to be vacated, it will be necessary to save the sum using the clock labelled LATCH. The sum would then presumably be moved using conservative techniques or erased dissipatively. However, it is interesting and important to note that as long as the inputs A and B are available, the sum can be erased non-dissipatively. Fig. 9 shows the clock sequence which accomplishes this. This technique can be used with any retractile cascade to erase as well as to produce the value given the inputs.

Retractile cascades can be used for virtually all the medium-scale components of a standard computer architecture. Fig. 10 shows a PLA and a barrel shifter implemented as retractile cascades. Figure 11 shows a register bank implemented in a hybrid technique: the addressing is a retractile cascade, but the data to be read and written is treated conservatively.

The major drawback of the retractile cascade for larger-scale use is that it precludes pipelining. It is possible, however, to pipeline a sequence of cascades of modest depth. Each stage consists of the function and its inverse, and at each step, the inverse function is used in its “erasing mode” to remove the input so the next value can use the stage. The latency of the pipeline of retractile cascades is not significantly worse than that of an irreversibly implemented one, but the successive data items are separated by the depth of the individual retractile cascades making it up.

In Figure 12, a pipeline of adders is being used to multiply. Each line now represents a word or other group of wires. As before, the small buffer symbols represent alpha-beta pass gates or sequences of them. The numbers are propagated along the top two lines, while the circuit below accumulates partial products. At each stage, the appropriate bit of A is used to gate B or 0 into the adder. The clock symbols represent the entire set of clocks for the adder, as in Fig. 9, including the latch.

At time T1, the first sum has appeared at the output of the stage 1 adder and been latched. Now C1b begins the process of computing the input while C2a begins the process of computing the next stage. Simultaneously, the adder in stage 1 retracts. It finishes at time T2, freeing the input value to be erased by C1b. Also at T2, C2a has produced and latched the value for the second stage, and the process continues in this fashion.

### *Architecture-level Reversibility*

In a complete reversible computer architecture, memory could be implemented along the same lines as the register set above. The appropriate method to access memory would be “exchange with register” rather than load and/or store, which of course are irreversible in general. Merkle [1991] notes that many register to register operations such as  $A=A+B$  are reversible. Such instructions can be implemented by making the memory cycle that produces B entirely retractile rather than hybrid.

The instructions that present problems are testing and flow of control. The existence

of a `JUMP` instruction makes every other instruction irreversible! Reversible flow of control can be accomplished by constrained `GOTO` and `COMEFROM` pairs, and the use of explicit jump/don't jump flags. Removing these is now a problem the programmer can handle in a number of ways.

Unconstrained jumps are like conventional linked lists in that each can imply confluent state sequences (i.e., irreversibility). Many applications that require linked lists can be implemented reversibly by a two-way pointer technique: Each data item has a pointer that is the `XOR` of the addresses of the next and previous items. Then assuming that registers `A` and `B` hold the addresses of successive items in the list, they can be advanced by the reversible sequence

```
XOR A, PTR(B)
EXCH A, B
```

However, using a completely reversible instruction set is quite cumbersome at best and invalidates a wide range of programming techniques. It is possible to have a much more conventional instruction set which limits but does not entirely eliminate irreversible operations. For example, the classic PDP-10 instruction set can be implemented (except for floating point operations) so as to require the destruction of one word or less of information per cycle.

The energetic cost can be reduced even further by taking advantage of the fact that fullword values usually have more 0 bits than ones. If in the electroid model, we irreversibly erase a bit by connecting it to ground through a resistance, we dissipate only when the bit was 1. Thus we can obtain quite a low average, probably less than 10 bits per cycle. This is to be compared with the average conventional computer architecture, implemented without regard to reversibility, which destroys between 10,000 and 100,000 bits per cycle.

## References

- Bartle, Thomas C.: **Computer Architecture and Logic Design** McGraw-Hill, New York, 1991
- Bennett, Charles H.: *Time/Space Tradeoffs for Reversible Computation* SIAM J. Comput. V. 18 No. 4, 766-776, August 1989
- Bennett, C.H. and Rolf Landauer: *The Fundamental Physical Limits of Computation* Scientific American 253 pp 48-56, July 1985
- Bennett, Charles H.: *Logical Reversibility of Computation* IBM J. Res. Devel. 17, pp525-532, 1973
- Bennett, Charles H.: *Notes on the History of Reversible Computation* IBM J. Res. Devel. V. 32 No. 1, 1 Jan 1988
- Conrad, Michael: *On Design Principles for a Molecular Computer*, CACM v28 n5, May 1985, pp. 464-480
- Drexler, K. Eric: *Rod Logic and Thermal Noise in the Mechanical Nanocomputer*, Proceedings of the Third International Symposium on Molecular Electronic Devices; Elsevier North Holland, 1988
- Drexler, K. Eric: **Nanosystems: Molecular Machinery, Manufacturing, and Computation** John Wiley and Sons, New York, 1992
- Fredkin, E., Toffoli, Tommaso: *Conservative Logic* MIT/LCS/TM-197 Cambridge, MA May 1981 (see in Int. J. Theor Phys, fall 81)
- Garrett, A.J.M.: *Macroirreversibility and Microreversibility Reconciled: The Second Law in Maximum Entropy in Action*, Buck and Macaulay, eds.; Clarendon Press, Oxford, 1991; pp. 139-170
- Landauer, Rolf: *Irreversibility and Heat Generation in the Computing Process* IBM J. Res. Devel. 5 pp 183-191, 1961
- Landauer, Rolf: *Dissipation and Noise Immunity in Computation and Communication* Nature, V. 335 pp 779-784, 27 Oct 1988
- Leff, Harvey S. and Andrew F. Rex: **Maxwell's Demon: Entropy, Information, Computing** Princeton University Press, Princeton, NJ, 1990
- Merkle, Ralph C.: *Mechanical Reversible Logic* Xerox PARC (Privately distributed), Oct 1991
- Merkle, Ralph C.: *Reversible Electronic Logic Using Switches* Submitted to *Nanotechnology* 1992
- Ressler, Andrew L.: *The Design of a Conservative Logic Computer and a Graphical Editor Simulator* M.S. thesis, M.I.T., Jan. 1981
- Seitz, Charles L. *et al*: *Hot-Clock nMOS*, Proceedings of the 1985 Chapel Hill Conference on VLSI, computer Science Press, pp.1-17
- Toffoli, T. and N. Margolus: **Cellular Automata Machines**, (particularly chapters 14 and 18) MIT Press, 1987
- Weste, Neil and K. Eshraghian: **Principles of CMOS VLSI Design** Addison-Wesley, Reading, Mass., 1985