Mechanical Computing Systems Using Only Links and Rotary Joints

RALPH C. MERKLE ROBERT A. FREITAS JR. TAD HOGG
THOMAS E. MOORE MATTHEW S. MOSES JAMES RYLEY

January 12, 2018

Abstract

A new paradigm for mechanical computing is demonstrated that requires only two basic parts, links and rotary joints. These basic parts are combined into two main higher level structures, locks and balances, and suffice to create all necessary combinatorial and sequential logic required for a Turing-complete computational system. While working systems have yet to be implemented using this new paradigm, the mechanical simplicity of the systems described may lend themselves better to, e.g., microfabrication, than previous mechanical computing designs. Additionally, simulations indicate that if molecular-scale implementations could be realized, they would be far more energy-efficient than conventional electronic computers.

1 Introduction

Methods for mechanical computation are well-known. Simple examples include function generators and other devices which are not capable of general purpose (Turing-complete) computing (for review, see [22]), while the earliest example of a design for a mechanical general purpose computer is probably Babbage's Analytical Engine, described in 1837 [23]. One of the very first modern digital computers was a purely mechanical device: the Zuse Z1, completed in 1938 [28].

While previous designs for mechanical computing vary greatly, the few designs capable of general purpose computing require a substantial number of basic parts, such as various types of sliding plates, gears, linear motion shafts and bearings, springs (or other energy-storing means), detents, ratchets and pawls, and clutches.

The use of many parts brings with it a number of potential problems, such as increased friction, higher mass, and increased device complexity. Such issues can reduce performance and increase the difficulty of manufacturing. However, reducing the number, complexity, and mass of parts in a mechanical computer is not a simple task due to the need to provide both universal combinatorial logic (e.g., AND, NAND, NOR, etc.) and sequential logic (memory).

Sequential logic in particular, being the basis for memory, requires the ability to conditionally decouple logic elements from current inputs. This is because memory cannot be only a deterministic result of just current inputs, otherwise previous states cannot be saved. Storing information, which is easily accomplished in electronic systems using latches or flip-flops, is not as easily accomplished in a mechanical system which may have to actually connect and disconnect parts of the system from each other (e.g., using a clutch-like mechanism) at appropriate times.

This paper demonstrates that mechanical computers can be greatly simplified by using only two parts: links and rotary joints.

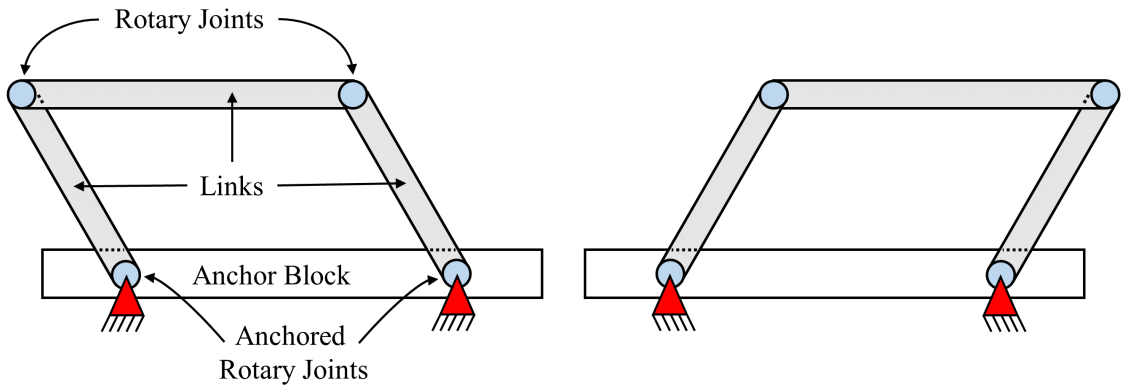


Figure 1: A 4-Bar Linkage in two configurations, left-leaning (left) and right-leaning (right)

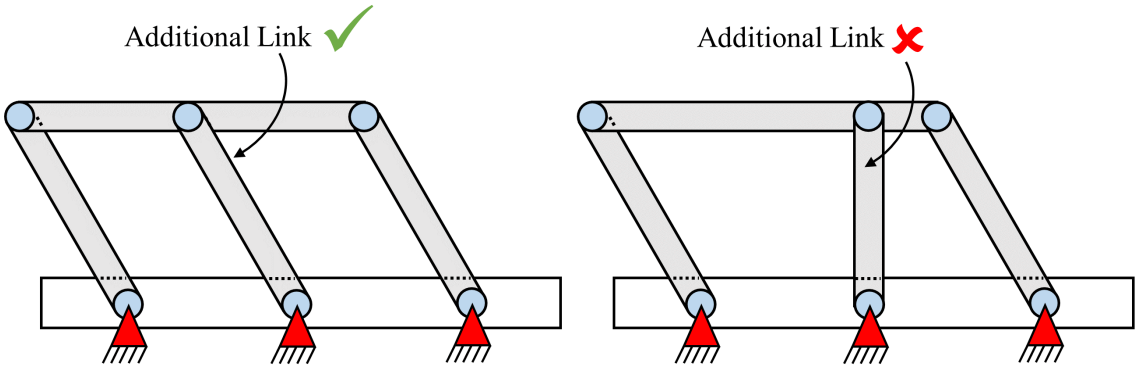


Figure 2: The mobile linkage (left) is free to move, while the non-mobile linkage (right) is static.

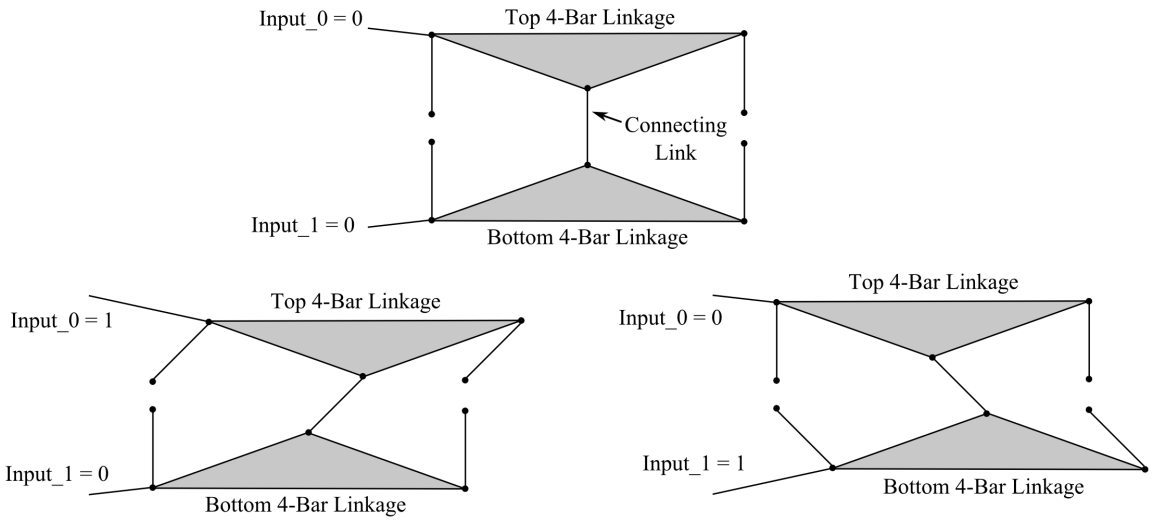


Figure 3: A lock in the (0, 0) position (top), a lock in the (1, 0) position (bottom left), and a lock in the (0,1) position (bottom right). The (1,1) position is prohibited by the linkage geometry.

2 Computing With Only Two Parts

Links are stiff, beam-like structures. Rotary joints are used to connect links in a manner that only allows rotational movement in a single plane. Perhaps counterintuitively, no type of clutch-like mechanism is required. All parts of the system can remain permanently connected and yet still provide all necessary combinatorial and sequential logic.

2.1 4-Bar Linkages

To demonstrate how this can be accomplished, we start with a simple, well-known mechanism, the 4-bar linkage (also referred to as a 3-bar linkage – the 4th bar is provided by the anchor block or base and is sometimes ignored for naming purposes). The 4-bar linkage relies only upon links and rotary joints. A 4-bar linkage can rotate around its anchored rotary joints (denoted by a circle with a triangle, while unanchored rotary joints are just circles), allowing, for example a "left-leaning" configuration to rotate into a "right-leaning" configuration (see Figure 1).

Note that, while a 4-bar linkage could assume many positions, we focus on the use of two distinct positions. This is because two positions can be used to signify 0 and 1, which is convenient when creating a system for binary computing. The actual angle traversed by the links when moving from, e.g., left-leaning to right-leaning is not critical. As diagrammed, it is approximately 45 degrees, but could be more or less as long as the design supports reliably differentiating between two positions; one representing 0, and the other representing 1.

It is important to note that in a parallelogram-type (meaning, the two side links must be the same length) 4-bar linkage, if an additional link is added to the center of the linkage, as long as the length and angle is the same as the side links, the linkage will still rotate around the anchored rotary joints. However, if the additional link is not the same length, or is not at the same angle, as the other links, the mechanism will not move (see Figure 2). This is because the additional link will be attempting to pivot through a different arc than the side links. The effect of this is that the side links and the center link are trying to move the top link in two different directions at once. This results in the mechanism binding up, or "locking." This locking behavior is important in the creation of "locks," one of two main higher-level mechanisms used by the system we describe (the other being the "balance").

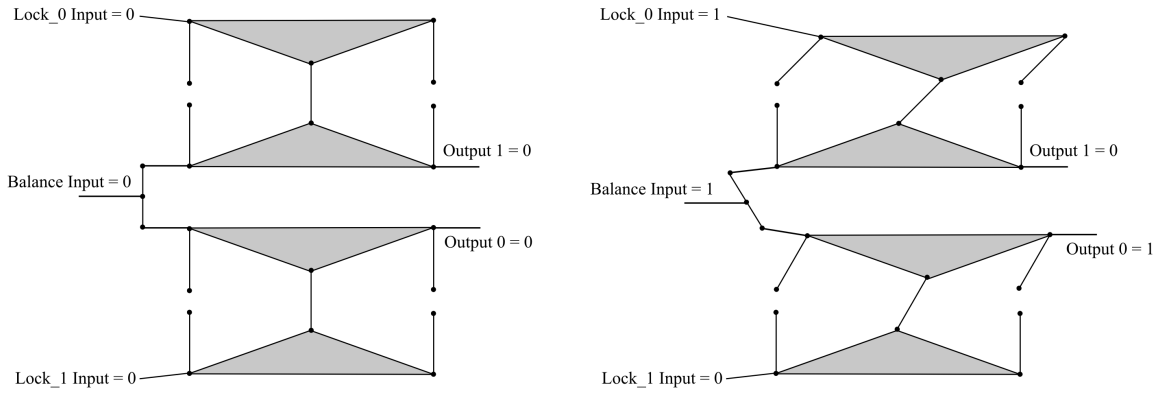


Figure 4: A balance coupled to two locks. The inactive configuration is shown on the left. On the right, Input 0 has been activated, followed by activation of the Balance Input, which in turn activates Output 0.

2.2 The "Lock"

A lock is a mechanism composed of two 4-bar linkages, connected in the center via a connecting link (see Figure 3). The connecting link is the same length as the two side links of each 4-bar linkage, and, in the starting position, is parallel to all four side links.

In the implementation depicted, the connecting link is affixed to two extra links at the top, and two at the bottom, each pair of which form a rigid triangle with the rest of the respective 4-bar linkage. This triangular projection allows the connecting link to be the same length as the side links. If the triangular projection were not present and the connecting link had to connect directly to the horizontal links of the 4-bar linkages, it would not be the same length as the side links. As already mentioned, this would cause the 4-bar linkages to bind up, or lock. To function in the intended manner, all of the vertical links must be the same length and must, at least initially, be parallel to each other.

Obviously, linkages need not have their two positions be left-leaning and right-leaning. Rather, for binary computing purposes, they just need to have two distinct positions which can represent 0 and 1. Figure 3 uses the convention that an input of 0 results in the side links being vertical, while an input of 1 would cause them to lean to the right.

Representative inputs, in the form of linear slides $Input_0$ and $Input_1$, have been added to Figure 3, to show the direction of actuation and where input could be connected. It is assumed that there is one input for each 4-bar linkage, or one input each for the top and bottom of the lock. In an actual system, lock inputs would be connected to other parts of the system (e.g., data from memory, or clock signals). Linear slides are not required.

The lock position depicted in the left of Figure 3 would be called the (0,0) position, referring to the state of the inputs. The righthand portion of Figure 3 shows a depiction of the (1,0) state, where $Input_0$ has been set to 1 and thereby pushed its respective 4-bar linkage into the right-leaning configuration. Flipping the figure upside down would represent the (0,1) state.

The (1,1) state is not possible, and this is a key aspect of a lock. Once one of the inputs has been set to 1, one side of the lock rotates around both its anchored rotary joints, and its connecting link. Crucially, while the connecting link never changes length, it does change angle. As depicted in Figure 3, once the lock moves into the (1,0) position, the connecting link is still parallel to the side links of the top 4-bar linkage, but is no longer parallel to the side links of the bottom 4-bar linkage. Due to the requirement that, if a 3rd link is present, this link be both the same length and at the same angle as the side links, once either the top or bottom 4-bar linkage moves, the other cannot; it is locked. It is for this reason that the (1,1) position is impossible. Once the lock has moved from (0,0) to (1,0) or (0,1), the only possible movement is back to the (0,0) position. Once back to the

(0,0) position, either input could be set to 1, but both inputs can never be set to 1 at the same time.

Note that flexures [13, 14] could take the place of rotary joints, allowing a lock (and other structures) to be monolithic, potentially simplifying manufacture using, for example, MEMS or NEMS techniques [3, 4, 6, 7, 18, 20, 25]. One might even argue that by using flexures, an entire computing system can be largely-monolithic, but as that raises the question of just what a part is, we will use links and rotary joints for clarity. Note that both flexures and rotary joints can be quite energy-efficient as they largely avoid sliding friction (which can have the added benefit of reducing wear [26]).

2.3 The "Balance"

The balance, so named because it is superficially similar to a classic pan balance, connects an input to a link which has three rotary joints: one at each end, and one in the center. The input is connected to the center rotary joint. The two end rotary joints are connected to other structures. Most commonly, these other structures are locks. The result is that, when the balance input is changed, one of the side rotary joints remains stationary, while the other moves. Which side is stationary and which side moves can be determined by data inputs. How this works in practice is best illustrated by example. Figure 4 depicts a balance (green) connected to two locks (blue and gray). While the diagrammatic representations have been simplified slightly, it will be obvious that each lock corresponds to the lock mechanism depicted in Figure 3.

The only difference between the locks in Figure 4 and the lock in Figure 3 is that in Figure 4, rather than having two individual inputs, each lock has one input that is specific to a given lock, and the other input is supplied indirectly by the balance, which also has an input. Note that input to balances will often be supplied by a clock system. Like conventional computers, a multiphase clocking system is required for the described computing system. Such a clocking system is assumed to be present; describing how to implement such a system is beyond the scope of this document, although this too can be accomplished with only links and rotary joints, if desired.

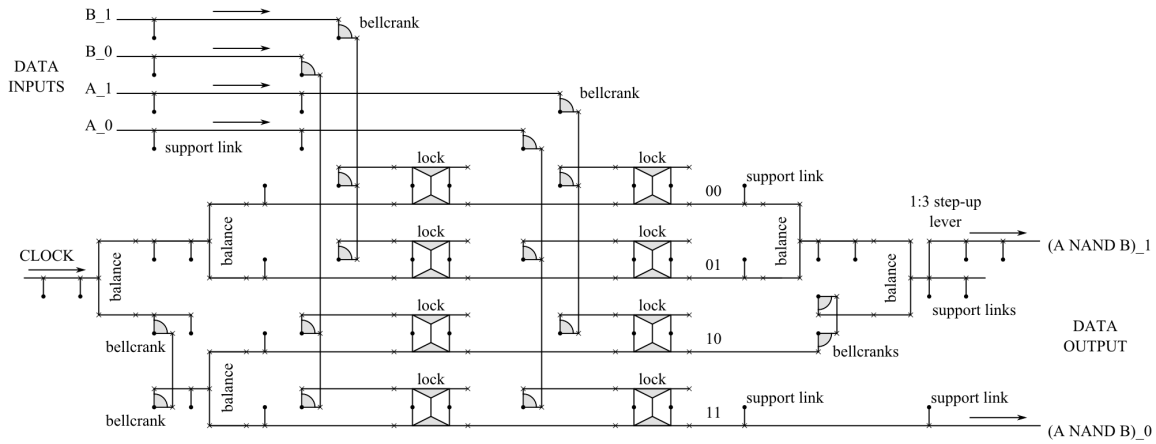
There are assumed to be two main rules governing the mechanism shown in Figure 4, and these rules would be enforced by the overall system; they are not inherent in the mechanism depicted:

1. Either the $Lock_0\, Input$, or the $Lock_1\, Input$, but never both, must be set to 1.
2. The lock inputs and the balance input must be set sequentially, not simultaneously.

Given these rules, the operation of the mechanism is as follows. Each step could be thought of as a different clock phase (which implicitly enforces the second rule above):

1. The mechanism starts with all inputs set to 0.
    - This means that neither lock is locked.
2. Either the $Lock_0\, Input$ is set to 1, or the $Lock_1\, Input$ would be set to 1.
    - This results in locking one of the locks.
    - Locking one of the locks constrains which side of the balance can move.
3. The balance's input is set to 1.
    - Since only one side of the balance is free to move, the balance input is transmitted down the path which is free to move.
    - This produces an output of 1 at either $Output\, 0$ or $Output\, 1$, depending on which lock input was set to 1.

This provides one example of performing simple logic that can, for example, be used to route data, shunting inputs down one path or another based on the states of $Lock_0$ and $Lock_1$.



DATA INPUTS						DATA OUTPUT		
A	A_1	A_0	B	B_1	B_0	A NAND B	(A NAND B)_1	(A NAND B)_0
zero	0	1	zero	0	1	one	1	0
zero	0	1	one	1	0	one	1	0
one	1	0	zero	0	1	one	1	0
one	1	0	one	1	0	zero	0	1

Figure 5: A balance- and lock-based NAND Gate, using a two-link per bit input/output design.

Note that this simple logic and conditional routing has been accomplished using only links and rotary joints, which are solidly connected at all times. No gears, clutches, switches, springs, or any other mechanisms are required (keeping in mind that the input mechanisms shown are representative; an actual system does not require linear slides).

3 Universal Combinatorial Logic

Although the previous example could be thought of as performing simple logic, it is perhaps more useful for routing data. The mechanism in Figure 4 cannot provide all the logic necessary for a complete computational system. However, it is possible to create all necessary logic using nothing but locks and balances (and a few extra links and rotary joints to route and/or copy data).

Any traditional 2-input logic gate, including AND, NAND, NOR, NOT, OR, XNOR and XOR, can be created directly from the appropriate combination of locks and balances. While most of these gates are illustrated in the Appendix, there is no need to address each in detail to prove that universal logic can be created using links and rotary joints. This is because it is well-known that NAND alone suffices to create all necessary combinatorial logic (i.e., all other logic can be created from combinations of NAND gates [8]). Therefore, as a proof of the fact that links and rotary joints suffice to create the combinatorial logic required for a Turing-complete computing system, Figure 5 shows how a NAND gate can be implemented. (Note that reversible gates can also be created using only links and rotary joints, and a Fredkin gate is also demonstrated in the Appendix).

While substantially more complex than the previous example, the NAND gate functions on the same principle of using locks and balances to implement logic. In this example, a set of inputs are connected to a set of locks. The inputs determine which side of each lock is locked. Another input (a clock signal in this example) is then used to actuate a main balance, the movement of which cascades through a series of additional balances and locks. Each balance moves either its top side or its bottom side, in accordance with the state of the locks to which it is connected. This results in a final output lock either having its top half or its bottom half move forward.

In Figure 5, conceptually there are two binary inputs, A and B, and one binary output, X. The physical model used for the inputs and output is that a single 1-bit input or output is broken into

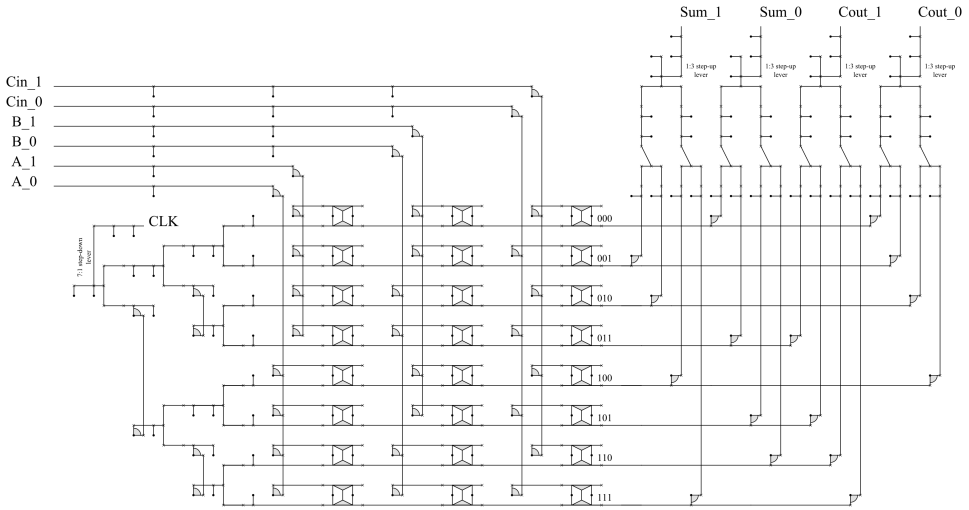


Figure 6: A 1-bit full adder, performing the logical operation described in Figure 7. A table of the schematic symbols used in this drawing is provided in the Appendix (Figure 25).

INPUTS			OUTPUTS	
A	B	Carry In	Sum	Carry Out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Figure 7: Logic table for the 1-bit full adder shown in Figure 6.

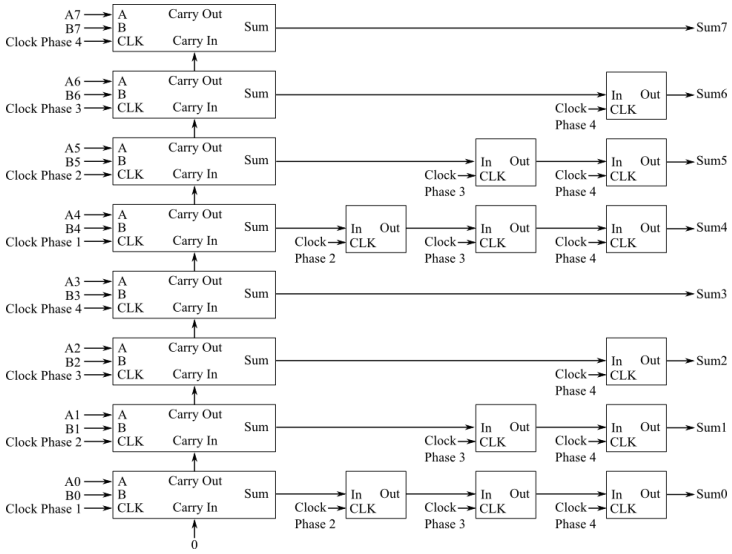


Figure 8: Eight 1-bit full adders (wide blocks) are cascaded using ripple carry. As described in Section 4.2, multiple blocks can be cascaded using a multiphase clock signal. In this drawing a four-phase clock is used. Narrow blocks contain shift register cells, which are used to form a delay line that stores portions of the results during computation. The final result appears on the outputs (right side) after two full clock cycles.

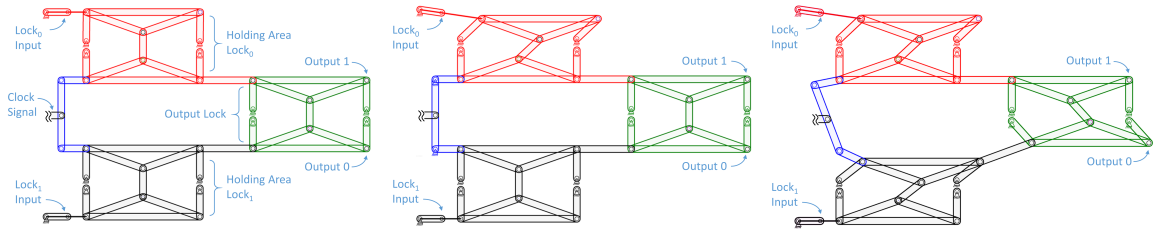


Figure 9: Left: Shift register cell in the (0,0) blank state; Center: Shift register cell with input (1,0) prior to clock actuation; Right: Shift register cell with input (1,0) after clock actuation

two separate input or output links (we will refer to this as the "two-input system"). For example, the A input is composed of inputs A_0 and A_1. If the value of A is to be set to 0, the A_0 input moves. If the value of A is to be set to 1, the A_1 input moves. A_0 and A_1 are not permitted to move simultaneously (which is logically obvious since a value cannot be set to 0 and 1 at the same time). The same applies to the input B and output X.

Note that each input is used twice to implement the desired logic, and so each input is represented twice in the diagram. While duplicating inputs in this manner is convenient for diagrammatic purposes, in an actual system it is simple enough to use a link and rotary joint structure that forks an input line, effectively copying the data to multiple locations as needed. There need not be actual duplicate inputs. For an example, see the Appendix, where the Fredkin gate illustrates one way of forking data, using 4-bar linkages. Also for diagrammatic clarity, the geometry of the links that connect the locks and balances is not what would be optimal in an actual system, but rather the use of straight, non-crossing lines has been favored. Although complex, the outputs that will result from any allowable set of inputs can be deduced by tracking which side of each lock locks, and which side of the balances move. It can be seen that the truth table in Figure 5 replicates that expected from a NAND gate.

Note that Figure 5 is just a single example, used as a proof due to the well-known universal nature of NAND. Using multiple NAND gates to create other logical functions may not always be efficient, and as previously mentioned, we have also shown that any of the standard 2-input logic gates can be implemented directly using locks and balances (see Appendix).

An example of a more complex logic function is shown in Figure 6. This device performs addition with carry on one bit (Figure 7). These devices can be cascaded together to perform addition on more than one bit, as shown in Figure 8.

4 Sequential Logic

Having demonstrated that all necessary combinatorial logic can be created using only links and rotary joints (assembled into locks and balances), we turn to sequential logic. The outputs of the NAND gate depicted in Figure 5 are dependent solely upon the current inputs. Such a mechanism provides no way of storing previous inputs or the results of previous logical operations because the outputs return to zero when the inputs and clock return to zero; it provides no means for creating memory. To demonstrate the creation of a simple memory mechanism, we describe the design of a small shift register, again only using links and rotary joints.

A shift register can be built by combining locks and balances to create cells which are the logical equivalent of electronic flip-flops. Each cell of a shift register is related to its neighbor by virtue of relying upon a preceding or succeeding clock phase, as appropriate. This enables the copying and shifting of data through the shift register, rather than deterministically setting the contents of the entire shift register simultaneously.

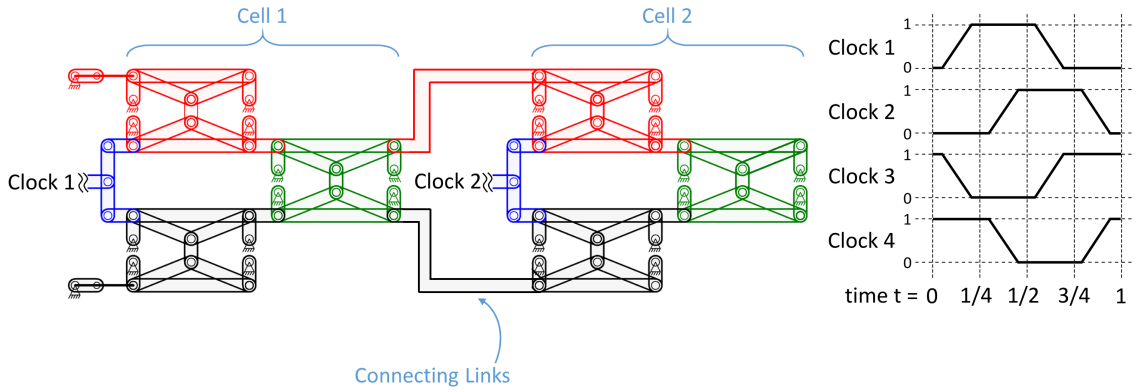


Figure 10: A two-cell shift register (left) shown with a plot of a four-phase clock cycle (right). In this example, the clock signal of Cell 1 is driven by Clock 1 and the clock signal of Cell 2 is driven by Clock 2.

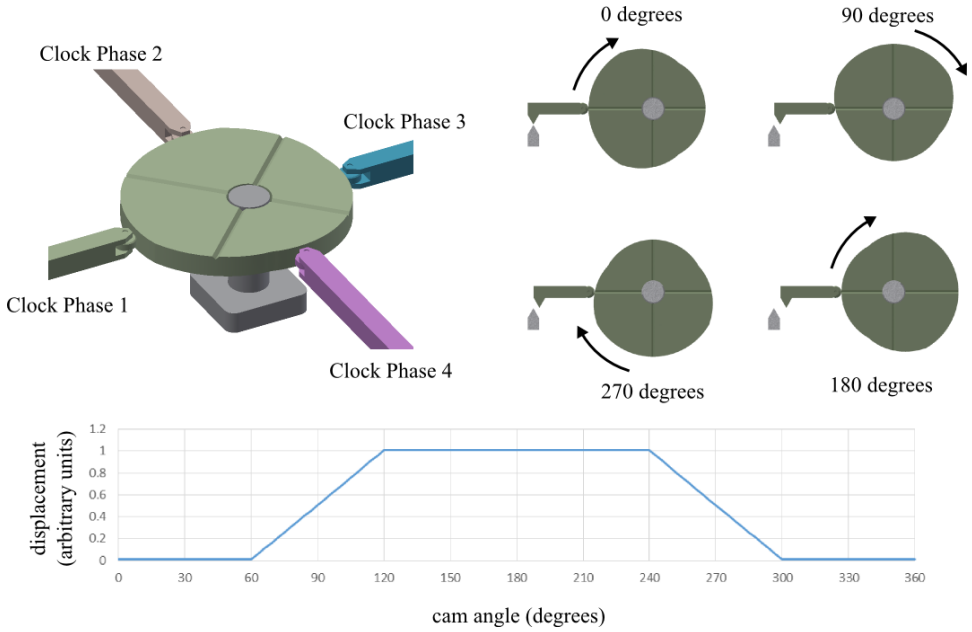


Figure 11: A suitable multiphase clock signal can be generated mechanically using cams and followers. Here, a four-phase clock signal is generated using four identical cams spaced 90 degrees out of phase. The four diagrams at the upper right show the cam at four rotations with only one of the four links. The collection of all four waveforms is shown in the right hand portion of Figure 10.

4.1 A Shift Register Cell

The left side of Figure 9 depicts a single shift register cell with an input of (0,0), or what may be referred to as the blank state. Using the two-input system, the other two possible inputs consist of zero, represented as (1,0), and one, represented as (0,1). The (1,1) state is generally not used, as that state would not permit either side of the balance to move. Note that the blank state suffers from the opposite problem: Both sides of the balance could move. However, the clock signal would not be driven to 1 while the locks are in the blank state, so this is not a practical problem.

Before delving into the actual use of shift register cells, a brief description of the major parts of a cell is in order. First, it is notable that the left-hand portion of a cell is identical to Figure 4, consisting of a balance which is connected to a top lock (Holding Area $Lock_0$) and a bottom lock (Holding Area $Lock_1$). The balance is actuated by a clock signal, again, just like Figure 4. The only difference is the addition of an extra lock, the Output Lock, which is connected to the outputs of the left side of the mechanism and in turn provides the final output for the cell. The easiest way to visualize how a cell works is to step through the movements.

Starting from the blank state, an input is set. Since this mechanism uses the two-input system, either the $Lock_0$ input is set to 1, or the $Lock_1$ input is set to one, but not both. If we assume that $Lock_0$ is set to 1, the movement results in the mechanism being in the state depicted in the center of Figure 9.

In the center of Figure 9, Holding Area $Lock_0$ has moved its upper side. This results in the lower side of that lock locking. However, since shift registers are dependent upon multiphase clocking, note that the input at $Lock_0$ has not yet had any effect on the Output Lock. For that to occur, the balance must be actuated, which does not occur until the next clock phase.

During the next clock phase, the balance is actuated (set from 0 to 1, in this case). Since $Lock_0$ has locked its lower half, which is connected to the upper side of the balance, upon actuation, the balance can only move its lower side. This movement propagates to the Output Lock, resulting in the state depicted on the right side of Figure 9.

The reason the left two locks are referred to as Holding Area Locks may now be apparent: They temporarily hold the input data prior to clock actuation. An input to these locks does not instantly result in an output at the Output Lock. Rather, the clock must actuate first, which results in copying the value, be it 0 or 1, from the holding area locks to the Output Lock. Note that we adopt the convention that, with respect to the input locks, the top lock is associated with an input of 0, while on the Output Lock, the top half is associated with an output of 1. This is because, due to how the mechanism is diagrammed, when an input of 1 is set at the top input lock, an output of 1 ends up at the bottom half of the Output Lock, and vice-versa. The mechanism could be easily altered to change this, but as it is currently represented, from a naming perspective it is easiest to have the 0 input result in a 0 output, and the 1 input result in a 1 output.

Thus far, we have only needed two clock phases: On the first phase, the inputs are set, and on the second phase, the balance actuates and the inputs are copied from the holding locks to the Output Lock. However, in an actual system, additional phases would be used. The subsequent examples assume four-phase clocking.

4.2 Connecting Cells

Figure 10 depicts a two cell shift register to illustrate how two cells would be connected and to explain how data would move from one cell to the next. In this figure, Cell 1 and Cell 2 are each equivalent to the mechanism depicted in Figure 9. Both cells have a connection to a clock signal (depicted as a partial link to indicate connection to a clock system that is not shown). The connecting links connect the output from Cell 1 to the inputs to Cell 2.

The operation of a single cell has already been described. Now, demonstrating how Cell 1 passes data to Cell 2 will illustrate the function of a minimal shift register. The sequence of events is as follows:

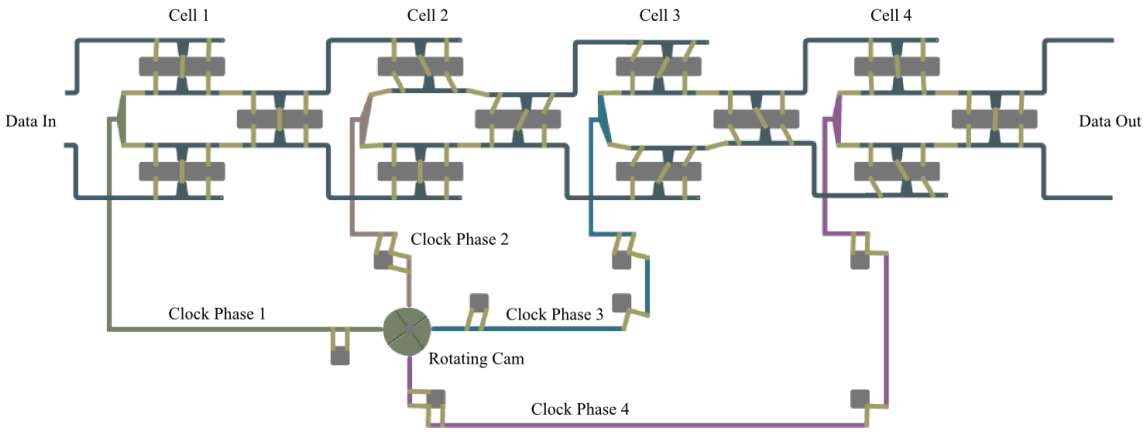


Figure 12: A 4-cell shift register driven by a four-phase clock, shown at time $t = 3/4$. The last three cells are set to state 1 and the first cell in the blank state. An animation of this mechanism is available at http://www.merkle.com/images/4_phase_shift_register_v4.gif and http://www.merkle.com/images/4_phase_shift_register_v4_reverse.gif

1. At time $t = 0$, the clock input for Cell 1 has been set to 0, and the data inputs have been set for Cell 1.
   - Either the upper or lower lock of Cell 1 is locked, depending on which input was set to 1.
2. During the transition from $t = 0$ to $t = 1/4$, the clock signal for Cell 1 is set to 1.
   - This results in the unlocked side of the Cell 1 balance moving, which in turn moves the upper (if the input value was 1) or the lower (if the input was 0) half of the output lock.
   - Cell 1's output lock in turn moves the appropriate connecting link, locking one of Cell 2's holding area locks. This has effectively copied the data from Cell 1's output lock into one of Cell 2's holding area locks.
   - Note that the output lock of Cell 2 still has not moved.
3. During the transition from $t = 1/4$ to $t = 1/2$, the clock signal for Cell 2 is set to 1.
   - This copies the data from Cell 2's holding area locks into Cell 2's output lock.
4. During the transition from $t = 1/2$ to $t = 3/4$, the clock for Cell 1 is reset to 0.
   - As a result, Cell 1's output lock, and hence the connecting links to Cell 2, retract to the 0 position (regardless of the values still stored in Cell 1's holding area cells).
5. During the transition from $t = 3/4$ to $t = 1$, the clock for Cell 2 is reset to 0.

This cycle then repeats itself as new data is input into Cell 1. In step 2 above, it is noted that the output lock of Cell 2 still has not moved. This behavior allows shift register cells to store previous data. This is a key difference between combinatorial and sequential logic. The state of the NAND gate described previously is completely determined by the current data and clock inputs. However, this is not true of shift registers. For example, a shift register with four cells that is driven by a four-phase clock is able to store one bit. One of the four clock phases will always be active for one of the four cells, and thus one of the four cells will contain information. When combined with transition logic to handle input and write enable, this allows the mechanism to act like a mechanical analog of an electronic flip-flop, thereby forming the basis for memory storage (see also Figure 14). To aid in the visualization of how this mechanism works, a 4-cell shift register is diagrammed in Figure 12, and an animated version is available online.

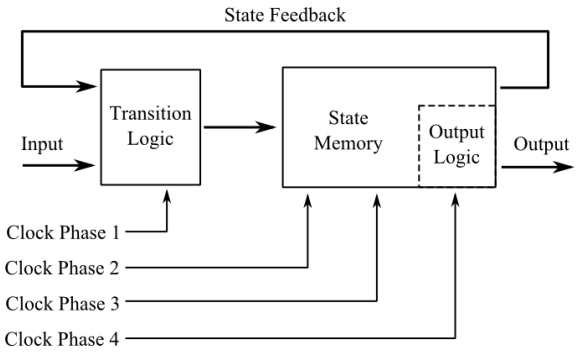


Figure 13: Block diagram of a generic Moore machine adapted for a four-phase clock. The state memory is implemented as a chain of shift register cells, similar to those shown in Figure 12 and Figure 8.

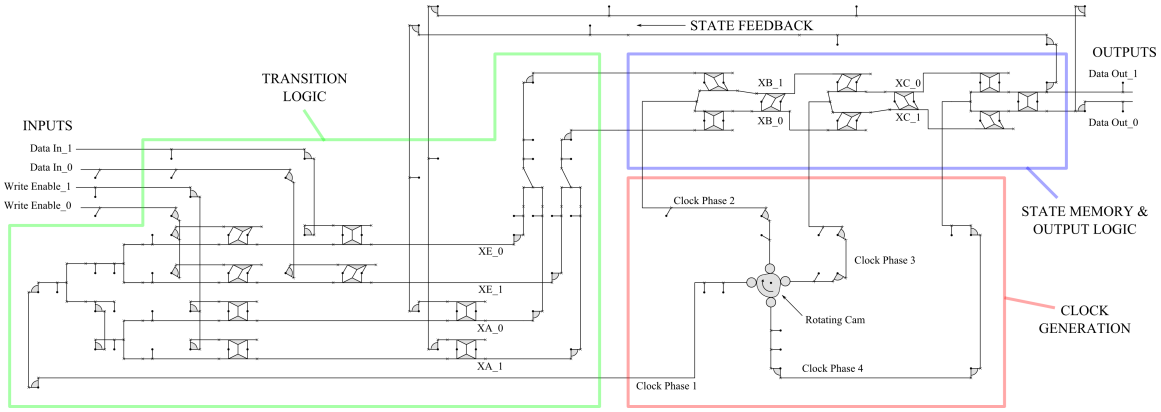
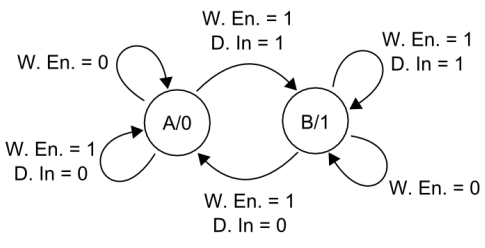


Figure 14: A simple state machine. The main components are highlighted for comparison with Figure 13. This state machine implements the transition table shown in Figure 15. A table of the schematic symbols used in this drawing is provided in the Appendix (Figure 25).



Current State	Inputs		Next State	Outputs
	Write Enable	Data In		Data Out
A	0	0	A	0
A	0	1	A	0
A	1	0	A	0
A	1	1	B	0
B	0	0	B	1
B	0	1	B	1
B	1	0	A	1
B	1	1	B	1

Figure 15: State transition diagram and table for the simple state machine of Figure 14.

4.3 Power Source and Clock Generation

The clock signal essentially functions as a synchronized power source, not unlike three-phase electric power used in electrical grids. An example multiphase clock signal is shown in Figure 10. The amplitude and shape of the clock waveform are somewhat arbitrary, provided that

1. There is sufficient overlap between adjacent clock phases when they are active: there must be a time when two adjacent clock phases are both active.
2. There is sufficient dwell time: there must be a duration of time when a clock phase is completely inactivated.

There are many ways that a suitable multiphase clock signal can be generated, both purely mechanically and otherwise. Examples include but are not limited to cams and followers (see Figure 11, Figure 12, and Figure 14); linkage-based dwell mechansisms; devices based on crankshafts, springs, and mechanical stops; and electrically driven MEMS comb actuators.

4.4 Finite State Machines

Combinatorial and sequential logic can be combined to build finite state machines. A block diagram of a generic Moore machine adapted for a four-phase clock is shown in Figure 13. Transition logic is implemented using the method described in Section 3. State memory is implemented using a chain of shift register cells as described in Section 4.2. A detailed example of a state machine is shown in Figure 14. This machine functions essentially as a one bit memory, as described in the state transition table in Figure 15.

4.5 High-density Memory

The state machine in Figure 14 is closely equivalent to a D flip-flop – it stores one bit of information on every clock cycle. This circuit is suitable for use in state machines, but it is not an efficient use of gates for high-density memory applications. While it departs from the links-and-pivots-only schema, the mechanism shown in Figure 16 uses far less gates to store a bit of information. The tradeoff is that the read/write process is somewhat slower and more involved.

4.6 Reversibility

Close inspection of the shift register chain in Figure 12 reveals that, if the clock generating mechanism is operated in reverse, information will propagate backward through the chain, from outputs to inputs. This is a characteristic and intentional feature of the mechanical link logic architecture. It is well known that reversible computers can be constructed from reversible logic gates such as the Fredkin gate [1]. Figure 17 shows how physically reversible Fredkin gates can be constructed from links and rotary joints (logic table in Figure 18). Reversibility can be exploited to create computers with extremely low energy dissipation. This potential application is discussed further in Section 5.4
.

5 Implementations and Applications

Mechanical computers have potential applications at the meso-, micro-, and nano-scales. Applications at the meso- to micro-scale include such disparate areas as soft robotics [14,21] and failsafe devices resistant to radiation and electromagnetic interference [19]. Mechanical computers constructed from nano-scale components could potentially dissipate much less energy than conventional CMOS devices, while providing comparable computing performance. Manufacturing at such small scales is challenging. The architecture described in this paper, consisting mainly of links and rotary joints,

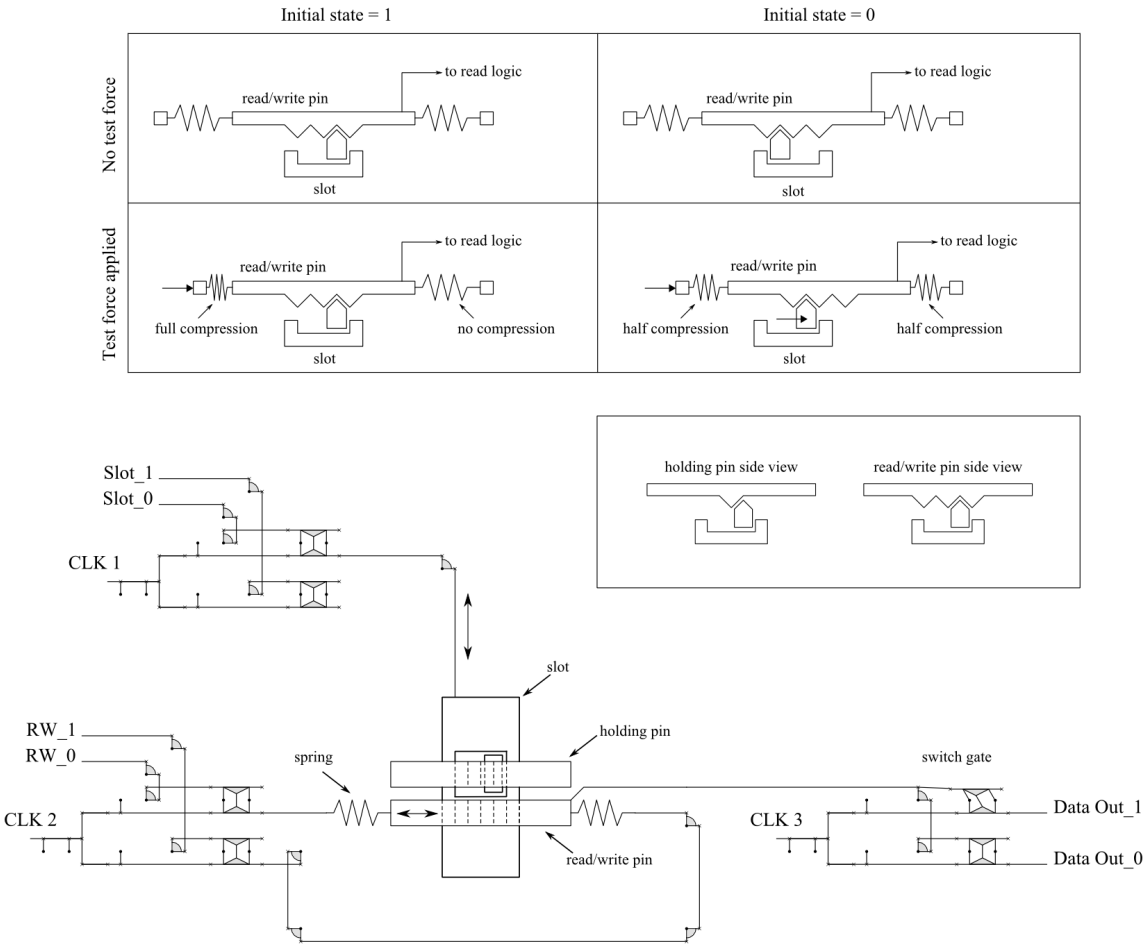


Figure 16: Logic circuitry for read and write operations on a high density memory storage cell.

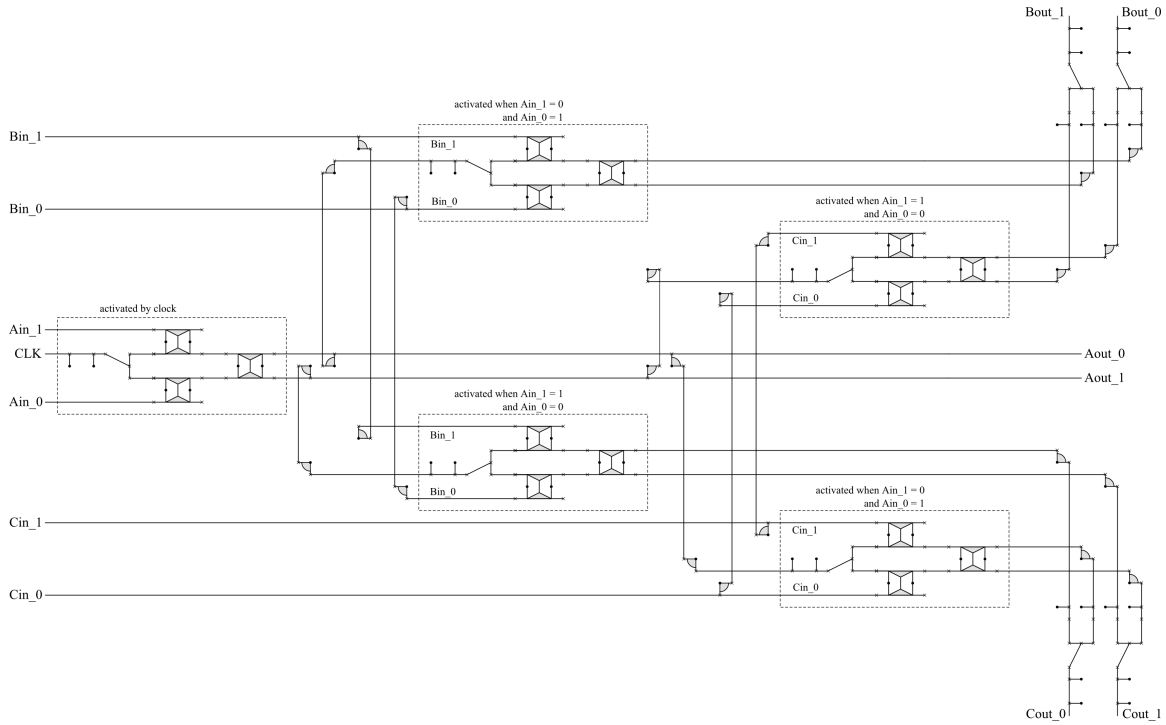


Figure 17: A balance- and lock-based Fredkin (CSWAP) Gate, using a two-link per bit input/output design, and duplicated inputs for diagrammatic clarity. This mechanical logic gate is logically and physically reversible. The logic table implemented is shown in Figure 18.

INPUTS						OUTPUTS					
Ain_1	Ain_0	Bin_1	Bin_0	Cin_1	Cin_0	Aout_1	Aout_0	Bout_1	Bout_0	Cout_1	Cout_0
0	1	0	1	0	1	0	1	0	1	0	1
0	1	0	1	1	0	0	1	0	1	1	0
0	1	1	0	0	1	0	1	1	0	0	1
0	1	1	0	1	0	0	1	1	0	1	0
1	0	0	1	0	1	1	0	0	1	0	1
1	0	0	1	1	0	1	0	1	0	0	1
1	0	1	0	0	1	1	0	0	1	1	0
1	0	1	0	1	0	1	0	1	0	1	0

Figure 18: Logic table for reversible Fredkin (CSWAP) gate (see Figure 17). Identical portions of the table are highlighted in the same color as a guide for the eye.

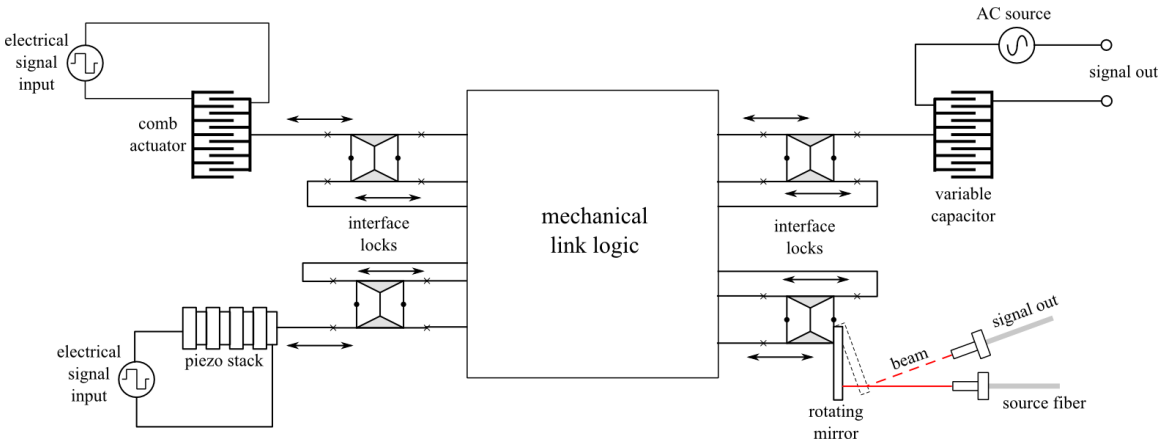


Figure 19: On the left, two concepts for inputting signals to the mechanical computer are shown; two concepts for outputting signals are shown on the right.

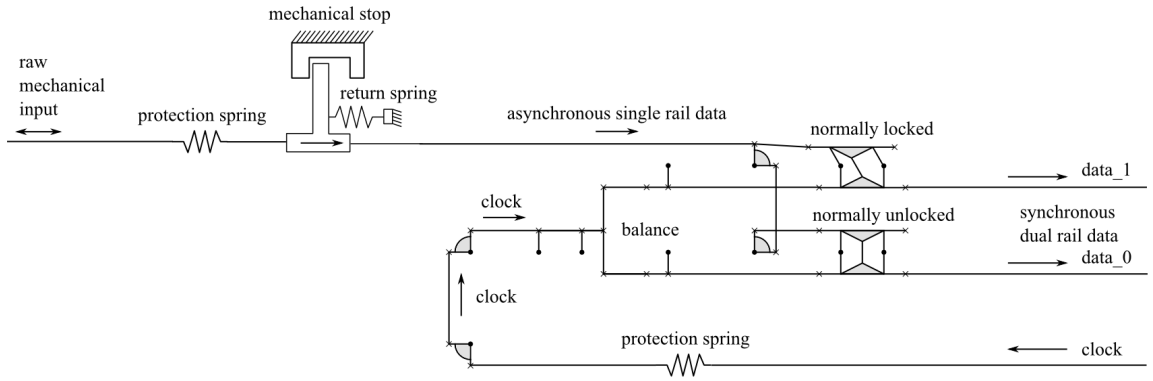


Figure 20: Asynchronous, noisy signals (such as from a mechanical sensor) can be conditioned so that synchronous dual rail mechanical logic gates can process them.

could be well-suited for such sizes since it does not require fabricating a large set of complicated components. In particular, a molecular version of this architecture could use stiff covalently-bonded nanotubes for the links and single bonds for the joints.

5.1 Input and Output

In many potential applications where a mechanical computer is constructed for its particular advantages (low energy dissipation, radiation hardness, etc) it may still be necessary to interface the mechanical elements to conventional electronic devices. Figure 19 illustrates a number of concepts for transferring information in and out of a mechanical computing device. In the top left, a conventional electrostatic MEMS comb actuator is used to move the (presumably very small) mechanical input links. In the lower left the input links are moved by a piezoelectric actuator. On the top right, mechanical links are used to move a MEMS variable capacitor, the value of which is read off by conventional electronics. On the lower left an optomechanical scheme is shown where the output links move a mirror to modulate an optical signal interfaced to standard optoelectronic components.

Typical input signals will be noisy and asynchronous, with amplitudes exceeding the allowed range of motion for the logic gates. These signals can be conditioned using mechanisms such as that shown in Figure 20.

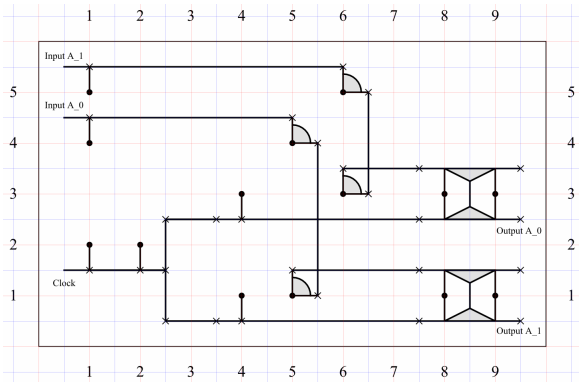


Figure 21: Schematic layout for a simple test mechanism containing a balance, two locks, and signal routing mechanisms.

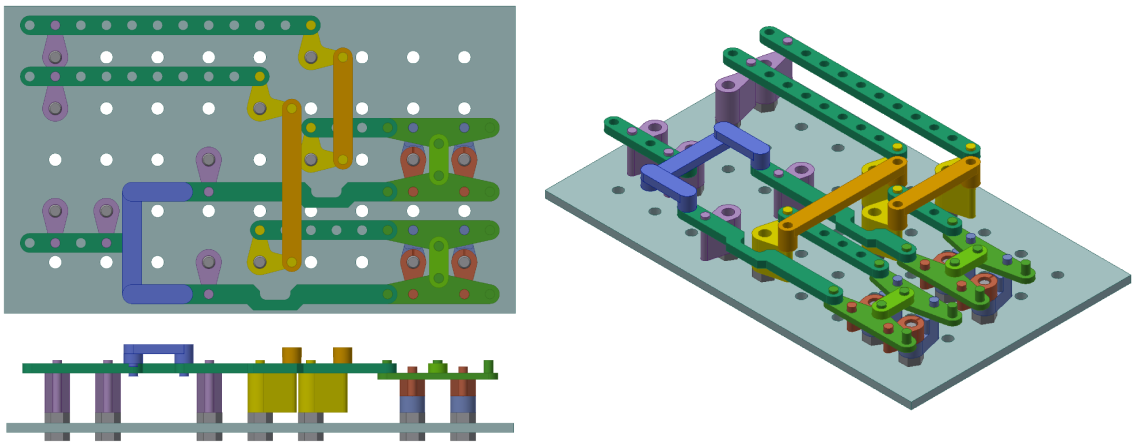


Figure 22: CAD model of the test system shown in Figure 21 implemented with 3D printed components assembled over quarter-inch bolts installed in a standard pegboard. This is a simple and low-cost method for testing and demonstrating mechanical link logic devices.

5.2 Macroscopic Components

A straightforward option for creating macroscale mechanical link logic devices is shown in Figure 21 and Figure 22. Standard pegboard material is used as the foundation, with 0.25 inch on 1.0 inch centers. Off-the-shelf 1/4-20 bolts serve as the pivots, and the remaining links and locks are made by 3D printing. While this is an effective method for quick demonstrations and educational purposes, it is unlikely to find much practical use outside these niche applications.

5.3 Flexure-based Designs

Flexure joints provide an alternative implementation with similar general performance to pivots. Flexures have the advantage that in many cases, particularly with MEMS, they are easier to fabricate and often more reliable than fully functional pivot joints. A conceptual design of a flexure-based mechanical link logic system is shown in Figure 23. A systematic method of design allows all necessary locks, balances, bell cranks, support links, and transmission links to be implemented in only two layers of material. These layers are shown as orange and blue in Figure 23. Spot-welds or rigid bonding between layers, as often used in flip-chip MEMS, holds the layers together at grid points. Each layer is a monolithic pure 2D pattern, making this approach well suited for

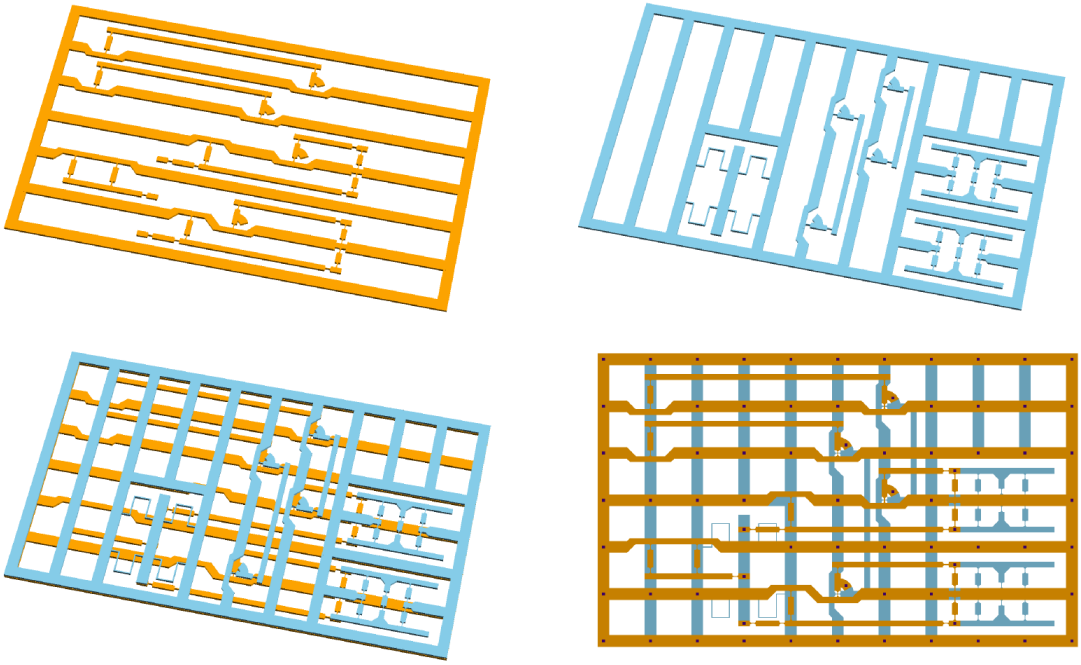


Figure 23: A flexure-based implementation of the system shown in Figure 21, made of two or three stacked layers (a second orange layer can be added to make an orange-blue-orange assembly for increased rigidity) . Interlayer bonds are shown as dark blue dots in the lower right image.

conventional microfabrication techniques such as LIGA, silicon micromachining, or high-resolution additive manufacturing.

5.4 Atomically Precise Manufacturing

Recently there has been some resurgence of interest [9, 15, 24] in molecular machines created by atomically precise manufacturing [5]. Apart from their ultimate miniaturization, molecular machines would offer the advantages of very low friction and zero wear. Mechanical computers constructed from molecular-scale atomically-precise components would be highly desirable because of their potential for combined high performance and low energy dissipation.

5.4.1 Drag on Molecular Rotary Joints

A key performance metric of computers is their energy dissipation. One contribution to dissipation is friction at the rotary joints in each logic gate. Due to the joint's small frictional drag, mechanical computers constructed from them can, in principle, dissipate orders of magnitude less power than conventional semiconductor computers, while still operating at relatively high speeds.

For instance, Figure 24 shows a lock containing bonded rotary joints [17]. Operating this lock involves rotation at the joints by up to $\Delta\theta \approx 1\,\text{rad}$. The model system analyzed in [11] is an excerpt of the links and joints shown in the closeup on the right of Figure 24. From [11, Eq. 2], this rotation dissipates about $2.4 \times 10^{-27}\,\text{J}$ per rotary joint when operating at $f = 100\,\text{MHz}$. Operation of the complete lock, involving multiple joints, would dissipate about an order of magnitude more. This dissipation is several orders of magnitude smaller than $k_\text{B}T = 4.1 \times 10^{-21}\text{J}$ at $T = 300\,\text{K}$. Thus at this operating speed, the rotor frictional dissipation per logic operation would be far below $k_\text{B}T$.

Fully exploiting the rotor's low dissipation for a computer requires avoiding other sources of dis-

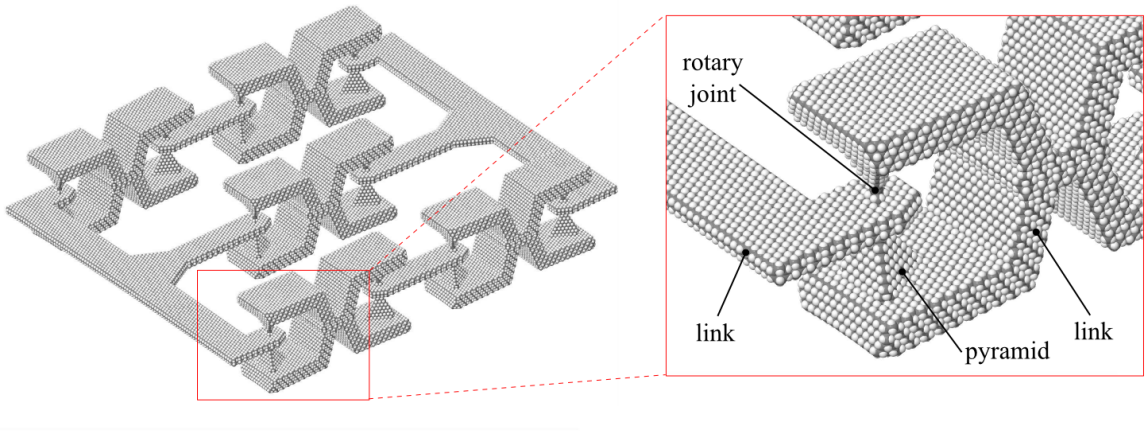


Figure 24: Part of a molecular mechanical logic gate. This molecular machine consists of 120695 atoms, 87595 carbon and 33100 hydrogen, and occupies a volume of about 27 nm × 32 nm × 7 nm. The nine rigid links are connected to each other via a pair of rotary joints.

sipation. For instance, the clock system driving the logic gates could store and release energy to the extent it might be needed to move up or over any potential barriers that might be encountered [11]. More fundamentally, the computer would need to use reversible logic gates based on this rotary joint, to avoid the minimum $k_B T \log 2$ dissipation from each logically irreversible bit operation predicted theoretically [1,10,16] and observed experimentally [2,12]. Such a machine could perform arbitrarily many computing operations per dissipated joule by operating at a sufficiently slow clock speed. The lowest possible energy dissipation would be aided by reducing or even eliminating potential barriers in those mechanical degrees of freedom involved in normal device operation.

A common performance metric for semiconductor devices is (energy dissipated per operation) multiplied by (time per operation). For dissipation due to velocity-dependent friction, as is the case for rotary joints, dissipation is proportional to speed, hence 1/time for operation requiring a fixed amount of motion, e.g., 60-degree rotation for the gate operations described in this paper. Hence the energy*time performance measure is a constant. From [11, Eq. 2], this constant is $E_{\text{dissipated}} t = k_{\text{rd}} \phi^2$ for rotation by angle ϕ. Supposing a logic operation corresponds to rotating about 10 joints by about a radian, this energy-time product is about 10^{-34} J s, using k_{rd} from [11].

For a machine using multiple rotary joints, such as a computer, interactions between nearby rotors could affect the dissipation. This could constrain how closely rotors can be placed before interactions significantly increase dissipation. An experimental study of arrays of molecular rotors on a surface [27] illustrates the effect of such interactions.

5.4.2 Effect of Large-Scale Design on Drag

The friction evaluation of [11] considered complete rotations of a rotor connected to a housing solely through the rotary joints. When used in the mechanical computer discussed in this paper (and as illustrated in Figure 24), the rotor is linked to other parts of the gate.

These links increase coupling to the environment, and could affect drag. For example, the link will reduce the amount of rotor axis tilt and shift, compared to simulations of an isolated rotor and housing. The link, in effect, stiffens the rotor against tilt, and is an alternative to local changes in the design that could also stiffen the joint, e.g., altering the housing size to place the bond in tension rather than compression. These alternatives are an example of how molecular machine design goals could be realized either locally or at larger scales by choice of how the molecular machine is embedded into its environment.

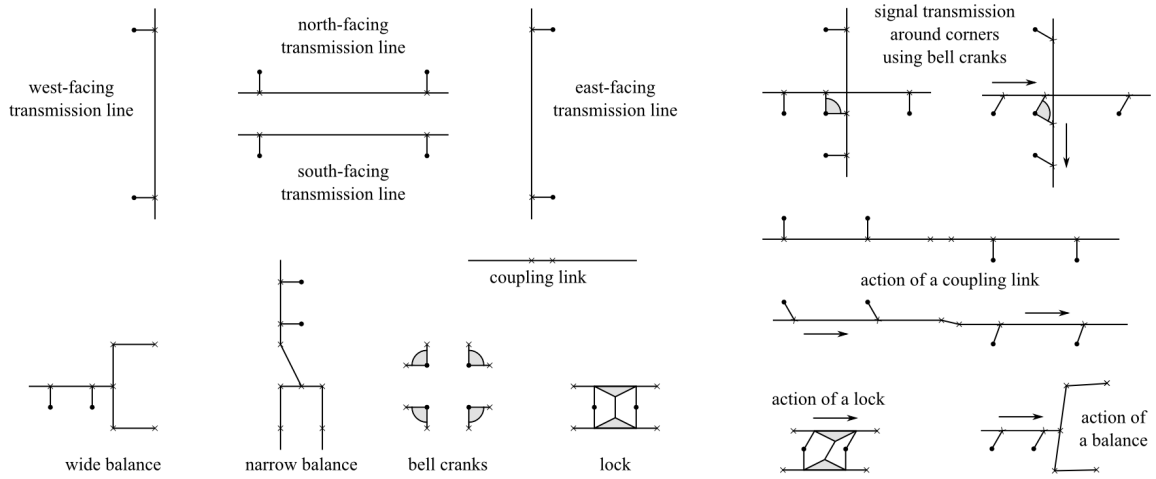


Figure 25: Schematic symbols for mechanical linkage logic components, including the lock, two versions of a balance, and mechanisms for routing signals.

Another instance of a higher-level design choice is arranging designs of neighboring rotary joints to help reduce dissipation. Specifically, the rotary joint has a small potential barrier to rotation [11] that could be a significant contribution to dissipation at low temperatures. That is, imperfect recovery of energy storage and release when moving over the rotor potential leads to dissipation. This could be reduced by arranging the rotors at each end of a link to be offset by 60 degrees, In this case, when one rotor is at a potential minimum the other is at a maximum. This would keep energy storage and release closer to the rotors than, say, a storage spring located near the clock input, which could be less effective at transmitting energy to and from the rotor due to "rubbery" links. This procedure is analogous to using counterweights with elevators.

6 Conclusions

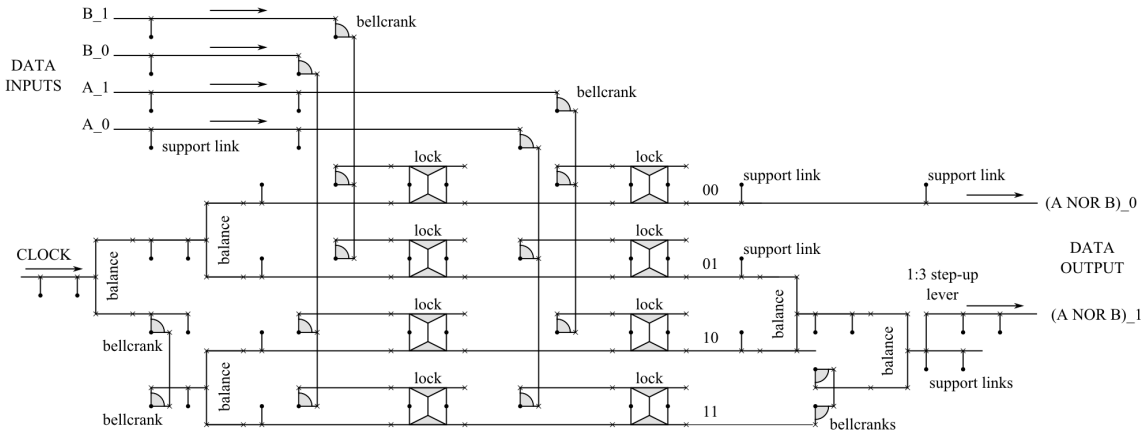
Universal combinatorial logic and sequential logic together are known to suffice for the creation of a general purpose, or Turing-complete, computational system. Subject to practical limits of time and memory (as in any computer), such a system can compute anything that can be computed.

We have demonstrated that, using only links and rotary joints, a Turing-complete computational system can be created. Universal combinatorial logic has been demonstrated with the design of a NAND gate, while sequential logic, mimicking electronic flip-flops and sufficient to create memory, has been demonstrated using cells combined into shift registers.

This design paradigm is far simpler than any other mechanical Turing-complete design of which we are aware. Additionally, due to the avoidance of substantial sliding friction, the "links and rotary joints" paradigm has the potential to be more power efficient than any previous design of which we are aware. In fact, simulations suggest that molecular-scale implementations of the described system would be far more power efficient than conventional electronic computers.

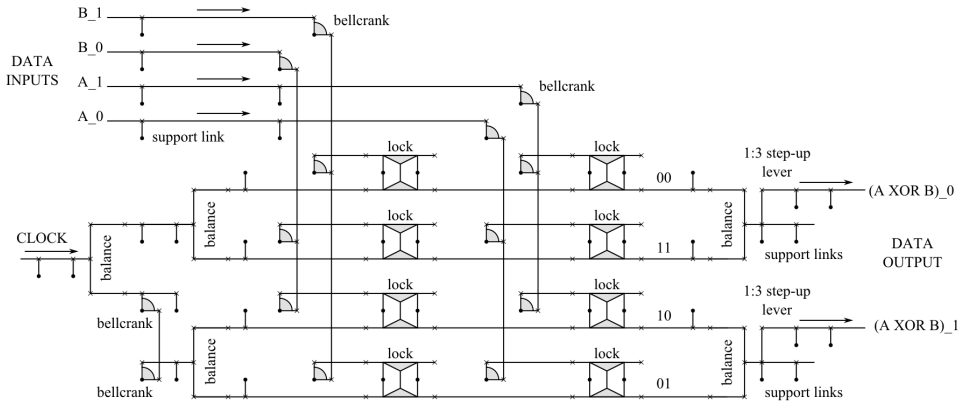
APPENDIX

This appendix contains diagrams that show how common logic gates can be constructed from locks and balances (Figure 25). The included gates are: NOR (Figure 26) and XOR (Figure 27). OR, AND, and XNOR gates can be implemented by inverting the gates shown - inversion is easily accomplished by simply switching the "one" and "zero" lines.



DATA INPUTS						DATA OUTPUT		
A	A_1	A_0	B	B_1	B_0	A NOR B	(A NOR B)_1	(A NOR B)_0
zero	0	1	zero	0	1	zero	0	1
zero	0	1	one	1	0	one	1	0
one	1	0	zero	0	1	one	1	0
one	1	0	one	1	0	one	1	0

Figure 26: A balance- and lock-based NOR Gate, using a two-link per bit input/output design.



DATA INPUTS						DATA OUTPUT		
A	A_1	A_0	B	B_1	B_0	A XOR B	(A XOR B)_1	(A XOR B)_0
zero	0	1	zero	0	1	zero	0	1
zero	0	1	one	1	0	one	1	0
one	1	0	zero	0	1	one	1	0
one	1	0	one	1	0	zero	0	1

Figure 27: A balance- and lock-based XOR Gate, using a two-link per bit input/output design.

References

[1] Charles H. Bennett. The thermodynamics of computation: a review. *International Journal of Theoretical Physics*, 21(12):905–940, 1982.

[2] Antoine Bérut, Artak Arakelyan, Artyom Petrosyan, Sergio Ciliberto, Raoul Dillenschneider, and Eric Lutz. Experimental verification of Landauer's principle linking information and thermodynamics. *Nature*, 483:187–189, 2012.

[3] Kenneth C Bradley. Mechanical computing in microelectromechanical systems (mems). Master's thesis, Air Force Institute of Technology, 2003.

[4] Faisal Khair Chowdhury. *Micro-electro-mechanical-systems-based single-device digital logic gates for harsh environment applications*. PhD thesis, The University of Utah, 2013.

[5] K. Eric Drexler. *Nanosystems: Molecular Machinery, Manufacturing, and Computation*. John Wiley & Sons, Inc., New York, 1992.

[6] K. L. Ekinci and M. L. Roukes. Nanoelectromechanical systems. *Review of scientific instruments*, 76(6):061101, 2005.

[7] Heiko Fettig, James Wylde, Ted Hubbard, and Marek Kujath. Simulation, dynamic testing and design of micromachined flexible joints. *Journal of Micromechanics and Microengineering*, 11(3):209, 2001.

[8] Richard P. Feynman. Quantum mechanical computers. *Optics News*, 11(2):11–20, 1985.

[9] David R. Forrest. Integrated nanosystems for atomically precise manufacturing. In *Workshop on Integrated Nanosystems for Atomically Precise Manufacturing (INFAPM)*, Berkeley, CA, August 2015. US Department of Energy Advanced Manufacturing Office.

[10] Michael P. Frank. Introduction to reversible computing: motivation, progress, and challenges. In *Proceedings of the 2nd Conference on Computing Frontiers*, pages 385–390. ACM, 2005.

[11] Tad Hogg, Matthew S. Moses, and Damian G. Allis. Evaluating the friction of rotary joints in molecular machines. *Molecular Systems Design & Engineering*, 2:235–252, 2017.

[12] Jeongmin Hong, Brian Lambson, Scott Duhey, and Jeffrey Bokor. Experimental test of Landauer's principle in single-bit operations on nanomagnetic memory bits. *Science Advances*, 2:e1501492, 2016.

[13] Larry L. Howell. *Compliant mechanisms*. John Wiley & Sons, 2001.

[14] Alexandra Ion, Ludwig Wall, Robert Kovacs, and Patrick Baudisch. Digital mechanical metamaterials. In *Proceedings of CHI 2017*. ACM, May 2017.

[15] Salma Kassem, Alan T. L. Lee, David A. Leigh, Vanesa Marcos, Leoni I. Palmer, and Simone Pisano. Stereodivergent synthesis with a programmable molecular machine. *Nature*, (549):374–378, September 2017.

[16] Rolf Landauer. Irreversibility and heat generation in the computing process. *IBM J. of Research and Development*, 5:183–191, 1961.

[17] Ralph C. Merkle, Robert A. Freitas Jr., Tad Hogg, Thomas E. Moore, Matthew S. Moses, and James Ryley. Molecular mechanical computing systems. techreport 46, Institute for Molecular Manufacturing, Palo Alto, CA, 2016.

[18] A. Modi, H. Shah, C. Amarnath, P.S. Gandhi, S.G. Singh, and R. Rashmi. Design, analysis and fabrication of a microflexural and gate. In *13th National Conference on Mechanisms and Machines (NaCoMM-07)*, pages 275–279, Bangalore, India, 2007. Indian Institute of Science.

[19] David Plummer and William Greenwood. The history of nuclear weapon safety devices. In *34th AIAA/ASME/SAE/ASEE Joint Propulsion Conference and Exhibit*, page 3464, 1998.

[20] Vincent Pott, Hei Kam, Rhesa Nathanael, Jaeseok Jeon, Elad Alon, and Tsu-Jae King Liu. Mechanical computing redux: relays for integrated circuit applications. *Proceedings of the IEEE*, 98(12):2076–2094, 2010.

[21] Jordan R Raney, Neel Nadkarni, Chiara Daraio, Dennis M Kochmann, Jennifer A Lewis, and Katia Bertoldi. Stable propagation of mechanical signals in soft media using stored elastic energy. *Proceedings of the National Academy of Sciences*, page 201604838, 2016.

[22] John H. Reif. Mechanical computation: it's computational complexity and technologies. In Robert A. Meyers, editor, *Encyclopedia of Complexity and System Science*, pages 5466–5482. Springer-Verlag, 2009.

[23] M. L. Roukes. Mechanical computation, redux? In *Electron Devices Meeting, 2004. IEDM Technical Digest. IEEE International*, pages 539–542. IEEE, 2004.

[24] Robert F. Service. Chemistry Nobel heralds age of molecular machines. *Science*, 354(6309), Oct 2016.

[25] A. Sharma, W. S. Ram, and C. Amarnath. Mechanical logic devices and circuits. In *14th National Conference on Machines and Mechanisms (NaCoMM-09)*, pages 235–239, Durgapur, India, 2009. Department of Mechanical Engineering, National Institute of Technology.

[26] James G. Skakoon. There's the rub: some surprising discoveries are made in the quest of a practically frictionless mechanical operation. *Mechanical Engineering*, 2009.

[27] Y. Zhang, H. Kersell, R. Stefak, J. Echeverria, V. Iancu, U. G. E. Perera, Y. Li, A. Deshpande, K.-F. Braun, C. Joachim, G. Rapenne, and S.-W. Hla. Simultaneous and coordinated rotational switching of all molecular rotors in a network. *Nature Nanotechnology*, 2016.

[28] Konrad Zuse. *The Computer – My Life*. Springer-Verlag, 1993.