# Particle Computation: Device Fan-out and Binary Memory

Aaron Becker, Rose Morris-Wright, Erik D. Demaine, Sándor P. Fekete

***Abstract*— Consider a 2D grid world, where all obstacles and robots are unit squares, and for each actuation, robots move maximally until they collide with an obstacle or another robot. We demonstrated *particle computation* in this world, designing obstacle configurations that implement AND and OR logic gates. By using dual-rail logic, we designed NOT, NOR, NAND, XOR, XNOR logic gates. However, we could not implement a FAN-OUT gate. This prevented us from creating arbitrary digital circuits. In this work we prove unit-sized robots cannot generate a FAN-OUT gate. We introduce $2 \times 1$ robots, which can create fan-out gates that produce multiple copies of the inputs. Using these gates we can create complex digital circuits. As an example we connect our logic elements to produce a 3-bit counter. We also implement a data storage element.**

## I. INTRODUCTION

Currently, micro- and nanorobot systems with many robots are steered and directed by a common control signal [1]–[7]. These common control signals include global magnetic or electric fields, chemical gradients, and turning a light source on and off. In this paper, we show how a common control signal, mobile particles, and unit-sized obstacles can implement a computer. We do not present particle logic as an alternative to electronic computing. Frankly, this form of computation is impractical. It is slow, requires large amounts of space, and is vulnerable to manufacturing defects. Rather, we quantify the computing power of mobile robotics at the most simple level in order to gain insight for massively-parallel, automated assembly at the micro and nano length-scales.

### A. Model

This paper builds on the techniques for controlling many simple robots with uniform control inputs presented in [8]–[10], using the following rules:

1) A planar grid *workspace* $W$ is filled with some unit-square robots (each occupying one cell of the grid) and some fixed unit-square blocks. Each unit square in the workspace is either *free*, which a robot may occupy or *obstacle* which a robot may not occupy. Each square in the grid can be referenced by its Cartesian coordinates $\bar{x} = (x, y)$.
2) All robots are commanded in unison: the valid commands are "Go Up" ($u$), "Go Right" ($r$), "Go Down" ($d$), or "Go Left" ($l$).

A. Becker is with the Department of Cardiovascular Surgery, Boston Children's Hospital and Harvard Medical School, Boston, MA, 02115 USA {first name.lastname}@childrens.harvard.edu, R. Morris-Wright rmorriswright@gmail.com, E. Demaine is with the Computer Science and Artificial Intelligence Laboratory, MIT, Cambridge, MA 02139, USA, edemaine@mit.edu, S. Fekete is with the Dept. of Computer Science, TU Braunschweig, Mühlenpfordtstr. 23, 38106 Braunschweig, Germany, s.fekete@tu-bs.de.

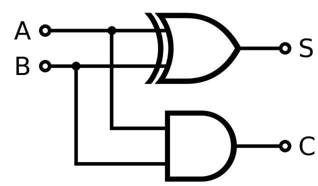


Fig. 1. The half adder shown above requires two copies of $\mathbf{A}$ and $\mathbf{B}$.

3) The robots all move in the commanded direction until they hit an obstacle or a stationary robot. A *command sequence* $M$ consists of a ordered sequence of moves $m_k$ where each $m_k \in \{u, d, r, l\}$ A representative command sequence is $\langle u, r, d, l, d, r, u, \ldots \rangle$. We assume the area of $W$ is finite and issue each command long enough for the robots to reach their maximum extent.

### B. Dual Rail Logic and Fan-out Gates

As shown in [8], AND and OR can be implemented with single particles. However, particle logic is *conservative*—particles are neither created nor destroyed–and we were unable to implement a NOT gate. To implement NOT gates and other logic we used *dual-rail logic*, where two lines for each input are supplied to explicitly represent the variable and its complement [10]. Here, we show that dual rail logic is necessary, as single rail logic is insufficient to produce a NOT gate.

The *fan-out* of a logic gate output is the number of gate inputs it can feed or connect to. With particle logic, as demonstrated in [9], each logic gate output could fan-out to only one gate. This is sufficient for *sum of products* and *product of sums* operations in CPLDs (complex programmable logic devices), but insufficient for more flexible architectures. Consider the half-adder shown in Fig. 1. The inputs $\mathbf{A}$ and $\mathbf{B}$ are needed to compute both the SUM and the CARRY bits, so the fanout of $\mathbf{A}$ and $\mathbf{B}$ is two. In this paper we prove the insufficiency of unit sized particles for the implementation of fan out gates, and design a fan out gate using $2 \times 1$ particles.

### C. Contributions

After a brief overview of related work, the contributions of this paper are as listed:

1) prove the necessity of dual-rail logic for Boolean logic (Section III)
2) prove the insufficiency of unit-size particles for gate fan-out (Section III)
3) design FAN-OUT gates (Section IV-B)
4) design memory latches (Section IV-C)

5) present architecture for device integration, design a common clock sequence, and present a binary counter (Section IV-D)

## II. Related Work

Our efforts have similarities with *mechanical computers*, computers constructed from mechanical, not electrical components. For a fascinating nontechnical review, see [11]. These devices have a rich history, from the *Pascaline*, an adding machine invented in 1642 by a nineteen-year old Blaise Pascal; Herman Hollerith's punch card tabulator in 1890; to the mechanical devices of IBM culminating in the 1940s. These devices used precision gears, pulleys, or electric motors to carry out calculations. Though our GRID-WORLD implementations may seem anachronistic, note that we require none of these precision elements—merely unit-size obstacles, and $2 \times 1$ and $1 \times 1$ sliding particles.

### A. Collision-based computing

Collision-based computing has been defined as *computation in a structureless medium populated with mobile objects*. For a survey of this area, see the excellent collection [12]. Early examples include the billiard-ball computer proposed by Fredkin and Toffoli using only spherical balls and a frictionless environment composed of elastic collisions with other balls and with angled walls [13]. Another popular example is Conway's game of life, a cellular automaton governed by four simple rules [14]. Cells live or die based on the number of neighbors. These rules have been examined in depth and used to design a Turing-complete computer [15]. Game of life scenarios and billiard-ball computers are fascinating, but lack a physical implementation. In this paper we present a collision-based system for computation and provide a physical implementation.

### B. Sliding-block puzzles

Sliding block puzzles use rectangular tiles that are constrained to move in a 2D workspace. The objective is to move one or more tiles to desired locations. They have a long history. Hearn [16] and Demaine [17] showed tiles can be arranged to create logic gates, and used this technique to prove P-SPACE complexity for a variety of sliding block puzzles. Hearn expressed the idea of building computers from the sliding blocks—many of the logic gates could be connected together, and the user could propagate a signal from one gate to the next by sliding intermediate tiles. This requires the user to know precisely which sequence of gates to enable/disable. In contrast to such a hands-on approach, with our architecture we can build circuits, store parameters in memory, and then actuate the entire system in parallel using a global control signal.

## III. Theory

First we provide terminology to define how robots interact with each other.

*a) Definition of hit:* During move $m_k$, robot $a$ *hits* robot $b$ if $a$ is prevented from reaching location $\bar{x} = (x, y)$ because robot $b$ occupies this location. Robot $b$'s location at the end of $m_k$ will be $\bar{x}$ and robot $a$'s location at the end of $m_k$ can be calculated as follows:

$$\begin{cases} (x-1, y) & \text{if } m_k = r \\ (x+1, y) & \text{if } m_k = l \\ (x, y-1) & \text{if } m_k = u \\ (x, y+1) & \text{if } m_k = d \end{cases}$$

*b) Definition of path:* As robot $r$ travels from $\bar{s}$ to $\bar{g}$, it passes through a sequence of locations $\{\bar{s}, \ldots, \bar{g}\}$. We call this sequence of locations $r$'s *path*.

*Lemma 1:* Given a workspace $W$, a command sequence $M$, and a robot $r$ traveling along a path beginning at location $\bar{s}$, this path can only be changed if $r$ hits a robot it did not hit before, or if $r$ fails to hit robots it did hit before.

*Proof:* By definition, the path is the sequence of locations occupied by the robot. The path is entirely determined by four factors: the starting location of the robot, the command sequence, obstacles in the workspace, and encounters with other robots already occupying free locations. If no hits with other robots are added or subtracted, then robot $r$ will start in the same place, receive the same command sequence and encounter the same obstacles, so robot $r$'s path will remain unchanged. Note that $r$'s path is still unchanged if it is hit by another robot. According to our definition fo a hit, if robot$q$ hits $r$, then $r$'s path is unchanged, while if robot $r$ hits $q$ then $r$'s path is changed. ■

Now we show that adding more unit sized robots cannot prevent a location form being occupied at the end of the command sequence.

*Theorem 2:* If given an workspace $W$ and a command sequence $M$ that moves a robot initially at $\bar{s}$ to a goal location $\bar{g}$, adding additional robots anywhere in $W$ at any stage of the command sequence cannot prevent $\bar{g}$ from being occupied at the conclusion of sequence $M$.

*Proof:*

Consider the effect of adding robot $b$ to workspace $W$. If $a$ never hits $b$, then by Lemma 1, $a$'s path remains the same. Therefore, at the conclusion of $M$, $a$ occupies $\bar{g}$.

Now suppose $a$ hits $b$. By the definition of a hit, $b$ prevents $a$ from reaching some location $\bar{x}$ because $b$ already occupies this location. After the hit, the command sequence will continue and so robot $b$ will continue on $a$'s original path, following the same instructions and therefore ending up in the same location, $\bar{g}$. By induction, we can clearly see that adding additional robots will have the same effect. If $b$ hits any other robot this robot will continue on the original path. Thus by adding more robots, it is impossible to prevent some robot from occupying $\bar{g}$ at the conclusion of $M$. ■

*Corollary 3:* A NOT gate without dual-rail inputs cannot be constructed

*Proof:* By contradiction. A particle logic NOT gate without dual-rail inputs has one input at $\bar{s}$, one output at $\bar{g}$, an arbitrary, possibly zero, number of asserted inputs which are

| | *Inputs* | | | *Outputs* | | |
|---|---|---|---|---|---|---|
| $A$ | $\overline{A}$ | 1 | $A$ | $A$ | $\overline{A}$ | $\overline{A}$ |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 1 | 0 | 0 |

TABLE I

FAN-OUT OPERATION. THIS CANNOT BE IMPLEMENTED WITH $1\times 1$ PARTICLES AND OBSTACLES. OUR TECHNIQUE USES $2\times 1$ PARTICLES.

| $A$ | $B$ | $A \vee B$ | $AB$ | $\overline{A \vee B}$ | $\overline{AB}$ | $A \oplus B$ | $\overline{A \oplus B}$ |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |

TABLE II

POSSIBLE BOOLEAN OPERATIONS IN DUAL-RAIL PARTICLE LOGIC.

all initially occupied, and an arbitrary, possibly zero, number of waste outputs.

In order for the NOT gate conditions to be satisified, given a command sequence $M$:

1) if $\bar{s}$ is initially unoccupied, $\bar{g}$ must be occupied at the conclusion of $M$
2) if $\bar{s}$ is initially occupied, $\bar{g}$ must be unoccupied at the conclusion of $M$

By Theorem 2, if $\bar{s}$ initially unoccupied results in $\bar{g}$ being occupied by soem robot $r$ at the conclusion of $M$, then the addition of a robot $q$ at $s$ cannot prevent $g$ from being filled, resulting in a contradiction. ∎

This shows that dual rail logic is necessary for the formation of NOT gates.

Additionally, we would like to show that $1 \times 1$ robots are insufficient to produce fan out gates. In order to do this we need to examine both the possibilities when we add additional robots to the scenario, as well as the possibilities when we remove robots.

*Theorem 4:* If given an workspace $W$ and a command sequence $M$ that moves two robots, $r_1$ and $r_2$, initially at $\bar{s}_1$ and $\bar{s}_2$ to respective goal locations $\bar{g}_1$ and $\bar{g}_2$, then removing one robot results in either $\bar{g}_1$ or $\bar{g}_2$ being occupied at the conclusion of $M$.

*Proof:* Without loss of generality, robot $r_1$ is removed. First suppose $r_2$ never hits $r_1$. Then the removal of $r_1$ will not affect the path of $r_2$. Robot $r_2$ has the same number of hits that it had before the removal of $r_1$ and so by Lemma 1, $r_2$ will follow the same path and occupy $\bar{g}_2$ at the conclusion.

Alternatively, suppose $r_2$ hit $r_1$ when $r_1$ was occupying location $\bar{x}$. Since $r_1$ is removed, it no longer occupies $\bar{x}$ during this move, and $\bar{x}$ becomes $r_2$'s new location. $r_2$ now proceeds along the path previously traveled by $r_1$. Effectively, $r_2$ has replaced $r_1$ and follows the path until it reaches $\bar{g}_1$. Successive hits between $r_2$ and $r_1$ in the original scenario are resolved in the same manner. ∎

*Corollary 5:* A conservative dual logic FAN-OUT gate cannot be constructed using only $1\times 1$ robots.

*Proof:* We assume such a FAN-OUT gate exists and reach a constradiction. Consider a FAN-OUT gate $W$, dual-rail input locations $\bar{s}_a$, $\bar{s}_{\bar{a}}$, and dual rail output locations $\{\bar{g}_{a_1}, \bar{g}_{a_2}, \bar{g}_{\bar{a}_1}, \bar{g}_{\bar{a}_2}\}$. Because $W$ is conservative there must also be one additional input location $\bar{s}_r$ and robot $r$. A FAN-OUT gate implements the truth table shown in Table I. Given an arbitrary command sequence $M$:

1) if $\bar{s}_a$ and $\bar{s}_r$ are initially occupied and $\bar{s}_{\bar{a}}$ vacant, at the conclusion of $M$ $\bar{g}_{a_1}$ and $\bar{g}_{a_2}$ are occupied and the locations $\bar{g}_{\bar{a}_1}$ and $\bar{g}_{\bar{a}_2}$ are vacant.
2) if $\bar{s}_a$ is initially vacant and $\bar{s}_{\bar{a}}$ and $\bar{s}_r$ are occupied, at the conclusion of $M$ $g_{a_1}$ and $g_{a_2}$ are vacant and the locations $g_{\bar{a}_1}$ and $g_{\bar{a}_2}$ are occupied.

We will now assume that condition 1, above, is the original scenerio and add and subtract robots, applying theorems 2 and 4, to show that it is impossible to meet condition 2.

Assume condition 1. Robots $a$ and $r$ start at $\bar{s}_a$ and $\bar{s}_r$ respectively and at the conclusion of $M$, the locations $\bar{g}_{a1}$ and $\bar{g}_{a2}$ are occupied. Now remove robot $a$. According to Thm. 4, either $\bar{g}_{a_1}$ or $\bar{g}_{a_2}$ must be occupied at the conclusion of $M$. Suppose without loss of generality that $\bar{g}_{a_1}$ is filled. By Thm. 2, adding an additional robot at location $s_{\bar{a}}$ cannot prevent $\bar{g}_{a_1}$ from being filled. However in order to meet condition 2, $\bar{g}_{a_1}$ must be vacant, thus no such gate is possible. ∎

## IV. DEVICE AND GATE DESIGN

This section describes how the clock signal, logic gates, and wiring were designed.

### A. Choosing a clock signal

The *clock sequence* is the ordered set of moves that are simultaneously applied to every particle in our workspace. We call this the clock sequence because, as in digital computers, this sequence is universally applied and keeps all logic synchronized.

A clock sequence determines the basic functionality of each gate. To simplify implementation in the spirit of Reduced Instruction Set Computing (RISC), which uses a simplified set of instructions that run at the same rate, we want to use the same clock cycle for each gate and for *all* wiring. Our early work used a standard sequence $\langle d, l, d, r\rangle$. This sequence can be used to make AND, OR, XOR, and any of their inverses. This sequence can also be used for *wiring* to connect arbitrary inputs and outputs, as long as the outputs are below the inputs. Unfortunately, $\langle d, l, d, r\rangle$ cannot move any particles upwards. To connect outputs as inputs to higher level logic requires an additional reset sequence that contains an $\langle u\rangle$ command. Including all four directions is a necessary condition for a valid clock sequence. We choose the simplest such sequence, $\langle d, l, u, r\rangle$, and by designing examples prove that this sequence is sufficient for logic gates, FAN-OUT gates, and wiring.

This clock sequence has the attractive property of being a clockwise rotation through the possible input sequences.

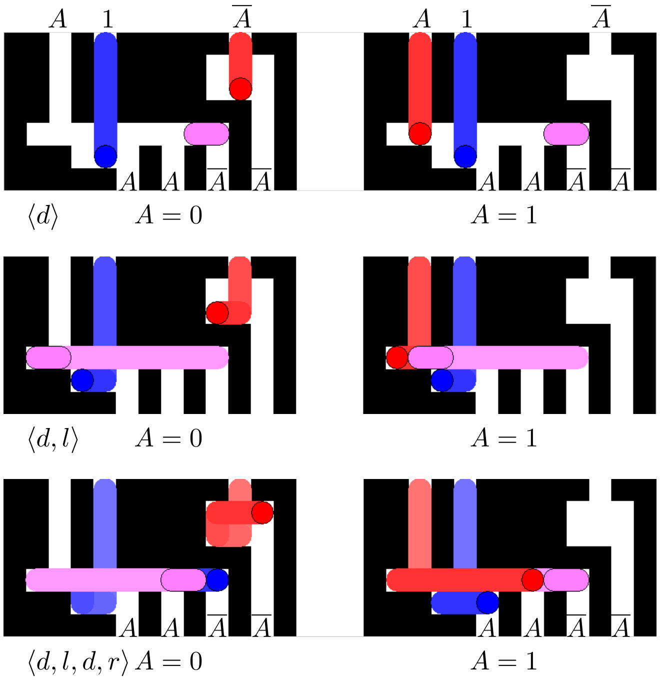


Fig. 2. A single input, two-output FAN-OUT gate. This gate requires a dual-rail input, a supply particle, and a $2 \times 1$ slider. The *clockwise* control sequence $\langle d, l, u, r \rangle$ duplicates the dual-rail input.

One could imagine our particle logic circuit mounted on a wheel rotating perpendicular to the ground. If the particles were moved by the pull of gravity, each counter-clockwise revolution would advance the circuit by one clock cycle.

### B. A FAN-OUT gate

A FAN-OUT gate with two outputs implements the truth table in Table I. This cannot be implemented with $1 \times 1$ particles and obstacles, by corollary 5. Our technique uses $2 \times 1$ particles. A single input, two-output FAN-OUT gate is shown in Fig 2. This gate requires a dual-rail input, a supply particle, and a $2 \times 1$ slider. The *clockwise* control sequence $\langle d, l, u, r \rangle$ duplicates the dual-rail input.

The FAN-OUT gate can drive multiple outputs. In Fig. 3 a single input drives four-outputs. This gate requires a dual-rail input, three supply particles, and a $2 \times 1$ slider. The *clockwise* control sequence $\langle d, l, u, r \rangle$ quadruples the dual-rail input. In general, an $n$ output FAN-OUT gate with control sequence $\langle d, l, u, r \rangle$ requires a dual-rail input, $n - 1$ supply particles, and one $2 \times 1$ slider. It requires $4(n+1) \times 2(n+1)$ area.

### C. Data Storage

A general-purpose computer must be able to store data. A $2 \times 1$ particle enables us to construct a read/writable data storage for one bit. A single-bit data storage latch is shown in Fig. 4 and implements the truth table in Table III. By combining an $n$-out FAN-OUT gate shown in Fig 3 with $n$ data storage devices, we can implement an $n$ bit memory. To maintain *conservative* properties of the computer, i.e. the same number of robots enter and leave each gate, single-bit data storage latches must be used in pairs to record the state and its inverse.

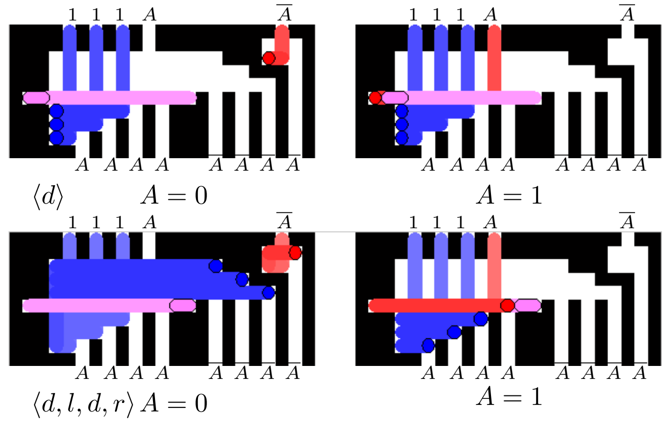


Fig. 3. The FAN-OUT gate can drive multiple outputs. Here a single input drives four-outputs. This gate requires a dual-rail input, three supply particles, and a $2 \times 1$ slider. The *clockwise* control sequence $\langle d, l, u, r \rangle$ quadruples the dual-rail input.

| $Q$ | $R$ | $S$ | $C$ | $Q$ | $Q_R$ | $W_1$ | $W_2$ | $\overline{Q}_R$ |
|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 |

TABLE III

A SINGLE-BIT DATA STORAGE LATCH WITH STATE $Q$.

### D. A binary counter

Using the FAN-OUT gate from Section IV-B we can generate arbitrary Boolean logic. The half-adder from Fig. 1 requires a single FAN-OUT gate.

We illustrate how many gates can be combined by constructing a binary counter, shown in Fig. 5. Six logic gates are used to implement a 3-bit counter. A block diagram of the device is shown in Fig. 6, and Fig. shows the results of each computation stage. The counter requires three FAN-OUT gates, two summers, and one carry. Six $1 \times 1$ particles and three $2 \times 1$ particles are used. The counter has three levels of gates $\langle d, l, d, r \rangle$ and requires three interconnection moves $\langle d, l, d, r \rangle$, for a total of 24 moves. Figure 7 shows the ending configuration for each iteration of the counter.

### E. Scaling issues

Particle computation requires multiple clock cycles, workspace area for gates and interconnections, and many particles. In this section we analyze how these scale with the size of the counter, using Fig. 6 as a reference.

*c) gates:* an $n$-bit counter requires $n$ FAN-OUT gates, $n - 1$ summers (XOR) gates, and $n - 2$ carry (AND) gates.

*d) particles:* we require $n$ $1 \times 1$ particles, one for each bit and $n$ $2 \times 1$ particles, one for each FAN-OUT gate.

*e) propagation delay:* the counter requires $n$ stages of logic, and $n$ corresponding wiring stages. Each stage requires a complete clock cycle $\langle d, l, u, r \rangle$ for a total of $8n$ moves.

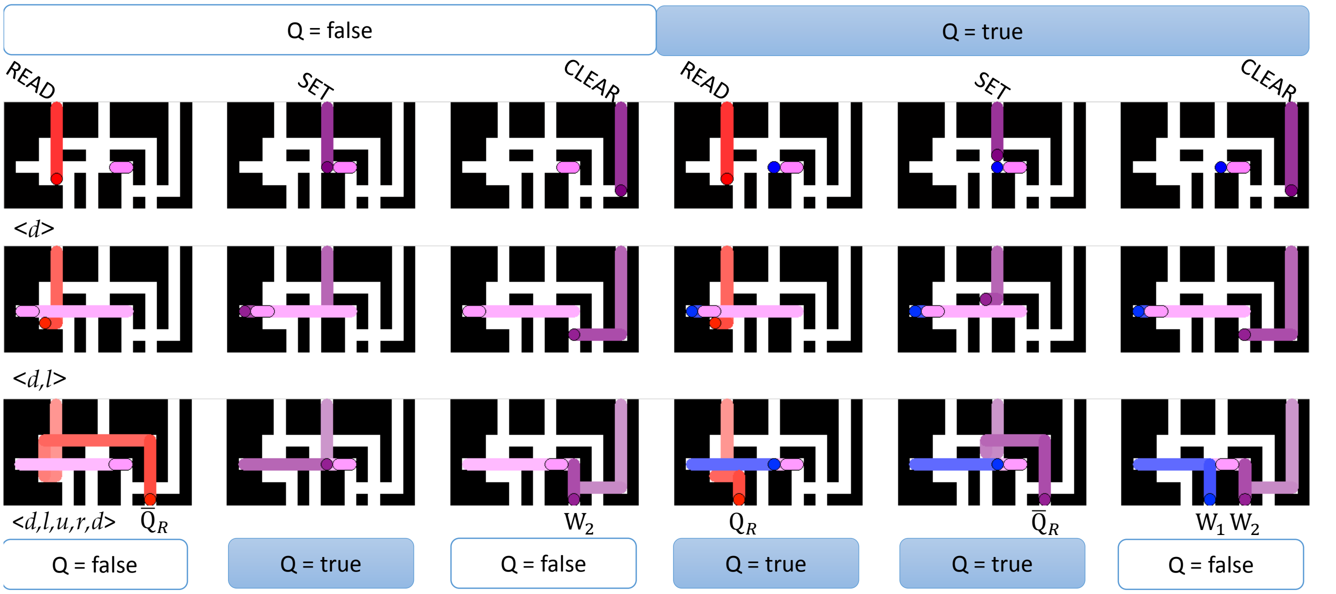


Fig. 4. A flip-flop memory. This device has three inputs, *Read*, *Set*, *Clear*, a state variable (shown in blue), and a $2 \times 1$ slider. Depending on which input is active, the control sequence $\langle d, l, d, r\rangle$ will read, set, or clear the memory.

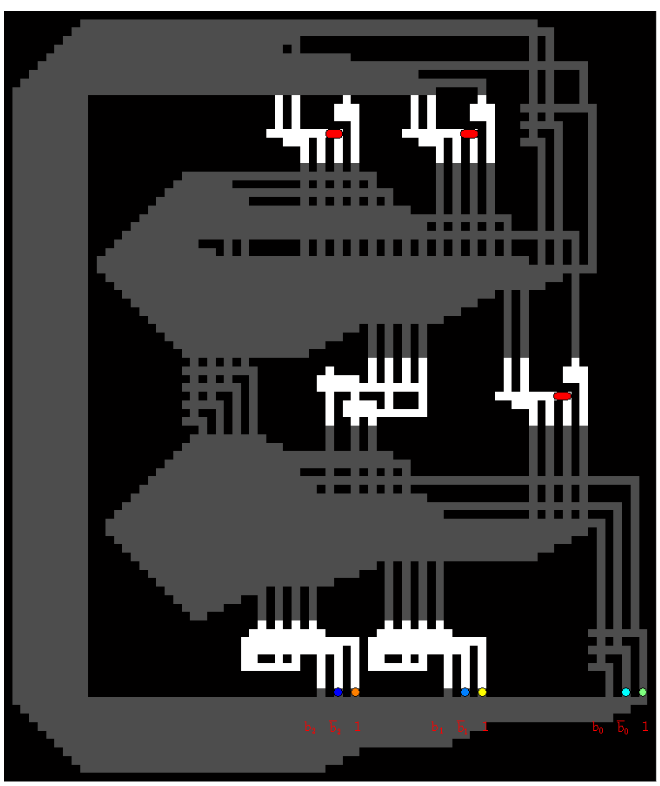

Fig. 5. A three-bit counter implemented with particles. The counter requires three FAN-OUT gates, two summers, and one carry. Six $1 \times 1$ particles and three $2 \times 1$ particles are used. The counter has three levels of gates actuated by CW sequence $\langle d, l, u, r\rangle$ and requires three interconnection sequences $\langle d, l, u, r\rangle$, for a total of 24 moves.

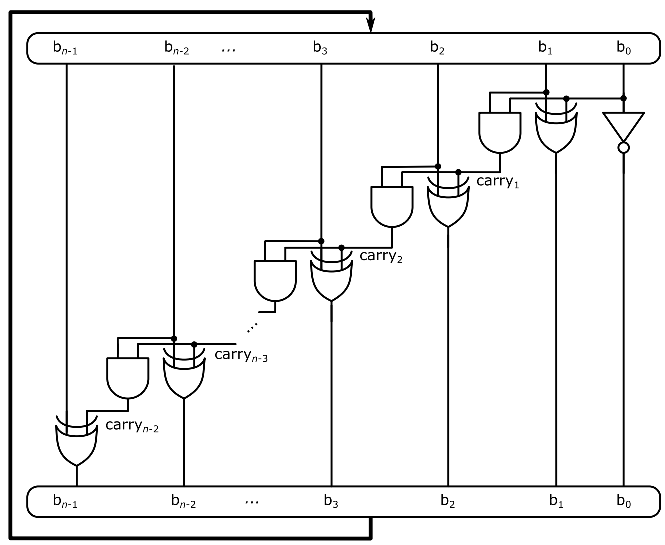


Fig. 6. Gate-level diagram for an $n$-bit counter. The counter requires $n-1$ XOR gates, $n-2$ AND gates, and 1 NOT gate.

These are comparable to a ripple-carry adder: the delay for $n$ bits is and requires $x =$ gates. Numerous other schemes exist to speed up the computation. The advantage of gates is that they are easily reused and connected. If speed was critical, instead of using discrete gates, we could engineer the workspace to directly compute logic.

### F. Optimal Wiring schemes

With our current CW clock cycle, we cannot have outputs at the same column as inputs – outputs must be either one to the right, or three to the left. Choosing one of these results in shifts horizontally at each stage and thus requires spreading out the logic gates. A better wiring scheme would cycle through three layers that each shift right one, followed by one layer that shifts left three. We also want the wiring to be tight left-to-right. If our height is also limited, *wire buses* provide a compact solution.

## V. Conclusion

In this paper we... (copy over from intro)

## References


[1] B. R. Donald, C. G. Levey, I. Paprotny, and D. Rus, "Planning and control for microassembly of structures composed of stress-engineered mems microrobots," *The International Journal of Robotics Research*, vol. 32, no. 2, pp. 218–246, 2013. [Online]. Available: http://ijr.sagepub.com/content/32/2/218.abstract

[2] P.-T. Chiang, J. Mielke, J. Godoy, J. M. Guerrero, L. B. Alemany, C. J. Villagómez, A. Saywell, L. Grill, and J. M. Tour, "Toward a light-driven motorized nanocar: Synthesis and initial imaging of single molecules," *ACS Nano*, vol. 6, no. 1, pp. 592–597, Feb. 2011.

[3] H.-W. Tung, D. R. Frutiger, S. Panè, and B. J. Nelson, "Polymer-based wireless resonant magnetic microrobots," in *IEEE International Conference on Robotics and Automation*, May 2012, pp. 715–720.

[4] E. Diller, J. Giltinan, and M. Sitti, "Independent control of multiple magnetic microrobots in three dimensions," *The International Journal of Robotics Research*, vol. 32, no. 5, pp. 614–631, 2013. [Online]. Available: http://ijr.sagepub.com/content/32/5/614.abstract

[5] W. Jing, N. Pagano, and D. Cappelleri, "A tumbling magnetic microrobot with flexible operating modes," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, May 2013, pp. 5514–5519.

[6] Y. Ou, D. H. Kim, P. Kim, M. J. Kim, and A. A. Julius, "Motion control of magnetized tetrahymena pyriformis cells by magnetic field with model predictive control," *Int. J. Rob. Res.*, vol. 32, no. 1, pp. 129–139, Jan. 2013.

[7] D. de Lanauze, O. Felfoul, J.-P. Turcot, M. Mohammadi, and S. Martel, "Three-dimensional remote aggregation and steering of magnetotactic bacteria microrobots for drug delivery applications," *The International Journal of Robotics Research*, 11 2013. [Online]. Available: http://ijr.sagepub.com/content/early/2013/11/11/0278364913500543

[8] A. Becker, E. Demaine, S. Fekete, G. Habibi, and J. McLurkin, "Reconfiguring massive particle swarms with limited, global control," in *International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics (ALGOSENSORS)*, Sep. 2013.

[9] A. Becker, E. Demaine, S. Fekete, and J. McLurkin, "Particle computation: Controlling robot swarms with only global signals," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[10] A. Becker, E. D. Demaine, S. P. Fekete, G. Habibi, and J. McLurkin, "Reconfiguring massive particle swarms with limited, global control," in *Algorithms for Sensor Systems*, ser. Lecture Notes in Computer Science, P. Flocchini, J. Gao, E. Kranakis, and F. Meyer auf der Heide, Eds. Springer Berlin Heidelberg, 2014, pp. 51–66. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-45346-5_5

[11] S. McCourtney, *ENIAC, the triumphs and tragedies of the world's first computer*. United States of America: Walker Publishing, 1999.

[12] A. Adamatzky and J. Durand-Lose, "Collision-based computing," in *Handbook of Natural Computing*, G. Rozenberg, T. Bäck, and J. Kok, Eds. Springer Berlin Heidelberg, 2012, pp. 1949–1978. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-92910-9_58

[13] E. Fredkin and T. Toffoli, "Conservative logic," *International Journal of Theoretical Physics*, vol. 21, no. 3-4, pp. 219–253, 1982. [Online]. Available: http://dx.doi.org/10.1007/BF01857727

[14] E. R. Berlekamp, J. H. Conway, and R. K. Guy, *Winning Ways for your Mathematical Plays, 2nd edition*. A. K. Peters Ltd., 2001–2004.

[15] A. Adamatzky and P. Rendell, *Turing Universality of the Game of Life*. Springer London, 2002, pp. 513–539. [Online]. Available: http://dx.doi.org/10.1007/978-1-4471-0129-1_18

[16] R. A. Hearn, "The complexity of sliding-block puzzles and plank puzzles," *Tribute to a Mathemagician*, pp. 173–183, 2005.

[17] E. D. Demaine and R. A. Hearn, *Games of No Chance 3*. Mathematical Sciences Research Institute Publications, Cambridge University Press, 2009, vol. 56, ch. Playing Games with Algorithms: Algorithmic Combinatorial Game Theory, pp. 3–56. [Online]. Available: http://arXiv.org/abs/cs.CC/0106019

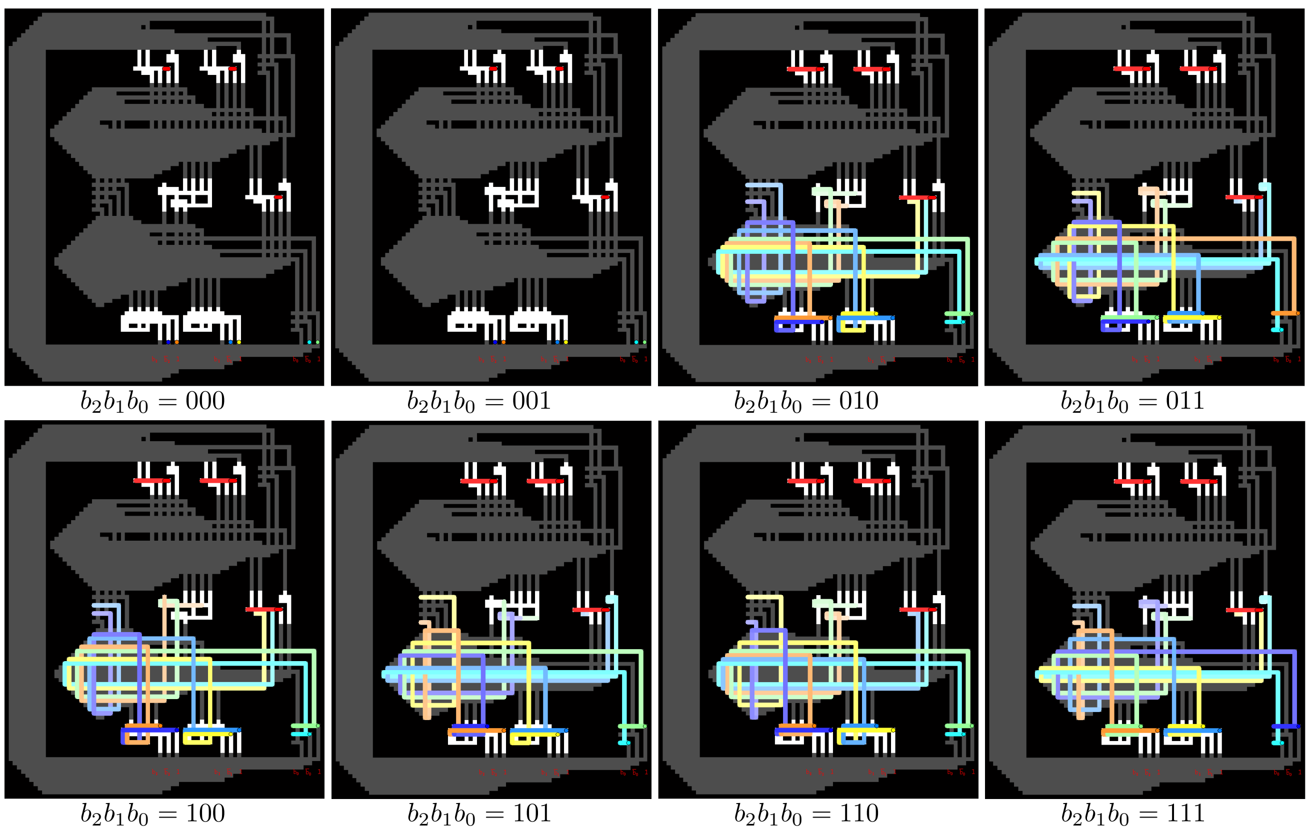


Fig. 7. Ending configuration for each stage of the computation.