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, OR, NOT, NOR, NAND, XOR, XNOR **logic gates. We could not implement a** FAN-OUT **gate because such a gate depends on particle priority, and our particles are indistinguishable. This prevented us from creating arbitrary digital circuits. In this work we introduce 2×1 robots. We now 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.**

```
Add section on optimal wiring schemes --
with our current CW clock cycle, we
cannot have outputs at the same column
as inputs -- they must be either 1 to
the right, or 3 to the left. Choosing
one of these results in logic that
shifts horizontally at each stage and
spreads out the logic. A better wiring
scheme would cycle through 3 layers that
go right 1, followed by one layer that
goes left 3. We also want the wiring to
be tight left-to-right. What if our
height is also limited? In such a case
wire buses would be a compact solution.
```

I. Introduction

Currently, micro- and nanorobot systems with many robots are steered and directed by a common control signal [?]. 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, being both slow, requiring large amounts of space, and being vulnerable to manufacturing defects. Rather, we want to 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. This paper builds on the techniques for controlling many simple robots with uniform control inputs presented in [1]–[3], using the following rules:

1) A planar grid is filled with some unit-square robots (each occupying one cell of the grid) and some fixed unit-square blocks.
2) All robots are commanded in unison: the valid commands are "Go Up" (u), "Go Right" (r), "Go Down" (d), or "Go Left" (l). The robots all move in the commanded direction until they hit an obstacle or a stationary robot. A representative command sequence is $\langle u, r, d, l, d, r, u, \ldots \rangle$. We assume we know the maximum dimension of the workspace and issue each command long enough for the robots to reach their maximum extent.

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 (give section)
2) prove the insufficiency of unit-size particles for gate fan-out (give section)
3) design FAN-OUT GATES (give section)
4) design memory latches (Section IV-B)
5) present architecture for device integration, design a common clock sequence, and present a binary counter (Section IV-C)

A. Fan-out

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 [2], 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 **A** and **B** are needed to compute both the SUM and the CARRY bits, so the fanout of **A** and **B** is two.

AND and OR can be implemented with particles, but other logic requires *dual-rail logic*, where two lines for each input are supplied to explicitly represent the variable and its complement. Dual-rail logic is required because particle logic is *conservative*—particles are neither created nor destroyed. Unfortunately particle logic is not reversible. The OR gate outputs a 1 if either input is asserted, but afterwards we cannot determine which input was high. This makes implementing a FAN-OUT gate impossible with unit size particles and obstacles.

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 is with the 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.

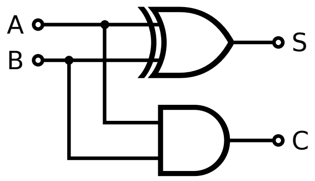


Fig. 1. The half adder shown above requires two copies of $\mathbf{A}$ and $\mathbf{B}$.

B. *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. Our early work used a standard sequence $\langle d, l, d, r\rangle$. This sequence can be used to make AND, OR, XOR, 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 such as $\langle d, l, u, r\rangle$. 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. A clock sequence without orthogonal inputs, i.e. $\langle u, d, u, d\rangle$ can only implement assertions. Using two orthogonal inputs, i.e. $\langle d, l, d, l\rangle$ cannot implement FAN-OUT gates, and particles cannot be returned u or r. Therefore, 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 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. 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.

II. Related Work

Our efforts have similarities with *mechanical computers*, computers constructed from mechanical, not electrical components. For a fascinating nontechnical review, see [4]. 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 GRIDWORLD implementations seem an anachronism, note that we require none of these precision elements—merely unit-size obstacles, and 2×1 and 1×1 sliding particles.

A. *Collision-based computing*

```
refers to.... For a survey of this area,
so the collection ....
```

One example is Conway's game of life (cite original) these simple rules have been examined in depth and used to build a Turing-complete computer [5]. These scenarios are fascinating, but lack a physical implementation. They require *cells* that live or die based on the number of neighbors.

B. *Sliding-block puzzles*

Sliding block puzzles use rectangular tiles that are constrained to move in a 2D environment. The objective is to move one or more tiles to desired locations. They have a long history. Hearn [6] and Demaine [7] 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

Theorem 1: If given an environment E and a command sequence M that moves a robot initially at s to a goal location g, then adding additional robots anywhere in E at any stage of the command sequence cannot prevent g from being occupied at the conclusion of sequence M.

a) Definition of hit: During command m_i, robot a *hits* robot b if a's movement is stopped by b. A hit changes a's location at the end of move step m_i to be b-m_i.

Robots must hit in to change the outcome of a command sequence. If marbles do not hit, there is no change because the robots have not interacted.

Lemma 2: If two robots hit, one robot follows the original path

```
need to specify path and original path.
Maybe: ``one robot follows the path
either robot would have followed if the
other did not exist" or If robots a and
b hit, a robot will still occupy m...
```

Proof: In any hit either b hits a or a hits b. If b hits a, there is no change to a's path and a will still end at g. If a hits b, b is where a would have been and b follows a's original path. ■

Robots can only interact via hits. By lemma 2, a hit cannot prevent an occupancy, therefore the proof of 1 follows.

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 I
POSSIBLE BOOLEAN OPERATIONS IN DUAL-RAIL PARTICLE LOGIC.

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 II
FAN-OUT OPERATION. THIS CANNOT BE IMPLEMENTED WITH 1×1 PARTICLES AND OBSTACLES. OUR TECHNIQUE USES 2×1 PARTICLES.

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

Proof: define a NOT gate,

explain how this follows

■

Theorem 4: If given an environment E and a command sequence M that moves robots initially at s_1 and s_2 to respective goal locations g_1 and g_2, then removing either robot results in either g_1 or g_2 being occupied at the conclusion of M.

Proof: If robots s_1 and s_2 never hit when both exist, then the remaining robot continues to its goal location unchanged.

If robots do hit when both exist

show that this cannot lead to a goal location g_3, when only one robot is used, it goes to the goal location occupied by the last robot to be hit when there are two.

■

Corollary 5: A FAN-OUT gate cannot be constructed using only 1×1 robots.

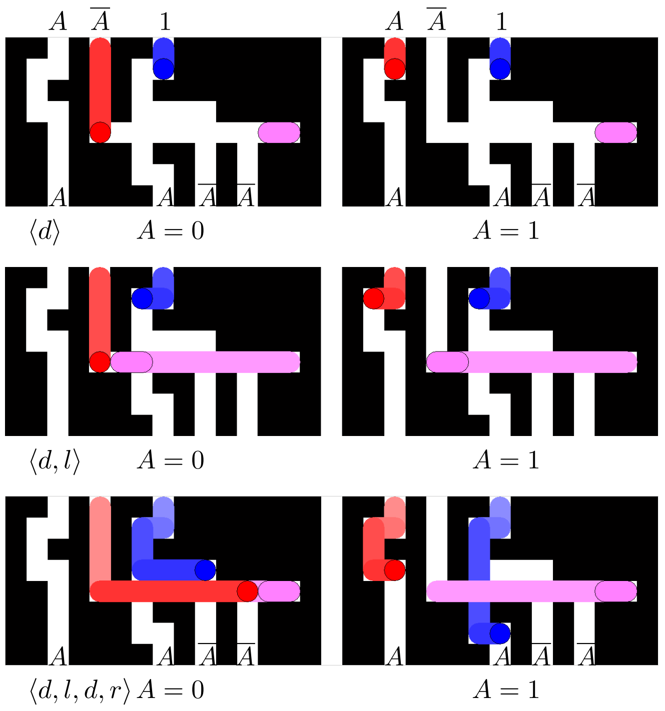
Proof: By contradiction. Consider a FAN-OUT gate E and an arbitrary sequence M such that with input s_a initially occupied and one or more supply robots, at the conclusion of M, the output location g_{a1} and g_{a2} are occupied and the locations $g_{\bar{a}1}$ and $g_{\bar{a}2}$ are vacant. Also, under the same command sequence M, but with input s_a initially vacant, $s_{\bar{a}}$ initially occupied and the same supply robots, at the conclusion of M, the output locations $g_{\bar{a}1}$ and $g_{\bar{a}2}$ are occupied and the locations g_{a1} and g_{a2} are vacant.

If input s_a is initially vacant, by Thm. 4 , either g_{a1} and g_{a2} is occupied at the conclusion of M. By Thm. 1, adding an additional robot at location $s_{\bar{a}}$ cannot prevent one of g_{a1} and g_{a2} being filled, thus arriving at a contradiction. ■

IV. DEVICE AND GATE DESIGN

A. A FAN-OUT *gate*

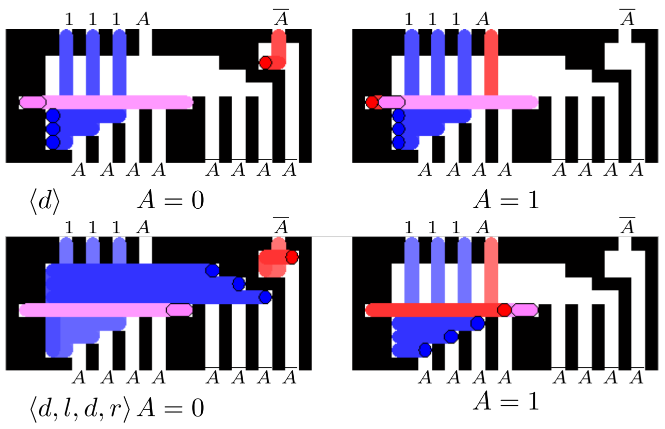
A FAN-OUT gate with two outputs implements the truth table in Table II. Our implementation is shown in Fig 2.



A $\overline{A}$ 1
A A $\overline{A}$ $\overline{A}$
$\langle d\rangle$ $A=0$
$A=1$
$\langle d,l\rangle$ $A=0$
$A=1$
$\langle d,l,d,r\rangle$ $A=0$
$A=1$

replace image with CW gate

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



1 1 1 A $\overline{A}$
A A A A $\overline{A}$ $\overline{A}$ $\overline{A}$ $\overline{A}$
$\langle d\rangle$ $A=0$
$A=1$
$\langle d,l,d,r\rangle$ $A=0$
$A=1$

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 particle, and a 2×1 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.

B. *Data Storage*

A general-purpose computer must be able to store data. A 2×1 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.

C. *A binary counter*

Using the FAN-OUT gate from Section IV-A 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 1×1 particles and three 2×1 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.

D. *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.

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

c) particles: we require n 1×1 particles, one for each bit and n 2×1 particles, one for each FAN-OUT gate.

d) 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.

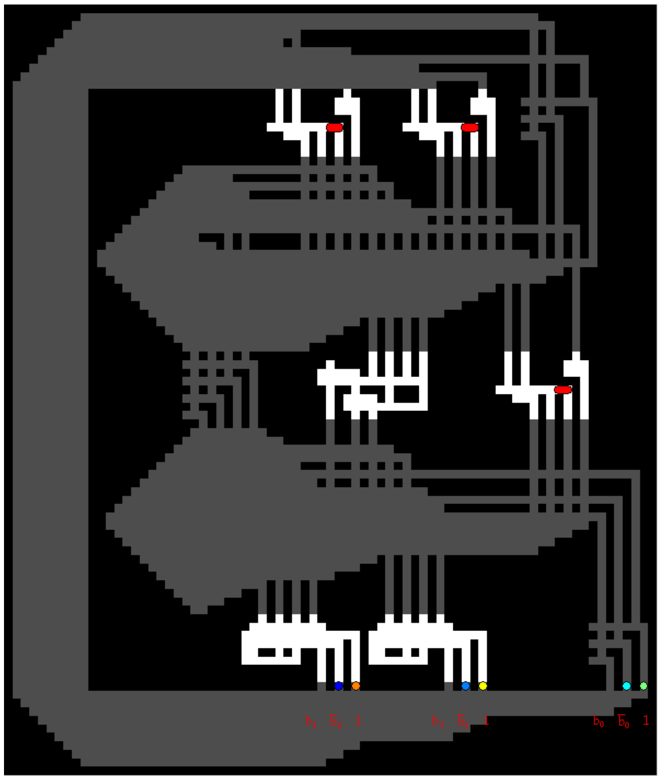


Fig. 5. A three-bit counter implemented with particles. The counter requires three FAN-OUT gates, two summers, and one carry. Six 1×1 particles and three 2×1 particles are used. The counter has three levels of gates actuated by CW sequence $\langle d, l, u, r\rangle$ and requires three interconnection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

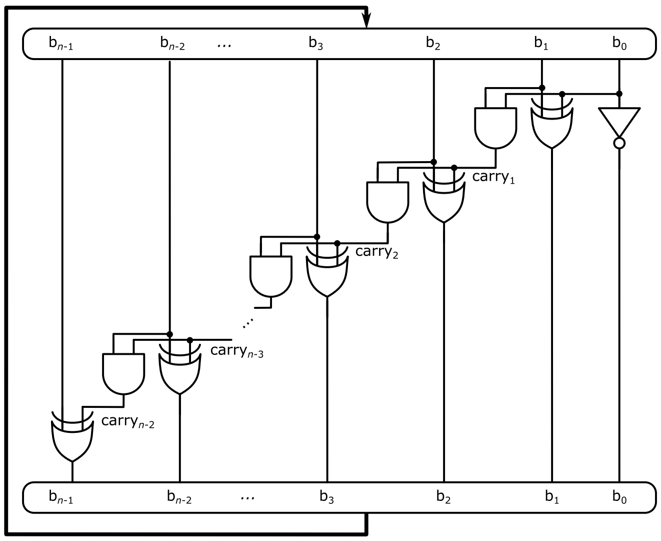


Fig. 6. Gate-level diagram for an n-bit counter. The counter requires $n-1$ XOR gates, $n-2$ AND gates, and 1 NOT gate.

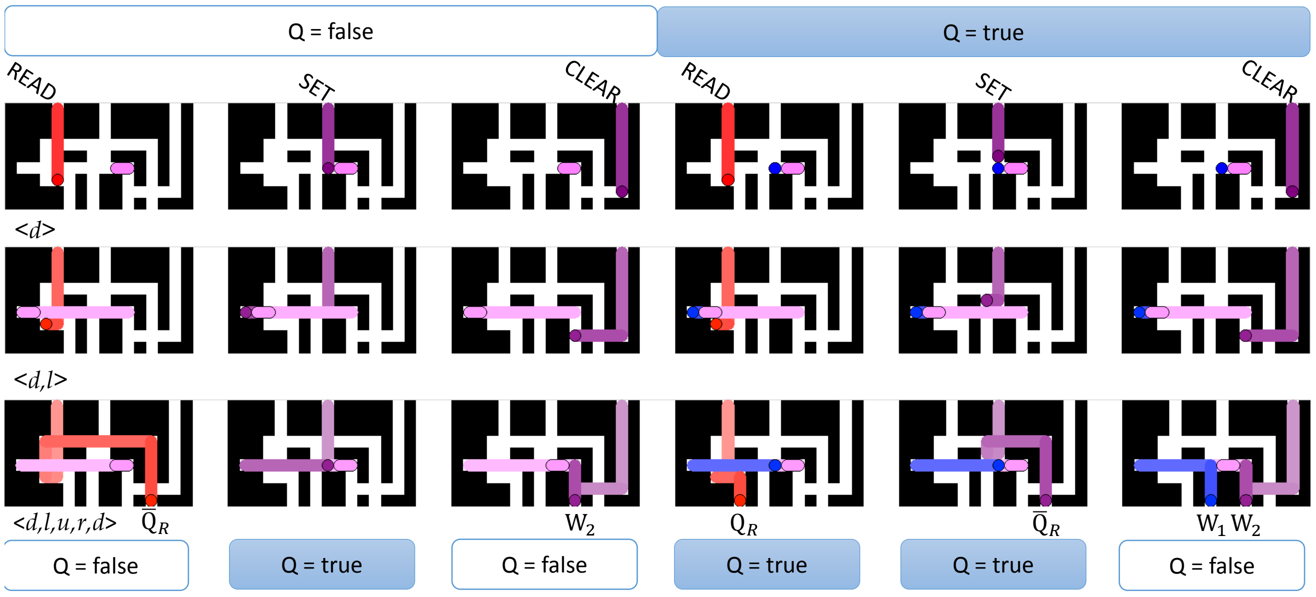


Fig. 4. A flip-flop memory. This device has three inputs, *Read*, *Set*, *Clear*, a state variable (shown in blue), and a 2×1 slider. Depending on which input is active, the control sequence $\langle d, l, d, r \rangle$ will read, set, or clear the memory.

`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. Instead of using discrete gates, we could engineer environment to directly compute each stage. The advantage of gates is that they are easily reused and connected.`

References

[1] 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.

[2] 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.

[3] 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

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

[5] 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

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

[7] 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

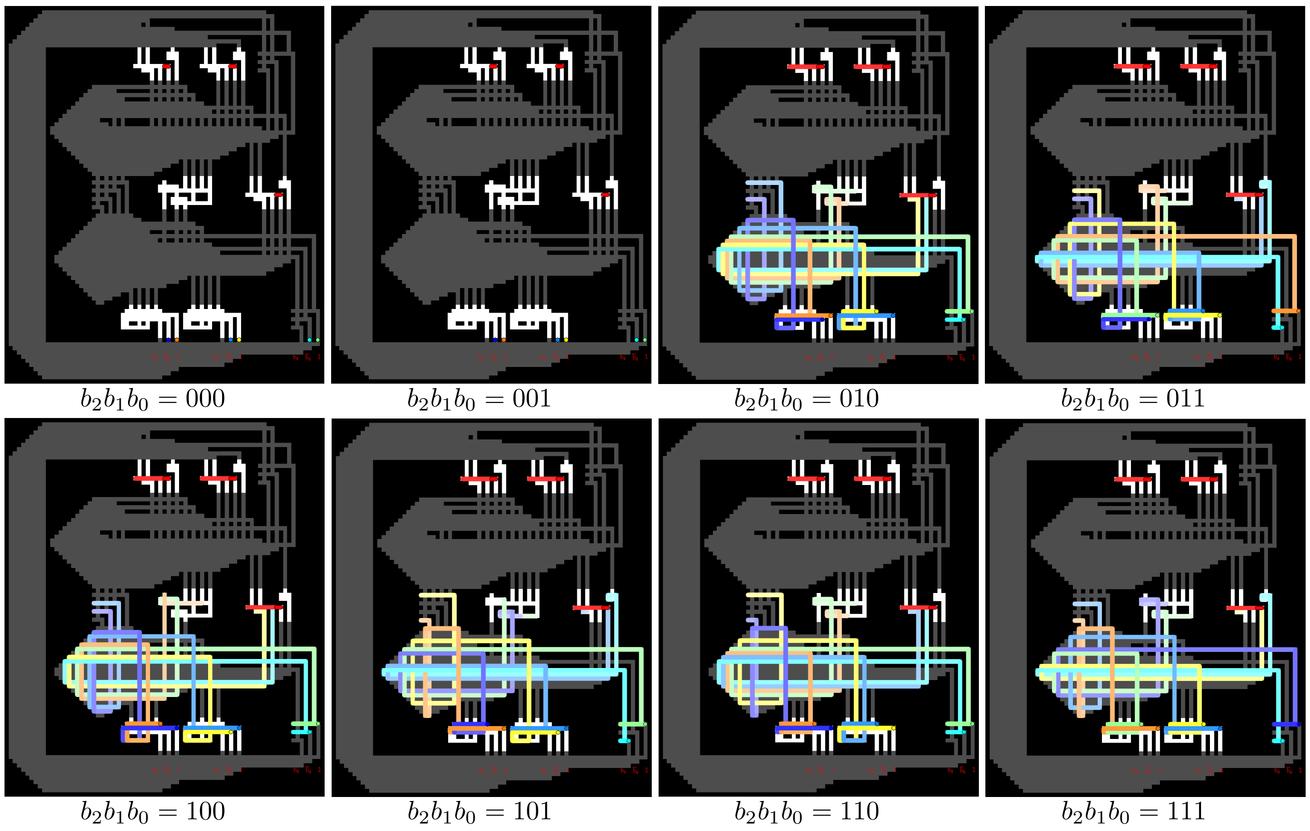


Fig. 7. Ending configuration for each stage of the computation.