Asynchronous logic automata

David A. Dalrymple, Neil A. Gershenfeld, and Kailiang Chen

Massachusetts Institute of Technology

Abstract. We present a class of device, termed Asynchronous Logic Automata (ALA), for practical reconfigurable computing that may be categorized as an asynchronous lattice gas automaton, as a Petri net, as a field-programmable gate array, and as charge-domain logic. ALA combine the best features of each: locality, consistent asynchronous behavior, easy reconfiguration, and charge-carrier conservation. ALA are thus "closer to physics" (at least, the physics used in classical computation) than classical computers or gate arrays with global interconnect and clocking, and may therefore be physically implemented with more desirable properties such as speed, power, and heat dissipation.

1 Introduction

Physics, above the atomic level, is inherently local, and computation, like every other process, relies on physics. Thus, programming models which assume non-local processes, such as data buses, random access memory, and global clocking, must be implemented at a slow enough speed to allow local interactions to simulate the non-local effects which are assumed. Since such models do not take physical locality into account, even local effects are limited to the speed of the false non-local effects, by a global clock which regulates all operations.

In computing today, many observers agree that there is a practical physical speed limit for the venerable von Neumann model (see for instance [1]), and that the bulk of future speed increases will derive from parallelism in some form. Chipmakers are currently working to pack as many processors as they can into one box to achieve this parallelism, but in doing so, they are moving even further from the locality that is necessary for a direct implementation as physics. At the other end of the abstraction spectrum, while sequential programming models can be generalized to use multiple parallel threads, such models are often clumsy and do not reflect the physical location of the threads relative to each other or memory.

In addition, research has long suggested that asynchronous (or "self-timed") devices consume less power and dissipate less heat than typical clocked devices [2]. However, traditional microarchitectures require significant book-keeping overhead to synchronize various functional blocks, due to the nature of their instructions, which must be executed in sequence. Most asynchronous designs to present have derived their performance benefits from clever pipelining and power distribution rather than true asynchrony – known as "globally asynchronous, locally synchronous" design – and often this is not enough to offset the overhead [3].

These shortcomings are accepted because of the tremendous body of existing code written in sequential fashion, which is expected to run on the latest hardware. However, by removing the assumption of backwards compatibility, there is an opportunity to create a new, disruptive programming model which is more efficient to physically implement. In particular, such a model could scale favorably to an arbitrary number of parallel elements, to larger problem sizes, and to faster, smaller process technologies. Potentially, this may have eventual impact across the computing industry, particularly in high-performance computing. In addition, it could conceivably be an enabling technology for the Singularity (see [4]).

In Sect. 2, we describe the Logic CA, a synchronous cellular automaton which is the basis of the ALA. In Sect. 3, we introduce the ALA as a modification of the Logic CA. In Sect. 4, we discuss the relationship to past work, and in Sect. 5 we identify future work.

2 Logic CA

Asynchronous Logic Automata (ALA) are based on an earlier model, a cellular automaton (CA), known as the Logic CA. It may be convenient to understand first the Logic CA, which has closer ties to previous work (e.g. [5]), particularly if the reader is familiar with these types of constructions.

The Logic CA consists of cells with 8 neighbors and 9 bits of state. The state bits are divided into 8 configuration bits and 1 dynamic state bit. The configuration bits are further divided into 2 gate bits which choose among the four allowed Boolean functions ($\{AND$, OR, XOR, $NAND\}$) and 6 input bits which choose among the 36 possible pairs of (potentially identical) inputs chosen from the 8 neighbors ($\frac{1}{2}\cdot 8\cdot(8-1)+8$). At each time step, a cell examines the dynamic state bit of its selected inputs, performs the selected Boolean operation on these inputs, and sets its own dynamic state to the result.

Mathematically, an instance of the Logic CA can be described as a series of global states S_t ($t \in \mathbb{N}_0$) each composed of local states $s_{t,(i,j)} \in \{0,1\}$ ($i,j \in \mathbb{Z}$) and a set of constant configuration elements

$$\begin{aligned} c_{(i,j)} \in \mathcal{C} &= \left(\{AND, OR, XOR, NAND\} \times (\{-1,0,1\}^2 - \{(0,0)\})^2\right) \\ &= \{AND, OR, XOR, NAND\} \\ &\quad \times \{(1,0),(1,1),(0,1),(-1,1),(-1,0),(-1,-1),(0,-1),(1,-1)\} \\ &\quad \times \{(1,0),(1,1),(0,1),(-1,1),(-1,0),(-1,-1),(0,-1),(1,-1)\} \end{aligned}$$

(note that there is a bijection between $\mathcal{C}$ and $\{0,1\}^8$, 8 bits) such that

$$s_{t+1,i,j} = \begin{cases} \text{if } (c_{(i,j)})_1 = AND & s_{t,(i,j)+(c_{(i,j)})_2} \wedge s_{t,(i,j)+(c_{(i,j)})_3} \\ \text{if } (c_{(i,j)})_1 = OR & s_{t,(i,j)+(c_{(i,j)})_2} \vee s_{t,(i,j)+(c_{(i,j)})_3} \\ \text{if } (c_{(i,j)})_1 = XOR & s_{t,(i,j)+(c_{(i,j)})_2} \oplus s_{t,(i,j)+(c_{(i,j)})_3} \\ \text{if } (c_{(i,j)})_1 = NAND & \neg(s_{t,(i,j)+(c_{(i,j)})_2} \wedge s_{t,(i,j)+(c_{(i,j)})_3}) \end{cases}$$

Although the Logic CA is useful for many applications, we identified two major problems with it, leading to the development of ALA:

1. **Lack of Reversible/Adiabatic Logic.** The system does not employ conservative logic [6] or adiabatic computing [7], which is necessary to truly represent physical resources.
2. **Global Clock.** The clock is global – clearly a non-local effect. Cellular automata are not fundamentally required to have a global clock to perform universal computation [8, 9].

3 Asynchronous logic automata

We have discovered a new approach, inspired by both lattice-gas theory [10] and Petri net theory [11], that resolves the above problems.

By "lattice gas" we mean a model similar to cellular automata in which the cells communicate by means of particles with velocity as opposed to broadcasted states. Practically, this means that the information transmitted by a cell to each of its neighbors is independent in a lattice gas, where in a cellular automaton these transmissions are identical. By convention, a lattice gas also has certain symmetries and conservation properties that intuitively approximate an ideal gas [12], and in some cases, numerically approximate an ideal gas [13].

Meanwhile, Petri nets are a broad and complex theory; we are primarily concerned with the subclass known as "marked graphs" (a detailed

explanation can be found in [14]). In short, a marked graph is a graph whose edges can be occupied at any given time by zero or more tokens. According to certain conditions on the tokens in edges neighboring a node of the graph, the node may be allowed to "fire" (at any time as long as the conditions are met), by performing some operations on the tokens (such as moving a token from one of its edges to another or simply consuming a token from an edge).

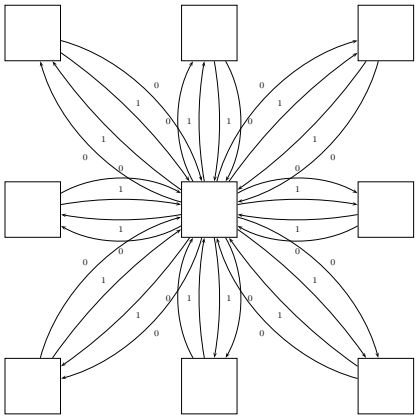


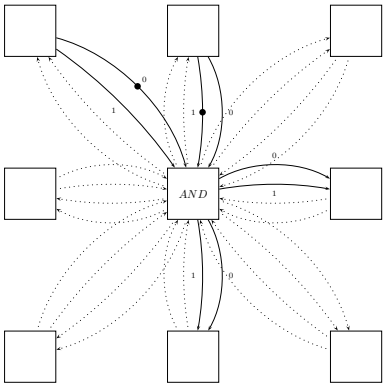
Fig. 1. Edges of one cell in the new approach

Our new approach merges these with the existing Logic CA as follows. We remove the global clock and the bit of dynamic state in each cell, and replace the neighborhood broadcasts with a set of four edges between neighboring cells, each containing zero or one tokens, thus comprising a bit of state (see Fig. 1). Between each pair of cells, in each direction, we have a pair of edges, one to represent a "0" signal, and the other a "1" signal. Note that each pair of edges could be considered one edge which can carry a "0" token or a "1" token. Instead of each cell being configured to read the appropriate inputs, this data is now represented by an "active" bit in each edge. Then, each cell becomes a stateless node (except the gate type) in this graph, which can fire on the conditions that all its active inputs are providing either a "0" token or a "1" token and that none of its active output edges is currently occupied by a token of either type. When firing, it consumes the input tokens, removing them from the input edges, performs its configured function, and deposits the result to the appropriate output edges (see Fig. 2 for an example of a 2-

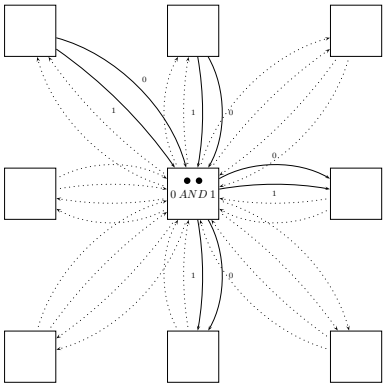
input, 2-output AND gate firing). As it is a marked graph, the behavior of this model is well-defined even without any assumptions regarding the timing of the computations, except that each computation will fire in some finite length of time after the preconditions are met. The model now operates asynchronously, and removes the need not only for a global clock, but any clock at all.

We have also introduced explicit accounting for the creation and destruction of tokens instead of implicitly doing both in every operation, as with traditional CMOS logic. For instance, in Fig. 2, since there are equally many inputs and outputs, no tokens must be created or destroyed. While the model still uses the same irreversible Boolean functions, these functions can be thought of as being simulated by conservative logic which is taking in constants and dispersing garbage [6], enabling an easy pricing of the cost of non-conservatism in any given configuration.

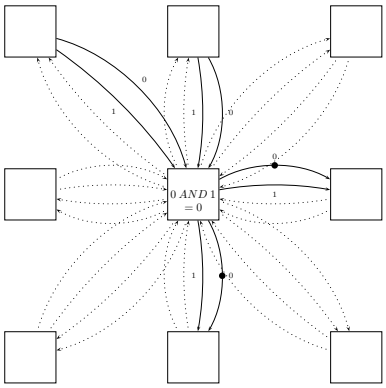
In addition, this model adapts much more easily to take advantage of adiabatic logic design; for instance, when a cell is being used only to ferry tokens from one place to another (e.g. an inverter, shown in Fig. 3), it can do so physically, instead of using a traditional, charge-dumping CMOS stage.



(a) Firing conditions met

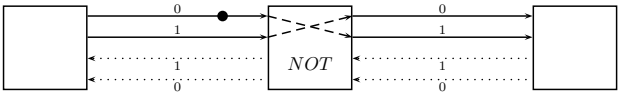


(b) Collect inputs; perform computation

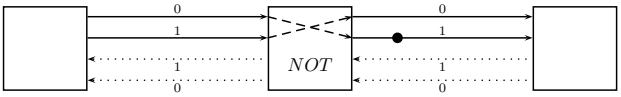


(c) Distribute result

Fig. 2. A cell firing; note that despite the loss of information, tokens are conserved in this example



(a) A token is output by the cell to the left



(b) The token now passes to the right cell

Fig. 3. A bit travels left to right through an inverting cell

Figures 4 and 5 show the general concept of how such cells could be implemented using so-called "bucket brigade", or charge-domain logic.

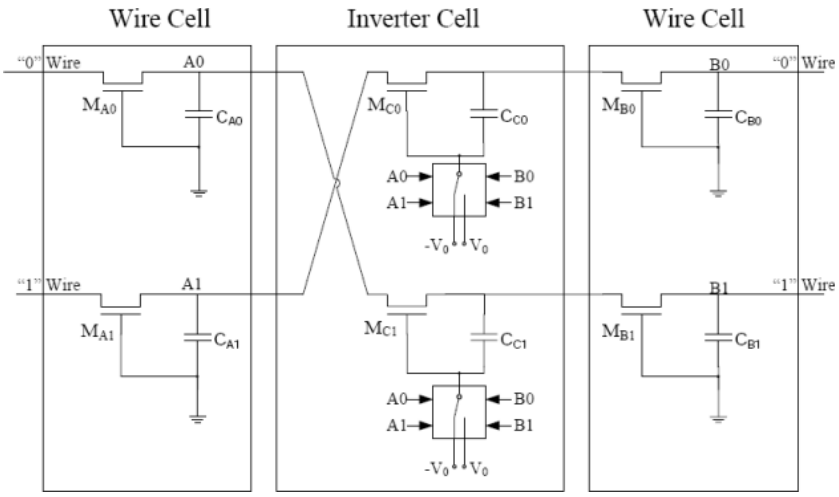


Fig. 4. Transistor-level effects of configured wire and inverter cells

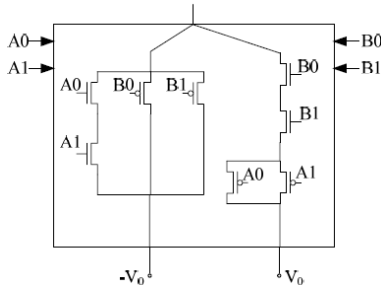


Fig. 5. Expansion of "switch" block from Fig. 4

Note the following possible ALA variations:

1. **No Diagonals.** Connections may only be present between vertically or horizontally adjacent cells, to simplify circuit layout.
2. **Multiple Signals.** More than four token-storing edges may connect neighboring cells, allowing the conveyance of more parallel information in the same period of time.
3. **More Functions.** The class of possible functions executed by each cell need not be limited to {AND, OR, XOR, NAND} but may include any function $f : \{0, 1, \emptyset\}^n \to \{0, 1, \emptyset\}^n$ where n is the number of neighbors of each cell (for $n = 8$ there are 43046721 possible functions). A cell executing function f may fire if f's present output is not $\emptyset^n$ and every non-empty element of the output points to either an inactive or empty set of output edges. Then each of those output edges would become populated with the value specified by f's output. There is a tradeoff between the number of functions allowed and the number of configuration bits in each cell needed to specify the function.

4 History

The history begins with the cellular automata (CAs) of von Neumann [5], designed to explore the theory of self-replicating machines in a mathematical way (though never finished). Note that this was some time after he completed the architecture for the EDVAC project [15], which has come to be known as "the von Neumann architecture." Many papers since then can be found examining (mostly 2-state) CAs, and there are a few directions to prove simple CA universality – Alvy Ray Smith's [16], E. Roger Banks' [17], and Matthew Cook's more recent Rule 110 construction [18]. However, while interesting from the point of view of computability theory, classical CAs clearly over-constrain algorithms to beyond the point of practicality, except in a small class of problems related to physical simulation (for instance, see [13]).

Another related sub-field is that of field-programmable gate arrays (FPGAs). Gate arrays have evolved over time from sum-product networks such as Shoup's [19] and other acyclic, memory-less structures such as Minnick's [20] to the complex, non-local constructions of today's commercial offerings, yet skipping over synchronous and sequential, but simplified local-effect cells.

The tradition of parallel programming languages, from Occam [21] to Erlang [22] to Fortress [23] is also of interest. Although they are designed for clusters of standard machines (possibly with multiple processors sharing access to a single, separate memory), they introduce work

distribution techniques and programming language ideas that are likely to prove useful in the practical application of our work.

Finally, the Connection Machine [24] was designed with a similar motivation – merging processing and memory into a homogeneous substrate – but as the name indicates, included many non-local connections: "In an abstract sense, the Connection Machine is a universal cellular automaton with an additional mechanism added for non-local communication. In other words, the Connection Machine hardware hides the details." We are primarily concerned with exposing the details, so that the programmer can decide on resource trade-offs dynamically. However, the implementation of Lisp on the Connection Machine [25] introduces concepts such as xectors which are likely to be useful in the implementation of functional programming languages in our architecture.

One key element of our approach that is not present in any of these models is that of formal conformance to physics:

- classical CAs are an "overshoot" – imposing too many constraints between space and time above those of physics;
- gate arrays have become non-local and are trending further away from local interactions;
- practical parallel languages accept the architecture of commercial computers and simply make the best of it in software; and
- the Connection Machine allows non-local communication by hiding physical details.

Also, at least as important as this is the fact that our model operates precisely without a global clock, while the four models above do not. This decreases power requirements and heat dissipation, while increasing overall speed.

5 Future work

The primary disadvantage to practical fabrication and use of ALA in their present form is the need to simultaneously initialize all cells with the configuration data before useful computation can be performed. We are currently investigating various approaches to solving this problem, such as a protocol for loading the data in to uninitialized space from the edges, by specifying a forwarding direction after each cell is configured and then propagating a final start signal when initialization is finished and computing can begin. We are also developing a hierarchical, module-based design environment for computing this configuration data on a traditional PC.

6 Conclusion

We have presented a new model which merges lattice gases, Petri nets, charge-domain logic, and reconfigurable logic. This model represents a simple strategy for asynchronous logic design: making the operations asynchronous at the bit level, and not just at the level of pipelines and functional blocks. It is also a potentially faster, more efficient, and lower-power alternative to traditional FPGAs, or to general-purpose computers for highly parallelizable tasks.

References

[1] Ronen, R., Mendelson, A., Lai, K., Lu, S.L., Pollack, F., Shen, J.P.: Coming challenges in microarchitecture and architecture. Proceedings of the IEEE **89**(3) (2001) 325–340
[2] Werner, T., Akella, V.: Asynchronous processor survey. Computer **30**(11) (1997) 67–76
[3] Geer, D.: Is it time for clockless chips? Computer **38**(3) (March 2005) 18–21
[4] Kurzweil, R.: The Singularity Is Near : When Humans Transcend Biology. Viking Adult (September 2005)
[5] von Neumann, J.: Theory of Self-Reproducing Automata. University of Illinois Press (1966)
[6] Fredkin, E., Toffoli, T.: Conservative logic. International Journal of Theoretical Physics **21**(3) (April 1982) 219–253
[7] Denker, J.S.: A review of adiabatic computing. In: Low Power Electronics, 1994. Digest of Technical Papers., IEEE Symposium. (1994) 94–97
[8] Morita, K., Imai, K.: Logical universality and self-reproduction in reversible cellular automata. In: ICES '96: Proceedings of the First International Conference on Evolvable Systems, London, UK, Springer-Verlag (1996) 152–166
[9] Lee, J., Peper, F., Adachi, S., Morita, K., Mashiko, S.: Reversible computation in asynchronous cellular automata. In: UMC '02: Proceedings of the Third International Conference on Unconventional Models of Computation, London, UK, Springer-Verlag (2002) 220–229
[10] Toffoli, T., Capobianco, S., Mentrasti, P.: When – and how – can a cellular automaton be rewritten as a lattice gas? (September 2007)
[11] Petri, C.A.: Nets, time and space. Theoretical Computer Science **153**(1-2) (January 1996) 3–48
[12] Hénon, M.: On the relation between lattice gases and cellular automata. In: Discrete Kinetic Theory, Lattice Gas Dynamics, and Foundations of Hydrodynamics. World Scientific (1989) 160–161
[13] Frisch, U., d'Humières, D., Hasslacher, B., Lallemand, P., Pomeau, Y., Rivet, J.P.: Lattice gas hydrodynamics in two and three dimensions. In: Lattice-Gas Methods for Partial Differential Equations. Addison-Wesley (1990) 77–135

[14] Murata, T.: State equation, controllability, and maximal matchings of petri nets. IEEE Transactions on Automatic Control **22**(3) (1977) 412–416
[15] von Neumann, J.: First draft of a report on the EDVAC. IEEE Annals of the History of Computing **15**(4) (1993) 27–75
[16] Smith, A.R.: Cellular Automata Theory. PhD thesis, Stanford University (1970)
[17] Banks, E.R.: Cellular Automata. Technical Report AIM-198, MIT (June 1970)
[18] Cook, M.: Universality in elementary cellular automata. Complex Systems **15**(1) (2004)
[19] Shoup, R.G.: Programmable Cellular Logic Arrays. PhD thesis, Carnegie Mellon University (1970)
[20] Minnick, R.C.: Cutpoint cellular logic. IEEE Transactions on Electronic Computers **EC-13**(6) (December 1964) 685–698
[21] Roscoe, A.W., Hoare, C.A.R.: The laws of Occam programming. Theoretical Computer Science **60**(2) (September 1988) 177–229
[22] Armstrong, J., Virding, R., Wikström, C., Williams, M.: Concurrent Programming in Erlang, Second Edition. Prentice-Hall (1996)
[23] Steele, G.L., Allen, E., Chase, D., Luchangco, V., Maessen, J.W., Ryu, S., Tobin-Hochstadt, S.: The Fortress Language Specification. Technical report, Sun Microsystems (March 2007)
[24] Hillis, W.D.: The Connection Machine. MIT Press, Cambridge, MA (1985)
[25] Steele, G.L., Hillis, W.D.: Connection Machine Lisp: fine-grained parallel symbolic processing. ACM Press (1986)