4-5 Nanoelectronics Architectures

Ferdinand Peper, LEE Jia, ADACHI Susumu, ISOKAWA Teijiro,
TAKADA Yousuke, MATSUI Nobuyuki, and MASHIKO Shinro

The ongoing miniaturization of electronics will eventually lead to logic devices and wires with feature sizes of the order of nanometers. These elements need to be organized in an architecture that is suitable to the strict requirements ruling the nanoworld. Key issues to be addressed are (1) how to manufacture nanocircuits, given that current techniques like optical lithography will be impracticable for nanometer scales, (2) how to reduce the substantial heat dissipation associated with the high integration densities, and (3) how to deal with the errors that are to occur with absolute certainty in the manufacturing and operation of nanoelectronics? In this paper we sketch our research efforts in designing architectures meeting these requirements.

Keywords
Nanoelectronics, Architecture, Cellular automaton, Heat dissipation, Fault-tolerance, Reconfigurable

1 Introduction

1.1 Background

The advances over the past decades in densities of Very Large Scale Integration (VLSI) chips have resulted in electronic systems with ever-increasing functionalities and speed, bringing into reach powerful information processing and communications applications. It is expected that silicon-based CMOS technology can be extended to up to the year 2015; however, to carry improvements beyond that year, technological breakthroughs will be needed. Though this is a major motivation for research into devices on nanometer scales, it is equally important to find out how such devices can be combined into larger systems. Systems based on nanodevices will require very different architectures (designs), due to the strict requirements of nanometer scales. Three major factors influencing designs of nanocircuits are: ease of manufacturing, minimization of heat dissipation, and tolerance to errors.

1.2 Ease of Manufacturing

It will be difficult to manufacture circuits of nanodevices in the same way as VLSI chips, i.e., by using optical lithography. The reason is that light has too large a wavelength to resolve details on nanometer scales. As a result, an alternative technique called self-assembly of nanocircuits is increasingly attracting the attention of researchers. The idea of this technique is to assemble circuits using the ability of molecules to interact with each other in such a way that certain desired patterns are formed, according to a process called self-organization. Though self-assembly is a promising technology in development, it comes with strings attached: the patterns that can be built tend to have very regular structures, like tiles in a bathroom. Consequently, nanocircuits constructed by self-assembly will likely need to have very regular structures.

1.3 Minimizing Heat Dissipation

The huge integration densities made possi-

ble by nanotechnology will have far-reaching consequences for the heat that can be dissipated by nanodevices. Even nowadays heat dissipation is a growing problem in VLSI chips. For nanoelectronics the situation is much worse: it is anticipated that heat dissipated per unit of surface will surpass that of the sun, if similar techniques as in current VLSI are used. Most notably, the use of clock signals to synchronize the operations in circuits is a major factor causing heat dissipation. Each time clock signals are distributed over a circuit, energy needs to be pumped into wires, and much of this energy tends to get lost in the form of heat. One technique to cope with this problem is to remove the clock. Circuits not requiring synchronization by clock signals are called asynchronous. Roughly speaking, in such circuits devices are activated only when there is work for them to do, unlike in clocked circuits, in which most devices are constantly busy processing clock signals, even in the absence of useful work. Provided they are well-designed, asynchronous electronics can reduce power consumption and heat dissipation substantially.

Another technique to reduce power consumption and heat dissipation is reversibility of operation, as was first claimed by Landauer [8]. The idea behind reversibility of a device is that it can recapture the energy spent in the device's operation by undoing that operation (i.e. doing the operation in reverse) afterwards. Though this principle has been known since the early 60's, it is hardly applied in practical circuits nowadays, perhaps because the gains it promises for VLSI do not justify the overhead in hardware it requires. For nanoelectronics the story may be different, however, since interactions on nanometer scales (for example interactions between molecules or other particles), are typically reversible, and such interactions are of fundamental importance to the operation of nanodevices. Consequently, interest into reversibility has revived together with the nanotechnology boom.

1.4 Tolerance to Errors

Both the manufacturing and the operation of nanocircuits will suffer from the unreliability of interactions on nanometer scales. In manufacturing this manifests itself in defects: some devices will permanently fail to work and some of the interconnections between devices will not be correctly made. To cope with defects, an architecture needs to be defect-tolerant. In other words, it needs to have redundent (spare) devices and interconnections available that take over the tasks of defective ones. An architecture also needs to be reconfigurable such that defective devices and interconnections can be made inaccessible in favor of redundant ones.

Errors occuring during the operation of nanocircuits will, unlike manufacturing defects, typically be transient. Transient errors occur only now and then, even in perfectly manufactured devices, due to factors such as thermal noise, signal noise, quantum mechanical effects, radiation, etc. The capability of architectures to cope with transient errors is called fault-tolerance. It can be provided by redundancy in a circuit, such that even when errors occur, they can be detected and corrected using the remaining (correct) information in the circuit. Error correcting codes are a well-known technique in this context, and they have been used extensively to achieve reliable performance in communications and in present-day memory circuits.

1.5 Nanoelectronics Architectures

The realization of nanoelectronics thus requires architectures with a regular structure that employ techniques like asynchronous timing or reversibility to minimize heat dissipation, while also offering tolerance to defects and transient errors. The architectures we have proposed for this purpose, are based on so-called cellular automata, which are regular arrays of identical cells, each cell of which can conduct a very simple operation. This paper shows how cellular automata can be used as architectures for nanoelectronics.

2 Cellular Automata

Cellular automata were first proposed by von Neumann [15] with the aim of describing the biological process of self-reproduction using a simple mathematical formalism. The appeal of cellular automata is that they can model complex behavior, while only simple local interactions between individual neighboring cells take place. In this paper we use a cellular automaton that is specially designed with implementation of nanoelectronic circuits in mind. This cellular automaton is a 2-dimensional array of identical cells, each of which has four neighboring cells, at its north, its east, its south, and its west. Each side of a cell has a memory of a few bits attached to it, as in Fig. 1.

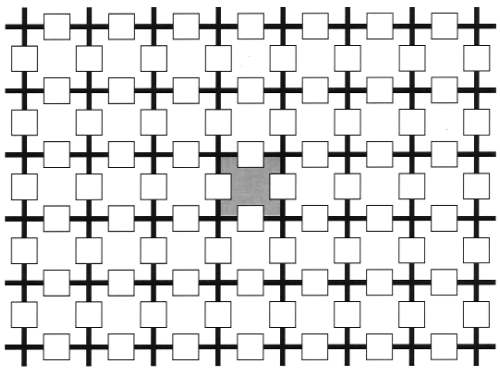


Fig.1 *Cellular automaton consisting of cells (the big squares with fat borders, one cell being shaded), each having access to four memories (the small squares). Shared by two cells, a memory stores a few bits of information.*

The four memories at the side of a cell are said to be associated with the cell. The value stored in a memory is called the state of the memory. A cell may change the states of the four memories associated with it according to an operation called a transition. The transitions a cell may undergo are expressed by a table of transition rules. A transition rule describes how a cell may change the states of the memories associated with it.

Figure 2 shows a typical transition rule: it consists of a left hand side, which is the part

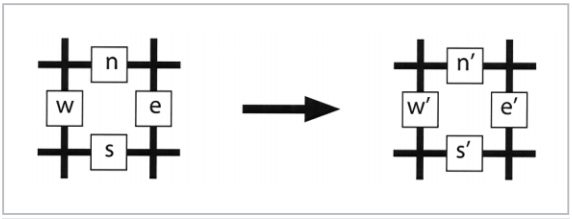


Fig.2 *Transition rule describing a transition the memories associated with a cell can undergo. If the memory states of a cell match the states n, e, s, and w in the left hand side of the rule, the rule may be applied to the cell, as a result of which the states of the cell's memories are changed into the states n′, e′, s′, and w′, respectively, depicted in the right hand side.*

left of the arrow, and a right hand side. If the left hand side matches the combination of states of the memories associated with a cell, the transition rule is said to apply to the cell. The states of the memories may then be replaced by the states in the right hand side of the transition rule. Certain desired behavior of a cellular automaton can be obtained by setting its memories in proper states and defining transition rules that lead to state changes corresponding to this behavior. For example, one may design a cellular automaton that behaves in the same way as a certain digital electronic circuit.

Cellular automata in which all cells undergo transitions simultaneously in synchrony with a central clock signal are called synchronous. They are the most widely studied type. When transitions of the cells occur at random times, independent of each other, we obtain asynchronous cellular automata, which is the type employed in this paper. In an asynchronous cellular automaton, a cell may undergo a transition when its memory states match the left hand side of a transition rule; this transition is randomly timed. If there is no transition rule whose left hand side matches a cell's combination of memory states, the cell remains unchanged. Though transitions of cells are timed randomly and independently of each other, they are subject to the condition that two neighboring cells never undergo transitions simultaneously. This ensures that two neighboring cells will

not attempt to set the memory common to them to different states at the same time.

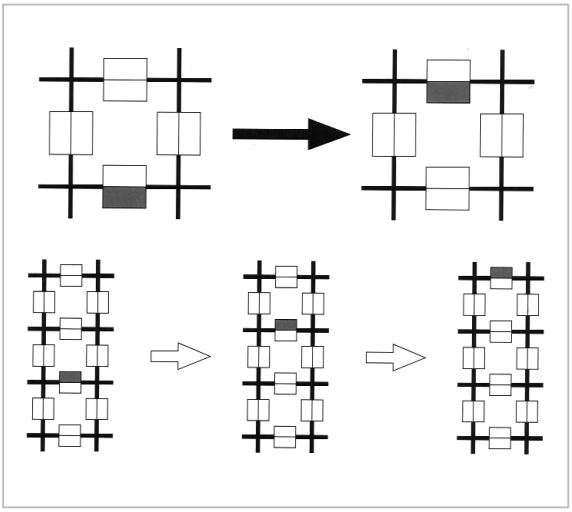


Fig.3 *(a) Transition rule for signal propagation, and (b) its application twice to a configuration of cells, each time giving rise to a signal successively moving one cell towards the north. Here a memory contains two bits, each of which is indicated by a block that is shaded for the value 1 and white for the value 0. The transition rule operates on all bits in the four memories associated with a cell.*

Figure 3(a) shows an example of a transition rule that moves the contents of a memory of two bits, one bit being 0 and the other 1, by one cell to the north. In a memory in this example, the two bits are represented by a white block and a black block, to denote a 0-bit and a 1-bit, respectively, and the bits are separated by a thin line. When this transition rule is applied to the configuration on the left of Fig. 3(b), a 01-bit pattern in a memory is moved one cell to the north (Fig. 3(b) middle configuration). Applying the transition rule once more moves the bit pattern one more cell to the north (right side of Fig. 3(c)). Since this 01-bit pattern steadily moves to the north over time, it can be used to transfer information from the south to the north in the cellular automaton. For this reason, this pattern is called a signal. Together the cells over which the signal can move form a so-called path. The rotated or reflected version of a rule may also be used for transitions. This allows the above transition rule to be used for transmitting signals in directions towards the south, east, or west as well.

3 Operations

The cellular automaton described up to this point is extremely simple: it can only transmit signals along straight paths. To do some useful work by the cellular automaton, we need a way to operate on signals. Operating on signals is more difficult than just transmitting them, because it may involve multiple signals. For example, we may have an operation that requires one signal to be input, and that produces two output signals as a result. Such an operation is called a fork, because a fork (the variety used for eating) has one bar at one end, reflecting the input side, and multiple bars at the other end, reflecting the output side. The opposite of a fork is an operation called a join. A join receives two signals as input, and produces one signal as output. If a join receives only one signal as input, it just waits until the second signal arrives before taking any action. How long should a join operator wait for a second signal? The last question addresses an important point for the operations we use in our cellular automaton. Our answer is: we allow a join to wait for a second signal as long as it is necessary. Even stronger, we allow any operator to wait for any input signal at least as long as is necessary for that signal to arrive. In other words, any arbitrary delay experienced by a signal is supposed not to alter the logical outcome of the operations conducted on the signal. A circuit in which delays of signals do not alter the logical outcome of circuit operations is called delay-insensitive (DI). DI circuits are an important class of asynchronous circuits. To operate correctly, such circuits do not require a central clock sending out clock signals. Rather, DI circuits are driven by data signals: each circuit operator is inactive, until it receives an appropriate set of input signals, after which it processes the signals, outputs appropriate signals, and becomes inactive again. Operators in DI circuits are similar to

the operators in normal digital circuits, like AND-gates and NOT-gates, except that DI operators need no clock signals to ensure their inputs and outputs are properly timed. DI circuits are thus promising candidates for reducing heat dissipation in physical realizations.

To implement a fork operator on our cellular automaton, we employ a cell of which one of the memories contains two 1-bits and the other three memories contain all 0-bits.

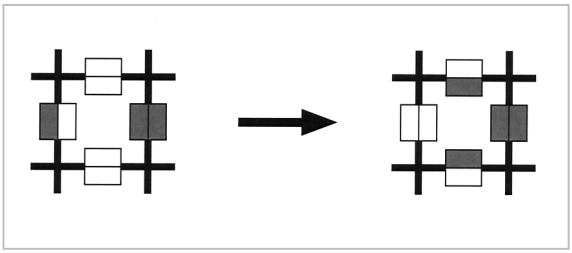


Fig.4 *Transition rule defining the operation of a fork. The fork is represented by a memory of which both bits are 1.*

The transition rule in Fig. 4 is able to operate on such a fork cell after the cell receives an input signal, like in Fig. 5, and this results in two output signals. Unfortunately, a fork alone is insufficient to create arbitrary circuits: other operators are necessary, like the join operator described above. It turns out that arbitrary valid DI circuits can be constructed using only a small set of operators. Such a set is called universal, and it can be as small as three operators, as shown in [11] (see also [10]). For reasons of space we will not go into details, other than mentioning that this set can be implemented on our cellular automaton using only four more transition rules [16] in addition to the two rules in Fig. 3(a) and Fig. 4. To see how actual circuits can be made on our cellular automaton, like a 1-bit memory or a DI AND-gate, we refer to [16]. Implementation of other, higher level, circuits and program structures, like counters, for-loops, etc., is discussed in [17].

4 Reconfigurability

Reconfigurability of a nanoelectronics architecture is its ability to reorganize its devices and the interconnections between its devices according to some functional specification given by a user. In the case of using a cellular automaton as architecture, it may happen that it conducts for example at one time a pattern recognition task, at another time a communications task, and at still another time some number crunching task like deciphering an encrypted message. Given that a cellular automaton has a very homogeneous structure with all identical simple cells, how can it be made to do such a wide variety of tasks? In the previous section we have seen that the functionality of each cell can be determined by setting the memories associated with it to appropriate states. Taken together, a configuration of cells with memories in certain states thus forms a DI circuit with a certain functionality. While this explains how a cellular automaton can have different functionalities depending on the states of its memories, it leaves open the question as to how the memories can be set to the appropriate states that give rise to a certain desired functionality, and later be reset to different states reflecting a different functionality. This problem boils down

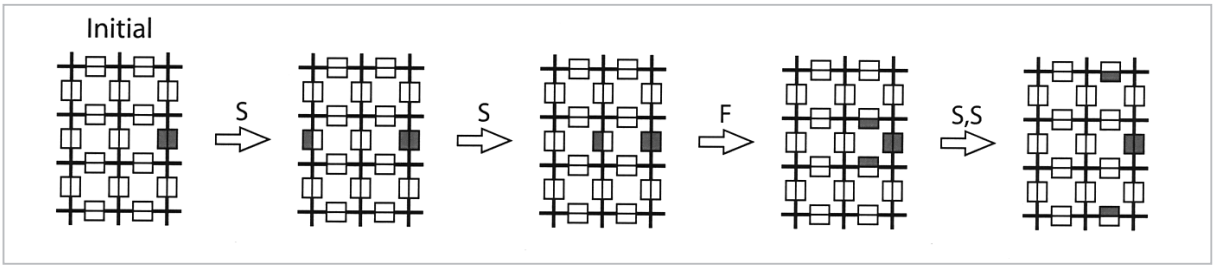


Fig.5 *Sequence of operations in which a fork receives one input signal on its input path from the west, after which it produces one signal to each of its output paths, i.e., to the north and the south. Each time a transition rule is used, its label appears above the corresponding arrow, whereby S denotes the transition rule for signal propagation and F that for the fork.*

to moving a certain pattern of information to a certain location in the cellular automaton. In this case this pattern of information is a configuration of cells that represent a DI circuit layout. How can this problem be addressed?

Fortunately, the field of cellular automata has a rich history, and one of the major achievements is the implementation of self-reproduction, which is the ability to copy and move certain patterns of information stored in the cells' memories to other cells' memories. It thus closely resembles reconfigurability. As we pointed out in section 2, self-reproduction was the defining problem that started the field of cellular automata. The method used by Von Neumann to implement self-reproduction on cellular automata [15], though, is extremely complicated, up to the point that even the mere simulation on modern-day computers is difficult in terms of memory resources and computation time. Successors of this research, however, like [2] [3] [19], have achieved much simpler self-reproducing cellular automata. To investigate the potential of our cellular automaton for self-reproduction, we have implemented self-reproduction on a simplified version of our cellular automaton [20] [21]. Note that our cellular automaton is different from the cellular automata used in the past for self-reproduction [2] [3] [15] [19], in the sense that it works asynchronously. This makes it much more difficult to control the timing by which cells undergo transitions, and consequently, it complicates implementations of self-reproduction.

Our implementation [20] [21] uses the same basic idea as von Neumann's. That is, it is based on a so-called Turing machine, which is a very simple model of a computer. A Turing machine consists of two parts: a control mechanism and a tape unit. The control mechanism does all the "logical work", and can be described in terms of DI circuits, so it is easy to implement on our cellular automaton, using the same six transition rules as are necessary for implementing DI circuits. The second part of a Turing machine, i.e., the tape unit, is a kind of memory that stores the machine's program, data, and temporary results. The tape unit is more complicated than the control mechanism: it requires a head that moves about the tape and that reads and writes information from and to tape cells. To do self-reproduction by a Turing machine, one additional mechanism is needed. Called a construction arm, this mechanism can be extended towards possibly far-away cells to copy information to their memories, and it can be withdrawn afterwards. Unfortunately, the implementation of a construction arm on our cellular automaton requires additional transition rules, making the cellular automaton more complicated. The same holds true for the tape unit. Fortunately, the tape unit is very similar to the construction arm: it turns out that they can be both made by using the same transition rules, making a total of 39 transition rules [20] [21]. Though this is much more than the six rules required for merely implementing DI circuits, we believe that the number of rules can be reduced, but this is left as future research.

5 Tolerance to Errors

Reconfigurability not only endows a cellular automaton with a flexible functionality, but also makes it more tolerant to defects, because it gives freedom to use only nondefective cells when configuring DI circuits on the cellular automaton. Well-known results on defect tolerance have been obtained with the Teramac [4], which is a parallel computer based on field programmable gate arrays (FPGAs). The Teramac is able to achieve high-performance computing, even if a significant number of its components are defective. However, the Teramac requires a master computer to set up a table of routes around defects and to configure the hardware accordingly, so it is not autonomous. Especially with defects that occur after manufacturing, i.e., during operation, it would need a master computer to be connected permanently to it to keep the routing table error-free. A master computer based on nanoelectronics, however, would

suffer from defects too, so this solution is not self-contained. We thus prefer a method in which scanning for defects and hardware reconfiguration is done by nanoelectronics hardware itself rather than by a master computer. We have conducted some preliminary research for this purpose [7], by designing a cellular automaton in which cells follow a certain sequence of states over time. As defective cells typically fail to change states, they can be identified in this way. Defective cells are then "isolated" by marking their non-defect neighbors by states that are exclusively used for this purpose. This allows DI circuits to be configured such as to avoid marked cells.

When errors are transient, other strategies than the above are needed. In [16] we have constructed a fault-tolerant version of the cellular automaton of sections 2 and 3 by encoding the memories associated with the cells by error correcting codes. The central idea of error correcting codes is to include redundant information in the encoding, and use this information to reconstruct the original contents of a memory in case some of the memory bits become erroneous.

Figure 6 shows one such encoding: the original four values of a 2-bit memory in the bottom row, i.e., the values 00, 01, 10, and 11, are encoded by the four values in the top row that are expressed by 14 bits each. So, to make a memory fault-tolerant, we need 14 bits instead of the original two bits. The expense of seven times more bits in a memory gives us in return the ability to correct up to four bit errors. For example, flipping four arbitrary bits in one of the memories in the top row of Fig. 6 from 0 to 1 or the other way around, gives a combination of bit values that more closely resembles the original than any of the other three memories in the top row (see Fig. 7). So, even with four bit errors, we can deduce what the original contents of a memory should have been.

As is pointed out in [16], most 5-bit errors and some 6-bit errors can also be corrected in these 14-bit memories. Our scheme is flexible in the sense that it can be made tolerant to more (or less) bit errors by expending more (or resp. less) bits in each memory. The idea of our error correcting scheme is to block transitions on a cell as long as at least one of the memories associated with it contains at least one error. Errors in each memory are corrected at random times; when all memories associated with a cell become error-free, the precondition is then created for a transition to take place on the cell. In other words, operations of cells take only place once all the errors in their memories are corrected. We refer to [16] for more details of this scheme, and to [6] for a similar scheme based on a different type of error correcting code.

6 Discussion

The construction of nanoelectronics circuits requires a serious consideration of the architectures that are to be used. To facilitate manufacturability, architectures will need to

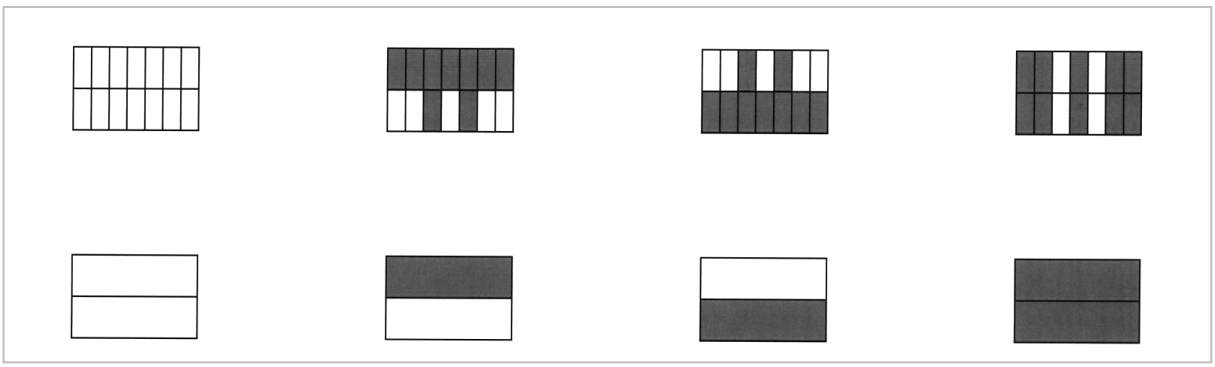


Fig.6 *Graphical notation of an error correcting code. The bottom row contains the original 2-bit memories, the top row the corresponding memories that are encoded by 14 bits each. A dark block denotes a bit with value 1, a white block a bit with value 0.*

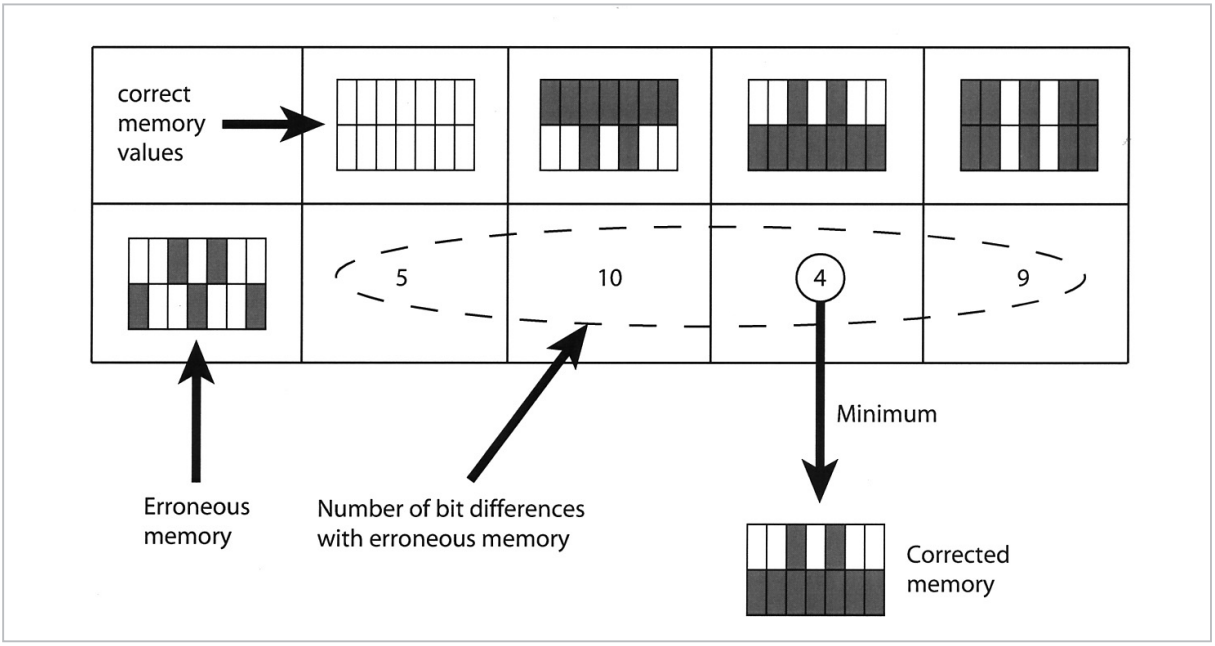


Fig.7 *Occurrence and correction of four bit errors in a 14-bit memory. The four correct values of a 14-bit memory in the top row differ with the erroneous memory in 5, 10, 4, and 9 bits, respectively. So, the corrected value of the erroneous memory is most likely the value that differs in only 4 bits with it.*

be regular. Additionally, the reduction of heat dissipation is important, as well as the tolerance to errors during manufacturing and operation. Our results achieved thus far suggest that cellular automata form an attractive basis to satisfy these demands. With respect to heat dissipation we have investigated cellular automaton models with an asynchronous mode of operation. The randomness by which transitions on cells take place eliminates the need for distributing a central clock signal over the cells. Though this has the potential to reduce heat dissipation substantially, it also complicates the operation of a cellular automaton. We have developed a method to cope with this: to simplify operation, we configure the cells such that they implement delay insensitive circuits. As compared to past methods [14], this is a remarkably effective way to coordinate cells' activities on local scales, while allowing them to be independent of each other on global scales. Our method can be applied to a wide range of asynchronous cellular automaton types, for example as in [1] [9].

We have also investigated an alternative way to reduce heat dissipation, i.e. by using reversibility of circuit operation, but we do not report extensively on it in this paper, other than mentioning that we have designed asynchronous cellular automata [12] and delay insensitive circuits [13] that are reversible. Though these designs are not very efficient in terms of computational power, this research has deepened our understanding of the nature of reversible operation without the use of a central clock, and it has the potential to deliver some unexpected results. Moreover, since reversible operation is a precondition for quantum computing, our research may also have some impact on that field.

The feasibility of our cellular automaton approach will be decided by how efficiently individual cells can be manufactured and put together into regular structures. One promising way to achieve efficient manufacturing is the so-called Nanocell approach [5]. In this approach, the internal of a cell consists of nanodevices connected in a random network of nanowires. To achieve a certain desired functionality, the cell is subjected to appropriate signals at its inputs and outputs that burn certain pathways in the network, according to a process that resembles neural network train-

ing. Once manufactured and trained, cells may be put together by using DNA tiling techniques like those advanced by Seeman et al. [18]. Though many new techniques important to physical implementations may emerge in the coming decade, it seems clear at this point that a key issue will be making cells as simple as possible, i.e. making them operate with a minimum set of transition rules. For this purpose, simpler delay insensitive circuits and simpler self-reproducing techniques need to be designed, and this is an ongoing effort in our research.

What can we eventually expect from nanoelectronics? The high integration densities that are possible have the potential of substantial performance gains, as compared to current VLSI. Most of these gains will come from the huge number of available devices, though increased switching speeds of devices may also contribute. Applications that can effectively utilize such huge numbers of devices will be the main beneficiaries of nanoelectronics. We think about applications suitable to be subdivided in many independent subproblems that can be processed in parallel. Typical in this context are Artificial Intelligence (AI) problems, search problems, optimization problems, and many pattern recognition problems. Portability of applications will be greatly facilitated by nanoelectronics' limited heat dissipation, since the latter implies low power consumption. One can thus envision devices that do not need recharging other than occasionally by solar cells. Such devices can be carried continuously by their users, keeping them wirelessly connected to the world. Due to their powerful abilities, these devices will be indispensible in augmenting the intellectual abilities of their users, for example with respect to retrieving faces, recognizing speech, etc. Other applications that may be made possible by nanoelectronics are networks on submillimeter scales that connect miniscule sensors, actuators, and communication hubs that interface with the outside world. Such networks may find use in the control of miniscule environments.

References

1 S. Adachi, F. Peper, and J. Lee, "Computation by asynchronously updating cellular automata", Journal of Statistical Physics, Vol. 114, Nos. 1/2, pp. 261 – 289, 2004.

2 E.R. Banks, "Universality in cellular automata", Proc. IEEE 11th Annual Symposium on Switching and Automata Theory, pp. 194 – 215, 1970.

3 E.F. Codd, Cellular Automata, Academic Press, New York, 1968.

4 J.R. Heath, P.J. Kuekes, G.S. Snider, and R.S. Williams, "A defect-tolerant computer architecture: opportunities for nanotechnology", Science, Vol. 280, pp. 1716 – 1721, 1998.

5 C.P. Husband, S.M. Husband, J.S. Daniels, and J.M. Tour, "Logic and memory with Nanocell circuits", IEEE Transactions on Electron Devices, Vol. 50, No. 9, pp. 1865 – 1875, 2003.

6 T. Isokawa, F. Abo, F. Peper, S. Adachi, J. Lee, N. Matsui, and S. Mashiko, "Fault-tolerant nanocomputers based on asynchronous cellular automata", International Journal of Modern Physics C, in press, 2004.

7 T. Isokawa, F. Abo, F. Peper, N. Kamiura, and N. Matsui, "Defect-tolerant computing based on an asynchronous cellular automaton", Proc. SICE Annual Conference, Fukui, Japan, pp. 1746 – 1749, 2003.

8 R. Landauer, "Irreversibility and heat generation in the computing process", IBM Journal of Research and Development, Vol. 5, pp. 183 – 191, 1961.

9 J. Lee, S. Adachi, F. Peper, and K. Morita, "Embedding universal delay-insensitive circuits in asynchronous cellular spaces", Fundamenta Informaticae, Vol. 58, Nos. 3/4, pp. 295 – 320, 2003.

10 J. Lee, F. Peper, S. Adachi, and K. Morita, "Universal delay-insensitive circuits with bi-directional and buffering lines", IEEE Transactions on Computers, Vol. 53, No. 8, pp. 1034 – 1046, 2004.

11 J. Lee, F. Peper, S. Adachi, and S. Mashiko, “Universal delay-insensitive systems with buffering lines”, IEEE Transactions on Circuits and Systems I, in press, 2004.

12 J. Lee, F. Peper, S. Adachi, K. Morita, and S. Mashiko, “Reversible computation in asynchronous cellular automata”, Unconventional Models of Computation, Lecture Notes in Computer Science, Vol. LNCS 2509, 220 – 229, 2002.

13 J. Lee, F. Peper, S. Adachi, and S. Mashiko, “On reversible computation in asynchronous systems”, in: Quantum Information and Complexity: Proceedings of the 2003 Meijo Winter School and Conference, T. Hida (Ed.), World Scientific, in press, 2004.

14 K. Nakamura, “Asynchronous cellular automata and their computational ability”, Systems, Computers, Controls, Vol. 5, pp. 58 – 66, 1974.

15 J. von Neumann, Theory of Self-Reproducing Automata, University of Illinois Press, 1966.

16 F. Peper, J. Lee, F. Abo, T. Isokawa, S. Adachi, N. Matsui, and S. Mashiko, “Fault-tolerance in nanocomputers: a cellular array approach”, IEEE Transactions on Nanotechnology, Vol. 3, No. 1, pp. 187 – 201, 2004.

17 F. Peper, J. Lee, S. Adachi, and S. Mashiko, “Laying out circuits on asynchronous cellular arrays: a step towards feasible nanocomputers?”, Nanotechnology, Vol. 14, pp. 469 – 485, 2003.

18 N.C. Seeman, “Nanotechnology and the double helix”, Scientific American, Vol. 290, No. 6, pp. 64 – 75, June 2004.

19 T. Serizawa, “Three-state Neumann neighbor cellular automata capable of constructing self-reproducing machines”, Systems and Computers in Japan, Vol. 18, pp. 33 – 40, 1986.

20 Y. Takada, T. Isokawa, F. Peper, and N. Matsui, “Universal construction and self-reproduction on self-timed cellular automata”, International Journal of Modern Physics C, 2005, accepted.

21 Y. Takada, T. Isokawa, F. Peper, and N. Matsui, “Universal construction on self-timed cellular automata”, Cellular Automata for Research and Industry, Lecture Notes in Computer Science, LNCS3305, pp. 21-30, 2004.

Ferdinand Peper, *Ph.D.*

Senior Researcher, Nanotechnology Group, Kansai Advanced Research Center, Basic and Advanced Research Department

Nanotechnology, Computer Science

LEE Jia, *Ph.D.*

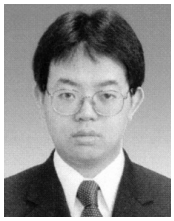
Expert Researcher, Nanotechnology Group, Kansai Advanced Research Center, Basic and Advanced Research Department

Computer Science, Nanotechnology

ADACHI Susumu, *Ph.D.*

Expert Researcher, Nanotechnology Group, Kansai Advanced Research Center, Basic and Advanced Research Department

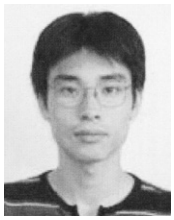
Computer Science, Nanotechnology



ISOKAWA Teijiro, *Dr.Eng.*

Research Associate, Division of Computer Engineerng, Graduate School of Engineering, University of Hyogo

Computer Science



TAKADA Yousuke

Student, Division of Computer Engineerng, Graduate School of Engineering, University of Hyogo

Computer Science

MATSUI Nobuyuki, *Dr.Eng.*

Professor, Division of Computer Engineerng, Graduate School of Engineering, University of Hyogo

Computer Science, Physics

MASHIKO Shinro, *Dr.Eng.*

Director of Kansai Advanced Research Center, Basic and Advanced Research Department

Laser Spectroscopy, Nanotechnology