# Reconfiguring Circuits Around Defects in Self-Timed Cellular Automata

Tadashi Kunieda[1], Teijiro Isokawa[1], Ferdinand Peper[2,1], Ayumu Saitoh[1], Naotake Kamiura[1], and Nobuyuki Matsui[1]

[1] Division of Computer Engineering, Graduate School of Engineering, University of Hyogo, 2167 Shosha, Himeji, 671-2280, Japan
isokawa@eng.u-hyogo.ac.jp
[2] Nano ICT Group, National Institute of Information and Communications Technology, Japan

**Abstract.** For the realization of nanocomputers it will be important to have built-in defect-tolerance, which is the ability to overcome the unreliability caused by defective components. This paper explores defect-tolerance for nanocomputers based on Self-Timed Cellular Automata—an asynchronously timed CA of which the functionality can be expressed through a small number of transition rules. The proposed method assumes that defects are coped with in an initial phase by detecting and isolating them in cellular space from non-defective cells. The phase after this—the main topic of this paper—includes a scheme to efficiently lay out circuits on the cellular space in areas that are not affected by defects. The scheme is self-contained, i.e., it is carried out through the transition rules defined for the CA and does not require external circuitry.

## 1 Introduction

Cellular Automata(CA) have much potential as architectures for computers built with nanometer-scale feature sizes (nano-computers), because their regular structures and local connectivity are very suitable for manufacturing by molecular self-assembly [1,2]. Two important issues for realizing nanocomputers are (1) the reduction of power consumption and heat dissipation and (2) reliability. The first issue has resulted in models governed by an asynchronous scheme of timing, which offers an energy-efficient alternative to the clock signals commonly used in synchronous schemes. Models combining the key advantage of CA's regularity with the advantages of asynchronous timing thus offer a promising road to the realization of nanocomputers. One such a model is the Self-Timed Cellular Automaton (STCA), which requires only a small number of transition rules to express the functionality required for computation. Computations on this model are typically carried out by simulating so-called Delay-Insensitive circuits on the cell space [3,4], which are a kind of circuit that is robust to delays in the signals processed by it.

The second issue concerns the reduced reliability of nanometer-scale devices as compared to their VLSI counterparts, due to defects arising during manufacturing. In a previous study on defects in STCA models [5], defective cells in

H. Umeo et al. (Eds): ACRI 2008, LNCS 5191, pp. 200–209, 2008.

the cellular space are detected and isolated in an initial phase, followed by the placement of circuit modules in free areas of the cell space that lie between isolated defects. This scheme is, however, rather inefficient in its use of cell space, since the placement of modules is rudimentary, allowing modules to be placed only along a narrow zone that can be extended in one dimension only, but not in a second dimension. This would leave some non-defective areas unused for the placement of modules. The scheme in [5] also lacks completeness, in that the placement of circuit modules is not self-contained. A more complete mechanism would require, for example, a so-called universal constructor [6] in cellular space to direct the detection and measurement of defect-free areas and to direct the placement of modules.

In this paper, we present a novel scheme for defect tolerance on STCA that is an improved version of the scheme in [5]. A constructor for placing circuit modules is implemented that has the ability to adjust the placement location of each module by measuring the sizes of the free areas in the cellular space. All two dimensions of the cell space are efficiently used for the placement of the modules into which the circuit can be subdivided. In the proposed scheme, the circuit modules communicate with each other through a combination of strategies. First, signals have the ability to go around defective areas autonomously, in a similar way as in [5]. This ability is implemented through the model's transition rules. Second, for the case modules' placements causes their input and output lines to be misaligned with each other, there is a bus configuration implemented on the cell space that carries signals from source to the appropriate destination in a flexible way. Lacking in the model in [5], this bus lies at the basis for the proposed model's ability to efficiently use the two dimensions of the cell space for laying out circuit modules.

## 2 Preliminaries

### 2.1 Self-Timed Cellular Automaton

A self-timed cellular automaton (STCA)[3] is a two-dimensional asynchronous CA of identical cells, each of which has a state that is partitioned into four parts in one-to-one correspondence with its neighboring cells. Each part of a cell has a state. Figure 1 shows a cell in STCA where the states of the partitions are denoted as $p_n$, $p_e$, $p_s$, and $p_w$. For simplicity the diagonal lines indicating partitions in cells are left out in the remainder of the paper. Each cell undergoes transitions in accordance with a transition function $f$ that operates on the four parts of the cell $p_n$, $p_e$, $p_s$, $p_w$ and the nearest part of each of its four neighbors $q_n$, $q_e$, $q_s$, $q_w$. The transition function $f$ is defined by

$$f(p_n, p_e, p_s, p_w, q_n, q_e, q_s, q_w) \rightarrow (p'_n, p'_e, p'_s, p'_w, q'_n, q'_e, q'_s, q'_w), \tag{1}$$

where a state symbol to which a prime is attached denotes the new state of a partition after update (see Fig. 2). Function $f$ can be described by a finite set of

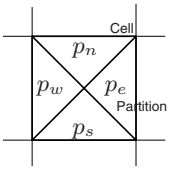



**Fig. 1.** A cell in Self-Timed Cellular Automaton

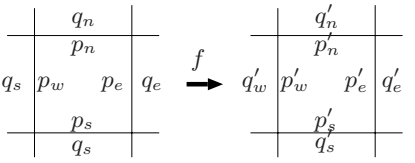
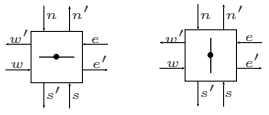
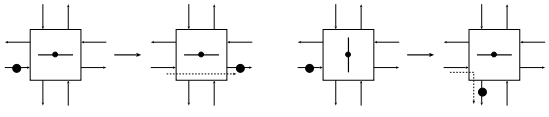



**Fig. 2.** Transition rule in accordance with the function $f$




(a) H-state (b) V-state

**Fig. 3.** A Rotary Element



(a) Parallel case (b) Orthogonal case

**Fig. 4.** The operation of an RE on a signal

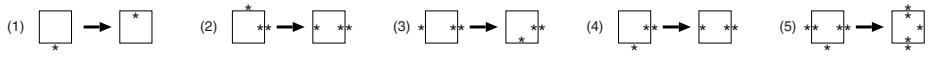



**Fig. 5.** Transition rules implementing the RE. A '$*$' symbol in a partition denotes the corresponding state '$*$', and a blank denotes the state '  '.

transition rules on the STCA. Dummy transitions, which are transitions without any changes of the partition states, are not included in a set of the transition rules, so we assume that the left-hand side of Fig. 2 always differs from the right-hand side. Furthermore, we assume that transition rules on an STCA are rotation-symmetric, thus each of the rules has four rotated analogues.

In an STCA, transitions of the cells occur at random times, independent of each other. Furthermore, it is assumed that neighboring cells never undergo transitions simultaneously to prevent a situation in which such cells write different values in shared partitions at the same time.

There are several approaches to perform computation on STCAs, such as simulating synchronous CA on them, or embedding delay-insensitive circuits on them[7,4]; we will use the latter approach, because of its lower overhead in terms of the number of required cell states and transition rules. We use Morita's Rotary Element [8] for conducting computation. This element has been proven to be computational universal.

**Rotary Element(RE) and Its Implementation on STCA.** A *Rotary Element (RE)* [8] is a logic element with four input lines $\{n, e, s, w\}$, four output lines $\{n', e', s', w'\}$, and two states—the H-state and the V-state—which are displayed as horizontal and vertical rotation bars respectively (Fig. 3). A signal coming from a direction parallel to the rotation bar passes straight through to the opposite output line, without changing the direction of the bar (Fig. 4(a));

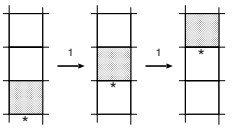


**Fig. 6.** Signal transmission northwards

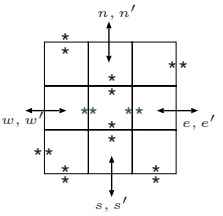



**Fig. 7.** RE realized on STCA

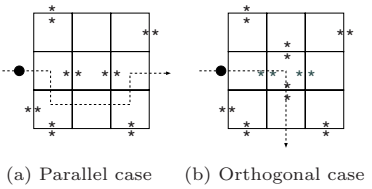



**Fig. 8.** Signal passing through RE

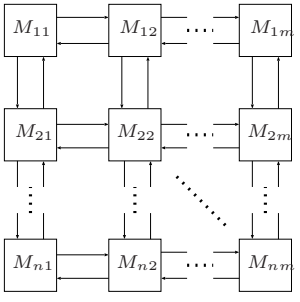



**Fig. 9.** Modules are arranged in two-dimensional space according to a mesh-connected scheme

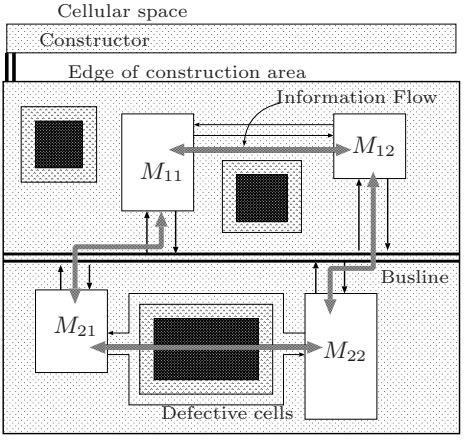



**Fig. 10.** Cellular space with four circuit modules ($M_{ij}$) and three isolated areas of defective cells (black surrounded by gray layers)

a signal coming from a direction orthogonal to the bar turns right and rotates the bar by 90 degrees (Fig. 4(b)).

Embedding an arbitrary RE-circuit on the cellular space of STCA can be achieved by the two states of a partition and the five transition rules shown in Fig. 5 (see [6] for details). Rule #1 defines a signal used for ordinary computation. Applying it to the successive shaded cells in Fig. 6 results in a signal proceeding northwards on a straight continuous path of cells, all bits of which are 0. Transmitting a signal towards the east, south, or west is done similarly, because of the rotation-symmetry of transition rules. Rules #2–4 are responsible for routing a signal according to the direction from which the signal comes from. The operation of the rotation bar in an RE is conducted by rule #5.

A rudimentary RE is shown in Fig. 7. The input/output paths at the sides of this RE are bi-directional. Connecting so-called Input/Output Multiplexers [6] to these paths results in a regular RE with four input and four output paths, all unidirectional. Figure 8 illustrates the traces of a signal input to a rudimentary RE from a line (a) parallel with or (b) orthogonal to the rotation bar. Since all the

elements of RE-circuits can be realized by stable partition pairs, any circuit on this STCA can be represented by a certain configuration of stable partition-pairs.

## 3 Laying Out Circuit Modules on STCA

In our model circuit modules are connected through a mesh-connected scheme, with the nodes in the mesh represented by modules (Fig. 9). The input (resp. output) ports of a module are matched to the output (resp. input) ports of its neighboring modules in a one-to-one fashion, i.e., the signal lines between modules are not merged or bifurcated.

In the underlying STCA model, each cell partition can be in one of 18 states, which are denoted by a set of symbol $\{0, \cdots, 9, A, \cdots, H\}$. The partition state '$*$' and ' ' in section 2 are mapped to the partition state 2 and 3 in this section, respectively.

Defects in the CA are of the so-called stuck-at-fault type [5], i.e., a cell's state cannot be updated by the cell itself or by an outside source, but it can still be read by neighboring cells. Defects are detected and isolated through a procedure outlined in [5]. This results in a cell space (Fig. 10) in which defective areas (colored black) are isolated by a layer of cells (the gray cells surrounding the black areas) all of which are in the same state.

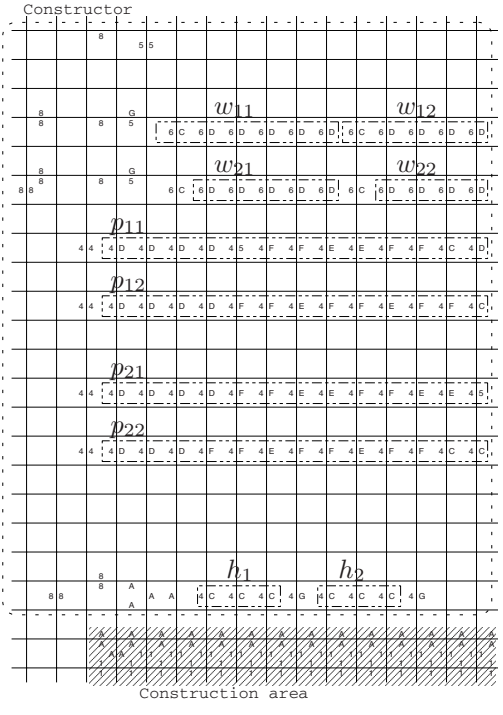



**Fig. 11.** Signals in constructor. For reasons of space only the left hand side of the constructor configuration is shown.

The areas between defects are used for the placement of circuit modules. Fig. 10 shows four modules placed in a cell space with three defective (and isolated) areas. There is also a constructor at the top, which directs the construction of the circuit on the cell space. Modules exchange computational signals between each other via signal lines. Module $M_{11}$ and $M_{12}$ are directly connected to each other, without any obstacles in between. Modules $M_{21}$ and $M_{22}$, on the other hand, have a large area of defective cells in between, which obstructs signals between them. Communication

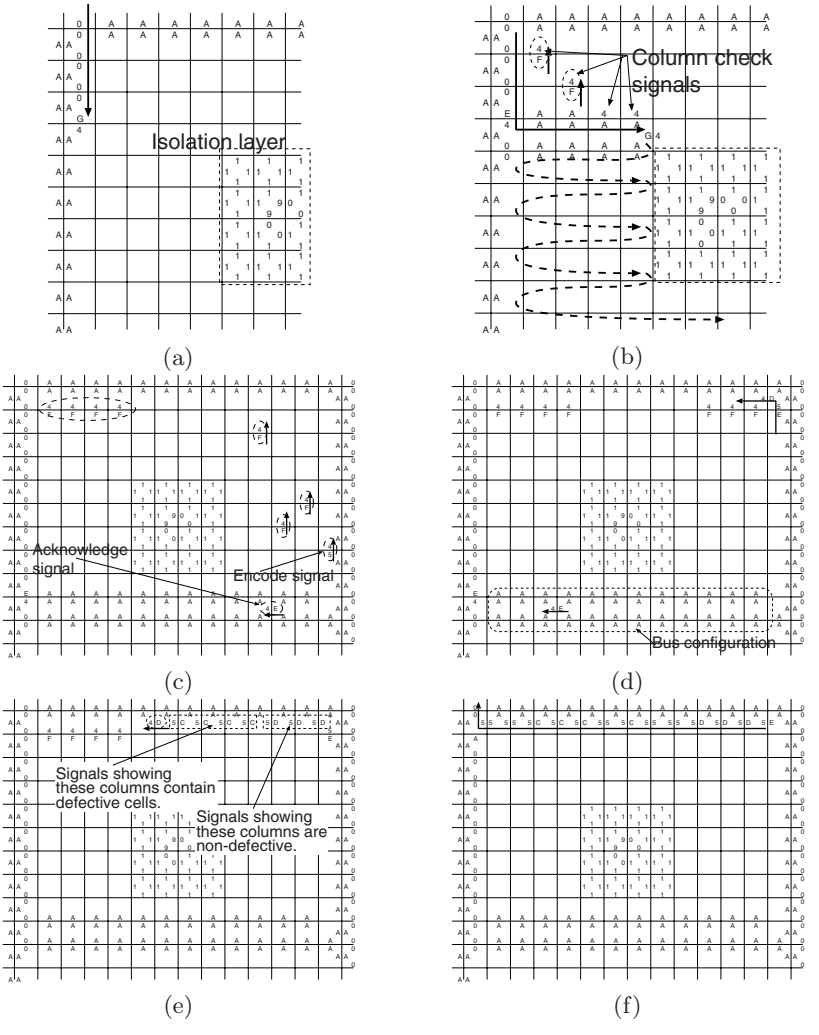



**Fig. 12.** Scanning free areas, followed by placing a bus. (a) *h*-signal injected from the top-left proceeds to the south. (b) When the encoded height is reached, the head starts to scan horizontally. Each step right generates a *column-check signal*, which, moving upwards, will be collected at the top of the area, but only at those columns that do not contain defects. Upon running into the isolation around defects, the head will move down one row, and starts a new scan. (c) When a whole row is free of defects it will be used for the bus line, which is created as part of the scanning process. At the right side of the area a so-called *Encode-signal* moves up. An *Acknowledge-signal* moves to the left to open up the area below the bus line for a similar process if a new *h*-signal is available. (d) The Encode-signal transforms the column-check signals at the top of the area, (e) into a form suitable to move out of the construction area and (f) move back into the constructor.

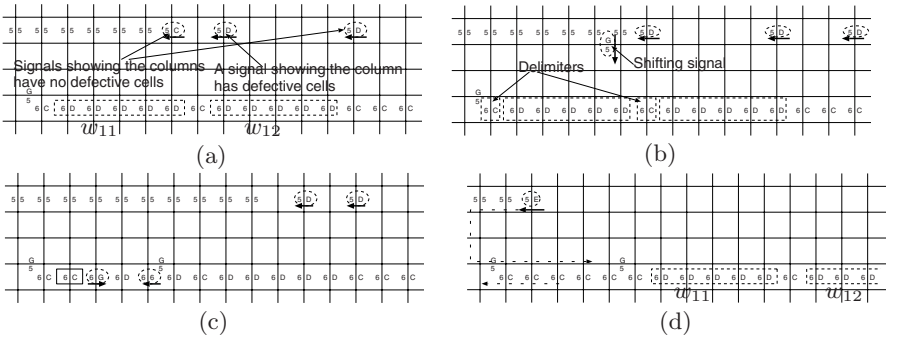



**Fig. 13.** Preparation for placement of circuit modules. (a) The train of signals encoding the positions of defect-free columns enters the constructor. Each of these signals encodes the absence resp. presence of defects in its corresponding column. (b) A signal encoding defects in a column generates a *Shift-signal*, which is dropped onto the $w$-signal below. (c) The Shift-signals cause the $w$-signal to be internally shifted, such as to reflect the proper positioning of modules in defect-free areas. (d) The train of signals and the dropped Shift-signals are cleaned up after use. Also, the $w$-signal starts to move toward the construction area to place the writing head in the appropriate position to begin processing of the $p$-signal.

is made possible in this case by signals having the ability to autonomously find their way around defects, as in [5]. Yet another case occurs between modules $M_{11}$ and $M_{21}$ (as well as between $M_{12}$ and $M_{22}$). Even though there are no defective areas between these modules, they are misaligned due to their forced placement between defective areas that are not aligned properly. Communications between these modules take place through a *Bus line* configuration, which is described in Section 3.2.

### 3.1 Scanning Non-defective Areas and Placing Modules

To place modules in the non-defective areas, we need to investigate the sizes of these areas and adjust the constructions signals such as to make the modules fit. Directly below the area in which a row of modules is to be placed we place a bus line, as part of the construction process. The constructor doing all this is attached at the top of the construction area (Fig. 10), in which the construction signals are injected in an appropriate order.

The constructor and the signals in it are shown in Fig. 11. In the initial stage, the signals are still static: they do not move until they receive an appropriate start signal. There are three types of signals: $h$-signals, denoted by $h_i$, express the maximum height in unary coding of the row $i$ of modules to be placed; $w$-signals, denoted by $w_{ij}$, express the width of module $M_{ij}$ in unary coding; and $p$-signals, denoted by $p_{ij}$, express the circuitry in module $M_{ij}$ (see Fig. 11).

These signals are processed at the writing head: they move the head in the construction area and direct the head to write appropriate information in the cells of the CA. The written information usually has the form of a stable pair of states in adjacent cells.

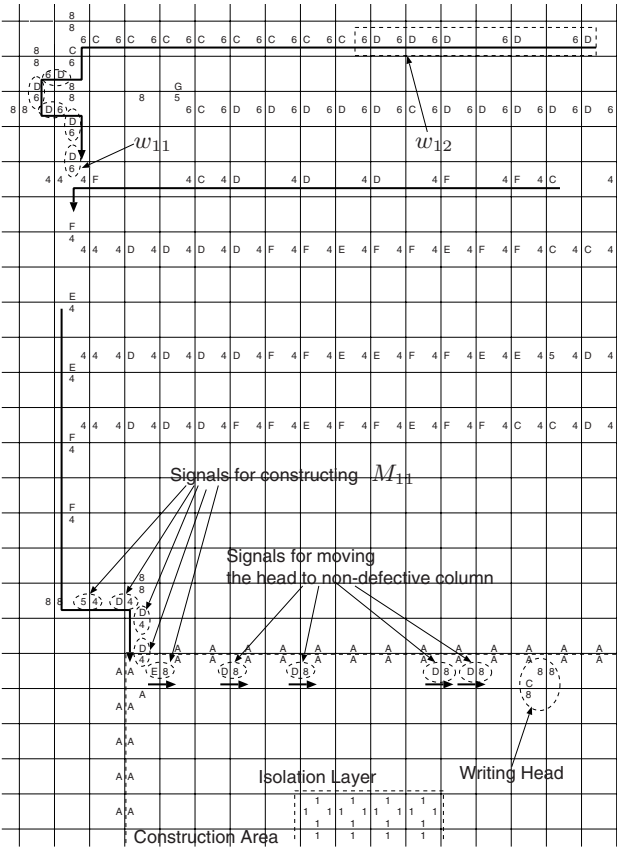



**Fig. 14.** Positioning of head by $w$-signals, followed by the placement of modules by $p$-signals

To place bus lines in the construction area, the constructor will scan the area for defects (Fig. 12).

Moving downward, this scanning process places a bus line at the first completely empty row of cells that it finds (for details see caption of Fig. 12).

An important product of the scanning process is a train of signals that encodes for each column of cells in the area whether there is a defect. This train of signals moves back into the constructor, and is placed in a position above the corresponding construction signal $w$, with the purpose to encode the positions of the defect-free areas in $w$ (Fig. 13).

The actual construction of the modules on the cell space starts with a $w$-signal moving into the construction area to position the writing head. The $p$-signal following directly after this will then result in the modules being written at the appropriate position, i.e. in a defect-free area (Fig. 14). Details of the construction process can be examined on the web page in [9].

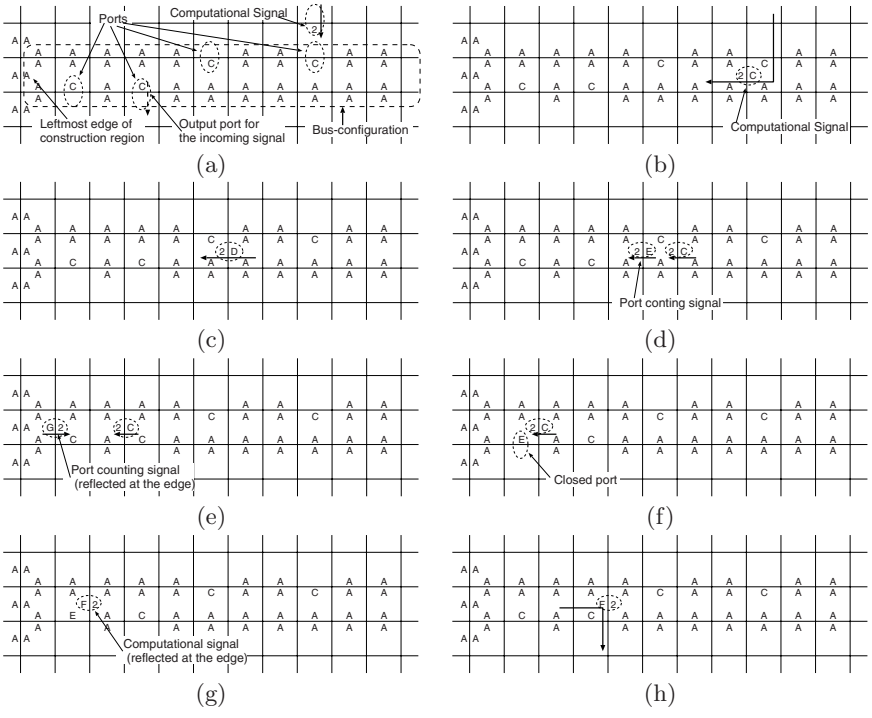



**Fig. 15.** A computational signal going through a bus. (a) Basic structure of a bus line. In this case the bus line contains two ports at its top and two ports at its bottom. (b) Signal entering right-most port at the top. This signal is supposed to exit from the right-most port at the bottom. (c) Signal moves along the bus line. (d) When encountering a port at its right, the signal emits a counting signal at its head. (e) Counting signal travels left and is reflected by the left side of bus. (f) The reflected counting signal closes the next port at its right and is annihilated. (g) The computational signal is also reflected, and it passes closed ports at its right, in the process opening them for the next signal. (h) Upon encountering an open port at its right, the computational signal exits via this port. By using counting signals to keep track of the number of ports to be passed, the computational signal can always find its appropriate output port.

### 3.2 Computation by the Modules

Once constructed, the modules can receive a computational signal, which will start the computation. Since this process is similar to that described in [5], we will not go into the details here, except for describing how communication via the bus line takes place, due to its novelty. The bus line contains ports that line up with the ports of the constructed modules. These ports on the bus line have been placed there by the construction process. A pair of ports on the bus line—one at the top of the bus line and one at the bottom—then connects two terminals, thus forming a connection between the corresponding modules. Transition rules are designed such that signals always move between matching pairs. The details of this process are outlined in Fig. 15. Again, details are on the web page in [9].

## 4 Conclusion

We have presented a novel scheme for defect tolerance on STCA that is able to flexibly place circuitry in cell space. The scheme is able to cope with defects in the construction area, but not with defects in cells used by the constructor itself or cells over which signals pass to and from the constructor, at the edges of the construction area. The cells that should be defect-free may be realized by reliable technology in implementations.

A novel element in our construction is a bus line configuration, which is used for communication between modules that are vertically unaligned. Due to these improvements, the circuit can be subdivided into modules in two-dimensional space and can be placed in non-defective areas more efficiently than in the previous scheme in [5].

The number of states per partition is 18 and the number of transition rules is 342. Both numbers are much larger than those in [5]. One reason for this is due to the use of an increased number of signal types and the need to appropriately control all these different signals. We expect that this number can be reduced if the bus line configuration can be applied not only for vertical connections between modules, but also for horizontal connections, since this would make redundant the previously proposed mechanism in [5] to guide signals around defective areas in an autonomous way.

## References

1. Biafore, M.: Cellular automata for nanometer-scale computation. Physica D 70, 415–433 (1994)
2. Durbeck, L.J.K., Macias, N.J.: The cell matrix: an architecture for nanocomputing. Nanotechnology 12, 217–230 (2001)
3. Peper, F., Lee, J., Adachi, S., Mashiko, S.: Laying out circuits on asynchronous cellular arrays: a step towards feasible nanocomputers? Nanotechnology 14, 469–485 (2003)
4. Lee, J., Peper, F., Adachi, S., Morita, K., Mashiko, S.: Reversible computation in asynchronous cellular automata. In: 3rd Int. Conf. on Unconventional Models of Computation 2002, pp. 220–229. Springer, Heidelberg (2002)
5. Isokawa, T., Kowada, S., Takada, Y., Peper, F., Kamiura, N., Matsui, N.: Defect-Tolerance in Cellular Nanocomputers. New Generation Computing (2007)
6. Takada, Y., Isokawa, T., Peper, F., Matsui, N.: Universal construction and self-reproduction on self-timed cellular automata. Int. J. of Modern Physics C 17(7), 985–1007 (2006)
7. Peper, F., Lee, J., Abo, F., Isokawa, T., Adachi, S., Matsui, N., Mashiko, S.: Fault-Tolerance in Nanocomputers: A Cellular Array Approach. IEEE Trans. on Nanotechnology 3(1), 187–201 (2004)
8. Morita, K.: A simple universal logic element and cellular automata for reversible computing. In: Margenstern, M., Rogozhin, Y. (eds.) MCU 2001. LNCS, vol. 2055, pp. 102–113. Springer, Heidelberg (2001)
9. `http://www.eng.u-hyogo.ac.jp/eecs/isokawa/acri08`