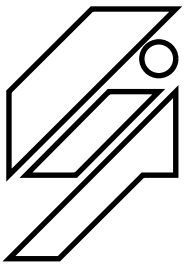


Laboratoire de l'Informatique du Parallélisme

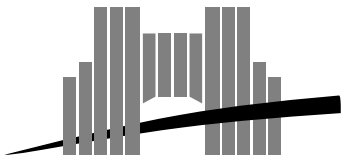
Ecole Normale Supérieure de Lyon
Unité de recherche associée au CNRS n°1398

High Speed Implementation of Cellular Automaton

Marc Daumas
Jérôme-Olivier Durand-Lose
Louis-Pascal Tock

Janvier 1996

Technical Report N^{o} 96-01



Ecole Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : (+33) 72.72.80.00 Télécopieur : (+33) 72.72.80.80
Adresse électronique : lip@lip.ens−lyon.fr

High Speed Implementation of Cellular Automaton

Marc Daumas
Jérôme-Olivier Durand-Lose
Louis-Pascal Tock

Janvier 1996

Abstract

Theoretical research on cellular automata aims at designing a class of new powerful tools for a wide range of applications. Yet, nothing is possible without efficient implementation. We present in this work an overview of the implementation of Wolfram rule 54 cellular automaton on reconfigurable hardware. Performances of the design are analyzed and perspectives are drawn on near-future cellular automaton implementations.

Keywords: FPGA, Computer Architecture, Cellular Automaton

Résumé

Les recherches théoriques sur les automates cellulaires ont pour but de définir un classe d'outils puissants qui puissent être utilisés par un grand nombre d'applications. Néanmoins, rien n'est possible sans une implantation efficace de ces automates cellulaires. Nous présentons dans ce travail l'implantation de l'automate 54 de Wolfram sur un circuit reconfigurable. Les performances du circuit sont analysées afin de présenter des perspectives sur les implantations possibles dans un futur proche grâce à cette technologie.

Mots-clés: Prédifusés Actifs, Architecture des Ordinateurs, Automates Cellulaires

High Speed Implementation of Cellular Automaton

Marc Daumas[†], Jérôme-Olivier Durand-Lose[*] & Louis-Pascal Tock[‡]

[†]Laboratoire de l'Informatique du Parallélisme - CNRS
École Normale Supérieure de Lyon
Lyon, France 69364

[*]Departamento de Ingeneria Mathematica
Universidad de Chile
Santiago, Chile

[‡]Université Claude Bernard de Lyon
Lyon, France 69622

Abstract

Theoretical research on cellular automata aims at designing a class of new powerful tools for a wide range of applications. Yet, nothing is possible without efficient implementation. We present in this work an overview of the implementation of Wolfram rule 54 cellular automaton on reconfigurable hardware. Performances of the design are analyzed and perspectives are drawn on near-future cellular automaton implementations.

Introduction

Implementations of cellular automata (CA) may be used for experimentation or as production tools. There are two classes of implementation: the programs from the first class generate a time-space diagram of the evolution of the automaton; the ones from the second class are only evolving toward a final steady state of the automaton. In any case, the speed of the program is critical and many simple automata are not used only because they could not be implemented efficiently.

In the 80's, Toffoli and Margolus have designed the CAM-6 card for IBM-PC compatible to simulate 1D and 2D cellular automata with performance comparable to the CRAY-I [4]. They offer a software library to drive the chip and build the space-time diagram along with some utilities. Their card is able to simulate a 256×256 array of binary cells with the 8-nearest-cell neighborhood and generate new data at the rate of about 6 *M bit per second*. But this hardware is especially designed for CA.

Programmable Active Memories (PAM) including the Digital PeRLe 1 board [1, 2] offer a good platform to implement cellular automaton. The PeRLe board is a general purpose reconfigurable hardware coprocessor. It is composed of a 4×4 grid-connected matrix of Xilinx XC 3090 chip Field Programmable Gate Arrays (FPGA) [6].

The cellular automaton based on Wolfram's rule 54 is a two-state ring-connected automaton with a relatively small neighborhood [5]. We have implemented 1024 cells on the PAM. Each cell updates every 125 ns, this leads to the global computation power of 8 *G cell-update per second*. The program generates the time-space diagram of the automaton for any input and as many as 10^5 transitions in a matter of 1 second.

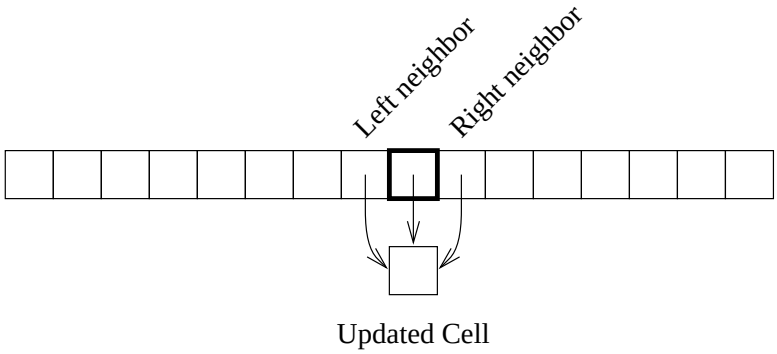


Figure 1: Cellular Automaton Architecture

State 0	Left neighbor 0	Left neighbor 1
Right neighbor 0	1	1
Right neighbor 1	1	0
	New state	

State 1	Left neighbor 0	Left neighbor 1
Right neighbor 0	0	0
Right neighbor 1	0	1
	New state	

Figure 2: Wolfram Rule 54 Truth Table

In Section 1, we present W54 automaton and Digital PeRLe 1 board. Then, in Section 2, we present the implementation of W54 on the PAM and figures of performance. Finally, we suggest directions for future research toward building both experimental automata and applications.

1 Environment

1.1 Wolfram Rule 54

Wolfram widely studied all one dimensional binary cellular automata with neighborhood { -1,0,1 }. There are exactly 256 such automata. This number is deduced from the concatenation of the images of the height elements of $\{0,1\}^3$.

W54 is a one dimension cellular automaton. The state of each cell is only stored with one bit. The update rule of the automaton involves only the left and right closest neighbors (see figure 1). The rule really involves all neighbors and is not trivial to compute. W54 automaton is a good representative to 3-neighbors, 2-state CA. Moreover, its global behavior leads to interesting space-time diagrams.

The truth table of the rule is presented in figure 2. It can be coded with the boolean expression below.

$$\textit{updated state} = \text{not } \textit{active state} \text{ xor } (\textit{left state} \text{ and } \textit{right state})$$

The automaton is known for presenting some pattern on specific configurations. The configuration ${}^{\omega}(0111)^{\omega}$ is two-periodic, with the associated configuration ${}^{\omega}(1000)^{\omega}$. Figure 3 presents a small time space diagram of the automaton for this initial state expanded below, for the diagram we have coded cell state 1 with a black square.

$$\ldots 0111011101110111011101110111\ldots$$

One of the interesting fact on W54 automaton is that, if we cut and paste two infinite parts of this configuration, the *border* will regularly move right or left. It will behave like some sort of particle with some interesting properties when colliding: the configuration just keeps going or remains stuck. It may happen that a third particle restores the two previous ones; many other types of interesting behaviors are observed.

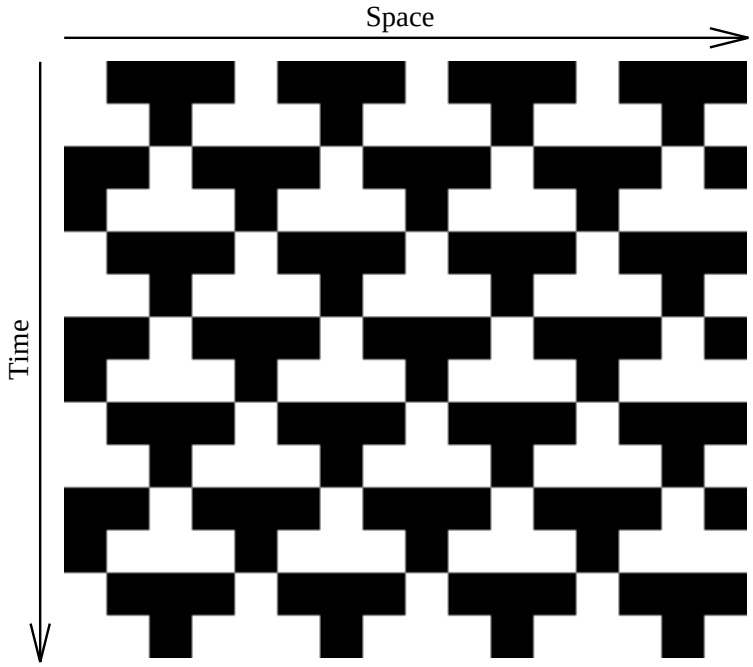


Figure 3: Example of W54 Evolution

The following example shows that the W54 automaton is not injective and therefore not reversible: these two configurations evolve towards the same state.

...0101010101010101010...	...0000000000000000000...
...0000000000000000000...	...0000000000000000000...

1.2 Board Architecture

The Digital PeRLe-1 board is one example of Programmable Active Memory (PAM). It is based on Field Programmable Gate Array (FPGA) technology and can be used as a universal hardware coprocessor which is coupled to a standard host computer. The board loads its configuration (the nature of the automaton) and exchanges data to be computed (initial state and intermediate states) with the host through the fast TURBOchannel interface. The PeRLe 1 board designed as a general purpose reconfigurable hardware coprocessor featured very good behavior to simulate cellular automata.

FPGA Xilinx XC3090 The board is based on the Xilinx XC3090 chip. It is a high density IC that can be configured by the user and that belong to the Logic Cell Array (LCA) family [7].

User-specified logic functions are performed by the 20×16 Configurable Logic Blocks (CLB) of the chip. Data can come in to or go out from the chip via 144 I/O Blocks (IOB). Three kinds of programmable interconnection resources are provided to connect IOBs and CLBs, and CLBs among themselves: general purpose interconnections, direct connections and long lines. Figure 4 presents the structure of an LCA.

The 20×16 CLBs of the chip perform some user-specified logic functions chosen from two functions of up to 4 variables with one variable common to both functions, any function of 5 variables or some functions of seven variables. Each CLB features five logic inputs and two flip-flops (see figure 5).

General purpose interconnections use switching matrices to drive a signal over the CLB array. They are supported by five horizontal and five vertical wires located between the rows and columns of CLBs and IOBs. The number of interconnect combination each time a vertical vector crosses an horizontal vector is limited to 20.

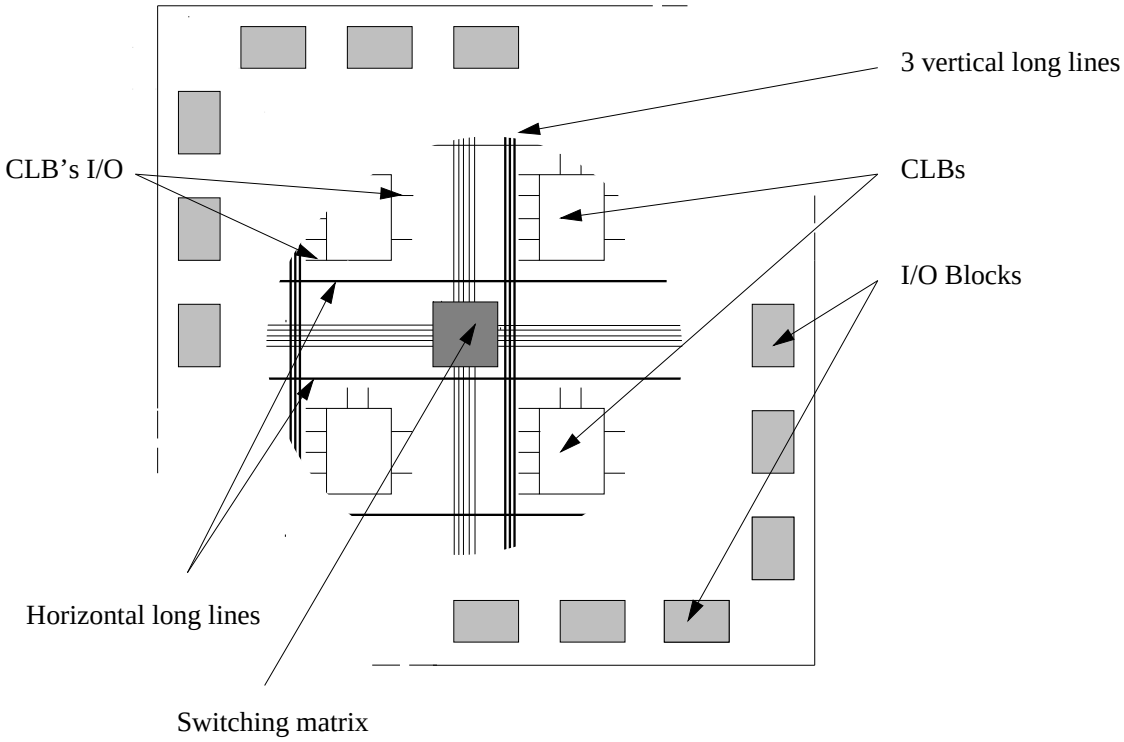


Figure 4: CLB's Signals.

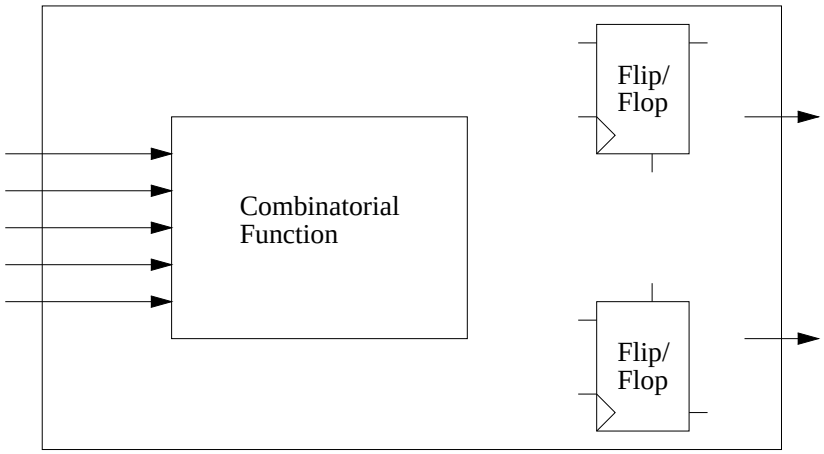


Figure 5: Configurable Logic Block.

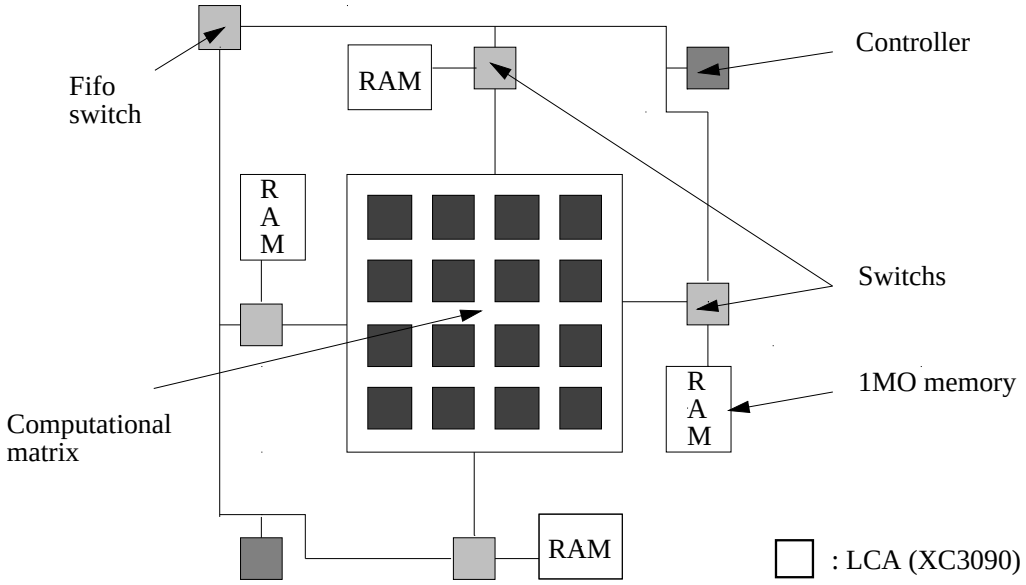


Figure 6: DEC PeRLe-1 Board.

Direct interconnections are used to route a signal between adjacent blocks. The propagation delay is minimal and there is no use of general purpose interconnection. Circuits involving such connections will maximize their speed. So, automata using 4-nearest-cell neighborhood will have the fastest implementation.

Internally, data are spread over the chip via horizontal and vertical long lines which drive a signal for a long distance (allowing a high level of fan-out). The long lines enable one to connect every CLB of a row or a column. Each CLB's column has three vertical long lines, and each CLB's row has two horizontal long lines. Three-state buffers also allow the use of horizontal long lines to form on-chip wired-OR and multiplexed buses.

Each IOB includes input and output storage elements and I/O options to deal with the many way inputs or outputs are performed: direct, registered, inverted, 3-state configured...

DEC PeRLe-1 This board [8] consists of 4×4 Xilinx XC3090 chips organized as a computational matrix and 4 banks of fast static memory. Seven more Xilinx XC3090 chips are present on the board: the 4 *switches* are dedicated to implement data routing and preformatting functions; the *fifo switch* connects to the board I/O system; the two *controllers* command the data path and the board electric control signals. Communication with the host is done through two 32-bit wide FIFO devices, one for each direction. An overall DEC PeRLe-1 board architecture view is given in figure 6.

All these basic programmable resources are interconnected. They allow data distribution over the board and permit one to establish a variety of different data paths. Such flexible features permit us to efficiently map the automaton to the board.

The major data processing is made by the 4×4 array of LCAs called *computational matrix*. Each LCA of the matrix follows a regular scheme of interconnection (see figure 7). Three kinds of user-programmable interconnection resources are provided: the matrix direct connections which are used between adjacent LCAs, the matrix buses which connect one side (north, east, west or south) of all LCAs in the same row or column, and the matrix ring buses which connect all the LCAs.

The propagation delays are different according to the interconnection resource used. The matrix direct connection features the fastest connection: 24 ns. The matrix bus has a propagation delay of 28 ns, and the matrix ring bus has a propagation delay of 43 ns. High performance circuits will avoid the matrix ring buses and usually involve nearest neighbor communications.

The surrounding logic is driven by five switches and two controllers which provide the user with a powerful data path control. These user-programmable devices manage data inputs and outputs, the four banks of

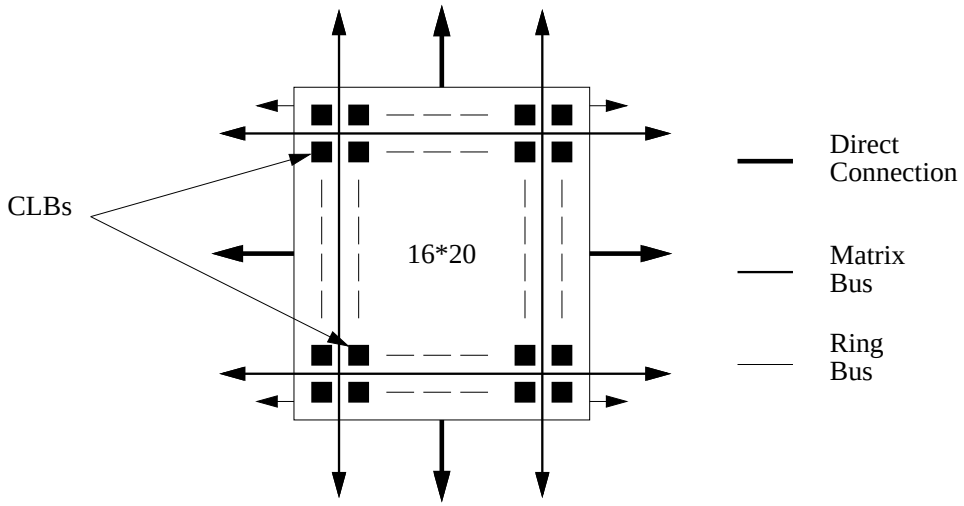


Figure 7: Interconnection Scheme of one LCA.

memory, and can be used for data preformatting. The four switches are connected to the computational matrix via 64-bit wide matrix busses, and connected to the memory banks via 32-bit wide buses. Two 32-bit wide buses connect the switches to the two controllers and the fifo switch.

The board features four banks of 1 MBytes static memory. They are 32-bit wide and fast enough for working with the computational matrix without altering performances. They are accessed through the switches.

A 32-bit wide FIFO link connects the board to the host. This link can be accessed through a DMA channel to exchange data at a speed of 100 Mbytes/s. Lastly, the board is cadenced by a user-programmable clock able to reach 40 MHz in case of high speed designs.

1.3 Programming the Board

The PeRLe1DC library [9] provides the PeRLe 1 board designer with an easy way to configure the board. The library is written in C++ and takes advantages of many features of the language.

Logic Logic is specified in terms of boolean equations using constants, variables, registers for synchronous circuits, clocks, multiplexers and the classic logic operators like logical and, logical or, etc...

Data Structure Advanced data structures such as static and dynamic vectors of boolean variables, equation handlers (to create classes returning complex boolean expressions), design hierarchy involving C++ inheritance are imported from the C++ possibilities.

Place/Route Automatic logic placement and routing of the data on the LCAs is done by the Xilinx CAD tools. Yet any part of the circuit can be placed and/or routed by the user. It is also possible to tag some critical signal for a careful route by the Xilinx tools.

I/O Pin Assignment A signal can be assigned to an internal buses of the board as well as a pin of a chips providing the user with full data path control.

Designs made with this library can be simulated at different stages of the definition. The generated files are compatible with the Xilinx net-list format and the Xilinx CAD tools are used at early stages of the design.

2 Implementing W54

We have directly mapped the automaton ring to the board. With 5120 CLBs on the computational matrix, 1024 cells of 4 CLBs each has been implemented, leaving 1024 CLBs for *glue logic*. One elementary automaton

cell was built and replicated across every CLBs of the matrix. The challenging part of this work was to connect the different cells: we have introduced an original scheme to load the initial state from the host and to save each intermediate state on a disk file.

Our goal was to update the 1024 ring connected cells of the automaton each clock cycle and to store their state stepwise every n clock cycles. The process is divided into three parts:

1. Vector initialization.
2. Updating n time the automaton state x[i] for $0 \leq i < 1024$.

$$x[i] = f(x[i], x[i-1], x[i+1]) \; where \; f(a,b,c) = \overline{a} \; xor \; (b \; and \; c).$$

3. Saving the vector's states and returning to step 2.

The boards never gets a chance to stop producing some new results. Yet the board works as long as its output buffer is not full. When the fifo output buffer is full, a mechanism integrated to the board stops the clock until some data are read from the board. This mechanism ensure that the board produces data as long as the client program on the host reads the results.

2.1 Board Control

At the board level, we distinguish between three states indicated each by an active signal: the automaton is working (`work`); it is loading its initial state (`enL`); the current state is presently being saved (`enS`).

The board is configured as a 32×32 grid of elementary cells that we will describe latter. During a load or save operation, the state of the 32 cells of one column of the grid are send from the west switch to the matrix or *vice versa*. The operation is carried away in 32 cycle by addressing each cycle a new column of the matrix.

During the first 32 clock cycles after the initialization of the board, a 1 shifting through a register activates the signals `enR[i]` (enable read) for each column, $i = 0 \ldots n-1$. The `enL` (enable load) signal is also activated, hence the input fifo reads one 32 bit word from the connection with the host each clock cycle. The word is sent to the west switch which dispatches it to the matrix. Each cell of the i^{th} column activated by the signal `enR[i]` save the corresponding bit of the word as its new state. The remaining cells of the matrix are kept unchanged. This first step achieves the loading of the automaton initial state.

After-while, the board enters in the working state. The `work` signal is activated; all the cells are updated every clock cycle. A counter keeps the number of cycle spend working. When the automaton has updated n consecutive clock cycles, the automaton is stopped. The counter is reseted and the board starts sending the automaton state to the host to be saved on a disk file.

In a way similar to the first step, the 32 `enW[i]` (enable write) signals are activated, one per clock cycle, and the automaton state is put in the fifo output which is activated by the `enS` (enable save) signal. When the state is finally saved, the board automaton is reactivated (see figure 8).

The process loops until the fifo output buffer is full. The image of one state of the automaton takes up 32 of the 256 words available in each fifo buffer. Eight states can be stored together in the buffer before the process stops. Figure 9 presents how the signals are generated by process. The actual implementation of the `work` signal uses a decrementing register.

2.2 Elementary Cell

Each component of the ring is set up as an *elementary cell*. The elementary cell stores the state `x` of the corresponding cell of the automaton. Three control signals define the instant behavior of a cell from its point of view, they are shared by all the cells of a same column inside the 32×32 matrix: `work[c]`, `enR[c]` and `enW[c]`. One bus, `xload[r]`, is shared by all the cell of one row inside a chip. A chip hold a sub-matrix of 8×8 cells.

The logic code managing the cell is presented bellow using a syntax closed to the PeRLe1DC library. Figure 10 presents such a cell.

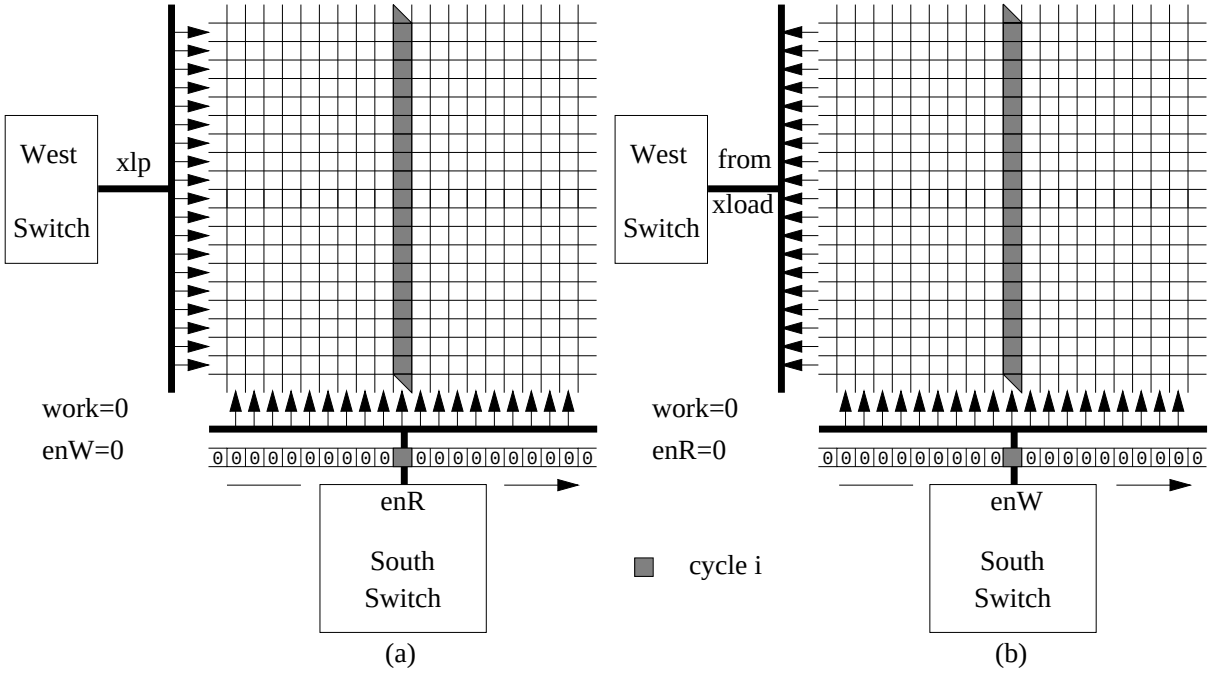


Figure 8: Load and Save Operations.

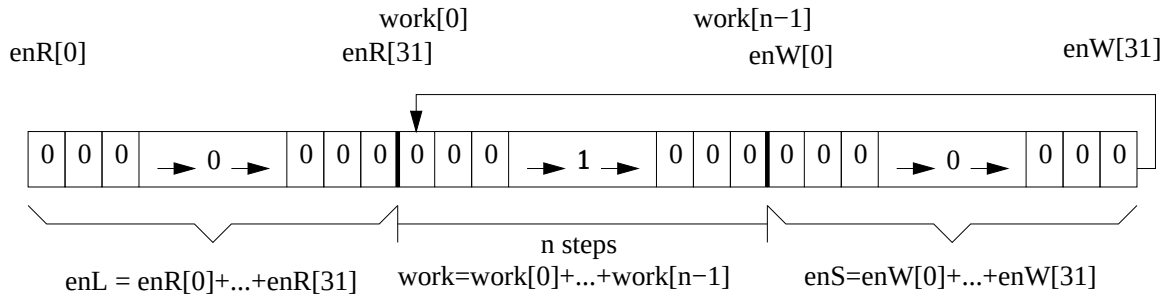


Figure 9: Generating Signal Process.

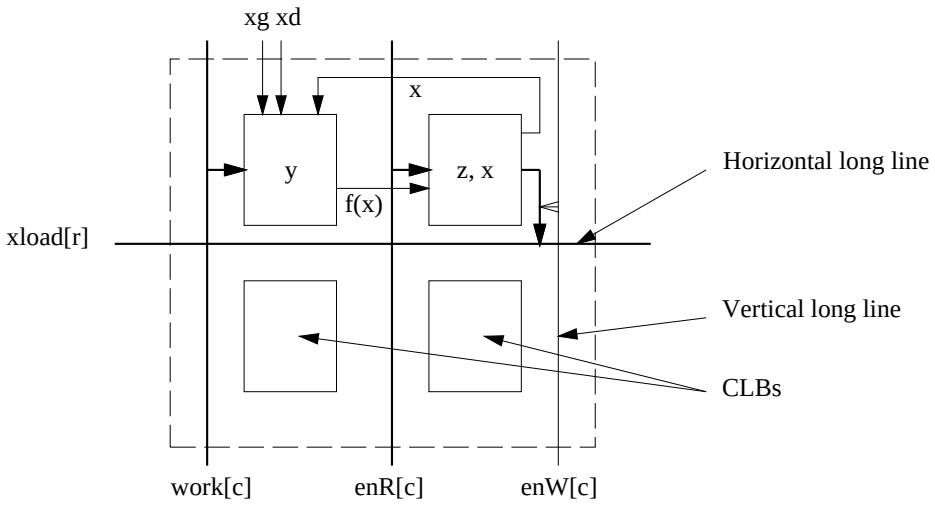


Figure 10: Cell(r,c).

```
y         = mux (work[c], f(x,  xg, xd), x); // Normal transformation
z         = mux (enR[c], xload[c], y);       // Load mode
x         = reg (z);                         // Synchronous operation
xload[r] += TriState (x, ~enW[c]);           // Save mode
```

When `work` is active then the cell operates in normal mode, the state `x` is updated each clock cycle (125 ns) using the W54 automaton rule. Otherwise the state `x` of the cell is frozen. When `enR` is active the cell state is read from the three-state bus `xload`.

Every time the intermediate state needs to be stored on the host disk file, the `enW` signal of the cell is activated. The current state value is written on the horizontal three-state bus `xload`. In the mean time, the access to this bus for the other cells of the same line is disabled.

One can have different automata simply by changing the `f` function. All the ring connected two state automata studied by Wolfram can thus be implemented efficiently without the need to design a new circuit.

2.3 Ring Connection

We have implemented three different elementary cell connection pattern for a Xilinx XC3090 chip depending on the position of the chip in the computation matrix. The pattern are named L, N and U depending on their form (see figure 11). The L pattern features only direct vertical connections between the elementary cells. The two other patterns connect some vertical line together on the upper row (LCA N) or on the lower row (LCA U).

Connecting all the chips using only the fast direct connection of the PeRLe 1 board as presented figure 12 builds a 1D ring on the 2D computational matrix. In order to close the ring, a link is set up between the ends of the vectors on the upper left and the upper right corner.

The data transmitted by the matrix in a load or save operation is the direct transcription of the automaton state. The state vector is coded on 32 words of 32 bits as $x_0 \ldots x_{31}, x_{32} \ldots x_{63} \ldots x_{1023}$. One word among two has to be reversed in order to adapt to the physical ring mapping on the computational matrix. This treatment occurs in the west switch. It uses a toggling control activated only in the load or save mode.

2.4 IO Management

Figure 11 shows the external signals received by each LCA from its point of view. Each chips sees 8 bit wide signals, but at the board level the buses are merged to a 32 bit wide bus connected to the corresponding switch. The control signals are sent on the vertical buses of the board by the south switch. As they are

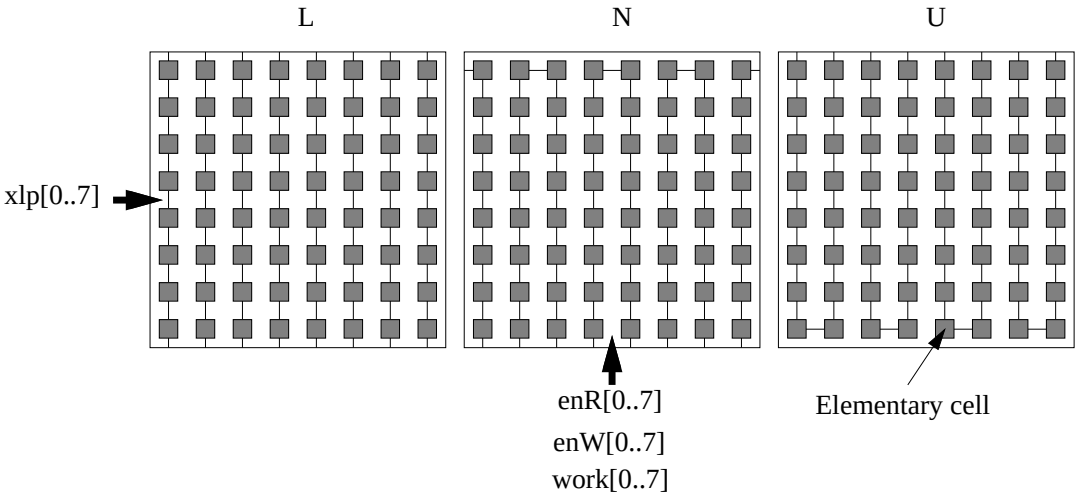


Figure 11: LCA Patterns.

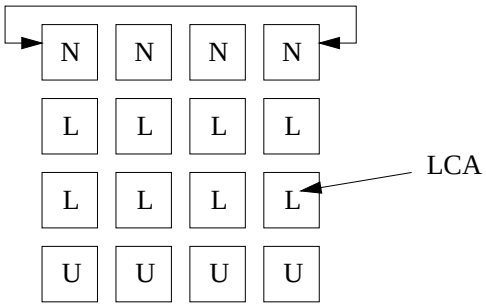


Figure 12: W54 Mapping.

transmitted on long line of the chip, they reach every elementary cell of a given column (8 columns per LCA). The `xlp` signals, which are used during the load and save operations, are driven in each chip through the horizontal bus `xload`.

Loading the state in the automaton, the control signal works has a cursor, and the datum to be read is found on the horizontal bus. No cell writes on the `xload` bus. The information is read from the `xlp` bus on the board. Saving the state, only one column of cells of the entire board is active. Hence, only one column of 4 chips out of the 16 chips holds some active cells. The `xload` bus of these chips is not driven by the `xlp` signal of the board. The buses are set by the active cells. The buses output connection to the `xlp` signal on board are activated, and the `xlp` signal is set according to the state of the active cells. The other chips, including the west switch monitor the value of `xlp`, hence the value of the active cells is retrieved on the switch.

2.5 Pipelining

The commuting time of the elements of the design is very small. Yet the critical path of one load or save operation involves transistors from the south switch down to the fifo buffer transiting inside the computational matrix. Sending a signal from one chip to another on the board is a slow operation compared to the power available inside every chip. To be able to use a high clock frequency, we have pipelined the IO operations.

By introducing some latch in the data path from the fifo switch to the computational matrix, we have been able to sustain operation on a much higher clock frequency. Some logic has been added to the switches to make sure that the signals that were synchronous without the retiming barriers arrive together. For example the control signals for the output fifo are delayed a few cycle for the information to arrive from the computational matrix to the fifo switch. No retiming barrier has been introduced inside the computational matrix because it would require state prediction to cross the retiming barriers. Although state prediction was simple with W54, we have restrained from implementing it. Our generic implementation shows that any cellular automaton up to 1024 cells with a relatively small neighborhood (up to 8 neighbors) can be simulated with comparable results.

The board produced excellent results on this problem, updating 1 GB/s. If we compare our implementation clock cycle of 125 ns to the clock speed usually obtained on the PeRLe board (cycles from 25 ns to 100 ns) and our link of 32 MB/s with the host to the maximum bandwidth of the DMA channel (100 MB/s) our architecture fits correctly the PeRLe board. We present in figure 13 a larger sample of W54 evolution on the standard configuration, and figure 14 illustrates the evolution of the automaton on a random configuration.

3 Conclusion

We have presented our implementation of Wolfram rule 54 cellular automaton. The target PAM architecture is Digital PeRLe 1 board. In this design, the size of the automaton was not critical, we have shown that a degree of integration twice or four times higher could still have been considered. The resolution of the screen limits the size of the time-space diagram that can usefully be computed.

We are moving toward automata that are able to recognize one given state and to trigger an action on this state. This functionality implemented in the two remaining CLBs could easily be used to automatically investigate the periodicity of a cellular automaton on a given state.

More complex automata are possible with a larger neighborhood. The communication switches are far from saturated and CA with a neighborhood of 4 or 8 will achieve equivalent performance. More promising are problems where the state of the automaton should only be saved after a large number of iteration. The saving could be triggered by a counter or by some dedicated cells of the automaton.

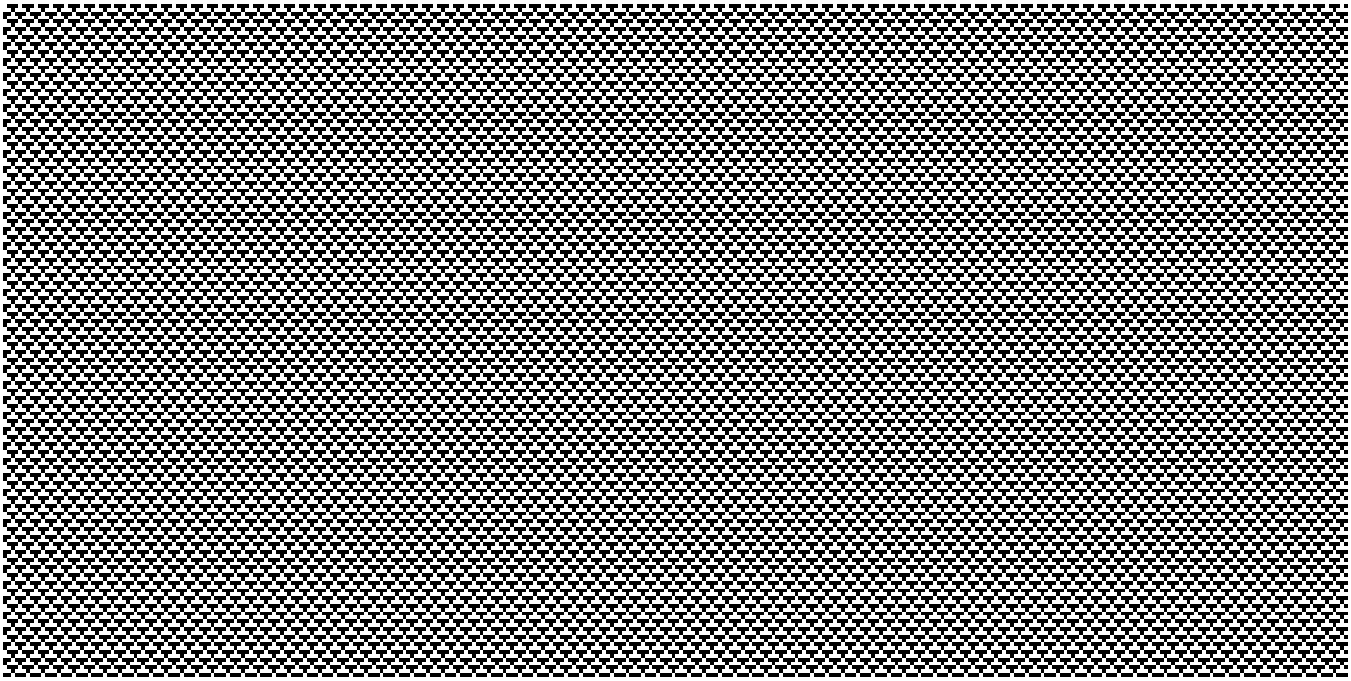


Figure 13: W54 Time-Space Diagram on Standard Configuration (350×175).

References

[1] P. Bertin, D. Roncin & J. Vuillemin, "Introduction to programmable active memories", *Systolic Array Processors*, Prentice Hall, also available from *Paris Research Laboratory*, PRL-RR 3, June 1989.

[2] P. Bertin, D. Roncin & J. Vuillemin, "Programmable Active Memories: a Performance Assessment", *Paris Research Laboratory*, PRL-RR 24, March 1993.

[3] B. Chopard & M. Droz, "Cellular Automata Approach to Non-Equilibrium Phase Transitions in a Surface Reaction Model: Static and Dynamic Properties", *Journal of Physics*, Math. Gen. 21, 1988.

[4] T. Toffoli & N. Margolus, "Cellular Automata Machine - A New Environment for Modeling", *MIT press*, Cambridge Mass, 1987.

[5] S. Wolfram, "Theory and Applications of Cellular Automata", *World Scientific*, 1986.

[6] Xilinx Inc., "The programmable gate array data book", *Product Briefs, Xilinx*, 1987.

[7] Xilinx, "The Programmable Gate Array Data Book", *2100 Logic Drive, San Jose, 95124 California*, 1992.

[8] P. Bertin and P. Boucard, "DECPeRLe-1 Hardware Programmer's Manual", *Digital Equipment Corporation, Paris Research Laboratory*, 1993.

[9] Hervé Touati, "Perle1DC: a C++ Library for the Simulation and Generation of DECPeRLe-1 Designs", *Digital Equipment Corporation, Paris Research Laboratory*, 1993.

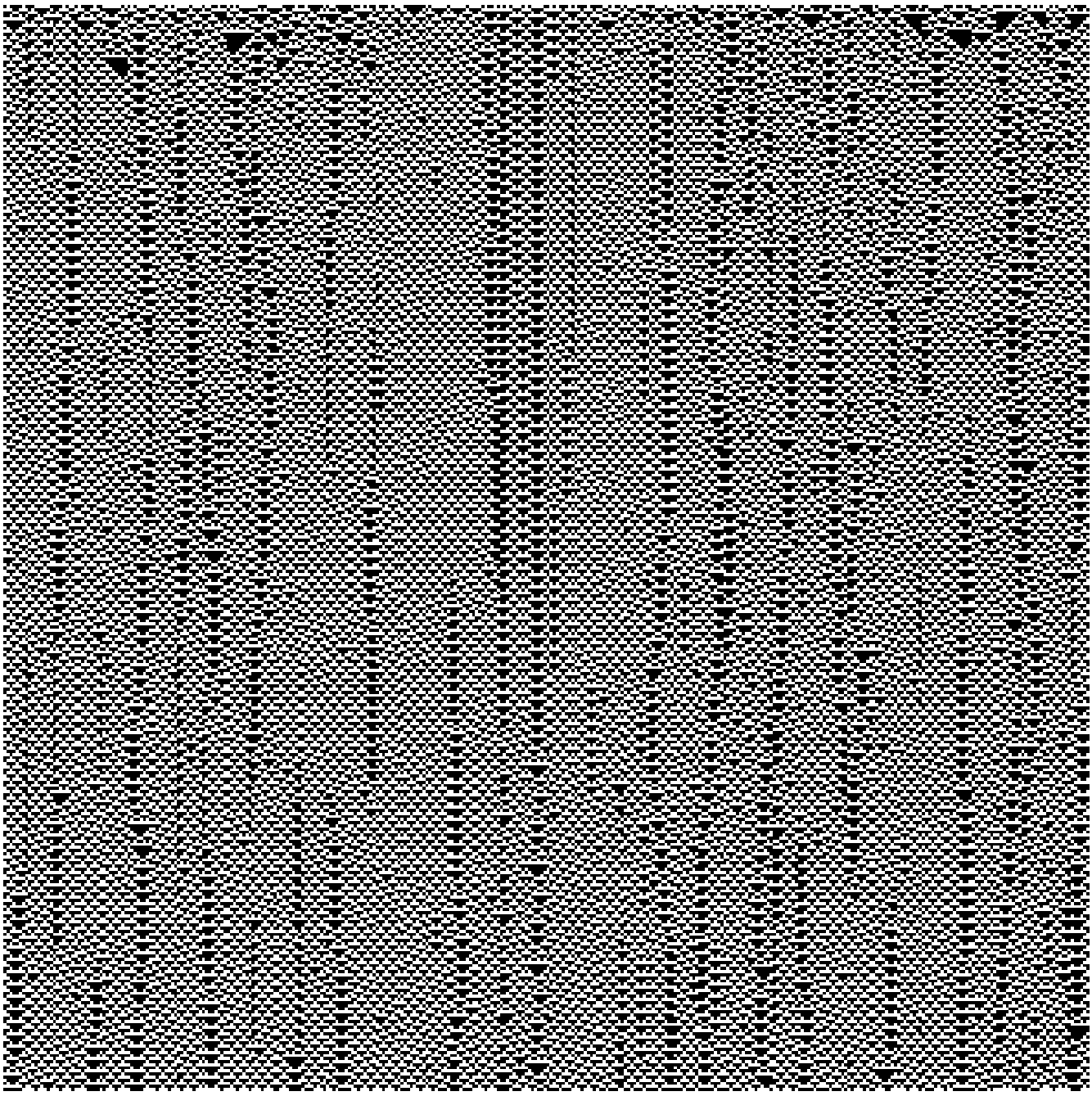


Figure 14: W54 Time-Space Diagram on Random Configuration (350×350).