RAIRO-Inf. Theor. Appl. **40** (2006) 177-206
DOI: 10.1051/ita:2006002

DIFFERENT TIME SOLUTIONS FOR THE FIRING SQUAD SYNCHRONIZATION PROBLEM ON BASIC GRID NETWORKS *

Jozef Gruska[1], Salvatore La Torre[2], Margherita Napoli[2] and Mimmo Parente[2]

Abstract. We present several solutions to the Firing Squad Synchronization Problem on grid networks of different shapes. The nodes are finite state processors that work in unison with other processors and in synchronized discrete steps. The networks we deal with are: the line, the ring and the square. For all of these models we consider one- and two-way communication modes and we also constrain the quantity of information that adjacent processors can exchange at each step. We first present synchronization algorithms that work in time n^2, $n \log n$, $n\sqrt{n}$, 2^n, where n is a total number of processors. Synchronization methods are described through so called *signals* that are then used as building blocks to compose synchronization solutions for the cases that synchronization times are expressed by polynomials with nonnegative coefficients.

Mathematics Subject Classification. 68Q80, 68Q10.

1. Introduction

The famous Firing Squad Synchronization Problem (FSSP), is an old problem, posed by Myhill in 1957 (in print in [19]). Using terminology of Cellular Automata, a line of n identical cells (finite state machines) is given that work synchronously and at discrete time steps. Initially, a distinguished cell (so called *General*) starts communication with its neighbour while all other cells are in a so called quiescent

* *Work partially supported by the grant "*Metodi Formali ed Algoritmi per la Verifica di Sistemi Distribuiti*", Università degli Studi di Salerno. The first author is also supported by the grants GAČR, 201/04/1153, MSM0021622419 and VEGA 1/0172/03.*

[1] Faculty of Informatics, Masaryk University, Brno, Czech Republic.

[2] Facoltà di Scienze Matematiche, Fisiche e Naturali, Università degli Studi di Salerno, Italia.

state. At each time step, any cell sends/receives to/from its neighbours some information about their states at the previous time step. The task is then to let all cells to enter simultaneously, and all for the first time, in the so called *firing time*, a special state called the *firing* state.

In the literature, many variations and solutions of the original FSSP have been considered. All early results focused on synchronization in minimal time: Minsky, in [18], showed that any solution to the FSSP requires at least $2n - 1$ time steps. Goto [5], Waksman [23] and Balzer [1] gave the first solutions in such a minimal time and Mazoyer, in [14], constructed a minimal time solution with the least number of states of cells to date: six. Moreover, in [1], it has also been shown that at least five states are necessary for any solution. That is, there is no solution with four or less states. Therefore, determining whether a five states solution exists is an open problem.

Several papers have also dealt with various variations of the FSSP. These variations concern the geometry of the network, number and positions of Generals, and computational and/or communicational constraints. In the following we briefly recall some of them. For example, the FSSP has been studied on a (one-way) ring [4, 11], and on arrays of two- and three-dimensions [8, 22]. All results in these papers focused on lower and upper bounds concerning the minimal time for synchronization. In the quite recent paper [9], the cells of the network are placed along a path in the two-dimensional array space, and a well-known combinatorial problem (for which only exponential algorithms are known) is reduced to the existence of an optimal solution of the FSSP on that path. In [21], several solutions of FSSP for Cayley graphs are given and, in [20], a particular class of graphs is studied having a solution in time $3r + 1$ or $3r$, where r is the longest distance between the General and any other node (the radius) of the graph. Concerning Generals, basic models have one General positioned at some end of the underlying interconnection graph. However, several variants have also been explored with the General at some other, or even arbitrary, position or/and with several Generals whose initial steps are synchronized. For example, the two-end General problem is dealt with in [16]. Some variants of FSSP have dealt with reversible CA (*i.e.*, backward deterministic CA), [6], and CA with a so called *number-conserving property* (*i.e.*, a state is a tuple of positive integers whose sum is constant during the computation) [7]. Other types of constraints, which have been considered so far, concern the amount of information exchanged between any pair of adjacent cells. In [12, 15], the network is a line of cells that can exchange at each step only one bit. That is at each time step each cell sends/receives only one bit of information to/from the adjacent cells, instead of its whole state. Finally, let us mention also a stimulating work of [2], where the FSSP is studied in a distributed setting (no global clock, but lock-step synchrony) with bounds on the number of faulty processors.

In this paper, we consider again the FSSP on several networks (a line, a ring, a square), and with one- and two-way communication modes, but with a new approach. Namely, we assume that specific firing time is given and we ask for

a synchronization algorithm working exactly in such a time. This is an interesting and challenging theoretical problem, which is also directly connected to the sequential composition of cellular automata. Given two cellular automata A and B, computing functions f and g, respectively, the sequential composition of A followed by B is the cellular automaton obtained in the following way: at first A starts on a standard initial configuration and after A finishes its computation, B starts using the final configuration of A as B's initial configuration. The resulting automaton clearly computes $g \circ f$. In order to compose two automata, A and B, it is necessary to synchronize all cells that will be used by B at the time A finishes computation of f.

Some of the results presented here are an improvement and a generalization of some of the results of [11–13]. Moreover, we present here a whole framework of *signals*. Informally speaking, signals are sets of cells that at a given time receive or send a particular state. We then define several basic signals (building blocks) and give rules to combine them to obtain new signals. This modular approach allows to design synchronizing algorithms in a natural way and also simplifies their understanding and descriptions. Moreover, we introduce here also, as a parameter, the number of bits that can be simultaneously transmitted at each step. We study, for example, networks where at each step a cell can transmit to each of its neighbours at most c bits, where $c \geq 1$.

As mentioned above, communication between adjacent cells can be in both directions or only in one direction. We thus consider either networks where a cell can exchange information with all its neighbours, or networks where for each cell, only a predetermined half of its neighbours can send information to it while the other half can only receive information from it (in such a way that information flow is unidirectional). In such a second case, to guarantee communication from a cell to all other cells, we consider only networks with some circular type connections.

For each network under consideration, we prove first a lower bound on the synchronization time, and then we prove its tightness by describing a synchronization with a number of steps that matches the given lower bound. We also obtain families of solutions for the considered variants of FSSP in several times $t(n)$, where n is the number of nodes of the network. The approach we follow is compositional: we first describe basic synchronizing algorithms and then we give general rules to compose synchronizations. The basic synchronizations are obtained in turn by composing elementary signals, which can be seen as fragments of cellular automata. A *synchronization* is thus a special signal obtained as a composition of simpler signals. Compositional rules, for both signals and synchronizations, include *parallel* composition, *sequential* composition, and *iterated* composition. We also state some sufficient conditions under which these rules can be applied. In a parallel composition, several synchronizations or signals start in parallel, all at the same time. In some cases, such a composition can be used to select among different synchronizations depending on the number of cells in the network. A sequential composition appends a synchronization (or a signal) to the end of another signal, possibly with a constant time offset. This way we are able to construct a synchronization in time $t_1(n) + t_2(n) + d$, for some $d \geq 0$, if there exist

synchronizations in time $t_1(n)$ and $t_2(n)$. If we are given two synchronizations, in time $t_1(n)$ and $t_2(n)$, respectively, the iterated composition consists of iterating $t_2(n)$ times the synchronization in time $t_1(n)$, thus obtaining a new synchronization in time $t_1(n) \cdot t_2(n)$. Compositions of synchronizations are used to determine synchronizations in a "feasible" time expressed by polynomials with nonnegative coefficients. Finally, we give a construction how to "inherit" synchronizations on two-dimensional networks starting from synchronizations of the corresponding linear networks. We show, for example, that an $(n \times n)$ array of cells can be seen as many lines of $(2n-1)$ cells (each of them having as endpoints cells $(0,0)$ and $(n-1,n-1)$) and a given synchronization on a line can be executed simultaneously on all these lines. Thus, we can synchronize an $(n \times n)$ array in time $t(2n-1)$, provided that there exists a synchronizing algorithm for a line of k cells in time $t(k)$.

As building blocks for compositional rules we present synchronizing algorithms that work in times: n^2, $n\lceil \log n \rceil$, $n\lceil \sqrt{n}\, \rceil$ and 2^n. To synchronize a line of n cells in a time $t(n)$ we first design some basic signals and then we compose them to obtain an overall signal that starts from the leftmost cell and comes back to it in exactly $(t(n) - 2n + 1)$ steps; then a minimal time synchronization starts, synchronizing n cells in time $t(n)$. To obtain a synchronization in time $t(n)$ of an array of $(n \times n)$ cells we use the following approach: we first synchronize a row in time $t_1(n)$ then start a synchronization in time $t_2(n)$ on all columns such that $t(n) = t_1(n) + t_2(n)$.

It is worth to notice that the composition rules also apply to the general case of $(m \times n)$ arrays. This way many synchronizations given for an $(n \times n)$ array can be extended to an $(m \times n)$ array, considering the time of the synchronization as a function of either m or n.

The rest of the paper is organized as follows. In Section 2 we give basic definitions and introduce the notation we will use throughout the rest of the paper. In Section 3 we give tight lower bounds on the time synchronization of different types of CA's and solutions in minimal time. In Section 4 the framework of signals is presented formally. In Section 5 several composition rules for synchronizations are defined. In Sections 6 and 7 solutions in the given times n^2, $n\lceil \log n \rceil$, $n\lceil \sqrt{n}\, \rceil$ and 2^n are given for two-way and one-way communication models, respectively. As an application of the compositional rules to obtain new synchronizations, in Section 8 we show how to obtain polynomial-time synchronizations on all of the considered models of networks. The conclusions are in Section 9.

2. Preliminaries

In this section we give basic definitions, introduce models that are generalizations of the well known model of cellular automata, and define our synchronization problem – FSSP.

Models of cellular automata networks. A cellular automaton is an array of identical finite-state machines, called *cells* (or sometimes *processors*), which are connected to nearest neighbours and operate synchronously at discrete time steps.

We will consider both one-dimensional and two-dimensional cellular automata. Connections between cells may be either one-way or two-way links. We will consider several generalizations of the known cellular automata model in which the capacity of the channel may vary. We call a c-link a channel being able to transfer c bits simultaneously. All cells of a CA can be seen as identical (with the exception of the border cells that are aware that they are on a border and they can be seen as having their external signals fixed). In order to simplify description of the evolution of networks, we will mostly label their cells in some way. For example, in one dimensional arrays of n cells we will number cells starting from 0. In this case, the cell 0 and cell $n-1$ are boundary cells. Unless stated otherwise, in the following n is the number of cells of the one-dimensional cellular automaton.

The behaviour of each cell is specified by its finite state transition function that depends on the state of the cell and the outputs, at the previous time step, of some of the connected cells. We denote by N a neighborhood function $N : \{0, \ldots, n-1\} \to \{0, \ldots, n-1\}^*$ which determines the neighbouring cells on which the transition function of a given cell depends. This function depends also on whether the connections are one-way or two-way-links and may also vary for different cells (for example, in the case of the boundary cells). For a cellular automaton A, we denote by m_A the maximum length of $N(i)$, for $0 \leq i \leq n-1$.

A *c-Line* is a one dimensional cellular automaton where all connections are two-way c-links and where the i-th cell is connected to the $(i-1)$-th and $(i+1)$-th cells, for $0 < i < n-1$. The first cell is connected only to the second cell, and the last cell is connected only to the $(n-2)$-th cell. Thus, $N(i) = \{i-1, i+1\}$, for $0 < i < n-1$, $N(0) = \{1\}$ and $N(n-1) = \{n-2\}$ (see Fig. 1a). A *c-Ring* is a one dimensional cellular automaton with two-way c-links and with a connection also between the first and the last cell. Thus the length of $N(i)$ is two, for every i. Finally, a *c-ORing* is a one dimensional cellular automaton with one-way c-links such that a connection exists also between the last and the first cells. The c-ORing can also be seen as having a circular shape with $N(i) = \{i-1\}$, for every $i > 0$, and $N(0) = \{n-1\}$. Thus $m_A = 1$: that is the i-th cell receives only the output of the $(i-1)$-th cell (see Fig. 1b).

In a two-dimensional array of $n \times n$ cells, the cells are numbered (i, j), for $0 \leq i, j \leq n-1$. Each cell (i, j), except for the boundary cells, is connected to cells $(i-1, j)$, $(i, j-1)$, $(i+1, j)$ and $(i, j+1)$. In this case, if the connections are two-way links, then $N(i, i) = \{(i-1, j), (i, j-1), (i+1, j), (i, j+1)\}$ and, in case of one-way connections, $N(i, j) = \{(i-1, j), (i, j-1)\}$. In such a network the cell $(0, 0)$ is connected to the cells $(0, 1)$ and $(1, 0)$, while a cell $(i, 0)$, $0 < i < n$ is connected to the cells $(i, 1)$ and $(i-1, 0)$ and $(i+1, 0)$. Similarly, for cells $(0, i)$.

In case of a *c-Square*, with $n \times n$ nodes, connections are two-way c-links (see Fig. 1c). On the other hand, we can define the *c-Torus of c-Rings*, where, similarly to the first and last cells in the c-Ring, the boundary cells of each row and column are pair-wise connected, and *c-OTorus* where the connections are one-way c-links (see Fig. 1d). c-OTorus is as c-Torus, except for the connections that are one-way-links.

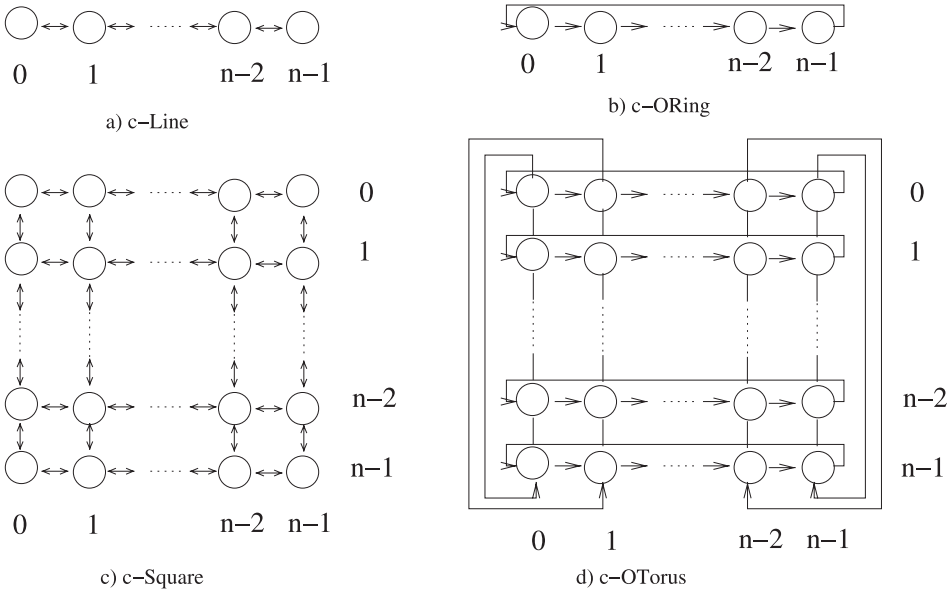

FIGURE 1. The one-dimensional and two-dimensional cellular automata.

	Non-circular	Circular
1-way		c-ORing, c-OTorus
2-way	c-Line, c-Square	c-Ring, c-Torus

FIGURE 2. Models of cellular automata.

For simplicity, we do not consider explicitly rectangular arrays of processors of size $m \times n$, where $m \neq n$, though many of the results in this paper can be extended to such a case. Figure 2 summarizes those models we will consider: non-circular versus circular and with two-way versus one-way links. Observe that we do not consider non-circular models with one-way links. To study FSSP on these models would not have too much sense.

To define behaviour of all models we introduce, we use the symbol Q to denote the set of states of a given cellular automaton A. Different transition functions are defined for cells with different boundaries. If we consider c-links, then for non-boundary cells a transition function is $\delta : D_A^c \rightarrow D_A^c$, where D_A^c is the set of tuples $(q, s_1, \cdots, s_{m_A})$ with $q \in Q$ and $s_j \in \{0,1\}^c$. In the non-circular models, we should define separately transition functions for boundary cells. We omit formal definitions of these functions since they are quite standard and can be easily obtained from the definition of the transition function for non-boundary cells.

The behaviour of a cell i can now be described as follows. $\delta(q, r_1, \cdots, r_m) = (p, s_1, \cdots, s_m)$ means that if the cell i is in the state q and receives inputs $r_1, \cdots, r_m$ from the cells in $N(i)$, then it enters the state p and sends the words $s_1, \cdots, s_m$

to its neighbours. (Observe that our way to describe behaviour of a cell is slightly different from a typical CA-like approach. Indeed, in the standard definition of a cellular automaton each cell sends in each step to its neighbouring cells just its state.) Therefore, in this paper, whenever we consider a model with link capacity c such that $c \geq \lceil \log |Q| \rceil$, we will omit the index c (that is, we will just speak about a Line, Square, etc., instead of a c-Line, c-Square, etc.). Some of the results presented in this paper hold for all models. Thus, when we speak about a c-CA, we mean any of the models introduced above with the link capacity c.

A *configuration* of a one dimensional cellular automaton with n nodes and with c-links is a mapping $C : \{0, \ldots, n-1\} \to D_A^c$. At each time t, a configuration specifies, for each cell i, the state of the cell i and binary words sent at this time. A *starting configuration* is a configuration at time 1. In the following we often write "(A, C)" to denote a cellular automaton A starting on a configuration C. To describe *time-unrolling*[1] of A during T steps, we use so called time-space array of dimension $n \times T$. (i,t)-th position of such an array, where $0 < i < n$ and $t \geq 1$, is called a *site*, and the element of the array in such a position denotes the state of the i-th cell at time t. and will be denoted by $state(i,t)$. Binary words sent to the neighbours by the i-th cell in time t will be denoted by $left(i,t)$ and $right(i,t)$. Sometimes, to avoid ambiguities, we will use notation $state_A(i,t)$, $left_A(i,t)$ and $right_A(i,t)$ to denote the state or the words sent by the i-th cell of the automaton A at the time t. A site (i,t) will be said to be *active* if either it changes its state at the next step, and/or sends/receives a word different from 0 to/from some of its neighbors.

In the two-dimensional cases configuration and unrolling are defined in a similar way and the state of the cell (i,j) at time t is denoted by $state(i,j,t)$.

The problem. We introduce now a synchronization problem which slightly generalizes the very original Firing Squad Synchronization Problem (FSSP). The generalization mainly consists in considering the General state in any cell, not only in the leftmost one.

We assume that among the states of the considered cellular automaton there are three distinguished states: G – the *General* state, L – the *Latent* state, and F the – *Firing* state. The state L, often also called as *quiescent* state, has the property that if a cell in state L receives all-zeros words from its neighbours, then it remains in the same state and sends the all-zero word 0 to its neighbours. A *standard configuration* is the configuration where the leftmost cell 0 (respectively the cell $(0,0)$ in the two-dimensional case) is in the state G and sends a word different from all-zeros to each neighbour and all the other cells are in the state L and each sends out the all-zero word.

A solution to the FSSP on a network in time $t(n)$ is a cellular automaton on such a network such that, starting from a standard configuration, all its cells enter the firing state F at time $t(n)$ and each cell does so for the first time. In such a case we will speak about a synchronization of a c-Line, c-Square, etc. when the underlying cellular automaton is on a c-Line, a c-Square, etc. Moreover, a cellular

[1]Often called also *space-time* diagram.

automaton which provides a synchronization in time $t(n)$ is also called a *solution in time* $t(n)$ of the FSSP, or simply a *solution*.

We introduce now two variations of the FSSP solutions which are sometimes useful to synchronize cellular automata. A solution to the *Two-End Synchronization Problem in time* $t(n)$ is a cellular automaton on a Line such that at time $t(n)$ all cells enter for the first time the state F, starting from the initial configuration which differs from the standard one because both cells 0 and $n-1$ are in the state G. A solution to the *Four-End Synchronization Problem in time* $t(n)$ is a cellular automaton on a Square such that at time $t(n)$ all cells enter for the first time the state F, starting from the initial configuration having all cells $(0,0)$, $(0,n-1)$, $(n-1,0)$, $(n-1,n-1)$ in the state G and all other cells in the Latent state.

It is easy to see that the synchronizations of cellular automata with different link capacities are not unrelated problems. Actually, a synchronization of a c-CA can be seen as a synchronization of a c'-CA for every $c' \geq c$. In particular, we will often use the following proposition:

Proposition 1. *If there is a synchronization of a c-CA in time $t(n)$, then there exists a synchronization of a c'-CA in time $t(n)$, for any $c' \geq c \geq 1$.*

Note that in the literature the time taken by a synchronization is sometimes expressed in terms of the number of steps, see for example [4,8], and sometimes in terms of the number of successive configurations, see for example [12,14]. In this paper the time is expressed by the number of configurations (these two measures differ only by one).

3. Minimal time solutions

In this section we give tight lower bounds on the time of synchronizations of c-CA and present methods for synchronization in the minimal time.

3.1. Lower bounds on the synchronization time

A synchronization of a c-Line requires at least time $2n-1$. Intuitively, this is the minimal time needed for the first cell to wake up all other cells and to get back the message that all cells have been awakened. Recall, that in the starting configuration each cell, except the first one, is in the Latent state and the cell i can not leave the Latent state before time $i+1$. Thus all cells are awake at time n, and the first cell can get this information back not sooner than in time $2n-1$.

Concerning two-dimensional cellular automata, Shinahr [22] has shown that the minimum time to synchronize a rectangular array of $m \times n$ cells is $n+m+\max(n,m)-2$, but this time gets reduced to $2n-1$ in the case of a Square. The following lemma summarizes these results.

Lemma 1. *Every synchronization of a c-Line or a c-Square has time greater than or equal to $2n-1$.*

The minimum time to synchronize a Ring, or a Torus, is at least, as above, the time required by the first cell to send a message to all other cells and to get back information that they received a message from a General.

Lemma 2. *Every synchronization of a c-Ring or a c-Torus has time greater than or equal to* $n+1$.

In the next lemmas we show that time $2n$ is necessary to synchronize a c-ORing and that time $3n-1$ is necessary to synchronize a c-Torus.

Lemma 3. *Every synchronization of a c-ORing has time greater than or equal to* $2n$.

Proof. Assume, by contradiction, that there exists a synchronization in time $\bar{t}(n) < 2n$ of a ORing (say A) and let B be an ORing which differs from A just by the size – B has $2n$ cells instead of n of A. Since, for all $t < n$, $state_A(n-1,t) = L$ and $state_B(2n-1,t) = L$, then $\bar{t}(n) \geq n$ and $state_A(i,t) = state_B(i,t)$ for all $0 \leq i \leq n-1$ and $1 \leq t < n$. Observe that the state of the cell $n-1$ at time $n+t$, for $0 \leq t \leq n$, depends on the states at time n of the following cells: the cells $n-1$ and $n-2$, when $t=1$, and the cells $n-1$, $n-2$ and $n-3$, when $t=2$, and, in general, on the states of the cells $n-1,\ldots,n-t-1$ for $2 < t < n$. As a consequence, $state_A(n-1,t) = state_B(n-1,t)$ for $1 \leq t < 2n$. If $\bar{t}(n) < 2n$, then at time $\bar{t}(n)$ the cell $n-1$ of both A and B enter the state F. Moreover, the cell $2n-1$ of B is at time $\bar{t}(n)$ still in the state L. Thus, we have a contradiction. □

Lemma 4. *Every synchronization of the c-OTorus has time greater than or equal to* $3n-1$.

Proof. Assume, by contradiction, that there exists a synchronization A in time $\bar{t}(n) < 3n-1$ of a OTorus and let B be a OTorus which differs from A by the number of cells. Let B have $2n \times 2n$ cells, instead of $n \times n$ of A. Since, for all $t < n$ and $0 \leq i \leq n-1$, $state_A(i,n-1,t) = state_A(n-1,i,t) = L$ and $state_B(i,2n-1,t) = state_B(2n-1,i,t) = L$, it holds that $state_A(i,j,t) = state_B(i,j,t)$ for all $0 \leq i,j \leq n-1$ and $1 \leq t \leq n$. Furthermore, for both A and B, the state of any cell (i,j) at time n is L if $i+j > n-1$. The state of the cell $(n-1,n-1)$ at time $n+t$, for $0 \leq t \leq \bar{t}(n)-n$, depends on the states at time n of the cells $(n-1-u,n-1-v)$, for $u+v \leq t$. As a consequence, at time $\bar{t}(n)$ the cell $(n-1,n-1)$ of both A and B enters the state F. However, since the cell $(2n-1,2n-1)$ of B at time $\bar{t}(n)$ is still in the state L, we have a contradiction. □

3.2. Minimal time synchronization for two-way communication networks

In this subsection we present minimal time synchronization for models of cellular automata whose inter-connections are two-way links. Due to the Proposition 1, it is sufficient to prove the statements only for the case $c = 1$.

Waksman, in [23], gave the first solution of the FSSP for a Line in the minimal time $2n-1$, and Mazoyer, in [15], showed that a minimal time synchronization

exists for a 1-Line. Moreover, Shinahr [22] has shown a minimal time solution for a Square. In [10], the approach of Shinahr is combined with the solution of Mazoyer to obtain a minimal time synchronization of a 1 -Square.

Lemma 5. *For every link capacity $c \geq 1$, there is a synchronization of a c-Line and of a c-Square in time $2n-1$.*

The above synchronizations can be used to obtain a Two-End synchronization of a Line and a Four-End synchronization of a Square in time n.

Lemma 6. *There are a Two-End synchronization of a Line in time n and a Four-End synchronization of a Square in time n.*

Proof. The Two-End synchronization in time n can be obtained by considering a line as being split into two halves and synchronizing each of them separately by a minimal time solution. This can be implemented by just starting a minimal time solution from both ends. In fact, each cell can determine that it belongs to a sub-line at the time it moves from the Latent state: this happens by a communication received from its left neighbour (in the case of cells of the left half-line), or from its right neighbour (in the case of cells of the right half-line). Note that in the case n is odd, the central cell belongs to both half-lines, while when n is even, the central cells start acting as the last cells of their half-lines with 1 unit of time delay (at the time they receive an input from the other half-line). Therefore, in both cases, the Line is synchronized in time n.

Consider now a Square and see it as consisting of n concentric frames, where the $(i+1)$-th inner frame is formed by the following four lines $(i,i)\ldots(i,n-i-1)$, $(i,n-i-1)\ldots(n-i-1,n-i-1)$, $(i,i)\ldots(n-i-1,i)$ and $(n-i-1,i)\ldots(n-i-1,n-i-1)$, see Figure 3. Suppose now that cells $(0,0),(0,n-1),(n-1,0)$ and $(n-1,n-1)$ are all in the same General state. The four lines of the first frame can all synchronize in time n, using the above result on the Two-End synchronization of a Line. During such synchronizations, after the first two steps, the four cells $(1,1)$, $(1,n-2)$, $(n-2,1)$ and $(n-2,n-2)$ all enter a General state and thus the four lines of the second frame can synchronize in time $n-2$. Repeating this argument, the i-th frame synchronizes in time $n-2(i-1)$, $1 \leq i \leq \lceil n/2 \rceil$. Since the last synchronization starts at time $2(i-1)+1$, the overall time to synchronize all cells is again n. □

Remark. The synchronizations sketched in the above proof do not work when the link capacity is 1. The main reason is that synchronizations of 1-CA make an important use of the information whether the time a bit 1 is received is even or not, to distinguish between different messages. In particular, each cell i expects an even time delay between the message sent by the General to wake up all cells and the reply sent by the last cell in the Line (in a minimal time solution the last cell replies as soon as it gets awakened). In the schema sketched in the proof of Lemma 6, for the Two-End synchronization of a Line, when n is even, the central cells delay the response of 1 time unit. Therefore, the reply message could be misunderstood by all the other cells, unless we delay it by another time unit. This

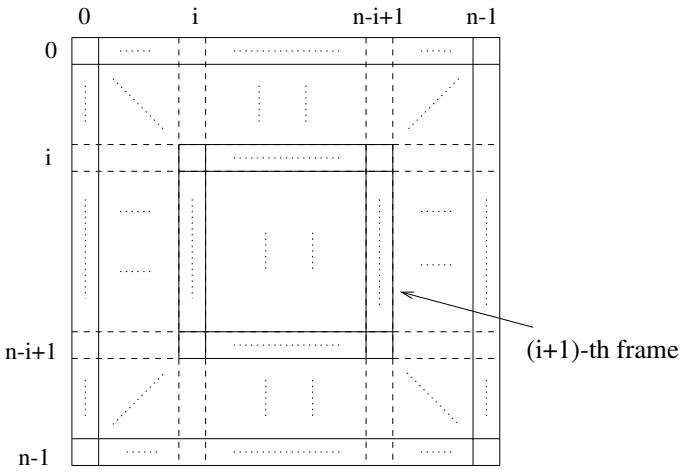

FIGURE 3. The frames in a Square of $n \times n$ processors.

is the idea exploited in the solution given in [10]. Therefore, we have the following lemma.

Lemma 7. *There are a Two-End synchronization of a Line in time* $2\lfloor n/2 \rfloor + 1$ *and a Four-End synchronization of a Square in time* $2\lfloor n/2 \rfloor + 1$.

Note that the minimal time synchronization of a 1-Line due to Mazoyer [15] can be modified to work for a 2-Line without relying on the parity of delays to recognize messages (we simply use the second bit to do that). Thus, it is easy to verify that the schema sketched in the proof of Lemma 6 can be adapted to work for a 2-CA using techniques similar to that used in [10] for 1-CA. Therefore, the following lemma holds.

Lemma 8. *For every link capacity* $c \geq 2$*, there are a Two-End synchronization of a Line in time* n *and a Four-End synchronization of a* c*-Square in time* n.

The following lemma states that the lower bounds given in the previous section for c-Ring and c-Torus are tight for $c \geq 2$. Note that for the Ring this is a more correct result than the one found in [4].

Lemma 9. *For every link capacity* $c \geq 2$*, there is a synchronization of a* c*-Ring and of a* c*-Toruses in time* $n+1$. *Moreover, there is a synchronization of a* 1*-Ring and of a* 1*-Torus in time* $2\lceil n/2 \rceil + 1$.

Proof. Since a c-Ring, $c \geq 2$, can simulate a Two-End synchronization of a c-Line of $n+1$ cells, the cell 0 can act as both boundary cells of the c-Line. Therefore, by Lemma 8, there exists a synchronization in time $n+1$.

A synchronization of a c-Torus in time $n+1$ can be obtained by considering such a Square as being split into three parts: the first row, the first column and the rest of the array, that is a sub-array of $(n-1) \times (n-1)$ cells. As we have

noticed above, the first row and the first column can be synchronized in time $n+1$. During these synchronizations (during their first two steps) the cells $(1,1)$, $(1,n-1)$, $(n-1,1)$, $(n-1,n-1)$ can enter a new state acting as a General state of a Four-End synchronization of a Square of $(n-1)\times(n-1)$ cells. Using Lemmas 8 and 7, and taking into consideration the fact that this last synchronization starts with a two step delay, we get the results claimed in the lemma. □

Let us now summarize main results of this section.

Theorem 1.

- *For every link capacity $c \geq 1$, there is a synchronization of a c-Line and of a c-Square in time $2n-1$. Moreover, every synchronization of a c-Line or a c-Square has time greater than or equal to $2n-1$.*
- *For every link capacity $c \geq 2$, there is a synchronization of a c-Ring and of a c-Torus in time $n+1$, and there is a synchronization of a 1-Ring and of a 1-Torus in time $2\lceil n/2 \rceil + 1$. Moreover, for every link capacity $c \geq 1$, every synchronization of a c-Ring or a c-Torus has time greater than or equal to $n+1$.*

Observe that there is still a gap between the lower and upper bounds shown in the above theorem for synchronization of a 1-Ring and a 1-Torus only for odd n.

3.3. Minimal time synchronization for one-way communication networks

The following two lemmas state that lower bounds given in the previous section for models with one-way links are tight.

Lemma 10. *There is a synchronization of a ORing in time $2n$.*

Proof. Using standard techniques, any computation on a Line A of n processors in time $t(n)$ can be executed by an ORing B in time $2t(n)$, provided that the initial configuration of A can be reached in one step from the initial configuration of B. To demonstrate that we use, informally, an induction on the number of steps. Let $state_B(i+1,1) = state_A(i,1)$ and $state_B(0,1) = state_A(n,1)$ and assume that $state_B(i+t,2t) = state_A(i,t)$. (To be more precise, since the cell $i+t$ of B has to simulate the cell i of A, then, if $i=0$ or $i=n-1$, the state of the cell $i+t$ of B encodes a state of A and the information that the simulated cell is the leftmost, or the rightmost, in the line). The i-th cell of A at time $t+1$ needs states of cells $i-1$ and $i+1$ at time t. The cell $(i-1)+t$ of B passes its current state, say p, to the cell $(i+t)$ and this one in turn forwards p, along with its own state, to the right neighbouring cell number $(i+1)+t$. This cell can therefore simulate the behaviour of the cell i of A at the step $t+1$. Thus $state_B(i+t+1,2(t+1)) = state_A(i,t+1)$. The overall simulation requires therefore a multiplicative delay factor of two.

Let us consider now a Two-End synchronization S of a Line. It takes time n and a synchronization of an ORing in time $2n$ can be obtained by the above simulation. Actually, as the first step, it lets the second cell to enter the General

state, so that the state of the cell $i+1$ after the first step is equal to the state of the cell i in the starting configuration of S. □

Lemma 11. *There is a synchronization of a OTorus in time* $3n-1$.

Proof. We will first give an easier to describe solution which takes time $3n$ and then we show how to save one time unit.

Using standard techniques (as in the previous proof), any computation of a Square A in time $t(n)$ can be executed by a OTorus B in time $3t(n)$ in the following way (we use again, informally, an induction on the number of steps). Assume that the cell $(i+1,k+1)$ in the third configuration of B contains the state the cell (i,k) has in the first configuration of A and that the cell $(i+j,k+j)$ of B, at the time $3j$, has the state the cell (i,k) of A has at the time j. Moreover, if the cell (i,k) is a border cell, *i.e.* if either $i \in \{0,n-1\}$ or $k \in \{0,n-1\}$, also such an information can be stored in the state of the cell $(i+j,k+j)$ of B. Note that the cell (i,k) of A at the j-th step computes the new state from its own state and the states of cells $(i-1,k)$, $(i,k-1)$, $(i+1,k)$ and $(i,k+1)$ at time j. Therefore, within three steps the cell $(i+(j+1),k+(j+1))$ of B can collect states that are, at time $3j$, in the cells $(i+j,k+j)$, $((i-1)+j,k+j)$, $(i+j,(k-1)+j)$, $((i+1)+j,k+j)$ and $(i+j,(k+1)+j)$. Namely:

(1) at the step $3j$, the cell $(i+j,k+j)$ of B stores two states p,q of cells $((i-1)+j,k+j)$ and $(i+j,(k-1)+j)$;
(2) at the step $3j+1$ states p,q are passed to the cells $((i+1)+j,k+j)$ and $(i+j,(k+1)+j)$ (note that in the previous step the state of the cell $(i+j,k+j)$ at the time $3j$ has been passed to these cells);
(3) at the step $3j+2$, the cell $((i+1)+j,(k+1)+j)$ simulates the cell (i,k) of A at the step j.

Therefore, the state of the cell $(i+(j+1),k+(j+1))$ of B at time $3j+3$ contains the state the cell (i,k) of B has at time $j+1$. The overall simulation takes thus a multiplicative delay factor of three.

Let now A be a Four-End synchronization as in Lemma 6. Recall that in this automaton the Square can be seen as being composed from concentric frames (see Fig. 3) that can be synchronized at the same time n. We can get an OTorus A' which reaches in the first two steps a configuration such that the states of all cells $(0,0),(0,1),(1,0)$ and $(1,1)$ contain the General state (recall that the states of the cells $(0,0),(0,n-1),(n-1,0)$ and $(n-1,n-1)$ in the starting configuration of the solution S are all the General state). Therefore A' simulates synchronization A within time $3n$.

Let us now briefly explain how can A' be modified to save one step at the synchronization, and this way to reach synchronization time $3n-1$. The first $3n-3$ steps (and thus the first $3n-2$ configurations) remain unmodified. After that we can observe the following facts:

(1) Each cell of A in the configuration j participates at the synchronization of the frame it belongs to. Actually, each cell participates at the synchronization either only of a row-line or only of a column-line of the frame

except for the four corner cells of the frame which participate at the synchronization of both lines they belong to. The same holds also for A' in the configurations $3j$ (due to the mapping between the cells of the configuration j of A and those of the configuration $3j$ of A').

(2) At time $3j+2$ in A', $1 \leq j < n$, a cell $(i+(j+1), k+(j+1))$ is aware of the states at time $3j$ of the following cells:
   a) $((i-1)+(j+1), (k-1)+(j+1))$, $(i+(j+1), (k-2)+(j+1))$, $(i+(j+1), (k-1)+(j+1))$ and $(i+(j+1), k+(j+1))$;
   b) $((i-1)+(j+1), (k-1)+(j+1))$, $((i-2)+(j+1), k+(j+1))$, $((i-1)+(j+1), k+(j+1))$ and $(i+(j+1), k+(j+1))$.

Thus, at the step $3n-2$, the cell $(i+n, k+n)$ can correctly simulate either the cell $(i, k-1)$ or the cell $(i-1, k)$ of S at the step $n-1$, hence entering the Firing state. In particular, the cell $(i+n, k+n)$ simulates the former one if the cell $(i, k-1)$ participates at the synchronization of a row-line, or simulates the latter one, if the cell $(i-1, k)$ participates at the simulation of a column line. (Note that at least one of these conditions must hold.) Therefore, there is an OTorus synchronization in time $3n-1$. □

Main results of this section are now summarized.

Theorem 2.

- *There is a synchronization of an ORing in time* $2n$ *and every synchronization of an ORing has time greater than* $2n-1$.
- *There is a synchronization of an OTorus in time* $3n-1$ *and every synchronization of an OTorus has time greater than* $3n-2$.

4. Signals

An exact formalization of the concept of *signal* has been introduced in [12] to simplify description of a synchronization process for FSSPs. This concept provides a handy way to modularize designs of synchronizations. Informally speaking, a signal is a set of cells that, at a given time, receive/send words, different from all-zero words, from/to the adjacent cells. In other words, a signal describes information flow in the space-time unrolling of a cellular automaton, and allows a modular description of the synchronization processes. Namely, it allows to start from some basic signals and then combine different signals to obtain new ones to describe, in a more natural and transparent way, synchronization process. (Let us note that an informal concept of signal is common in the FSSP literature. In fact, also in [3] and [17] it was used, but there the meaning was different. One should also note that an (informal) use of the various concepts of signals is common in cellular automata papers.) The scheme used to present some synchronization algorithms in time $t > 2n-1$ for a c-Line of n processors is then the following: Some signals are designed and then composed to obtain an overall signal that starts from the leftmost processor and comes back to it in exactly $(t-2n+1)$

time steps. Afterwards a minimal time synchronization process starts, to make the final synchronization of n processors in time t.

Consider now the time unrolling of a c-Line A with a starting configuration C – notation (A, C). Denote $t_i^{\max} = \max\{t \mid (i,t) \text{ is active}\}$ and $t_i^{\min} = \min\{t \mid (i,t) \text{ is active}\}$. We capture the boundaries of the active sites of (A, C) by the notions of *rear* and *front* of (A, C). The *rear* of (A, C) is the set of the earliest active sites of each cell i, that is, for each cell i such that there exists at least one active site (i, t) of (A, C), we define *rear* of (A, C) as the set of all sites $(i, t_i^{\min})$. Analogously, the *front* of (A, C) is the set of the latest active sites of each cell i, that is, for each cell i such that there exists at least one active site (i, t) of (A, C), we define *rear* of (A, C) as the set of all sites $(i, t_i^{\max})$. Moreover, we say that (A, C) is *tailed* if there exists a subset of Q, called *tail*(A, C), such that for all $i \in \{1, \ldots, n\}$, *state*$(i, t) \in \text{tail}(A, C)$ if and only if (i, t) belongs to the front of (A, C). The states in the tail(A, C) are called *tail* states. In words, a tail state appears for the first time (in the time unrolling of A) on the front of (A, C).

Two active sites $(i_1, t_1), (i_2, t_2)$ are said to be *consecutive* if $t_2 = t_1 + 1$ and $i_2 \in \{i_1 - 1, i_1, i_1 + 1\}$. A *simple signal* of (A, C) is a set S of consecutive sites with the property that if (A, C) is tailed, then $(i, t_i^{\max})$ belongs to S. Any union of a finite number of simple signals of a given (A, C) is called *signal* of (A, C).

A graphical representation of a simple signal S is obtained by drawing a straight line between:
(i) every pair of sites $(i, t) \in S$ and $(i, t + 1) \in S$ and
(ii) every pair of sites $(i, t) \in S$ and $(i + 1, t + 1) \in S$ (resp. $(i - 1, t + 1) \in S$) if *right*$(i, t) = 1$ (resp. *left*$(i, t) = 1$).

A graphical representation of a signal is obtained by the graphical representation of its simple signals. The *length* of a signal S is $t^{\max} - t^{\min} + 1$) where $t^{\max} = \max\{t \mid (i,t) \in S, 1 \le i \le n\}$ and $t^{\min} = \min\{t \mid (i,t) \in S, 1 \le i \le n\}$.

Remark. In the rest of the paper we sometimes refer to a signal without specifying an automaton and a starting configuration. For technical reasons, in this section, and also in Section 6, we will number the first cell as the cell number 1 (instead of 0 as said in the preliminaries).

The following examples show two signals: MAX and MARK. The former is the "fastest" signal (it touches one new cell each time unit), while the latter will be used to check the occurrence of an event (generally a signal crossing a given cell). Thus, if it is this case, it triggers a new signal (see Fig. 4).

Example 1. Let $i \neq j$ and MAX(i, j) be the set containing the sites $(i + h, h + 1)$ if $i < j$, or sites $(i - h, h + 1)$ otherwise, for $0 \le h \le |i - j| + 1$. This set is a simple signal, with length $|i - j| + 1$, of a tailed c-Line that starts from a configuration having the states of cells i and j different from all others.

Example 2. Given a positive constant $k < n$, the signal MARK$(n - k)$ is used to mark the cell $n - k$ starting from the leftmost cell after bouncing on the rightmost cell. The length of the signal MARK is $n + k$ (see Fig. 5). It can be easily seen that MARK is a signal of a tailed c-Line.

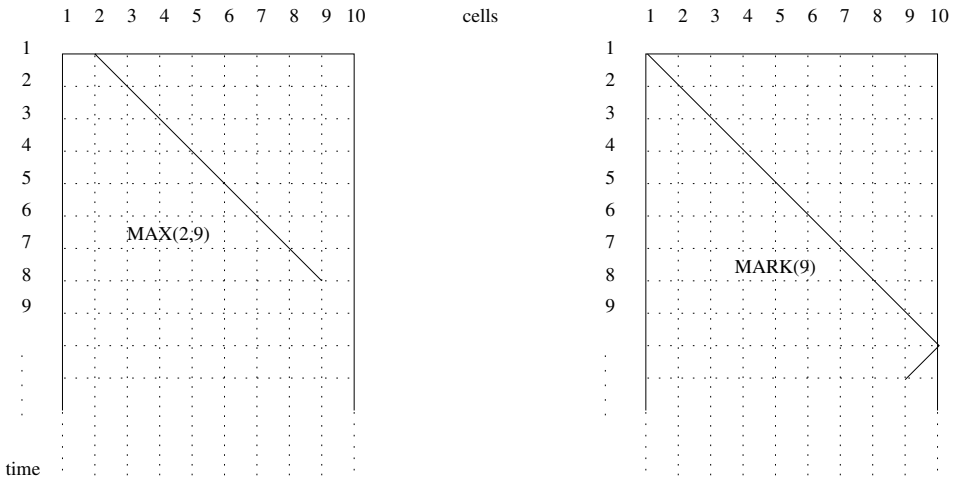

FIGURE 4. The signals MAX and MARK.

4.1. Composition of Signals

Signals can be composed in order to obtain new ones. Given two signals S_1 and S_2, we define their *concatenation*, $\mathrm{cat}_r(S_1, S_2)$, as the signal obtained by starting S_1 at time 1 and S_2 at time $r+1$. That is, S_2 is delayed r time steps. More formally, $\mathrm{cat}_r(S_1, S_2) = S_1 \cup \{(i, t+r) \,|\, (i,t) \in S_2\}$. For the concatenation of signals the following property is crucial. We say that a c-Line A_2 on C_2 *can follow* a tailed c-Line A_1 on C_1 if there exists a function h, defined over the $\mathrm{tail}(A_1, C_1)$, such that $h(p) = C_2(i)$ if $p = state(i,t)$. (When this property holds it is possible to switch from the front of (A_1, C_1) to C_2.)

The following lemma recalls some sufficient conditions for the existence of a tailed c-Line for a signal $\mathrm{cat}_r(S_1, S_2)$.

Lemma 12. *Let S_1 and S_2 be signals of tailed c-Lines (A_1, C_1) and (A_2, C_2), respectively. The signal $S = cat_r(S_1, S_2)$ is a signal of a tailed c-Line (A, C_1) if the following conditions hold:*

(1) *(A_2, C_2) follows (A_1, C_1);*
(2) *if a site (i,t) belongs to the front of (A_1, C_1), and (i, t') belongs to the rear of (A_2, C_2), then $t < t' + r$;*
(3) *if sites $(i,1)$ and $(j,1)$ belong to the rear of (A_2, C_2), then $t_i^{\max} = t_j^{\max}$ in (A_1, C_1).*

Proof. Let (i,t) be a site of a c-Line such that $t = t_i^{\max}$ in (A_1, C_1) and let $(i,1)$ belong to the rear of (A_2, C_2). Let $s = r - t + 1$. Due to the assumption 2, $s > 0$. A tailed c-Line (A, C_1) for $S = \mathrm{cat}_r(S_1, S_2)$, can now be obtained in the following way. At the beginning A behaves as A_1. On the states from $\mathrm{tail}(A_1, C_1)$, A counts up to $s-1$ and then enters the corresponding state of C_2. We recall that this step is well defined since s is a positive constant and the above condition 1 holds. At

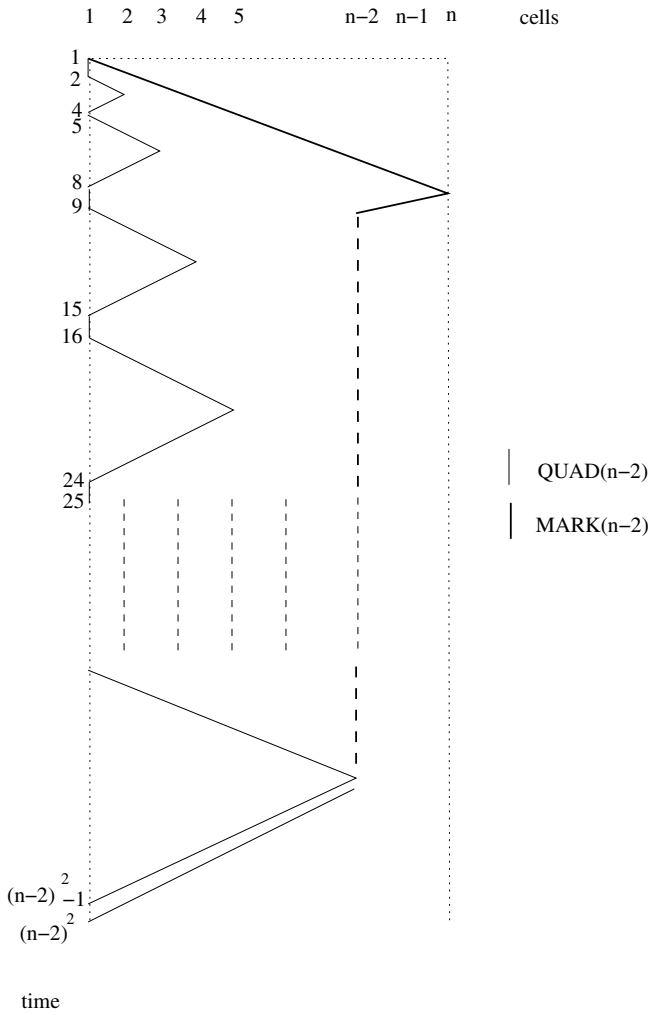

FIGURE 5. The signal $\mathrm{cat}_1(\textsc{Quad}(n-2), \textsc{Mark}(n-2))$.

this point A behaves as A_2. Clearly, (A, C_1) is tailed and S is a signal of (A, C_1). Notice that if there are cells corresponding to the active sites of (A_2, C_2) which do not correspond to the active sites of (A_1, C_1). It follows from the conditions of the above lemma that in both configurations C_1 and C_2 such cells are in the quiescent states.

4.2. Non trivial signals

We introduce here two non trivial signals of a c-Line that will be used later to get main synchronizations for FSSP. The first signal has a quadratic length and

the second has an exponential length, in the number of cells. In particular, from the Proposition 1 it follows that it is sufficient to consider only the case $c = 1$ (which is also the most difficult one).

The signal QUAD**.** Given a positive constant $k < n$, QUAD$(n-k)$ is a signal of a 1-Line A which can be described as follows:

- initially, the cell 1 sends the bit 1 to the right; then, if it receives a bit 1 from the right, it sends with a delay of one step (except for the first time, when there is no waiting), the bit 1 back to the right. The cell 1 eventually halts when it receives two consecutive bits 1;
- for $1 < h < (n-k)$, the cell h sends the bit 1 to the left when it receives, for the first time, the bit 1 from the left. Afterwards, if the cell h receives again a bit 1 from an adjacent cell, it sends a bit 1 to the other adjacent cell;
- the cell $(n-k)$ sends two consecutive bits 1 to the left when it receives the bit 1 from the left.

Notice that the designed 1-Line A can be tailed as well: in fact the cells from 1 to $(n-k)$ can enter a tail state when they receive two consecutive bits 1. The length of the QUAD signal is $(n-k)^2 - 1$. Let us note now that for an implementation of the QUAD signal the cell $(n-k)$ needs to be distinguished. In what follows we use only signals QUAD$(n-2)$ from Theorem 4 and QUAD$(n-1)$ from Theorem 3. Therefore, we only need to distinguish cells $(n-2)$ and $(n-1)$. This can be done by signals MARK$(n-2)$ and MARK$(n-1)$, for $n > 5$. For smaller n much easier and ad hoc algorithms can be given (see Fig. 5).

The signal EXP**.** In order to define, for $k > 0$ and $d > 0$, the signal EXP$(n-k, d)$, we use the concept of an *idle cell*. It is a cell which never sends a bit 1 before it receives a bit 1 from the left and in such a case it sends two consecutive bits 1 to the left.

Initially, the only idle cell will be the cell $(n-k)$. EXP$(n-k, d)$, as a signal of a 1-Line, can now be described as follows:

- First, the cell 1 sends a bit 1 to the right. Afterwards, whenever the cell 1 receives a bit 1 from the right, it immediately replies and sends back a bit 1. Finally, if the cell 1 receives two consecutive bits 1 from the right, then it changes into an idle cell;
- For $1 < h < (n-k)$, we distinguish two cases:
  - if the h-th cell receives a bit from the left, then it alternates the following two behaviours:
    (1) It sends a bit 1 back to the left. Call such cells as *peak cells* (though this is a property of the state this cell gets into).
    (2) It sends a bit 1 to the right.

    Each peak cell starts counting from 1 to $2^{d+1} - 2$ and when a number $2^{i+1} - 2$, $1 < i \leq d$, has just been reached and such a cell receives a bit 1 from the left at the next time unit, then it is the i-th cell in the line, and it is marked (see below for an explanation), in order to be distinguished this way later.

– If a bit 1 is received from the right, then the cell sends a bit 1 to the left. If at the next time unit the cell h receives another bit 1 from its right neighbour, then two other sub-cases need to be considered:
  if $h > d$, then the cell switches into an idle cell;
  otherwise, for $h \leq d$, the cell sends two consecutive bits 1 to the left. (Note that when such a case occurs, then cells number $h \leq d$ have already been marked as described in the Step 2 above.)

On the basis of the above facts, a proof by induction on $i \leq d$ can be given to show that each peak cell can be marked. In fact, the following property holds: the length of the interval from the moment a cell i gets to be a peak cell for the first time, and the instant it becomes a peak cell for the second time, is $2^i + \sum_{j=1}^{i-1} 2^j(i-j)$ (see Fig. 6, where $d = 3$, the cell 2 is marked at time 9 and the cell 3 is marked at time 20).

In order to implement a tailed 1-Line for $\textsc{Exp}(n-k, d)$, initially the cell $(n-k)$ must be distinguished. In what follows we will use the signals $\textsc{Exp}(n-2, \cdot)$ and $\textsc{Exp}(n-1, 1)$. The cells $n-2$ and $n-1$ can be distinguished by using $\textsc{Mark}(n-2)$ and $\textsc{Mark}(n-1)$, for $n > 5$. Observe also that cells number 1 to $(n-k)$ can enter a tail state after they received two consecutive bits 1. The length of $\textsc{Exp}(n-k, d)$ is $2^{n-k+1} - 2(n-k) - 2^{d+1} + 2(d+1)$ (see Fig. 6). In a very similar way we can define the signal $\mathrm{E}(n-k)$ of length $2^{n-k+1} + 1$ (see Fig. 7).

5. Composition of synchronizations

The design of synchronizations in non-minimal time is not obvious. A compositional approach to achieve that is very handy. In this section we discuss several ways to combine two or more synchronizations of networks. We start with a parallel composition, afterwards we study also sequential and iterated compositions.

In the following, if S_i is a synchronization of a c-CA, then G_i, L_i and F_i will denote the General, Latent and Firing states of S_i and Q_i, respectively, and δ_i will denote their set of states and transition functions. We will also use the cross product of automata as a mean to combine c-CA. Given a c_1-Line A_1 and a c_2-Line A_2, we denote by $A_1 \times A_2$ the $(c_1 + c_2)$-Line defined as the standard cross product of A_1 and A_2. In such constructions we keep distinct communication links of two underlying networks and therefore $A_1 \times A_2$ allows to run in parallel a synchronization of a c_1-Line along with a synchronization of a c_2-Line. Such constructions are extended to all the other models we consider in an obvious way. We will also slightly modify the above cross product construction to design synchronizations that choose among two different synchronizations according to a given condition $P(n)$. Examples of such conditions are the parity of the number of processors and the fastest/slowest synchronization. We call a c-Line as a *c-line selecting in time $t(n)$*, if it is a c-Line whose state set contains two disjoint subsets O_1 and O_2, called the *selection subsets*, such that starting from the standard configuration, any of its configuration at any time $t \geq t(n)$ contains either only

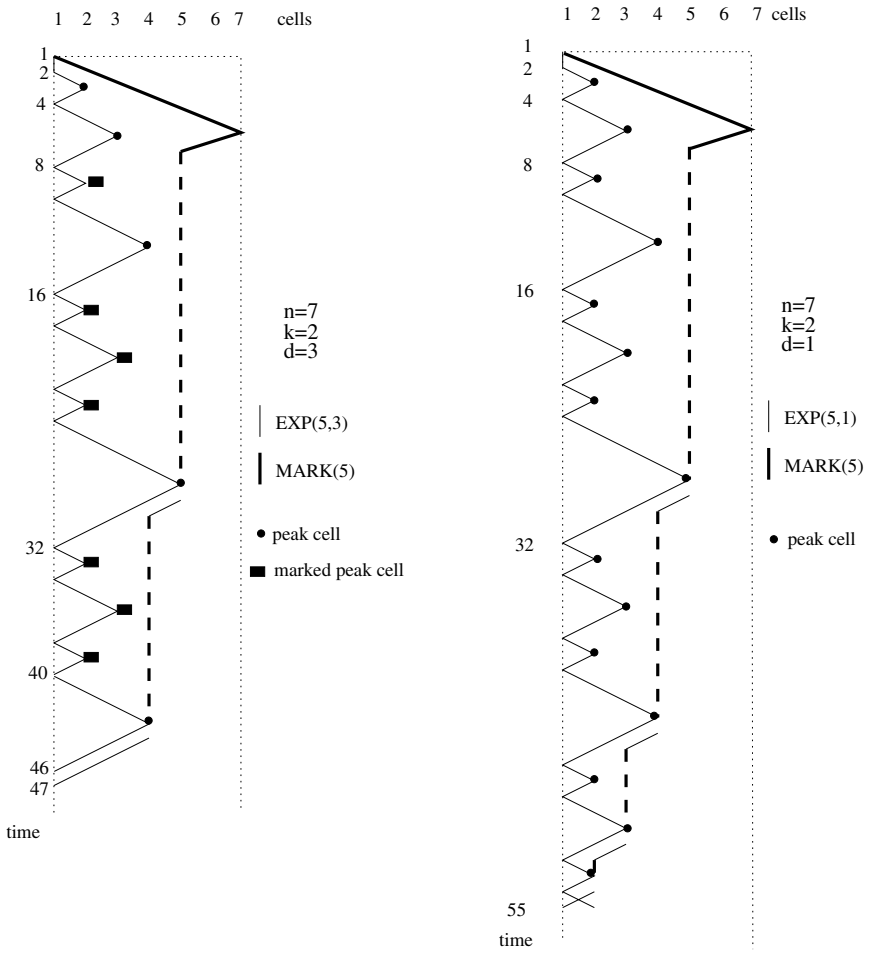

FIGURE 6. The signals $\mathrm{cat}_1(\textsc{Exp}(5,3), \textsc{Mark}(5))$ and $\mathrm{cat}_1(\textsc{Exp}(5,1), \textsc{Mark}(5))$.

states from O_1 or only states from O_2. This definition can be extended, in an obvious way, to all other models we will consider.

The following lemma shows how to design a c-CA that selects between two given synchronizations according to the number of cells it has. Clearly, by iterating this construction, a selection among more than two synchronizations can be obtained.

Lemma 13. *For $i = 1, 2$, let S_i be a synchronization on a c_i-CA in a time $t_i(n)$, and K be a selecting c_K-CA, in time $t(n) \leq \min\{t_1(n), t_2(n)\}$, with selection subsets O_1 and O_2. Then there exists a synchronization on a c'-CA in time $t'(n)$ such that $c' = c_K + c_1 + c_2$. Moreover, if any configuration of K at time $t \geq t(n)$ contains only states from O_1 (O_2), then $t'(n) = t_1(n)$ $9t'(n) = t_2(n)$).*

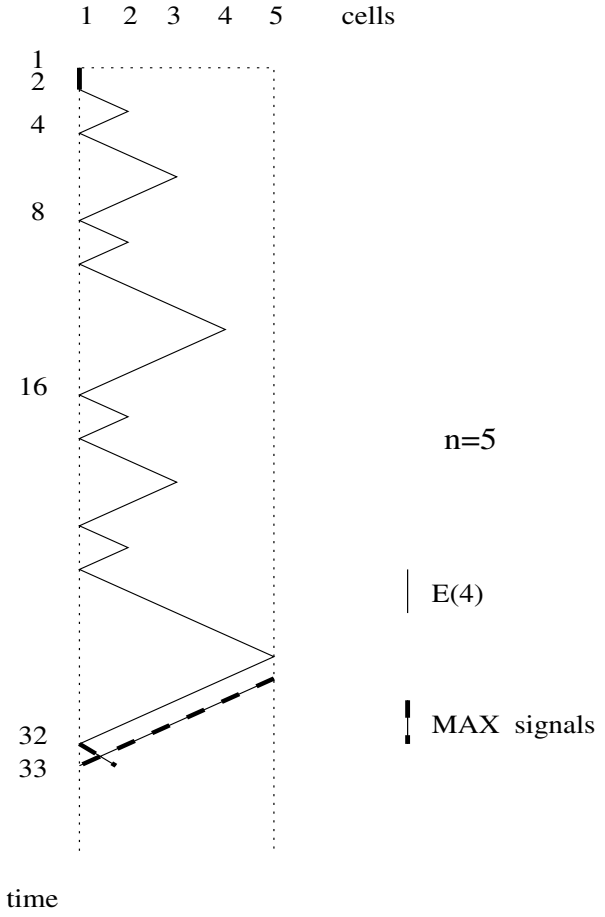

FIGURE 7. The signals $\mathrm{E}(n-k)$.

Proof. Let S be the c'-CA obtained by modifying $K \times S_1 \times S_2$ in the following way: for $i = 1, 2$, if a cell enters F_i and the selecting automaton K is in a state of O_i, then S enters the firing state of S. Clearly, if S starts on a configuration composed of the triples of the corresponding states of standard configurations for K, S_1 and S_2, then S synchronizes in the time stated in the lemma. □

As applications of the above lemma we deal with two examples. In the first example, we show how to solve the problem to obtain a synchronization which synchronizes at the maximum or at the minimum time of two synchronizations we consider. At first, we construct a selecting CA performing the test $t_1(n) \leq t_2(n)$. Then we show that such a selecting CA can be used to obtain synchronization in either the maximum or the minimum time of two synchronizations. In the

second example, a particular synchronization behaviour is selected depending on determination whether the number of the cells involved exceeds a given bound.

Example 3. For $i = 1, 2$, denote by S_i a synchronization in time $t_i(n)$. Let K be a selecting CA that checks the condition $t_1(n) \leq t_2(n)$ in time $t(n) = \min\{t_1(n), t_2(n)\}$. K can be seen as the cross product of S_1 and S_2 with the modification that once a synchronization in S_1 or S_2 enters the firing state, then K cycles on this state. This can be seen as choosing $O_1 = \{F_1\}$ and $O_2 = \{F_2\}$. Therefore, by Lemma 13, we have a synchronization in time $t_1(n)$, if $t_1(n) \leq t_2(n)$, and in time $t_2(n)$, otherwise. This way a synchronization in the minimum time between $t_1(n)$ and $t_2(n)$ is obtained. A modification of the above approach to get a synchronization in the maximum time between $t_1(n)$ and $t_2(n)$ can be obtained by choosing $O_1 = \{F_2\}$ and $O_2 = \{F_1\}$.

Example 4. We describe a selecting CA K that performs a test $n \leq h$, for a fixed integer h. Let $Q = \{G, L, p_1, \ldots, p_h, p_{\leq h}, p_{>h}\}$ where G and L are the General and the Latent state, respectively, and $O_1 = \{p_{\leq h}\}$ and $O_2 = \{p_{>h}\}$. The transition function of K can be informally described as follows (for two-dimensional models, K can be described in an analogous way). In the first step cells 0 and 1 enter states p_1 and p_2, respectively. Later, each cell in the Latent state enters the state p_{i+1} if its adjacent left neighbouring cell is in a state p_i, for $i < h$, but it enters the state $p_{>h}$ if the left neighbour is in the state p_h. Therefore, if each cell is in a state p_i for some $i \leq h$, then $p_{\leq h}$ can be seen as being propagated up to the cell 0 (this takes just one step in an ORing, but $n-1$ steps in a Line since this is the case if the cell $n-1$ is in a p_i for $i \leq h$). When a processor enters the state $p_{\leq h}$, or the state $p_{>h}$, all other processors will be forced to enter the same state within a time n. Obviously, this way a K can be designed that is a selecting CA in time $t(n) = n + \min\{h, n\}$.

In the next two lemmas we show how to compose two synchronizations in time $t_1(n)$ and $t_2(n)$ to obtain synchronizations in time $t_1(n) + t_2(n) + d$, for a given d, and in time $t_1(n)t_2(n)$.

Lemma 14. *If S_1 and S_2 are two synchronizations on a c-CA in time $t_1(n)$ and $t_2(n)$, then there exists a synchronization on a c-CA in time $t_1(n) + t_2(n) + d$, for a given $d \geq 0$.*

Proof. We define a synchronization S such that S behaves as S_1 from time 1 up to time $t_1(n)$, then, at time $t_1(n) + 1$, it switches to S_2. Such a S will be a synchronization in time $t_1(n) + t_2(n)$. Furthermore, given a synchronization S' in time $t(n)$ and with the Firing state F'_0, a synchronization in time $t(n) + d$ can be obtained from S' by adding the states $F'_1, \ldots, F'_d$ and the transition rules from F'_i into F'_{i+1} for $i = 0, \ldots, d-1$, and, finally, by picking F'_d as the Firing state of the resulting synchronization.

Lemma 15. *If S_1 and S_2 are two synchronizations on a c-CA in time $t_1(n)$ and $t_2(n)$, then there exists a synchronization on a c-CA in time $t_1(n) \cdot t_2(n)$.*

Proof. We prove the above result for a 1-Line – the proof is similar for all other models considered here. We construct a synchronization S as consisting of an Iterative phase, of length $t_1(n)$, executed $t_2(n)$ times. The set of states of S will be $Q_1 \times Q_2 \times \{0,1\}^2$, the General state will be $(G_1, G_2, 0, 1)$, the Latent state will be $(L_1, L_2, 0, 0)$ and the Firing state will be $(F_1, F_2, 0, 0)$. In the Iterative phase, the synchronization S keeps modifying the first component of its state according to the transition functions of S_1, until this component is F_1. At the end of this phase, S starts to execute a transition step, modifying the second component of the state according to the transition function of S_2. The bits sent according to the transition function of S_2 are being saved in the last two components of each state according to the order: left, right. Moreover, in the same step, S replaces F_1 with either G_1 or L_1 (depending on whether the cell is the one triggering, in the initial configuration, the firing signal of S_1) in the first component. So the Iterative phase can start again, until all cells get into the Firing state. So, the synchronization S_1 is iterated exactly $t_2(n)$ times and S synchronizes in time $t_1(n)t_2(n)$.

Finally, we show a construction that allows to obtain synchronizations on a c-Square in time $t(2n-1)$ provided that there exists a synchronization of a c-Line in time $t(n)$.

Lemma 16. *If there is a synchronization on a c-Line in time $t(n)$, then there exists a synchronization on a c-Square in time $t(2n-1)$.*

Proof. An $(n \times n)$ array can be seen as many lines of $(2n-1)$ cells, each of them having as endpoints cells $(0,0)$ and $(n-1, n-1)$. Each of these lines corresponds to a "path" from the cell $(0,0)$ to the cell $(n-1, n-1)$, going through exactly $(2n-3)$ other cells. Each cell (i,j) of these paths has as the left neighbour either the cell $(i-1, j)$ or the cell $(i, j-1)$ and as right neighbour either the cell $(i+1, j)$ or the cell $(i, j+1)$.

Notice that a cell (i,j) is the $(i+j-1)$-th cell from the left in all the lines it belongs to. This property allows us to execute simultaneously, on all these lines, a synchronization in time $t(n)$. Since the length of each line is $(2n-1)$, we have a synchronization of the c-Square in time $t(2n-1)$.

Of interest is also the following open problem. Given any computable function $t(n)$ between $2n-1$ and 2^n, does there exist a synchronization for FSSP for a c-Line? The answer is likely negative, but no proof is known. In particular, does there exist synchronization for such time as $2^n / \lg^* n$?

6. Two-way communication networks

In this section we compose signals introduced in the previous section to obtain synchronizations for FSSP in times n^2, 2^n, $n\lceil \log n \rceil$ and $n\lceil \sqrt{n} \rceil$ on a 1-Line and a 1-Square with two-way communications between neighbouring cells. Clearly, as a consequence we obtain synchronizations for FSSP in the same time for c-Lines and c-Squares as well as for circular c-Rings and c-Toruses.

Note. In order to simplify statements of reasoning, in this section we will number cells starting from 1 (instead of 0 as before).

Theorem 3. *There is a synchronization for FSSP on a* 1*-Line with* n *cell in time* n^2*.*

Proof. Synchronization process can be seen as divided into two phases. In the first phase, the following concatenation of signals is used: $\mathrm{cat}_1(\textsc{Mark}(n-1), \textsc{Quad}(n-1))$. It has length $(n-1)^2$ since $\textsc{Quad}(n-1)$ is delayed by one time step, see Figure 5. By Lemma 12, this phase can be seen as a signal of a tailed 1-Line that starts from a standard configuration. This can be seen as that the cell 1 has entered a tailed state, say G'. Such a state can now be considered as a General state that lunches, one step later, a minimal time synchronization process on the line, and that is the second phase. These two phases together yield a synchronization solution of FSSP in time n^2. □

Theorem 4. *There is a synchronization of a* 1*-Square with* n *cells in time* n^2*.*

Proof. Synchronization works as follows: at first a signal $\mathrm{cat}_1(\textsc{Mark}(n-2), \textsc{Quad}(n-2))$ starts on the first row. The length of the signal is $(n-2)^2$ since $\textsc{Quad}(n-2)$ is delayed one time step. This signal can be seen as a signal of a tailed 1-Line starting from a standard configuration (see Lem. 12). Therefore, after $(n-2)^2$ time units, the cell $(1,1)$ enters a tail state, say G'. G' is then taken as the General state and a minimal time synchronization on a linear array of n cells is executed on the first row, during $(2n-2)$ time steps. Once the corresponding Firing state F' is reached at the last synchronization, F' is then considered as the General state of minimal time synchronizations that gets then invoked on each column. These column synchronizations take another $(2n-2)$ time units, which adds up to a total synchronization time of n^2. □

Theorem 5. *There is a synchronization of a* 1*-Line with* n *cells in time* 2^n*.*

Proof. Synchronization process is divided into two phases. The first phase consists of the signal $\mathrm{cat}_1(\textsc{Mark}(n-1), \textsc{Exp}(n-1,1))$ of length $2^n - 2n + 2$, see Figure 6. By Lemma 12, such a phase is actually a signal of a tailed 1-Line starting from a standard configuration. At the end of this first phase, the cell 1 enters a tailed state, say G' and this state is then taken as a new General state and a minimal time synchronization process on a line is initiated; this is the second phase. These two phases together create a solution for the FSSP in time 2^n. □

Theorem 6. *There is a synchronization of a* 1*-Square in time* 2^n*.*

Proof. First a signal $\mathrm{cat}_1(\textsc{Exp}(n-2,3), \textsc{Mark}(n-2))$ is started on the first row, see Figure 6. After $(2^{n-1} - 2n - 3)$ time units, the cell $(1,1)$ enters a tail state, say H. This is a signal of a tailed 1-Line starting from a standard configuration (see Lemma 12). Now the cell $(1,1)$ enters a state G' and a minimal time synchronization on the first row is accomplished, using G' as the General state, thus taking other $(2n-1)$ time units. Once the Firing state F' is reached, each cell of the first row enters a state G'', and launches the signals $\textsc{Mark}(n-2)$ and $\textsc{Exp}(n-2,1)$ on each column, using G'' as new General state. This takes another $(2^{n-1} - 2n + 5)$ time units, which sums up to time $(2^n - 2n + 1)$. Finally, a minimal time synchronization on each column is accomplished, thus reaching time 2^n. □

The proof of the existence of a synchronization of a 1-Line in time $n\lceil \log n \rceil$ and in time $n\lceil \sqrt{n}\, \rceil$ is quite involved and long, see [12]. Here we recall synchronization for a 1-Square.

Theorem 7. *There is a synchronization of a* 1*-Square in time* $n\lceil \log n \rceil$ *and in time* $n\lceil \sqrt{n}\, \rceil$.

Proof. The algorithms resemble those used to synchronize a line of n cells at the same times in [12]. Therefore, here we only outline the main idea. For the synchronization in time $n\lceil \log n \rceil$, we use a signal to synchronize the first row in time $(n \log n - 2n)$ and then we apply a synchronization to each column in time $2n$ (that is a minimal time synchronization for a linear array with one more time unit).

Let us informally describe synchronization of the first row. Initially, the cells numbered as $(1, 5)$,$(1, \lceil n/2 \rceil)$, $(1, \lfloor n/2 \rfloor + 1)$ and $(1, n - 4)$ are marked: this can be easily accomplished in time $2n$. This way the row can be seen as being split into two halves and for each half a symmetric synchronization is done, in parallel. Therefore, we describe synchronization only for the left half. A phase is iterated $(\lceil \log n \rceil - 5)$ times: each iteration starts at time $((i+1)n+1)$, $1 \leq i \leq (\log n - 5)$, and has length n. During the i-th iteration, the test $(i+5) \geq \lceil \log n \rceil$ is performed in the following way: a signal of length $2^{(i+5)}$ on the linear array consisting of the first $(i + 5)$ cells and a signal MAX of length n, which is composed of MAX$(1, \lceil n/2 \rceil)$ and MAX$(\lceil n/2 \rceil, 1)$, are realized (see Fig. 8). We compose the two signals to give the signal MAX a higher priority. This means that if the exponential signal reaches a cell after the MAX signal, it is aborted. In this case the MAX signal finishes earlier than or at the same time as the exponential signal, and this means that $(i + 5) \geq \log n$ and therefore this is the last iteration. Otherwise (that is if MAX finishes later) cell $(i + 1)$ is marked and a new iteration starts (see Fig. 8). Omitting unessential details, at the end of the last iteration all cells are forced to be in tail states, what creates a standard configuration for a synchronization of a linear array of $\lceil n/2 \rceil$ cells in time n. The synchronization in time $n\lceil \sqrt{n}\, \rceil$ can be obtained in a very similar way by considering a quadratic signal, instead of an exponential one, to synchronize the first row in time $(n\sqrt{n} - 2n)$. □

Corollary 1. *There are synchronizations of a c-Line, c-Square and a c-Ring,* $c > 1$*, in time* n^2*,* 2^n*,* $n\lceil \log n \rceil$ *and in time* $n\lceil \sqrt{n}\, \rceil$.

7. One-way communication networks

In this section we present synchronization algorithms for circular networks, ORings and OToruses, in time n^2, $n\lceil \log n \rceil$, $n\lceil \sqrt{n}\, \rceil$ and 2^n. Synchronization algorithms in time 2^n are obtained by converting a synchronization for a Line. As in the previous section, for technical reasons, we start numbering cells from 1.

The following theorem yields synchronization in time n^2.

Theorem 8. *There is a synchronization of a ORing and of a OTorus in time* n^2.

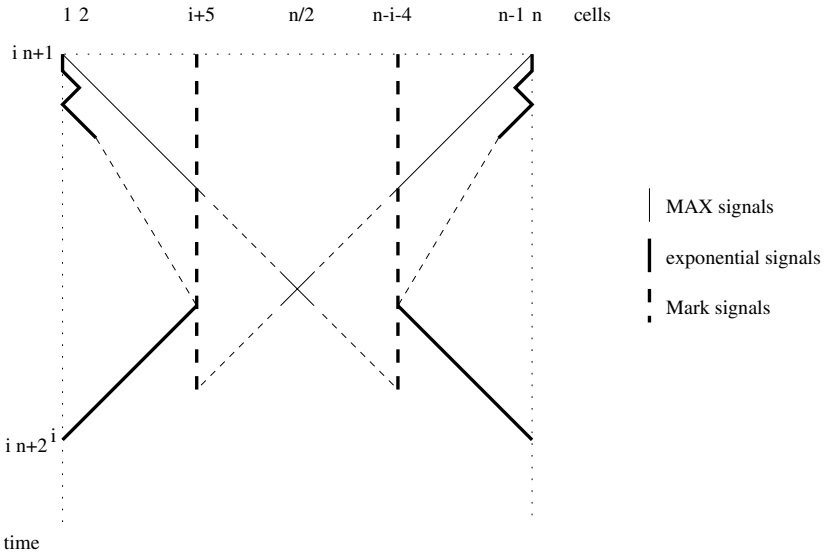

FIGURE 8. The phase in the i-th iteration, $i > 1$, and for n odd of the synchronization in time $n\lceil \log n \rceil$.

Proof. At first we consider ORings and assume $n \geq 3$. The case $n < 3$ can be dealt with a simple *ad hoc* strategy and we omit it (moreover, Lem. 13 can be used with the test $n \geq 3$ to select a synchronization, see Ex. 4). The algorithm is very intuitive, thus we will not give details of signals.

Synchronization will be divided into two phases: the *Counting* phase and the *Synchronization* phases. The Counting phase will have length $(n-2)n+1$ and can be seen as consisting of $n-2$ iterations of a sub-phase of n steps. This sub-phase is simply a MAX-signal going all along the ring, from the first cell to the last one. In the first iteration, the cell number 3 is marked with a marker M and at each successive iteration M is moved one cell to the right, so M is moved to the first cell when $n-2$ iterations have been executed, that is at the time $(n-2)n+1$. This phase is a signal of a tailed 1-Line starting from a standard configuration (see Lem. 12). The Synchronization phase consists of a minimal time solution (in time $2n$) on a ring and can start exactly at time $(n-2)n+1$. The total solution has therefore the length n^2.

Now let us consider a OTorus. Here we assume $n \geq 5$ and, as before, Lemma 13 is used to select the behaviour. The synchronization in time n^2 is easily obtained through the following two steps:

- the first row is synchronized in time $2n$ with a minimal time solution for a ring;
- a solution in time $n^2 - 2n$ is applied to each column.

The solution in time $n^2 - 2n$ is easily obtained from a solution in time n^2 on a ring and modifying the first iteration of the Counting phase in order to mark the cell 5

(instead of cell 3). In this way the Counting phase is formed by $n-4$ iterations of the sub-phase, thus saving $2n$ steps. □

Now we show how to obtain a solution in time $n \log n$. Let us recall that, by Lemma 5, there is a two-end synchronization of a c-Line in time n. Notice that it is easy to modify this algorithm in such a way that in the $(n-1)$-th configuration the processor $\lceil n/2^i \rceil - 1$, for a given i, is in a particular state which is different from all other states entered by any processor (actually this is a signal of type MARK). A similar result can be easily obtained for the ORing as well.

Lemma 17. *There is a synchronization of an ORing in time $2n$ such that in the configuration $2n-1$ the processor $\lceil n/2^i \rceil - 1$, for a given $i \geq 0$, is in a particular state which is different from the state of any other processor.*

Now we can give the synchronization in time $\lceil n \log n \rceil$.

Theorem 9. *There is a solution of an ORing and of a OTorus in time $n\lceil \log n \rceil$.*

Proof. First consider an ORing. Let us assume, for a moment, that $n > 8$. The solution is divided into three phases: the *Initialization*, the *Iterative* and the *Synchronization* phases. The Iterative phase is executed if $n > 16$, otherwise it is skipped. Informally speaking, the whole solution is described as follows.

In the Initialization phase, the cell $\lceil n/16 \rceil - 1$ is marked with a particular state, call it *marker*. Then the cell 0 is marked if and only if $n \leq 16$. Using Lemma 17 this phase can be realized in time $2n$.

In the Iterative phase, at the i-th iteration, the marker is moved from the cell $\lceil n/2^{i+3} \rceil - 1$ to the cell $\lceil n/2^{i+4} \rceil - 1$, for $i = 1, \ldots, \lceil \log n \rceil - 4$, and again the cell 0 is marked if $n \leq 2^{i+4}$. The i-th iteration starts at time $(i+1)n+1$ and ends at time $(i+2)n+1$. Note that the first step of the i-th iteration coincides with the last step of the $(i-1)$-th iteration. Thus the total time taken by this phase is $n(\lceil \log n \rceil - 4) + 1$. The third phase is actually a minimal time solution. Thus, the total time is $2n + n(\lceil \log n \rceil - 4) + 1 + 2n - 1 = n\lceil \log n \rceil$.

The case $n \leq 8$ can be easily solved with a particular strategy and the appropriate behaviour can be selected by using the method described in the proof of Lemma 13.

Now let us consider an OTorus. We assume that $n > 32$ and, as before, we can use the method from the proof of Lemma 13 to choose the behaviour. The solution in time $n\lceil \log n \rceil$ is then easily obtained through the following two steps:

- the first row is synchronized in time $2n$, with a minimal time solution on an ORing;
- a solution in time $n\lceil \log n \rceil - 2n$ is applied to each column.

The solution in time $n\lceil \log n \rceil - 2n$ is now easily obtained from the solution in time $n\lceil \log n \rceil$ on a Ring by modifying the Initialization phase in order to mark the cell $\lceil n/64 \rceil - 1$ (instead of cell $\lceil n/16 \rceil - 1$), thus saving $2n$ steps. □

Theorem 10. *There are synchronizations of an ORing and of an OTorus in time 2^n.*

Proof. A synchronization for an ORing in time 2^{n-1} can be obtained, from Theorem 5, by putting a General state in the second cell and then starting a synchronization on $n-1$ cells. In an analogous way it is possible to obtain a solution in time 2^{n-2}. Using standard techniques as in Lemma 1, any computation of a Line A in time $t(n)$ can be executed by a Ring B in time $2t(n)-1$. In fact, let us assume that the cell $i+j-1$ of B at time $2j-1$ has the state that the cell i of A has at time j. Now the cell i of A at step j needs the states of cells $i-1$ and $i+1$ at time j. The cell $(i-1)+(j-1)$ of B at step $2j-1$ passes its own state p to the cell $(i+(j-1))$ and this forwards p along with its state to the right neighbouring cell, the cell $(i+1)+(j-1)$, that at step $2j$ can simulate the cell i of A at step j. Now, by this simulation and Theorems 8 and 9, for an ORing, synchronization algorithms in time 2^n and 2^{n-1}, respectively, are achieved. Moreover, a synchronization of an OTorus in time 2^n can be obtained by first synchronizing the first row in time 2^{n-1} and then all columns, with the same algorithm as well. □

8. Composed solutions

In this section we briefly describe new synchronizations on a c-Square using known methods to synchronize a c-Line. Afterwards, we show how to construct synchronizations in any time expressed by polynomials with nonnegative integer coefficients.

In Section 6, we have given synchronizations for a c-Line in the following times: n^2, 2^n, $n\lceil\log n\rceil$, and $n\lceil\sqrt{n}\,\rceil$. Combining these results, using Lemma 16, we obtain the following corollary.

Corollary 2. *If $K = 2n-1$, then there are synchronizations on a c-Square of n cell in time K^2, 2^K, $K\lceil\log K\rceil$, and $K\lceil\sqrt{K}\,\rceil$.*

The following lemma is crucial to obtain the above synchronizations in polynomial time.

Lemma 18. *Given a synchronization on a c-CA with n cells in time $t(n)$, there exist synchronizations in times $t(n)+n$ and $n\cdot t(n)$.*

Proof. By Lemma 5, there exists a synchronization on a c-Line in time n, if the starting configuration has the General at both endpoints. We have shown, in Section 3.3, that there exists a synchronization on a c-Square in time n if the starting configuration has the General in all four corners. Clearly, these synchronizations hold also on a c-Ring and on a c-Torus. To obtain a synchronization in time n on a c-ORing, we split the ring in two halves and run the above synchronization on a c-Line in time n on both halves at the same time. This requires to start from a configuration where the General is at the cells 0, $\frac{n-1}{2}$, $n-1$, if n is odd and at the cells 0, $\lfloor\frac{n-1}{2}\rfloor$, $\lceil\frac{n-1}{2}\rceil$, $n-1$, otherwise. A synchronization in time n on a c-Square of ORings can then be obtained by running the above solution on all rows at the same time and starting from a configuration where the General is for $i=0,\ldots,n-1$ at cells $(i,0)$, $(i,\frac{n-1}{2})$, $(i,n-1)$, if n is odd and at cells $(i,0)$, $(i,\lfloor\frac{n-1}{2}\rfloor)$, $(i,\lceil\frac{n-1}{2}\rceil)$, $(i,n-1)$, otherwise. Since it is possible on various models to

mark, in time $t(n)$, all those cells we need to enter the appropriate configuration for the above synchronizations in time n, we have that, by Lemmas 14 and 15, synchronizations in times $t(n) + n$ and $n \cdot t(n)$ can be constructed. □

By that we have proved the following theorem.

Theorem 11. *Let $h \geq 2$ be an integer and $a_0, \ldots, a_h$ be natural numbers with $a_h \geq 1$. There is a synchronization in time $a_h n^h + \ldots + a_1 n + a_0$, where n is the number of cells in the linear networks and of the rows and columns on a square, on a c-Line, a c-Square, a c-Ring, a c-Torus, an ORing, and an OTorus.*

Proof. From Corollary 1, Lemma 18, and Theorem 8, a synchronization in time n^b can be obtained for every $b \geq 2$. By composing, using Lemma 14, these synchronizations in time n^b and the minimal time solutions given in Sections 3.2 and 3.3, the theorem follows.

9. Conclusions

We have presented several techniques to design synchronizations for the FSSP on different kinds of networks for several interesting/important and predefined synchronization times.

Our approach has been to define first a formal concept of a *signal* and, starting from several basic signals, we introduced several operations to compose signals in order to get new solutions. We have introduced also, as a parameter, the capacity of the link measured in bits. This has allowed us to classify network models in terms of the overhead on the amount of traffic on the links.

We have not dealt with the problem of the (minimum) number of states of various synchronizations – that problem was of primary concerns in our other papers. Another, and unexplored, question is how to synchronize a c-Line with some other neighbourhoods (for example (-3,-2,-1,0, 2)). This questions may have some connections with open questions of [21]).

Acknowledgements. This is to thank to anonymous referees for their valuable comments and suggestions

References

[1] R. Balzer, An 8-states minimal time solution to the firing squad synchronization problem. *Inform. Control* **10** (1967) 22–42.
[2] B.A. Coan, D. Dolev, C. Dwork and L. Stockmeyer, The Distributed Firing Squad Problem. *SIAM J. Comput.* **18** (1989) 990–1012.
[3] C. Choffrut and K. Culik II, On Real Time Cellular Automata and Trellis Automata. *Acta Inform* **21** (1984) 393–407.
[4] K. Culik, Variations of the firing squad problem and applications. *Inform. Process. Lett.* **30** (1989) 153–157.
[5] E. Goto, A Minimal Time Solution of the Firing Squad Problem. *Lect. Notes Appl. Math.* Harvard University **298** (1962) 52–59.

[6] K. Imai and K. Morita, Firing squad synchronization problem in reversible cellular automata. *Theoret. Comput. Sci.* **165** (1996) 475–482.
[7] K. Imai, K. Morita and K. Sako, Firing squad synchronization problem in number-conserving cellular automata, in *Proc. of the IFIP Workshop on Cellular Automata, Santiago (Chile)* (1998).
[8] K. Kobayashi, The Firing Squad Synchronization Problem for Two Dimensional Arrays. *Inform. Control* **34** (1977) 153–157.
[9] K. Kobayashi, On Time Optimal Solutions of the Firing Squad Synchronization Problem for Two-Dimensional Paths. *Theoret. Comput. Sci.* **259** (2001) 129–143.
[10] S. La Torre, J. Gruska and D. Parente, Optimal Time & Communication Solutions of Firing Squad Synchronization Problems on Square Arrays, Toruses and Rings, in *Proc. of DLT'04. Lect. Notes Comput. Sci* **3340** (2004) 200–211. Extended version at URL: `http://www.dia.unisa.it/∼parente/pub/dltExt.ps`
[11] S. La Torre, M. Napoli and D. Parente, Synchronization of 1-Way Connected Processors, in *Proc. of the 11th International Symposium on Fundamentals of Computation Theory, FCT 1997, Krakow, Poland, September 1–3, 1997*, edited by B. Chlebus and L. Czaja, *Lect. Notes Comput. Sci.* **1279** (1997) 293–304.
[12] S. La Torre, M. Napoli and D. Parente, Synchronization of a Line of Identical Processors at a Given Time. *Fundamenta Informaticae* **34** (1998) 103–128.
[13] S. La Torre, M. Napoli and D. Parente, A Compositional Approach to Synchronize Two Dimensional Networks of Processors. *Theoret. Inform. Appl.* **34** (2000) 549–564.
[14] J. Mazoyer, A six states minimal time solution to the firing squad synchronization problem. *Theoret. Comput. Sci.* **50** (1987) 183–238.
[15] J. Mazoyer, On optimal solutions to the firing squad synchronization problem. *Theoret. Comput. Sci.* **168** (1996) 367-404.
[16] J. Mazoyer and N. Reimen, A linear speed up theorem for cellular automata. *Theoret. Comput. Sci.* **101** (1992) 58–98.
[17] J. Mazoyer and V. Terrier, Signals in one dimensional cellular automata. *Research Report N. 94–50*, École Normale Supérieure de Lyon, France (1994).
[18] F. Minsky, *Computation: Finite and Infinite Machines*. Prentice-Hall (1967).
[19] E.F. Moore, *Sequential Machines, Selected Papers*. Addison-Wesley, Reading, Mass, (1964).
[20] Y. Nishitani and N. Honda, The Firing Squad Synchronization Problem for Graphs. *Theoret. Comput. Sci.* **14** (1981) 39–61.
[21] Z. Roka, The Firing Squad Synchronization Problem on Cayley Graphs, in *Proc. of the 20-th International Symposium on Mathematical Foundations of Computer Science MFCS'95, Prague, Czech Republic, 1995. Lect. Notes Comput. Sci.* **969** (1995) 402–411.
[22] I. Shinahr, Two and Three-Dimensional Firing Squad Synchronization Problems. *Inform. Control* **24** (1974) 163–180.
[23] A. Waksman, An optimum solution to the firing squad synchronization problem. *Inform. Control* **9** (1966) 66–78.