AN EXACT ALGORITHM FOR THE TECHNOLOGY FITTING PROBLEM IN THE APPLICATION SPECIFIC STATE MACHINE DEVICE

*Marek A. Perkowski +, Malgorzata Chrzanowska-Jeske +, Alan Coppola *, Edmund Pierzchala +*

+ Department of Electrical Engineering, Portland State University, P.O. Box 751, Portland, OR 97207.
* Cypress Semiconductor Inc., 12225 S.W. 2nd St. Beaverton, OR 97005.

ABSTRACT

In the paper the fitting problem for a new Application Specific State Machine Device, CY7C361, from Cypress Semiconductor is formulated and the solution is proposed. This fitting problem consists of mapping the netlist obtained from high-level synthesis into the chip's physical resources. In general, the mapping (fitting) problem can be formulated as one of the labeled graph isomorphism between the netlist graph and the sub-graph of the resources graph. However, the specific architecture-related constraints of the CY7C361 device cause the fitting problem to be generalized as a graph isomorphism problem with some additional mapping constraints (such as mapping some nodes of the netlist graph to more than one node of the physical graph - nodes' multiplication). Such formulation is quite general for a class of Electronically Programmable Logic Device (EPLD) fitting problems, and has not been found in the literature. We implemented an exact, constraint-based, tree searching algorithm with several kinds of backtracking.

1. INTRODUCTION.

Finite State Machines (FSM) are commonly used in VLSI chips, programmable devices, and board level designs to perform sequential logic functions. Such machines can be realized in various programmable architectures. In the development systems for each of those architectures there exist stages of *logic synthesis* and *physical synthesis*. In this paper we focus on the physical design problem for one particular type of device, namely the *Application Specific State Machine Devices (ASSMDs)*, and especially the CY7C36 chip, developed recently by Cypress Semiconductor [1]. The CY7C361 device has been optimized for very fast realization of asynchronous sequential distributed circuits. The high speed of the device is achieved through its unique architecture with *CDEC gates* and *shift registers* [2]. This architecture is different from both standard AND/OR EPLD architectures, and cell/channel FPGA devices. The main characteristic features of the CY7C361 device are: (1) the architecture is partitioned hierarchically into groups of state macrocells (state cells for short) and their excitation functions; (2) all state macrocells are different than standard flip-flops and can be all programmed as a shift register structure(s); (3) non-standard logic gates (the CDECs) are used to implement transition and reset functions of the state cells; (4) some reset gates must be multiplicated to satisfy the technology constraints imposed on the placement of macrocells and gates; (5) The output functions are ORs of the state signals from the state macrocells. (6) The CY7C361 device observed from outside behaves as an asynchronous circuit, it is, however, internally synchronized. Therefore, this architecture requires creation of a unique design methodology. Such methodology [2] is based on partitioning a design into smaller communicating machines, and possibly, realizing also each of those smaller machines as a parallel state machine.

In this paper we present one stage of CY7C361 design system: the *Fitter*, which places and routes the netlist resulting from logic synthesis to the cells and connections existing in the device Subsequently the device is programmed according to the results from the mapping stage.

2. THE IDEA OF THE APPLICATION SPECIFIC STATE MACHINE ARCHITECTURE.

The concept of a Tokenized State Machine generalizes the model of the FSM, and comes from the practical methods used by industrial companies such as IBM and Hewlett-Packard. This model has parallelism on two levels of description. First, the machine is a *network of FSMs*. Second, each of those FSMs in the network can be again a *Parallel Finite State machine (PFSM)*. Each component state machine from the network can have many *tokens* (i.e. "control points") in a "state" (node) at one time. Such machines are also called *"Multiple State Machines"* or *"Concurrent State Machines"*. The tokens can multiply and disappear, but not more than one token can exist in any *node* of the machine [2]. The tokens multiply when there are several edges with the same condition pointing out of a node. The token merge in JOIN nodes to a single token. This concept is close but not identical to *"safe Petri Nets", "Parallel Program Graphs"* of Karp and Miller, and several close formalisms that generalize Finite State Machines. We believe, however, that our model is easier to learn by industrial designers since it is very similar to state machines.

The user's development software for CY7C361 includes: VHDL compiler, high level synthesizer, logic optimizer, fitter, assembler, simulator, and device programmer [2,4]. The high level synthesis software takes a VHDL description of a design and converts it step-by-step to the netlist of excitation functions, memory macrocells and output cells. It is important at this point that if the user wants to specify a standard FSM, he can just describe it in VHDL - the user does not have to be concerned about what internal format is his description converted to.

The pattern of interconnections between the cells that results from the initial description satisfies always certain constraints on the type and connections of various types of cells. For instance, if a group of cells is a *C_IN chain* (a shift register) most of the cells in it are pulse generators (ST type storage macrocells). These properties are later on used by the Fitter program. Although resource mapping problems are well-known in designing the EPLD development software, we are not aware of any published algorithms to solve the fitting problem. Therefore, in our opinion, the most important contribution of the reported research is the formulation of the fitting problem and the first attempt at an *exact* algorithm to solve it.

3. THE ARCHITECTURE OF THE CY7C361 CHIP.

The task of this section is to present the peculiarities of the device that affect the process of physical design. Given below are the reasons why the CY7C361 chip may look unfamiliar to its user:

1. The reset functions and the excitation functions of the flip-flops in the state cells *are not* the general-purpose Sum-of-Products Boolean Functions nor similar logic in standard PLDs. To achieve high speed they are designed as unique gates called *Condition Decode (CDEC)*. CDEC gate is an AND of two, fan-in unlimited, gates, AND and NAND. These gates take primary inputs and state cell outputs as the arguments. The presence of CDEC gates when designing with the CY7C361 means that the designer cannot use the standard FSM synthesis methods [6], but special behavioral TSM transformations [2] and CDEC-based logic synthesis algorithms [5] must be used.
2. There are three types of storage elements in the state cells of CY7C361: Start (ST), Toggle (TO) and Terminate (TE). Only the **Toggle cell** corresponds directly to the familiar synchronized T type flip-flop, for which general design methods have been developed and are commonly used. The **Start cell** has one of three possible applications in TSM design: as a counterpart of D type flip-flop, as a "Dif-

0-7803-0593-0/92 $3.00 1992 IEEE

ferentiation Operator", which differentiates the input signal, and as an "extender" for SOP excitation functions of T and JK flip-flops (realized with TO and TE cells, respectively). The **Terminate cell** is somewhat similar to a synchronized JK flip-flop but it is set ON (J input) only by the C_IN "shifting" input from the previous state cell and it *terminates* (K input) when the CDEC-implemented condition of this cell is *satisfied*. The TE cell needs then at least one additional ST cell in order to emulate a JK flip-flop.

3. All state cells can be configured to shift registers by using C_IN inputs.
4. Even stronger restrictions are imposed on the output functions of the CY7C361 chip. At the first approximation one can assume that each output signal from the CY7C361 chip is only an OR function of the cell outputs. This restriction is very important as it means that one cannot realize the decoding of cell's outputs, such as $Q_1\overline{Q_2}$ or $Q_1\overline{Q_2}a + Q_3\overline{a}$, where a is a primary input (input 1)
5. The *reset logic* is realized by the CDEC gates as well. Reset can be *Global* (for all cells) or *Local* (for a group of cells). *Condition, Global Reset, and Local Reset* are the inputs signals to a state cell. A separate Condition signal exists for each cell, One Local Reset signal exists for four cells, Global Reset is for all state cells. (Currently in CY7C361 the Local Reset can be only used in a Toggle cell.)

The more detailed specification of the CY7C361 chip can be found in [1,2,3]. The available connections of the state cells are shown in Fig. 1. The architecture is partitioned into two groups of 16 state macrocells with their associated reset signals and CDEC planes. Each of these groups is partitioned again into two groups of 8 storage macrocells. The outputs from all cells in each of the 8-cell groups are available at the CDEC gates of all the cells belonging to that group. All groups of 8 include two groups of 4 with a separate local reset CDEC for each group. The groups of 4 are called *Physical LRE groups*. In each group of 16, the available connections between two groups of 8 are realized through two outputs of one group being available as inputs for the other group and vice versa. One output of each group of 4 is globally available to all other groups of 4.

As mentioned, each group of 4, the physical LRE group, has a common local reset signal. This causes that in each physical LRE group there are two possibilities:

1. After placing the netlist graph *(the symbolic graph)*, the cells of the group are an arbitrary collection of TE, ST, and TO type cells (or any subset of those types), and the TO cells (if present) *do not* use the local reset signal.
2. After placing, the cells are an arbitrary collection of TE and ST cells (or any subset of those types, possibly empty), and some TO cells. There exists a TO cell that uses certain symbolic local reset signal. In this case *all other TO cells in this LRE group must use the same local reset signal.*

This constraint has the most significant impact on the fitting problem.

Fig. 2 presents a *physical adjacency matrix* of all state cells (1...32), global reset cell (33) and local reset cells (34...41). The rows represent outputs of cells, the columns the inputs to cells' CDEC gates. This matrix has some symbolic graph mapped into it. Symbol | stands for a physical connection between the output of the row cell and the input to the column cell of the resources graph, which has not been used. Symbol E stands for a the physical connection that is "used" to place a netlist connection. Symbol * stands for an attempted connection which failed, because of lack of respective physical connection on the device. Empty space stands for no physical connection on the chip, and the corresponding resources graph which represents these connections is called the *Physical Graph).*

4. FORMAL SPECIFICATION OF THE FITTING PROBLEM.

The task of the Fitter is to fit the netlist of a TSM obtained in the logic synthesis stage into the resources of the device.

The netlist is represented by a directed graph *SG = (NOS, EDS)*, called the *symbolic graph.* It has a set of *symbolic nodes NOS* and a set $EDS \subset NOS \times NOS$ of *symbolic edges.* Each node of SG is labeled with a single label: ST, TE, TO, LRE, or GRE. ST, TE and TO are *state labels.* They correspond to symbolic nodes: Start, Terminate and Toggle, respectively. Labels LRE and GRE are for symbolic local reset and global reset nodes, respectively. Nodes of types ST, TE and TO are called *state nodes.* Nodes of types LRE and GRE are called *reset nodes.* Each state node of type TO has at most one symbolic LRE node connected to it. Each state node has one GRE node. In this graph the reset nodes cannot be connected with reset nodes (if *sn* is a reset node and $(sn, sm) \in EDS$ then sm is not a reset node). Let us also observe that there can be more than 4 (up to 32) state node successors of a LRE node (state cells cleared by the same LRE signal). This means that more than one physical LRE group must be used for placing these successors, and the symbolic LRE node must be multiplicated. **This fact, plus point 2 from the previous section constitute the essence of our problem's constraints of which distinguish it from isomorphism problems and require a special algorithm for solution.**

The physical resources are represented by a directed graph PG = (NOP, EDP), called a *Physical Graph.* NOP is an ordered set of *physical nodes,* and $EDP \subset NOP \times NOP$ is a set of *physical edges.* Graph PG is the one to which graph SG is mapped. State nodes in a set NOP are ordered as on the chip (when reconfigured to a shift register). Each node of graph PG has its *physical type. The types are: physical state nodes, physical LRE nodes,* and *a physical global reset (GRE) node.* Each physical LRE node provides a local reset for four physical state nodes. GRE node is a global reset of all state nodes.

Function *mapping* $M: NOP \rightarrow NOS$ from nodes of PG to sets of nodes of SG specifies the *placement of nodes* from set NOS to **sets of nodes** from set NOP, which means that a node from set NOS can be placed in more than one node from set NOP^+. The finding of this function, or in other words, assigning the nodes from NOS to the nodes from NOP, is the task of the Fitter. Some symbolic LRE nodes are placed in several physical LRE nodes, so our problem can be called *graph monomorphism with nodes multiplication.*

The "generic fitting problem" for "CY7C361-like" devices can be formulated as follows:

GIVEN is:

a. The Symbolic Graph SG = (NOS, EDS) as described above.

b. The set of *ordered sets* of symbolic state nodes from NOS that form the *C_IN chains.* A node in C_IN chain can be of ST, TE or TO type.

c. The directed Physical Graph PG = (NOP, EDP) (the PG for the CY7C361 chip has been specified in section 3).

FIND:

such mapping M: NOP -> { $NOS \cup phi$ }, where *phi* denotes an empty set, that:

A. Every symbolic node from NOS is a mapping of nodes from NOP (mapping of a node from NOP to *phi* means not using this node for a placement of any node from NOS).

B. Every symbolic state node or GRE node is the mapping of exactly one physical state node or GRE node. (More formally: for every node *sn* from NOS being not a symbolic LRE node there exists exactly one node $M^{-1}(sn)$ from NOP being not a physical LRE node). The symbolic GRE node is a mapping of exactly one physical GRE node. This simply means that the M function is a one-to-one function on the subset NOP - {all physical local reset nodes}.

C. Every symbolic LRE node is a mapping of one or more physical LRE nodes. (More formally: for every node *sn* from NOS being an LRE node there exist(s) one or more nodes pn_i from NOP, being physical LRE nodes, and such that $M(pn_i) = sn$.
This means that function M does not have to be one-to-one for the whole NOP set.

D. Mapping M restricted to physical state nodes plus physical GRE is a *monomorphism* from graph PG to graph SG:
For every edge *(sn, sm)* from EDS such that *sn* and *sm* are state or GRE nodes, there exist exactly one edge *(pn, pm)* from EDP such that $sn = M(pn)$, and $sm = M(pm)$.
Mapping M restricted to physical LRE nodes is a *many-to-one function* from graph PG to graph SG:
   I For every edge (sn, sm_j) $j = 1,...,r$ from EDS such that *sn* is an LRE node and $M(pn_1) = M(pn_2) = ... = M(pn_s) = sn$, there exists such node pn_i, $M(pn_i) = sn$, $i = 1,...,s$ and exactly one edge (pn_i, pm_j) from EDP such that $M(pm_j) = sm_j$.
   II For every edge (sn_j, sm), $j = 1,...,r$ from EDS such that *sm* is an LRE node and $M(pm_1) = M(pm_2) = ... = M(pm_s) = sm$, and

+ This formulation leads to simpler formalization than a more intuitive one, based on mapping from NOS to 2^{NOP}

every pm_i, $i = 1,...,s$ there exists an edge (pn_j, pm_i) from EDP such that $sn_j = M(pn_j)$, $j = 1,.....r$.

Comment 1. Condition I describes mapping of the connections that start from the multiplicated LRE nodes. By multiplication of a node we mean creating several copies of this node.

Comment 2. Condition II describes mapping of the connections that terminate in the multiplicated LRE nodes. It means that if pn_j is a placement of sn; $(pn_j, pm_i) \in$ EDP, for all $i = 1,...s$, $j = 1,....r$ or in other words pn_j is accessible in pm_i.

Comment 3. Multiplication of symbolic nodes is done by mapping of several physical LRE nodes to a single symbolic LRE node, and copying of edges pointing to such multiplicated symbolic LRE nodes. This of course requires checking that the connections to each of the multiplicated nodes (condition II) can be realized.

E. Every C_IN chain of state nodes $sn_1, sn_2, ..., sn_m$ from NOS is a mapping of a sequence $pn_i, pn_{i+1}, ..., pn_{i+m}$ of *successive state nodes* from NOP.

Comment 4. Recall the natural order of nodes from NOP. Symbolic C_IN chain must be placed in successive physical nodes.

F. In each *physical LRE group* (a group of state nodes from NOP which are fed from the same physical LRE node) all toggle nodes pm_j must have the same symbolic local reset *sr*, if the toggles use a local reset signal. In other words: if one toggle in the physical LRE group has a local reset, then all other toggles in this group must have the same local reset, but other types of state cells (not using local reset) are allowed in this LRE group. Allowed is also an LRE group with any type of cells, that do not use LRE signals at all. This is described formally as follows (pr_1 is a physical LRE node):

If:

$(pr_1, pm_1) \in EDP$
and
$M(pm_1)$ is a toggle node in SG
and
$sr = M(pr_1)$ is an LRE node in SG,
and
$(sr, M(pm_1)) \in EDS$,

Then

(for all pm_j, $j = 1,...,s$ *such that* $(pr_1, pm_j) \in EDP$*)*
if $M(pm_j)$ is a toggle *then* $(sr, M(pm_j)) \in EDS$.

5. THE ALGORITHM OF THE FITTER.

The Fitter algorithm is a constructive placement program which places exhaustively nodes of NOS in nodes of NOP, while additionally taking into account several constraints. Whenever a constraint is violated, program looks for the next possible placement of the currently considered symbolic node (this new placement attempt is called a *shift*), or backtracks to the previously placed symbolic node, if no more shifts are possible. Special mechanisms exist to backtrack from C_IN chains. The backtracking mechanism has been constructed in a very efficient way. The program backtracks *immediately* whenever any of the respective constraints are violated. It backtacks also when some specific conditions are satisfied, for which it was proven that their satisfaction causes future violation of some constraint. For instance, an attempt to place C_IN chain of length 4 starting in node 30 of NOP will cause immediate backtrack, since such chain is too long to be placed in the remaining two nodes of the structure. Another property of PG that speeds-up the search is based on the construction method applied in the layout of CY7C361. This property of graph PG can be described as follows:

Nodes from NOP are naturally ordered in such a way that if there is no edge from node *pn* to *pm*, and node *pm* is above the diagonal of the adjacency matrix of PG (which means, cell *(pn,pm)* is above the diagonal) then there is no edge from *pn* to any node above *pm*.

Because the Fitter can potentially search the entire solution space of all mappings, it can always find the solution if one exists. The approach is feasible, since the solution space is essentially restricted by the symmetries of the physical graph and the constraints imposed by the architecture.

The Fitter program is quite complex, its basic principles are as follows:

1. The fitter places symbolic state nodes, taking them one by one from the ordered set NOS and placing them in naturally ordered nodes of NOP from left to right: it maps node *sn* from NOS to node *p1*, next to node *p2*, and so on, of NOP. If node *sx* cannot be placed in node *px* because there is no possibility of finding a connection from node $M^{-1}(sa)$ of a previously placed node *sa* to node $px = M^{-1}(sx)$ and *px* is above the diagonal, then there is no need to check all placements $M^{-1}(sx)$ to nodes py_j, where $py_j > px$, and a backtrack is performed.
2. If node *sr* from NOS is a beginning of a C_IN chain, all its successors are attempted to be immediately placed in a sequence. If, however, this sequence cannot be fitted because of the violation of some constraint (for instance not enough space for this C_IN chain), the next *pn* node for node *sr* placement is tried, followed by an attempt to place its corresponding C_IN chain.
3. The symbolic state nodes are mapped only to physical state cells, and the symbolic reset nodes are mapped only to physical reset cells. Whenever a symbolic state node is placed, its respective symbolic reset node is immediately verified, and if the verification is positive, it is placed. If symbolic state node is placed in a physical node for which physical LRE is already mapped, the consistency of these nodes is verified. If the verification fails, all subsequent possible placements of the state node are attempted, with corresponding multiplications of its symbolic LRE node. This is performed as follows. When a toggle node *sx* with a local reset is placed in node *px*, it is verified if all physical nodes *pj* in the LRE group of *px* are mappings of symbolic toggle nodes with the same symbolic LRE node (see constraint F in section 4). If not, shift or backtrack is performed, depending on the situation. This is repeated until a mapping of node *sx* to node *px* is found, such that all constraints D - F from section 4 are satisfied. If the state node cannot be placed in this LRE group, a new physical LRE group is selected, which means the multiplication of the symbolic node LRE. Using backtracking and constraint verification, this mechanism can create all possible multiplications of the LRE signal. For instance, symbolic LRE with 5 successors can be split to two LRE groups: with 2 and 3 successors, or with 1 and 4 successors in a group, respectively (these groups are denoted as (2,3) and (1,4)). It can be also split to groups: (1,1,3), (2,2,1), (1,1,1,2), and (1,1,1,1,1). So one LRE node can be multiplied up to five physical LRE cells. Let us also observe that each above group has many final physical placements. For instance, group (1,1,3) can be placed as (1,1,3), (1,3,1), (3,1,1), (3,0,0,1,0,0,1) and so on, where 0 means a not used physical LRE group. Moreover, all placements within a physical LRE group are attempted. All these placements are found by the same backtracking/constraint mechanism of the Fitter. The execution of the algorithm depends heavily on the initial order of nodes in NOS. The basic variant of the Fitter starts from the order of symbolic nodes that is either defined by the user or found by the high-level synthesis programs.

In some cases when the designer wants to have more control over his design, he can avoid the high-level VHDL description. This can allow him to make some special design optimizations or apply some specific design tricks. Once this option is selected, the design is described in a "VHDL microcode" that has primitives directly corresponding to the logical resources of the chip, but does not specify the physical mapping, yet. Even in such cases the fitting task was often found to be too difficult to be performed by hand. So the Fitter was made available to process the data described in the VHDL microcode format as well.

Moreover, the feedback from the Fitter allows the designer to re-design his circuit by either modifying logically his high-level VHDL or microcode VHDL description, or by only changing the order of statements in the input specification. To help the designer to achieve this, the visualization facility, shown in Fig. 2, has been added that visualizes the used cells and the connections of the physical resources. When the microcode VHDL is used for initial description, the user can essentially influence the Fitter by the order of cells in his/her description. In the case of the high-level VHDL description, he can influence the Fitter by modifying the order of VHDL primitives. A controlling printout such as one on Fig. 2 can be used to find which symbolic nodes make most troubles and to obtain more clue where and how to change the microcode description. However, the VHDL modification method gives less predictable results.

The constraint-based Fitter described above has been implemented in C++ and runs on PC-compatibles and Sun/Sparc workstations. It has been tested on more than 22 examples. The Fitter found very quickly (in seconds) solutions in all but six cases of large and dense graphs. In those cases the work of the program was interrupted after 3 to 5 hours without solution. All those cases are extremely hard ones for testing the Fitter. The statistical

analysis of labeled symbolic graphs representing real-life netlists is given in Table 1.

6. CONCLUSIONS AND CURRENT WORK

The presented Fitter is under further improvements. Our current research is on adding other powerful heuristics to the program so that it will be able to find quickly solutions to all tough examples produced by the newest high-level synthesis programs [4,5]. We created algorithms that cooperate with the main backtracking strategy to start algorithm repeatedly from good starting points. The speed of finding the solution depends heavily on the order of nodes in set NOS. Our new algorithms create partial initial placements and NOS orderings. The algorithms include: (1) *Exact Algorithm for Hierarchical Bi-Partitioning*, which also uses backtracking but searches a smaller solution space. (2) *Approximate Algorithm based on Simulated Annealing*, (3) *Iterative Learning Algorithm*, in which the nodes making most troubles are placed at first and moved more often.

We keep looking for more efficient and more general approaches to solve this class of problems, since they are of increasing importance for several types of incoming new devices. On the other hand, it seems that the existing CAD algorithms will not be very useful, since the fitting problems are not similar to the well-known physical design and graph-theory problems. We would, however, appreciate any hints and ideas.

LITERATURE

[1] Anderson, R., "Programmable High-speed State Machine with Sequencing Capabilities", U.S. Patent No. 4,965,472, Oct. 23. 1990.

[2] Coppola, A., Perkowski, M.A., Anderson, R., Freedman J.S., and E. Pierzchala, "Optimal Synthesis of Tokenized State Machines into a Programmable Logic Device", *PSU EE Dept. Report*, April 1991.

[3] Cypress Semiconductor, Warp1 PLD Compiler.

[4] Cypress Semiconductor, WARP1 Beta Version 0.9 Release Note.

[5] Perkowski, M.A., and A. Coppola, "A State Machine PLD and Associated Minimization Algorithms", *Proc. of FPGA'92 Conference, Berkeley, Febr. 1992.*

[6] Kohavi, Z., "Switching and Finite Automata Theory", McGraw-Hill, New York, 1970.

AJC Data for Fitter					
Name	#LB	#Connects	Max-conn	#CINS	#Backtracks
busa	16	12	3	10	7
cntcmp	9	36	5	0	48
dees2	301	100	7	5	NC(>500k)
dram1	22	28	6	11	3101
epee1	24	35	8	13	13
pee1-res	27	53	13	13	NC(>250k)
spee-con	16	34	6	8	28
example8	10	64	9	0	8516
hotrea	12	10	5	6	0
longcto	26	170	18	0	NC(>500k)
m_fsm	14	10	1	3	0
mbarbnr_ba	31	28	4	22	NC(>500k)
mlt_fsm	21	22	3	5	7
reaword	15	23	4	5	11
rtl-ba	19	30	5	6	1380
seqdetec_fit	14	18	5	9	18
tgen-ba	25	60	10	0	NC(>500k)
tsr2-bug	18	19	2	2	7968
vmerq3	22	20	9	10	0
vmesup11	19	26	7	11	13
warpt	32	62	5	2	NC(>500k)
word	19	34	6	8	90

Table 1.

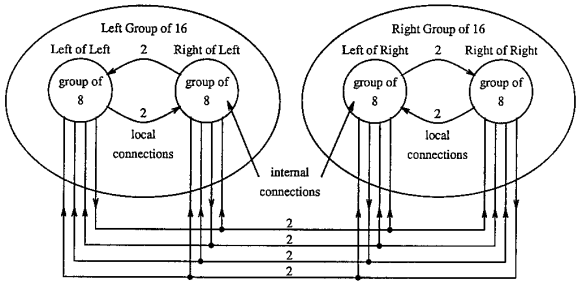

Figure 1.

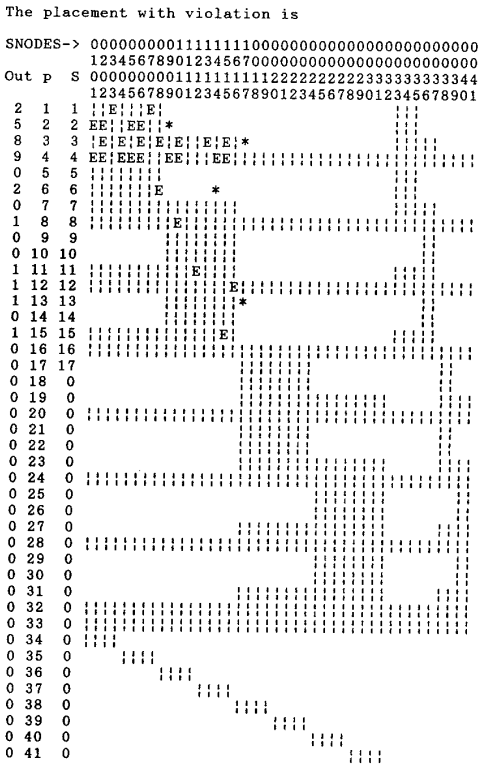

Figure 2.