VLSI DESIGN
1999, Vol. 9, No. 2, pp. 159–180
Reprints available directly from the publisher
Photocopying permitted by license only

System-level Time-stationary Control Synthesis for Pipelined Data Paths*

JONG TAE KIM[a], FADI J. KURDAHI[b,†] and NOH BYUNG PARK[c]

[a]*Department of Computer Engineering, Chonbuk National University, Chonju, Chonbuk, 560-756, Korea;*
[b]*Department of Electrical and Computer Engineering, University of California, Irvine, CA 92717, USA;*
[c]*Samsung Electronics, Korea*

(Received 16 August 1994; In final form 25 September 1997)

We address the prblem of time-stationary control synthesis for pipelined data paths. Control synthesis system accepts scheduled control data flow graph with conditional branches which are produced by high level synthesis tools such as Sehwa [1] as input specification and generates a FSM controller. First a scheduled control/data flow graph is analyzed and the various states are identified. Overlapped states are grouped together to produce L groups where L is the pipeline latency. Next, state transitions are identified and a state table is generated. Finally, a highly optimized FSM controller is implemented by performing horizontal partitioning and the corresponding stae encoding so as to minimize the total controller area. We compared our approach to published work on FSM generation and optimization and the results indicate that our method results in large savings in total controller area.

Keywords: Architectural, synthesis, pipelined systems, controller synthesis, partitioning

1. INTRODUCTION

Pipelining is a widely used approach for designing high performance digital circuts. As design size increases, pipelined architectures become quite complex and thus automatic pipeline design synthesis tools are necessary to cope with such complexity. pipelined data path synthesis problems have been well investigated in [1, 2, 5]. Sehwa [1] performs allocation of functional modules and scheduling of resources and estimates the cost of registers and interconnections. In [5] a method for module assignment and Register-Transfer level synthesis of pipelined data paths was presented. Once an RT-level data path is obtained, the corresponding controller can be synthesized.

A time-stationary control mechanism [6] provides the control signals for the entire pipeline

*This work was supported by NSF Grant#MIP-8909677 and by a TRW fellowship. Based on "Automatic synthesis of time-stationary controllers for pipelined data paths", by J.T. Kim, F. J. Kurdahi, and N. B. Park which appeared in International Conference on Computer Aided Design (ICCAD '91), November 1991. © 1991 IEEE.

†Corresponding author. Tel.: (714) 856-8104, Fax: (714) 856-4152, e-mail: kurdahi@balboa.eng.uci.edu

from a single source external to the ppeline. The main characteristic is that at each unit of time these controls govern the entire state of the machine. The design of this type of controller is a complex task since the controller must also remember the current pipe state in order to provide control signals to the pipe stages occupied by multiple overlapping tasks.

The controller is modeled as a Moore style Finite State Machine (FSM) in which state memory holds its present state, and the combinational parts decide the next state and the primary output functions. The combinational circuits can be implemented using PLAs or random logic. The structure of a time-stationary controller is shown in Figure 1. The controller is vertically partitioned into a Sequencing part and a Command part. The Sequencing circuit solely implements the next state function whereas the Command logic generates the output function. The Sequencing logic is partitioned horizontally into two parts since it was observed by Paulin [7] that such a partitioning minimizes the total controller area.

We describe a method to generate control specifications of the pipelined data paths which makes use of the node labelling and mutual exclusion testing techniques described in [1]. The synthesis of the sequencing part follows the Behavioral Synthesis tasks of scheduling and resource allocation. Once the RT-level-data path synthesis is done, (*i.e.*, the tasks of module assignment, and multiplexor and register allocation), the control requirements of each RT level component are known and thus the command logic can be synthesized. Figure 2 describes the flow of tasks in the overall high-level synthesis of data path and control.

Section 2 presents the control specification of pipelined data paths. The partitioning and state

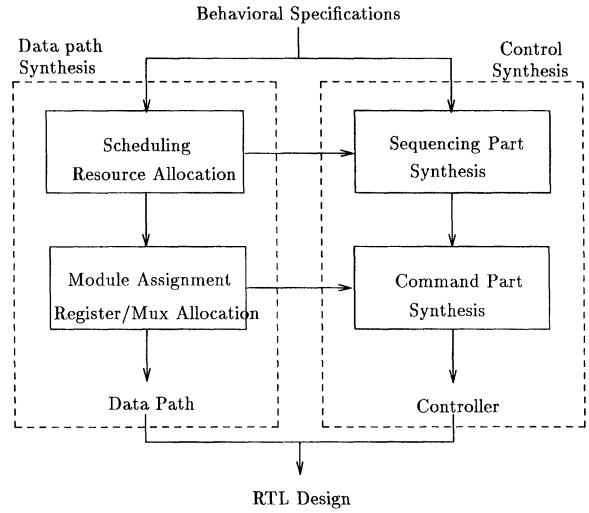


FIGURE 2 The pipelined high-level synthesis system.

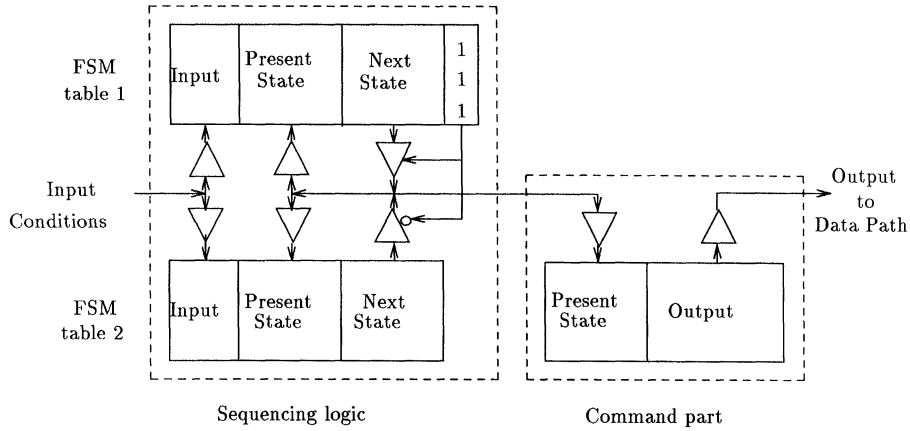


FIGURE 1 The structure of a time-stationary controller.

assignment algorithms are described in Section 3. Section 4 shows some experimental results. Conclusions are drawn in Section 5.

2. RELATED WORK

Control synthesis task can be divided into the generation of control specification and the FSM synthesis. The control specification generation and the FSM synthesis can be referred as control synthesis at the register transfer level and at the logic level, respectively.

Most of the previous work done in the control synthesis at the register transfer level was for nonpipeline systems. Automatic synthesis of microprogrammable control hardware was addressed in [10] using the two optimization algorithms, the autonomy algorithm and the attraction algorithm. It is constrained by the capacity of the microprogram storage, speed requirement, and the number of control signals that can be activated at the same time. The optimization techniques can be used to reduce the number of branches, to shorten conditional baranching time, and to reduce the number of micro cycles which leads to increase the performance of the micro engine. The HAL system tries to use this approach for minimizing the control path cost. The CONSPEC [8] dealt with the automatic production of control specifications from high-level behavioral descriptions in control and timing graph form and is designed for interface processors.

Bridge [9] is a high level synthesis system developed at AT&T Bell Laboratory and performs data path and control path allocation for nonpipelined systems by applying either a local slicing or a global slicing techniques. The Bridge system starts with a micro-architecture model and sometimes the implementation can be reduced to a finite state machine or a combinational circuit. Structural synthesis [12] in the Yorktown Silicon Compiler integrates data path and control synthesis for nonpipelined designs.

Several control synthesis works at logic level are reported in [3, 11]. A state table or an equivalent state transition diagram are used for input specification and the tasks involved are the state assignment and the logic minimization with or without topological optimization. The survey on PLA-based FSM optimization is presented in Section 4.2.

3. PLA-BASED FSM OPTIMIZATION SCHEMES

We can classify the PLA optimization approaches into two categories: Physical Topology Optimization (PTO) and Logical Topology Optimization (LTO). The PTO tools attempt to minimize PLA area by changing the physical layout of the PLA. An example of PTO methods in PLA folding. LTO methods include Vertical Partitioning, Counter Embedding, and Horizontal Partitioning. In practice, many of the LTO algorithms developed are combinations of the above listed methods and are usually associated with a state encoding scheme. Since PTOs do not modify the logic, they can be applied after processing a PLA with the LTO. We are focusing on LTO in this work.

A crucial step to prepare for the minimization is the task of state assignment. The codes of the states are assigned in such a way that results in boolean minimizations. De Micheli proposed a technique for state assignment of Finite State Machines based on symbolic minimization of the FSM combinational component and on a related constrained encoding problem [16]. The algorithm derives a set of coding constraints and encodes so that the associated state transitions can be implemented by one common product term. The distinctive approach that logic minimization of the symbolic description is applied before state encoding was proposed. His solution to the state assignment problem was to find the assignment of minimum code length among the assignments that minimize the number of rows of the PLA.

Amann presented state assignment algorithms that permit the synthesis of optimal counter-based PLA FSM's [18]. The algorithms are divided into two step: state chain calculation and state chain

coding. In state chain calculation the internal states of a finite state machine are ordered to maximally exploit the counter characteristics. In state chain encoding the internal states are coded so that state sequences are maintained. Numerous other works have been done on state encoding. Some of these works can be found in [4, 17, 3].

Vertical partitioning is a classical PLA optimization technique which separates the set of output functions into two or more PLA's while minimizing the number of redundant product terms in all the PLA's [21]. A common technique is the separation of state outputs from command outputs. An initial PLA personality matrix is partitioned to yield the sequence PLA and the command PLA which generate next states and the primary output functions, respectively.

Horizontal Partitioning was proposed by Paulin [7] and combines the advantages of traditional vertical partitioning and counter embedding. It allows the reduction of the number of input and/or output columns in the PLAs resulting from the partition. The technique also reduces the total number of product terms, as in counter embedding techniques. The concept of horizontal partitioning used in Paulin's algorithm is a generalization of Amann's work. The significant difference in procedures is how horizontal partitioning is performed. In paulin's algorithm, the final partitioning of the sequencing PLA into two PLAs is done by considering some boolean relations between various product terms in the sequence PLA. Paulin's algorithm was implemented with minor modifications and enhancements in [26]. In [7], Paulin proposed a horizontal partitioning scheme of PLA-based FSMs. The objectives for the horizontal partitioning are to:

1 Find a partition that satisfies all boolean relations, and
2 Find a partition that holds PTs which depend on common outputs and/or common inputs.

4. CONTROL SPECIFICATION

In this paper, we present a method to synthesize a Moore-style FSM controller specifications given a pipelined data path with conditional branches. The input is a scheduled data flow graph which shows operator-to-time step assignments and dependencies between operations. The output is a FSM specification in the form of a state table. A data flow graph (DFG) is a directed graph representing the functionality of a digital system or a computer program. In a DFG, a node represents an operation on values and a directed edge represents the flow of values between its source and sink nodes. There are many constructs which can represent conditional execution paths in DFGs. However, in this paper, we use OR-FORK and OR-JOIN (also referred to as a *distribute -join* (D-J) node pair [1])[1] to represent conditional execution paths in DFGs. Whenever an execution path is to be selected by some condition, a distribute node must be used to split the values to every possible execution path. Conditional branches can be nested as many levels deep as needed. When the execution path is no longer dependent on the branching condition, a join node is used to indicate the termination of conditional execution. A DFG which is augmented with these D-J constructs is referred to as a Control/Data Flow Graph (CDFG) since it now includes additional control information. At this point, and in order to avoid any ambiguity, it is important to define the term *latency* which will be extensively used throughout this paper.

DEFINITION 1 The number of time units between two consecutive initiations in a pipeline is called the ***latency***, L of the pipeline.

Loops can be modeled in several different ways, *e.g.* [13, 14]. Loops with a small number of iterations can be handled by unrolling as in [1]. Loops can be also treated as conditional blocks as

[1]Although the technique in this paper can easily be extended to multi-way D-Js, we will assume that the D-J pairs are two-way splits.

described in [15], for example. In this case, the inputs to a loop are selected between the inputs from outside the loop and those from previous loop interaction. The selection conditions can come from the loop counter (in a FOR-*loop*) or the corresponding conditions from the data path (in a WHILE-*loop*). Hence, loops can be scheduled without unrolling them at the expense of lower performance and throughput. In this paper, we assume that the system CDFG's to be synthesized are loop free, either as initially specified, or through the application of loop elimination techniques as described above. Extensions to this work towards more general CDFGs are currently underway.

The control specification procedure consists of three major steps: *preprocessing, state decision, and state transition*. In the remainder is this section, we describe these steps and present our approach to solving each one.

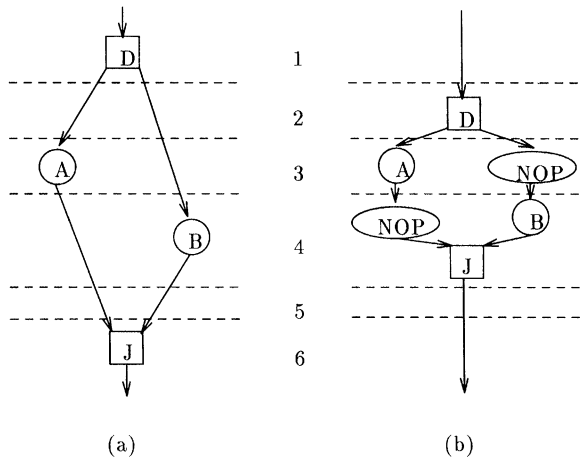


FIGURE 3 Preprocesing: (a) before, (b) after.

4.1. Preprocessing

Some pipeline scheduling schemes (such as Sehwa [1]) assume that the conditional nodes, D and J, have a negligible delay compared to other operations, and thus do not assign them to any specific time steps. In preprocessing we rearrange the D and J nodes, and insert NOP nodes in the CDFG. NOP nodes are inserted along execution paths in order to simplify the control synthesis tasks since now only nodes need to be considered (as opposed to nodes *and* edges). The procedure checks the node distance between two adjacent nodes and if the distance k is greater than 1 it inserts $k-1$ NOP nodes into $k-1$ stages between two nodes. The conditions for each D-J pair is kept in a one-bit memory and is assumed to be available when the pipeline is initiated or at some earlier time step. The number of state is dependent on the Scheduling of the D and J nodes. A subgraph of a CDFG is shown as an example in Figure 3. In Figure 3(a), the conditional branch occurs in time Step 1, nodes A and B are assigned to stages 3 and 4, respectively, and J node is assigned in stage 5. There are no operation nodes in stages 2 and 4, but there are two edges in those stages.[2] Edges with no operation nodes in D-J block increase the number of states in the FSM and our method to avoid this problem is keep the D and J nodes as close as possible, *i.e.*, move a D node to stage i where both or either one of the next nodes are located in stage $i+1$, and a J node to stage j where both or either one of previous nodes are scheduled. This always satisfies the availability of the condition for the conditional blocks, since a D is moved to a later time step. The corresponding condition is still available and the same can be said about the J nodes. Figure 3(b) shows such an arrangement, where the D is moved to time Step 2 and the J node to time Step 3. This reduces the number of edges in time Steps 2 and 4. Therefore reduces the number of states. Suppose that the latency is 2, then there are eight state in Figure 3(a), since NOP nodes and time steps are overlaped. But they can be reduced to four states after rearrangement of *distribute-join* block in Figure 3(b).

[2]In this paper, we will use the terms stage, time step, and control step interchangeably.

4.2. State Decisions

It is assumed that the CDFG schedule is pipelined with a latency of L and that the total number of stages (or time steps) is n_t. Conditional branches are handled by using an algorithm described in [1]. The algorithm assigns to every node a label consisting of a sequence of one or more integer codes. Using these labels, we can test for mutual exclusion between any pair of nodes (operations) in pseudo-constant time. Before going any further, we define the following terms which assume a CDFG scheduling with a latency L.

DEFINITION 3 Given two events in a data flow graph which occur conditionally. If the condition that selects one event always falsifies the condition selecting the other, and *vice versa*, then the two events are called *mutually -exclusive* with respect to each other.

DEFINITION 4 A set M of nodes is said to be a *mutual exclusion set* (MES) if all the nodes in M are pairwise-mutually exclusive and M is not included in any larger MES M'. For a given time step i, we weill denote by $M_{i,1}, M_{i,2}, \ldots$, the MES's which cover the nodes scheduled in i.

Based on Definition 4, MESs are the maximal groups of mutually exclusive operations within a given time step. Next, we find sets of operations which can be executed concurrently in each time step by picking one operation from each MES and combining them.

DEFINITION 5 Let $M_{i,1}, M_{i,2}, \ldots, M_{i,n}$ denote the MES covering time step i, a *Possible Execution Mode or* PEM, P is defined as a set of n operations, one from each MES. $P_i = \{o_1, \ldots, o_n | o_h \in M_{i,h}, h = 1, \ldots, n\}$. We will denote by $P_{i,1}, P_{i,2}, \ldots$, the different PEM's in time step i.

Thus, each PEM $P_{i,j}$ represents a subset of nodes that can be executed in parallel during time step i. Without loss of generality, we can assume that $i \leq L$. Since the schedule is pipelined, time steps i, $i + L$, $i + 2L$, ..., are overlapping and therefore, a *state* can now be defined as follows:

DEFINITION 6 Given $1 \leq i \leq L$, a state S_i is defined as a set of PEM's corresponding to overlapping time steps i, $i + L$, i $2L, \ldots S_i = \{P_{k,l} \mid \forall P_{k,l}, P_{m,n} \in S_i, k \bmod L = m \bmod L = i\}$, and S_i is not included in any larger state S_i'.

We will denote by $S_{i,1}, S_{i,2}, \ldots, S_{i,n_i}$ all the state can be generated by different combinations of PEM's in i and the time steps that overlap with it, and n_i is the number of such different combinations. Since $1 \leq i \leq L$, we can define groups of states $G_1, G_2, \ldots, G_i, \ldots, G_L$ such that $G_i = \{S_{i,j}, 1 \leq j \leq n_i\}$.

Figure 4 shows a scheduled CDFG with node labels. For convenience, we identify the operations in a time step by their node labels between parentheses. Table III shows the state identification procedure. In time step 2, we have 5 nodes and there are two mutual exclusion sets which are $M_{2,1}$ = (10, 110, 111) and $M_{2,2}$ = (20, 21). There

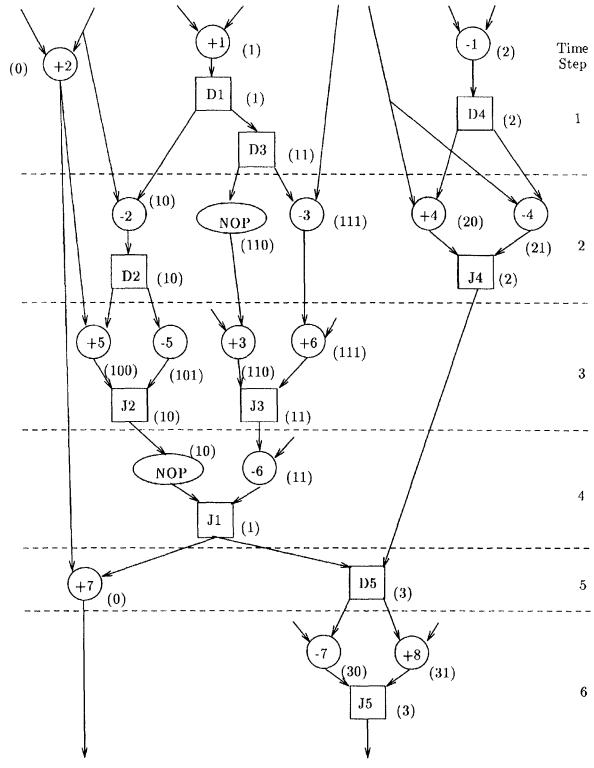


FIGURE 4 *Example* Sehwa-1-Scheduled CDFG with node labelling: L = 3.

TABLE I Values of $R_{L,n}$ for various n and L

L/n	1	2	3	4	5	6
1	1	x	x	x	x	x
2	1	1	x	x	x	x
3	1	3	1	x	x	x
4	1	7	6	1	x	x
5	1	15	25	10	1	x
6	1	31	95	50	15	1

are 6 different PEMs in stage 2. They are $P_{2,1} = (10, 20)$, $P_{2,2} = (10, 21)$, $P_{2,3} = (110, 20)$, $P_{2,4} = (110, 21)$, $P_{2,5} = (111, 20)$, and $P_{2,6} = (111, 21)$. Since the latency in 3, time steps 2 and 5 are overlapped. Note that time step 5 has only one PEM $P_{5,1} = (0, 3)$ since there are no mutual exclusion sets. From these PEMs we can bind operation nodes and states. We can find 6 states in G_2. For example, state $S_{2,1}$ governs nodes (10, 20) ($P_{2,1}$) in time step 2 and nodes (0, 3) ($P_{5,1}$) in time step 5.[3]

4.3. State Transitions

After identifying the states, we need to determine the state transitions. Given a CDFG pipeline-scheduled with a latency L, we observe that state transitions occur between adjacent groups of states in the following sequence: $G_1 \rightarrow G_2 \ldots G_i \rightarrow G_{i+1} \ldots G_L \rightarrow G_1$. This is mainly due to the pipelined nature of the scheduling and is shown in Figure 5. This is a key property in our optimization scheme, as will be discussed later. Another important factor affecting the control specifications are the distribution nodes (D). If the present state has m D nodes, there are 2^m possible combinations of input conditions.[4] Given a particular state, we now present a strategy for finding the next state by considering only node labels.

We define two nodes to be *compatible* if they are not mutually exclusive. Thus, the next state is the one which has all the compatible. (*i.e.*, not mutually exclusive) nodes of the present state. Using the state representation outlined in Section 2.2, can find compatible nodes by searching only the PEMs corresponding to the next time step within states in the next group. Starting with G_1, we choose a present state and find all the possible next states in G_2 for all possible input conditions. This procedure is repeated for all the states in G_1. Next, states in G_2 are considered in the same manner and so on until all L groups of states have been visited. A 0 (or false) condition in a D node implies that the left branch will be traversed and thus a transition must occur to one of the states corresponding to the left child of the D node. If a state is not dependent on a particular D node, then

Table II Efficiencies of the branch and bound and heuristic methods

Latency L	Heuristic # of calculation	Heuristic Saving (%)	Heuristic Execution time (sec)	Branch and Bound # of calculation	Branch and Bound Saving (%)	Branch and Bound Execution time (sec)	Exhaustive search # of calculation	Exhaustive search Execution time (sec)
5	9	40	0.0	12	20	0.0	15	0.0
6	14	55	0.0	25	19	0.0	31	0.0
7	37	41	0.0	56	11	0.0	63	0.0
8	53	58	0.0	104	18	0.0	127	0.0
9	120	53	0.0	199	22	0.0	255	0.0
10	193	62	0.0	381	25	0.0	511	0.0
11	284	72	0.0	686	33	0.0	1023	0.1
12	516	75	0.0	1,266	38	0.1	2,047	0.2
14	3,009	63	0.4	7,098	13	0.9	8,192	1.1
16	11,279	66	2.1	23,532	28	3.6	32,767	5.6
18	43,044	67	9.7	83,013	37	15.2	131,071	27.4
20	166,783	68	45.0	275,046	48	63.4	524,287	131.1
22	290,709	86	107.0	1,133,347	46	291.0	2,097,151	622.5

[3]A more detailed example will be presented in Section 4.
[4]We assume that the D operation itself takes negligible time to execute compared to the other operations.

Table III State decisions for example Sehwa-1

1. Mutual Exclusion Sets	$M_{2,1} = (10, 110, 111)$, $M_{2,2} = (20, 21)$, $M_{3,1} = (100, 101, 110, 111)$, $M_{4,1} = (10, 11)$, $M_{6,1} = (30, 31)$
2. Sets of PEMs	$P_{1,1} = (0, 1, 2)$, $P_{2,1} = (10, 20)$, $P_{2,2} = (10, 21)$, $P_{2,3} = (110, 20)$, $P_{2,4} = (110, 21)$, $P_{2,5} = (111, 20)$, $P_{2,6} = (111, 21)$,
	$P_{3,1} = (100)$, $P_{3,2} = (101)$, $P_{3,3} = (110)$,$P_{3,4} = (111)$
	$P_{4,1} = (10)$, $P_{4,2} = (11)$, $P_{5,1} = (0, 3)$,$P_{6,1} = (30)$, $P_{6,2} = (31)$
3. States	G_1 $S_{1,1} = (P_{1,1}, P_{4,1})$, $S_{1,2} = (P_{1,1}, P_{4,2})$
	G_2 $S_{2,1} = (P_{2,1}, P_{5,1})$, $S_{2,2} = (P_{2,2}, P_{5,1})$, $S_{2,3} = (P_{2,3}, P_{5,1})$ $S_{2,4} = (P_{2,4}, P_{5,1})$, $S_{2,5} = (P_{2,5}, P_{5,1})$, $S_{2,6} = (P_{2,6}, P_{5,1})$
	G_3 $S_{3,1} = (P_{3,1}, P_{6,1})$, $S_{3,2} = (P_{3,1}, P_{6,2})$, $S_{3,3} = (P_{3,2}, P_{6,1})$, $S_{3,1} = (P_{3,2}, P_{6,2})$, $S_{3,5} = (P_{3,3}, P_{6,1})$, $S_{3,6} = (P_{3,3}, P_{6,2})$, $S_{3,7} = (P_{3,4}, P_{6,1})$, $S_{3,8} = (P_{3,4}, P_{6,2})$

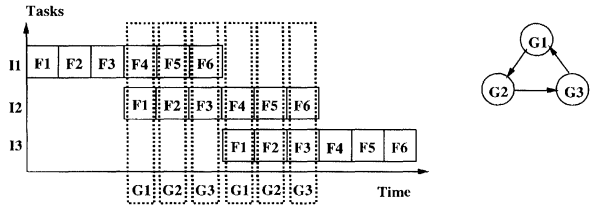


FIGURE 5 Overall timing and state transitions in a pipelined system.

the corresponding input condition in assumed to be a don't care, in which cases we just find a state which has the same node label or is compatible with the current state. The procedure is presented in Figure 6. Going back to the example described in Section 2.2, Figure 4 and Table II; assume that the current state is $S_{2,1}$, then the next state depends

```
Procedure State_Transition
inputs:    Groups of states G1 ··· GL
outputs:   State table which is set of TG1 ··· TGL
/*   TGi:  the transition groups corresponding to Gi
     Si,j:  jth state in Gi */
{
     For i=1 to L {
          While (Gi ≠ ∅) {
               Choose a state Si,j ∈ Gi and set Gi = Gi - {Si,j};
               Check Si,j to identify input condition Ds.;
               If there are such Ds
                    then  For (all possible input conditions)
                               Decide next state Si',j ∈ Gi';
                    else  find the next state Si',j;
                    /* where i' = (i + 1) modulo L */
               Put state transition informations in TGi;
          } /* while */
     } /* for */
} /* Procedure */
```

FIGURE 6 State transition procedure.

on two input conditions, D2 in time step 2 and D5 in time step 5. If the condition at D2 (whose label is 10) is 0, then the states in group G_3 which have a label of 100, (in this case states $S_{3,1}$ and $S_{3,2}$) are possible next states. At the same time, if the input condition at D5 (whose label in 3) is also 0, then we search the candidate states and select the ones which have a node label of 30 in that segment. $S_{3,1}$ satisfies both criteria, therefore it is the next state of $S_{2,1}$ when input conditions to D2 and D5 are both 0. Once the state table is obtained, we need to synthesize an FSM to implement the controller.

5. FSM OPTIMIZATION

As part of our approach to FSM optimization, we use a variation of the horizontal partitioning technique. In Paulin's algorithm, the second objective must be weighed against the first one. By exploiting the specific characteristics of the pipeline control synthesis problem, we developed a new algorithm for horizontal partitioning in which both objectives are satisfied without conflicts and therefore is more efficient in the situation above. While it is originally targeted at PLA optimization, his methodology can also be applied to non-PLA control structures, such as random logic, and would still result in area svaings as will be shown experimentally in Section 4. In addition, we extended the horizontal partitioning from a two-partitioning to multi-way partitioning. This enables

us to explore more optimization possibilities and thus obtain more area-efficient controller implementations.

5.1. The Partitioning Algorithm

The area of a controller logic can be reduced by reducing the number of states (row reduction in PLAs) and also by reducing the number bits/state (column reduction in PLAs). The first reduction can be achieved by placing all the possible binary relations (objective 1 in Paulin's approach) in either one of the partitions and the number of bits/state can be minimized by grouping together PTs which depend on common inputs (objective 2). In general these two are interdependent (and sometimes even competing) and thus can not be optimized simultaneously. However, in out model, the D nodes are scheduled in only one time step and therefore the inputs to each of the groups G_i are always mutually exclusive Thus, grouping overlapped stages in a pipelined data path has the important advantage that it solves the first objective in Paulin's algorithm without worrying about the second one. In other words, the input/output relations do not block any binary simplictions between terms because the inputs to each group are mutually exclusive.

We partition the groups into two subsets SP_1 and SP_2 such that the total area of the resulting partitioned FSM is minimized. If the controller is implemented as a PLA, in order to calculate the area we need to known the number of columns C_{P_1}, and C_{P_2} in SP_1 and SP_2, the number of product terms PT_1 and PT_2, and the number of binary relations in each partition. We can estimate the number of rows R_{p1} and R_{p2} in each partition by subtracting the total number of rows reduced by coding constraints from the total number of PTs in the partition. The total area is Area $= R_{p_1} C_{p_1} + R_{p_2} C_{p_2}$. If the controller is implemented in random logic, we can estimate the area of the layout by using the LAST area estimator [24] which can do so quite efficiently and within 5% accuracy for standard cell implementations.[5] In either case, the problem reduces to finding the partitioning which results in a minimum total Area.

5.1.1. *Exhaustive Search*

Since the partitioning scheme now deal with only L groups instead of a much larger number of states, the problem is greatly reduced in size. For small values of L we can find the optimal partitioning by exhaustive search. The number of distinct ways of partitioning L **distinct** objects $a_1, \ldots, a_L$ into n ($n \leq L$) non-empty partitions is given by

$$R_{L,n} = \frac{1}{n!} \sum_{\substack{(e_1,\ldots,e_n)| \\ e_1+\cdots+e_n=L, \\ e_1\geq 1,\ldots,e_n\geq 1}} \frac{L!}{e_1!\cdots e_n!} \tag{1}$$

Table II shows some numerical values of $R_{L,n}$ for values of L and n between 1 and 6. Since the values of $R_{L,n}$ are less than 100 in all the cases, an exhaustive search to obtain the optimal partition is feasible for small values of L.

The experimental results in Section 4 show that partitioning the groups of states onto more that two partitions can result in more area efficient implementations than the two-way partitioning. The multiway partitioning Algorithm is a simple modification of the two-way partitioning.

5.1.2. *Branch and Bound Method*

As noted in Section 2, the presence of loops in a CDFG schedule could easily make the latency L relatively large. Thus, exhaustive search would become computationally intractable. In these cases, the problem is reformulated as a branch and-bound alogirthm to reduce the search space. In our branch and bound method a decision tree is constructed as follows. The ith level in the tree represents a possible 2 way partition with partition

[5]The estimate does not take into consideration the effect of further logic minimization and is only an approximation of the final result.

size of i and $n - i$, where n is the total number of distinct objects (or groups). The labels on the edges denote the objects selected. For example, the first level shows the possible partitions of sizes 1 and $n - 1$, the second level represents the possible partitions of sizes 2 and $n - 2$, and so on as shown in Figure 7. We observe that there are $n - 1$ levels in the decision tree.[6] In order to implement an efficient branch and bound technique, we need to derive good bounding function which can be easily computed. In our case, the cost function to be minimized is the sum of the areas of the resulting partitions. In the following, we discuss this issue and propose a bounding technique to be used by the search algorithm. Here, we assume that the target implementation is done using PLAs.

Let's represent the unpartitioned PLA by a rectangle of which w $\propto$ (# of input conditions) and $h \propto$ (# of PTs). Since two groups or two sets of groups will have disjoint PTs input conditions, we can represent a partitioning as two rectangles connected by one point $P(a, b)$ as shown in Figure 8(a). Let w and h be the sides of the rectangle which shows the unpartitioned design. Let a and b be the sides of one partition as shown in Figure 8(a). Then the (shaded) area occupied by the partition is computed by $f(a, b) = ab + (w - a)(h - b) = 2ab + wh - ah - bw$. The unshaded portion of the rectangle shows the saving area by the partitioning. While the search is going down to the leave nodes one group is moved from lower partition to upper partition, so the point $P(a, b)$ can only move to a point $(a + \Delta a, b + \Delta b)$ where $\Delta a \geq 0$, $\Delta b > 0$ from the upper left corner of the unpartitioned rectangle, the point S in Figure 8(a). Let's have a closer look on the function $f(a, b)$ how it have changed in the space bounded by $x = 0$, $x = w$, $y = 0$, and $y = h$ line segments. Let's divide the space into four quadrant as shown in Figure 8(b). It is bounded by $x = w/2$ and $y = h/2$ line segments. We can make the following observations:

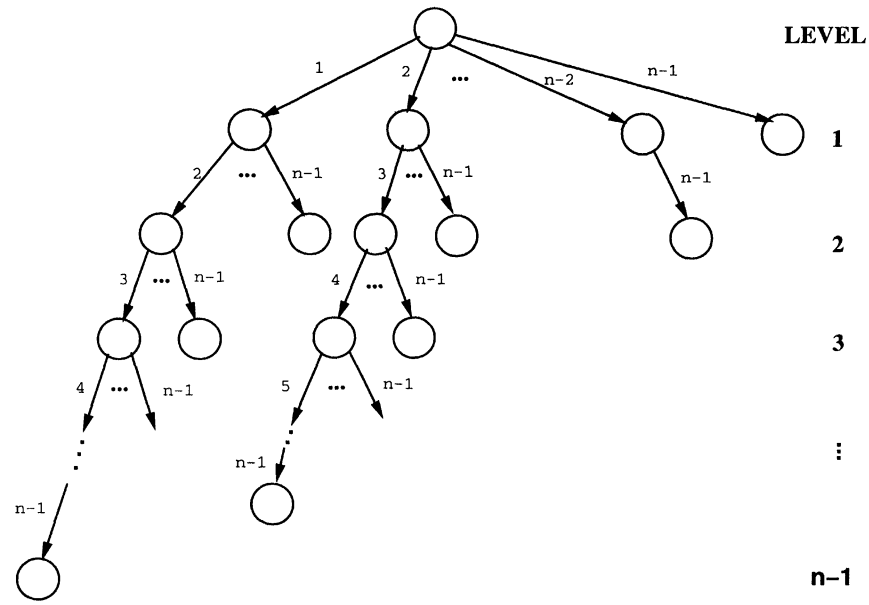


FIGURE 7 A decision tree.

[6]Note that the tree is not balanced.

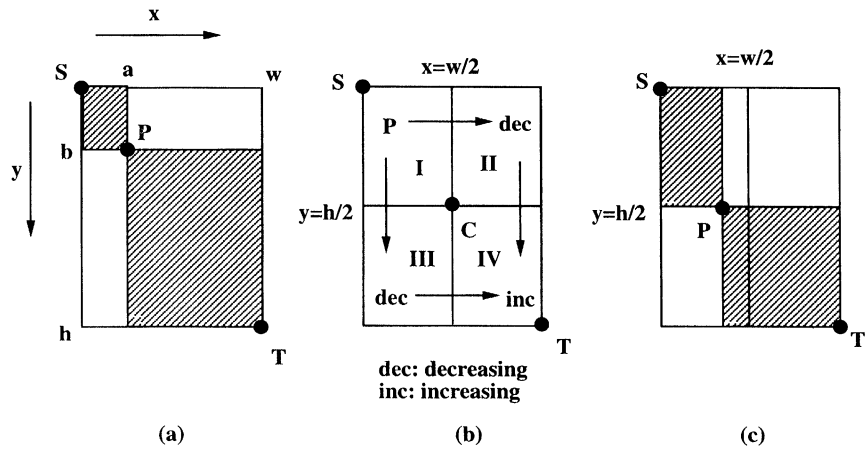


FIGURE 8 Analysis of horizontal partitioning design space.

OBSERVATION 1 *In Quadrant I, $f(a,b)$ is always decreasing with a and /or b. On the other hand $f(a,b)$ is always increasing in Quadrant IV.*

Proof Let's analyze $f(a,b)$ using the partial derivatives to see how $f(a,b)$ changes in each quadrant. $(\partial f(a,b)/(\partial a)) = 2b - h$

if $b < h/2, f(a,b)$ is decreasing function along with the x direction.
if $b > h/2, f(a,b)$ is increasing function along with the x direction.
$(\partial f(a,b)/(\partial b)) = 2a - w$
if $a < w/2 f(a,b)$ is decreasing function along with the y direction.
if $a > w/2 f(a,b)$ is increasing function along with the y direction.

Since in Quadrant I, $a < w/2$ and $b < h/2$, both $\partial f(a,b)/\partial a$ and $\partial f(a,b)/\partial b$ are less than zero, therefore, $f(a,b)$ is decreasing function in Quadrant I. On the other hand, in Quadrant IV, both $\partial f(a,b)/\partial a$ and $\partial f(a,b)/\partial b$ is greater than zero. Therefore, $f(a,b)$ is always increasing in the Quadrant IV while a and/or b is increasing. ■

OBSERVATION 2 *The point $C = f(w/2, h/2)$ in Figure 8(b) is a saddle point of $f(a,b)$ and C is unique.*

Proof We prove this using the partial differentiation. The sufficient condition for a saddle point [23] is

$$\frac{\partial f^2(a,b)}{\partial a^2}\frac{\partial f^2(a,b)}{\partial b^2} - \left[\frac{\partial f^2(a,b)}{\partial a \partial b}\right]^2 < 0.$$

Since the first order partial derivatives are $\partial f(a,b)/\partial a = 2b - h$ and $\partial f(a,b)/\partial b = 2a - w$, there is only one critical point which is $(w/2, h/2)$.

The second order partial derivatives are $\partial^2 f(a,b)/\partial a^2 = 0, \partial^2 f(a,b)/\partial b^2 = 0$, and $\partial^2 f(a,b)/\partial a \partial b = 2$.

$$\frac{\partial f^2(a,b)}{\partial a^2}\frac{\partial f^2(a,b)}{\partial b^2} - \left[\frac{\partial f^2(a,b)}{\partial a \partial b}\right]^2 = -4.$$

Therefore the point C is an saddle point and this point is unique. ■

OBSERVATION 3 *If P lies on either $a = w/2$ or $b = h/2$ line segments as shown in Figure 8(c), then the area $f(a,b)$ occupied by the partitions is $wh/2$.*

Proof

i) if the point P lies on segment $(a = w/2)$ the total area $f(w/2, b) = 2(w/2)b + wh - (w/2)h - wh/2$
ii) if the point P lies on segment $(b = h/2)$

the total area $f(a, h/2) = 2a(h/2) + wh - ah - (h/2)w = wh/2$

Therefore the area $f(a, b) = wh/2$. ■

The above observations show how $f(a, b)$ changes in Quadrants I and IV. In the case of moves in Quadrants II and III, $f(a, b)$s can be decreasing or increasing. We also find that $f(a, b)$ is greater than $wh/2$ in Quadrants I and IV, but it is less than $wh/2$ in Quadrants II and III.

Based on these observations we developed a new bounding technique to be used in the Branch and Bound method for our horizontal partitioning. While the decision tree is searched down to the leaf nodes, point $P(a, b)$ in Figure 8 moves from point S to point T through Quadrants II, III, or through point C. Subsequently, the search will go down on the subbranches until the point P reaches Quadrant IV. Beyond this point, any further move would increase the area $f(a, b)$ and further search through this bounds of the tree is not necessary as can be seen in Figure 9. The sufficient and necessary conditions to detect that point P has already reached Quadrant IV are that: (1) the resulting area of child node is larger than the parent node's, (*i.e.*, $f(a, b)$ is now increasing), and (2) $f(a, b)$ of the child node is greater than $wh/2$. These two conditions are our bounding factors for our branch and bound based algorithm. The less the levels which we need to investigate the better this branch and bound scheme works. In other words, if the minimum point is found at the earlier levels for each branch, then the search space is reduced significantly. To satisfy this objective, the objects are sorted according to the size of their diagonal, (*i.e.*, by increasing distance to Point S).

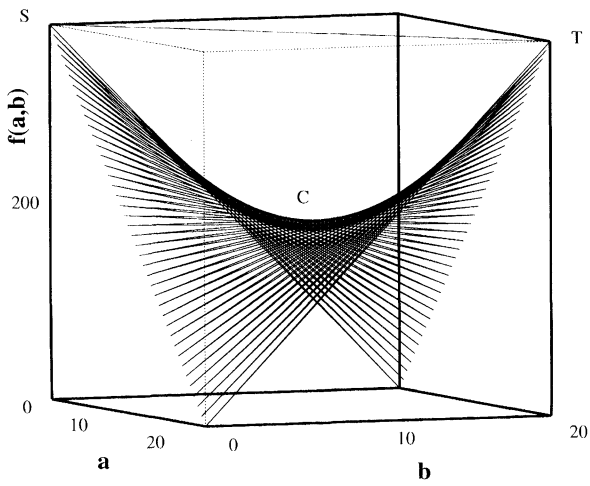


FIGURE 9 3-D plot of function $f(a, b)$ for h = w = 20.

5.1.3. *Heuristic Approach*

While the branch and bound approach can significantly cut the search time, its worst case performance is still exponential. In order to further reduce the partitioning runtime, we developed a heuristic algorithm which is a variation of branch and bound method. As we mentioned in the previous section, $f(a, b)$ is decreasing with both a and b in Quadrant I. It means that $f(a, b)$ is likely to be large in earlier levels since point P is now clear to point S in Quadrant I. So, by avoiding the computation of $f(a, b)$ in the earlier levels, we can reduce the overall amount of computation. While the Branch and Bound scheme prevents searching in Quadrant IV, the heuristic approach tries to reduce the search in both Quadrants I and IV. It controls the starting level of the next sub-branch in Quadrant I and adopts the Branch and Bound method in Quadrant IV. As depicted in Figure 10, the tree is searched in a depth first manner. If the minimum value of the previous sub-branch is found at level l, then in the next sub-branches we start the search from the same level l. Thus, savings can be achieved if the next sub-branch starts from level k where $(1 \leq k < l)$, in which case we save $l - k$ calculations. Figure 10 shows an example of the design space with a latency of 6, where the number inside the circle show the total normalized PLA area of each partitoning. Since the minimum partition is found in level 3 (1-2-5), only the solutions at levels greater than or equal to level 3 are computed, thus only the space between line 1 and 2 is the design space which the heuristic

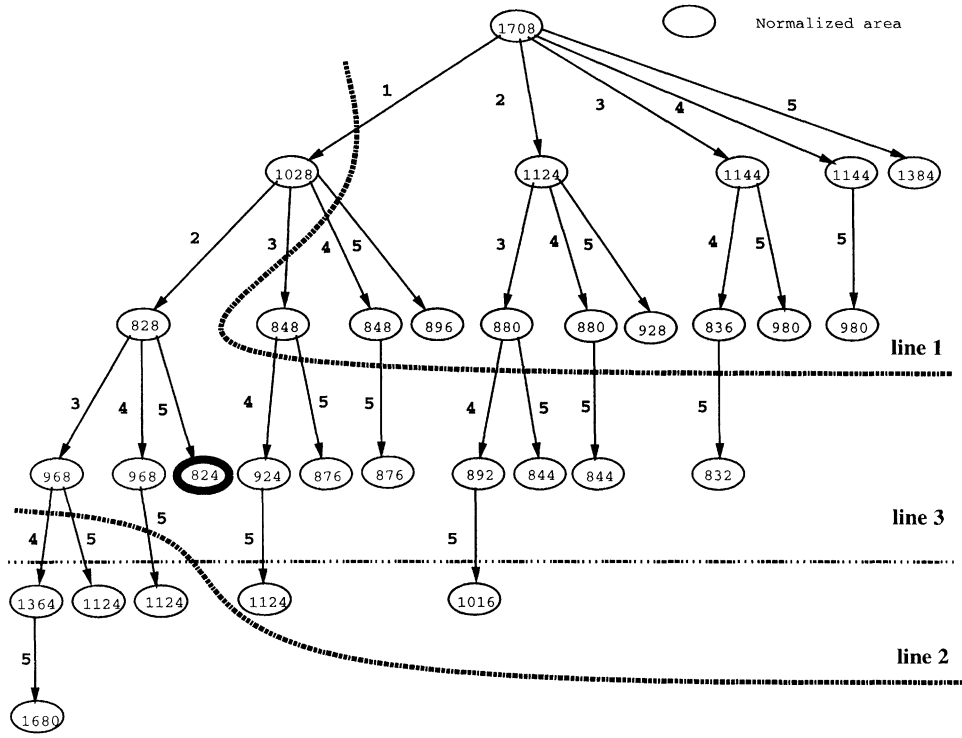


FIGURE 10 The decision tree for $L = 6$.

algorithm explores, and the space above the line 3 is the one for branch and bound method.

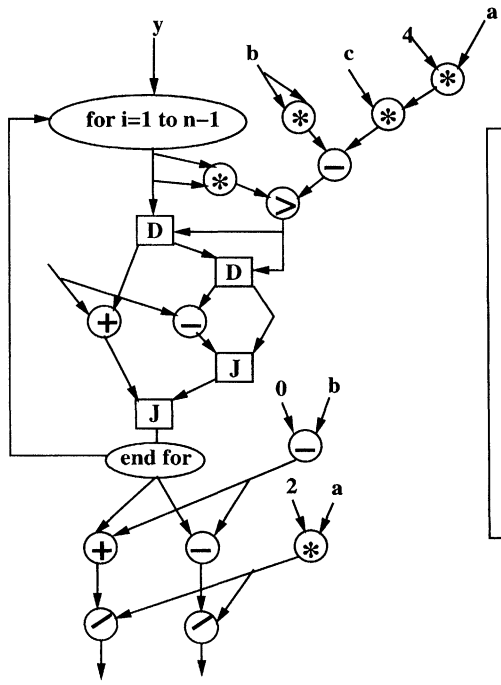
Both approaches were implemented in one program as shown in Figure 11. Currently, it is up to the user to decide which method will be used in partitioning. Table III compares the efficiency of the branch and bound method and the heuristic algorithm with respect to the exhaustive search on the quadratic equation solver example depicted in Figure 12. In this example we assume that the values a, b, and c in the equation ($ax^2 + bx + c = 0$) are 8-bit integers so that the number of iterations compute the square root is 7. We unrolled the loop completely and ran Sehwa to schedule the CDFG with different values of latency(L) up to 11. As explained in Section 2.2 there are L groups of states to be partitioned. In order to show the efficiency of our branch and bound and heuristic approaches we created additional cases (of latency more than 11) by duplicating the group, *i.e.*, we use each group twice. For example, the case with latency of 22 is made up by taking the scheduling with a latency of 11 and using each group twice. The heuristic always finds the optimal partition except when the latencies are 11 and 22. In these cases, the heuristic finds suboptimal solutions which are only 1.2% and 1.8% greater in area than the optimal solution, respectively.

5.2. State Encoding

Once the horizontal partitioning of the state table is done, we need to perform state encoding. Given a set of coding constraints, the objective of this procedure is to assign state codes so that the size of the sequencing logic is reduced. We generate coding constraint groups consisting of states having the same next state and matching primary

```
Procedure Horizontal_Partitioning(key)
key:        User select either branch & bound algorithm or heuristics.
inputs:     State table {TG_1 ··· TG_L}
outputs:    the partitions SP_1 and SP_2 which are the sets of
            TG_i's such that SP_1 ∩ SP_2 = ∅.
/*    PT_i:  the number of product terms in TG_i;
      CC_i,j:  jth coding constraint in TG_i;
      IC_i = No.  of input conditions to TG_i;
      Area_m:  the minimum area of partitioned sequencing FSMs; */
{
      For (i=1;i≤ L;i++) {
            compute PT_i and IC_i in TG_i;
            calculate the rows R_i = PT_i - Σ_{j=1}^{n_i} (|CC_i,j| - 1);
            total_R = total_R + R_i;  total_IC = total_IC + IC_i;
      } /* for */
      total_area = total_R × total_IC;
      For (i=1;i≤ L;i++) {
            compute f(IC_i, R_i) = IC_i R_i + (total_IC - IC_i)(total_R - R_i);
      Sort R_i's and IC_i's in descending order according to f(IC_i, R_i);
      if (key)    then BBPartition(1);
                  else HUPartition(1);
} /* Procedure */
BBpartition(nodenum)    /* Branch and Bound Method */
nodenum:    the starting node of the level, and the number of elements in one partition.
{
      For (i=nodenum;i ≤ n;i++) {
            Compute_area;
            if ((Area_p > Area_c) OR (Area_c < total_area/2)
            then  put Area_p into STACK;      Area_p = Area_c;
                  BBpartition(nodenum+1);
      } /* for */
} /* BBpartition */
HUPartition(nodenum)    /* Heuristic Algorithm */
{
      For (i=nodenum;i ≤ n;i++) {
            if (level_p ≤ nodenum)
            then  Compute_area;
                  if ((Area_p > Area_c) OR (Area_c < total_area/2)
                  then  put Area_p into STACK;      Area_p = Area_c;
                        BBpartition(nodenum+1);
      } /* for */
} /* HUpartition */
Compute_Area
{
      if (PLA controller)
      then
            R_p1 = Σ_{TG_m ∈ SP_1} R_m;      C_p1 = Σ_{TG_m ∈ SP_1} IC_m;
            R_p2 = Σ_{TG_m ∈ SP_2} R_m;      C_p2 = Σ_{TG_m ∈ SP_2} IC_m;
            Area_c= R_p1 C_p1 + R_p2 C_p2;
      else /*(random logic)*/
            Area(1) = Estimate_area (SP_1);   Area(2) = Estimate_area (SP_2);
            Area_c= Area(1) + Area(2);
      if (Area_c < Area_m) then Area_m = Area_c;
} /* Compute_Area */
```

FIGURE 11 Horizontal partitioning algorithms.




```
y=2^(n-1) ;
x=b^2 - 4ac ;
for i := 1 to n - 1 do
  if ( x - y^2 ) > 0 then y := y + 2^(n-i-1)
  else if ( x - y^2 ) < 0
  then y:= y - 2^(n-i-1)
end for;
roots: = (-b ± y ) / 2a ;
```

FIGURE 12 Quadratic equation example.

inputs. States in the same coding constraint group can be collapsed into one common PT, thus reducing the number of states. In addition to saving by horizontal partitioning, we can also reduce the number of bits/state in the two-way partitioning case by assigning even codes to all the next states of one partition (in a PLA, this will set the last column in the OR-plane to all zeros, and in random logic, this will reduce the gate count and the wiring). To decide on a candidate for this reduction, we compute the number of next states in each partition and check if it is less than $\lfloor \log_2(\text{total number of states}) \rfloor / 2$. If this applies to both partitions, choose the partition that can result in a larger reduction in area. This is always possible since the number of next states either partitions is less than equal to a half of the total number of states and also the number of available codes are always at least equal to the total number of states. Furthermore, PTs not in the current partition area included but their next states are set to don't cares. This allows further minimization by logic optimization tools such as Espresso [19] (for PLAs) or MIS [20] (for random logic) since it reduces the number of literals. The state encoding algorithms is shown in Figure 13. The encoding algorithm can be extended to the multi-way partitioning in a straightforward manner by dividing the partitions onto two blocks and assigning state codes to each as if it were a two-way partitioning.

6. RESULTS

In this section, we present some experimental results which were obtained by applying our approach to two design examples. The first example is from [1], the second is a reduced instruction set version of the M6502 microprocessor. In both cases, we show evidence that our

```
Procedure State_Encoding
input:      Coding Constraint Sets CCS_i in partition i and n_s
output:     encoded states
/*    n_s:  the total number of states;
code:  Set of code to be assigned to states;
CC_{i,j}:  Coding constraints in CCS_i;
S_{i,j}:  States in CC_{i,j};
BC: Starting number to encode CC_i; */
{
      code={ 0,1,···,2^⌈log2 n_s⌉ };
      While ((CCS_1 and CCS_2) ≠ ∅)
      {     If (CCS_1 ≠ ∅)
                  then  {pick max(|CC_i|) ∈ CCS_1;
                        CCS_1 = CCS_1 − CC_i;
                        even_code=true; }
                  else  { pick max(|CC_i|) ∈ CCS_2;
                        CCS_2 = CCS_2 − CC_i;
                        If N < (2^⌊log2⌈n_s⌉ − 1)
                              then  even_code=false
                              else  even_code=true
                        }
            BC=2^⌈log2|CC_i|⌉
            IF (even_code=true)
                  then  N=min {n | n = k(BC) and n ∈ code, k = 1,2,···};
                  else  N=min {n | n = k(BC)+1 and n ∈ code, k = 1,2,···};
            For j=1 to |CC_i|
            {     if N ∉ code
                        if (even_code={true})
                              then N=min { n | n = 2k and n ∈ code, k = 1,2,···};
                              else N=min{n | n ∈ code}
                  code=code-{N};
                  S_{i,j}=N;
                  CC_i=CC_i − {S_{i,j}}
                  N=N+2;
            } /* for j*/
      } /* while */

} /* procedure */
```

FIGURE 13 State encoding algorithm.

approach achieves better area savings compared to traditional synthesis methods.

6.1. The Sehwa Example

The first example CDFG [1] is shown in Figure 4. We used Sehwa to schedule this CDFG with different latencies. In Example Sehwa-1, the CDFG is scheduled with latency $L = 3$. Preprocessing adds two NOP nodes to the CDFG, one, (110), is added in time step 2 and other, (10), is in time step 4, so that the procedure deals with only operation nodes. There is a total of 16 states in this example. Starting from state S_{11} we construct the state table. Using the FSM synthesis algorithms of Section 3, the controllers are built with both PLAs and standard cells. For the PLA controller, partitioning can save us the second and fifth input

columns in partition 1, and the first, third, and fourth input columns in partition 2. The last column of the OR-plane in partition 1 is also reduced by the state encoding. We then minimized each partition with Espresso. The array area of each PLA can be estimated by $A_{pla} = (2 \times |n_i| + |n_o|) \times n_{pt}$, where n_i and n_o are the number of bits of inputs and outputs, respectively, and n_{pt} gives the number of product terms. Table IV(a) shows the PLA areas obtained by NOVA [17], by the modified Horizontal Partitioning [26], and by our algorithm in PLA area units (normalized). We added an estimate of the routing and buffering area for the last two approaches which produce multiple PLAs. We ran the i_exact encoding strategy for NOVA. The modified Horizontal Partitioning we use different sizes of cluster which are 8, 10, 12, 16, 20 and we choose the best result. In this particular example, our algorithm achieves PLA area savings of more than 24% as compared to the other two algorithms.

In Example Sehwa-2, the same DFG in Figure 4 is scheduled with latency $L = 2$. Here we use the io_hybrid encoding strategy for NOVA since the i_exact encoding was computationally infeasible (we ran for more than 70 hours on a Convex super computer). The savings are much greater in this case.

In the case of random logic controllers, we minimized the logic by using the MIS multi-level logic optimizer. Each partition was optimized separately using NAND, NOR, and INVERTER gates. The three partitions were then merged to one block and laid out using the GDT standard cell place and route tools [27]. Table IV(b) shows that our approach achieves savings of 9.5% and 72.9% in layout area over NOVA for Example Sehwa-1 and Sehwa-2, respectively. In each case, we chose the standard cell row configuration which resulted in minimum layout area. Again, we note that for Example 2, our comparison with NOVA is based on a sub-optimal io_hybrid run because an optimal run of NOVA was computationally infeasible.

TABLE IV(a) Experimental results, PLA controller

Program	No. of PLAs	PLA controller area (array units) latency 3		latency 2	
		area	savings	area	savings
NOVA	1	528	0%	1,175*	0%
M_Horizon	3	474	10.2%	619	47.3%
Proposed	3	397	24.8%	370	68.5%

* io_hybrid encoding.

TABLE IV(b) Experimental results, standard cell controller

Program	Standard cell controller area (μm^2) latency 3		latency 2	
	area	savaings	area	savings
NOVA	133,929	0%	267,246*	0%
Proposed	121,258	9..5%	72.446	72.9%

* io_hybrid encoding.

6.2. Reduced M6502 Example

The Sehwa example, while interesting in its structure, is synthetic (*i.e.*, it does not perform any useful computations). In order to benchmark our controller synthesis approach against realistic cases, we selected the MOSTEK 6502 microprocessor as another example. The specification in ISPS was obtained from the High Level Synthesis benchmark set [30]. In order to obtain a manageable size example (which can be handled by Sehwa), we reduced the instruction set to four instructions. This resulted in the CDFG shown in Figure 15. Also, the original specifications were based on a non-pipelined scheduling which was reflected in the assumed data path. In order to enable us to perform pipelined scheduling with high throughput, we made some modifications on the data path and the CDFG, as follows:

- we assume a dual ported memory in which two memory read operations are permitted to overlap. Only one memory write operation is permitted at a time, though.

TABLE V Data path and state table information of the M6502 example with $L = 4$ and $L = 6$

Latency	operators	Registers	Muxs	# inputs	# states	# PTs
4	4	16	14	13	664	1,368
6	4	14	12	13	196	305

- we increased the number of various resources, such as registers, in order to enable the overlapped execution of some register operations in the CDFG.

We used Sehwa to schedule the CDFG with latencies $L = 4$ and 6. For each scheduling, we generated a pipelined RT-level implemnetation of the data path. Table V shows some statistics on the data paths and state tables for both latency values. Figure 16 shows the data path for $L = 6$.

For both data path, we used our algorithm to synthesize several implementations of the control part using both PLAs and standard cells. [7] In each case, we generated layouts corresponding to various n -way partitionings of the groups of states for $n = 1, 2, 4, 6$. Table VI shows area and delay data of the various implementations at latency $L = 6$. The total area figures for the PLA implementations includes the sum of the areas of the individual partitions plus estimates of the buffering and routing areas. The delays of all the implementations are computed as the worst case delay. The PLAs were laid out using octtools [29] in SCMOS 3 μ technology whereas the Standard

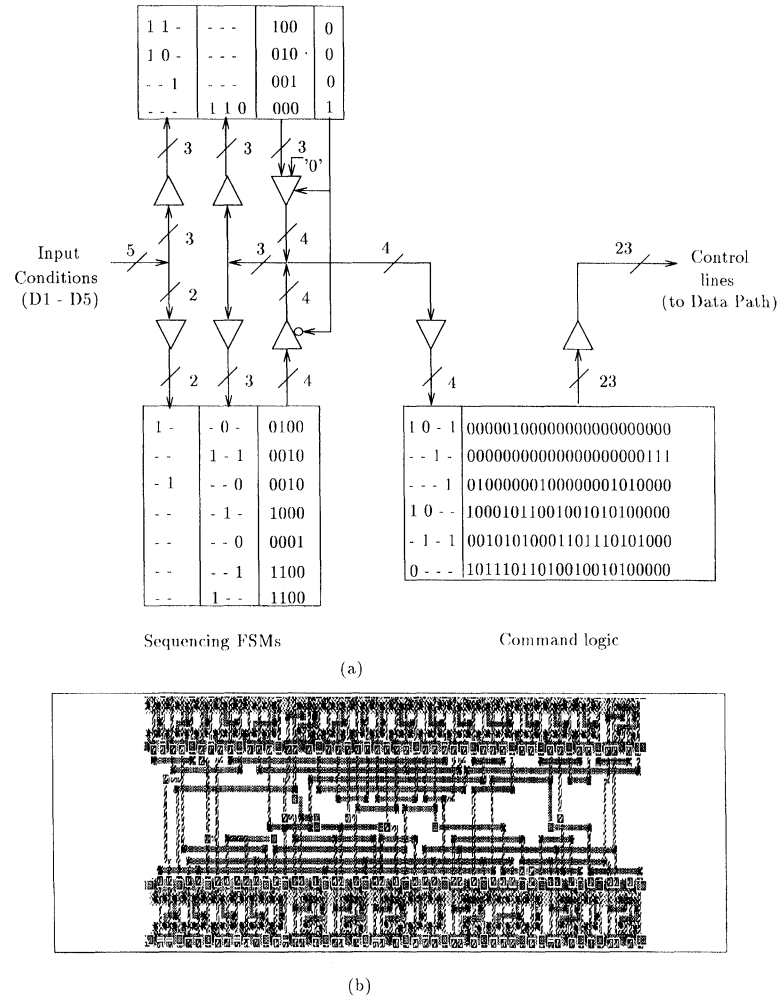


FIGURE 14 Time-stationary controller for Exaple Sehwa-1 (a) PLA, (b) Standard cell.

[7] Here, we could not compare against NOVA because the state table was too large for NOVA to handle, even in io_hybrid mode.

TABLE VI Experimental results for the M6502 example with $L = 6$.

No. of partitions	PLA		Standard Cells	
	Total area (mm^2)	Total delay (ns)	Total area (mm^2)	Total delay (ns)
1	12.53	94.6	3.87	46.9
2	9.82	68.5	3.70	36.7
4	9.39	26.1	3.11	33.9
6	10.11	26.1	3.61	34.0

TABLE VII Experimental results for the M6502 example with $L = 4$

No. of partitions	PLA		Standard Cells	
	Total area (mm^2)	Total delay (ns)	Total area (mm^2)	Total delay (ns)
1	24.08	159.8	9.76	57.3
2	19.42	101.4	6.91	43.3
4	19.52	100.6	6.09	43.0

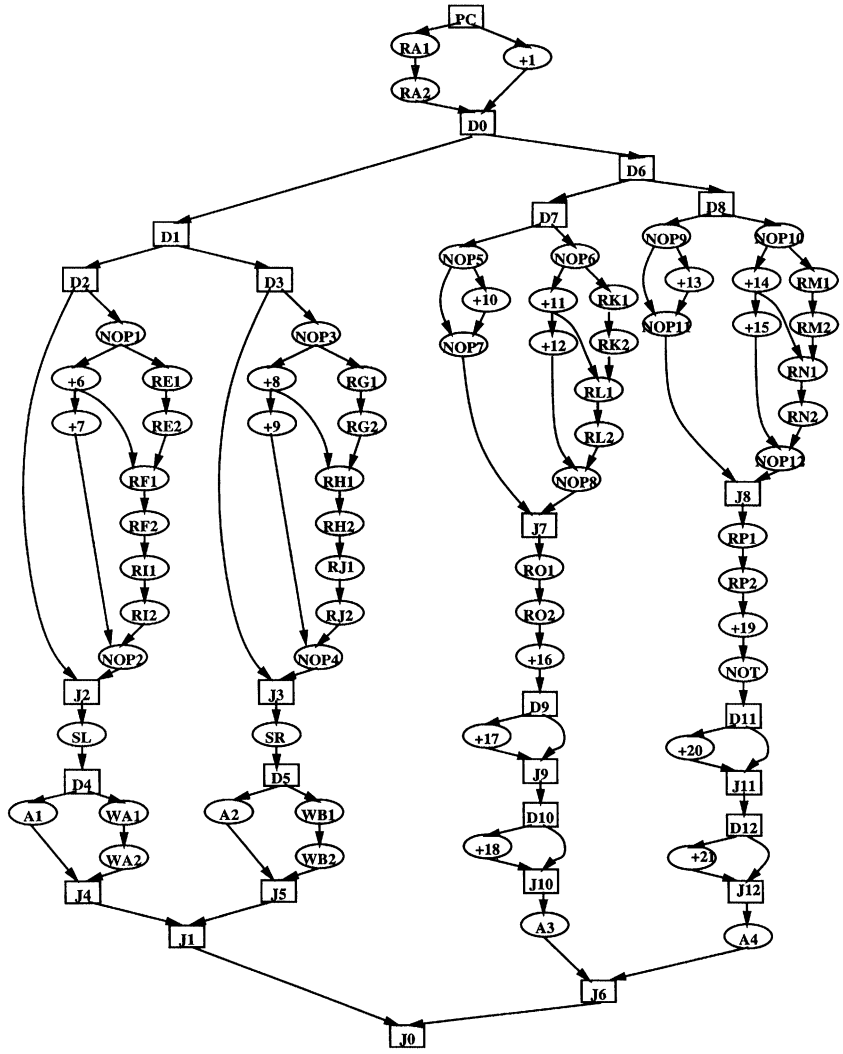


FIGURE 15 CDFG of the modified M6502 example.

Cells were laid out using the GDT CMOS 3 μ technology. The PLA delay figures were obtained using Crystal [28]. The Standard Cell delay figures were estimated by MIS. In both styles, the best area and performance were achieved using a four-way partitioning of the controller. The slightly

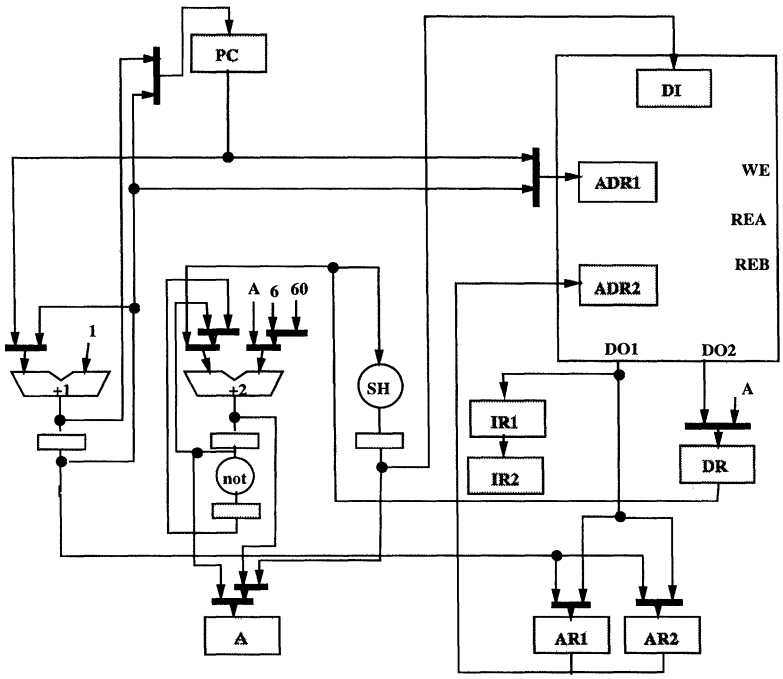


FIGURE 16 Data path of the modified M6502 example with $L = 6$.

differing technologies make it hard to compare the areas and delays of the two implementation styles. We note that the PLA delay figures include estimates of wiring delay from layout back-annotation. The delay figures produced by MIS for Standard Cells are pre-layout estimates and do not include wiring delay. Table VII shows similar figures for a scheduling of latency $L = 4$.

7. CONCLUSION

There are two types of basic control schemes for pipelined data paths: data-stationary control and time-stationary control schemes. We presented an approach to automatically synthesize a time-stationary control scheme for a pipelined data path. We developed an efficient model and methods to produce a control specification for a pipelined data path with conditional branches by detecting mutual exclusion between operations. A highly optimized FSM controller implementation is obtained FSM controller implementation is obtained by performing horizontal partitioning so as to minimize the total controller area. The examples presented indicate that our approach results in controller implementations which are more area efficient than the ones obtained by directly using traditional nonpartitioned logic design methods. We are currently researching the automatic design of data stationary controllers. we will analyze and compare the cost *vs.* performance tradeoff of these two control schemes and set up a basic guidelines for the choice of controllers given a data path design. We are also developing a unified methodology for combined data path and control synthesis of pipelined system.

References

[1] Park N. and Parker, A. (1988). Sehwa: a software package for synthesis of pipelines from behavioral specifications. *IEEE Transactions on CAD*, **7** (3), 356–370.

[2] Girczyc, E.,Loop winding-a data flow approach to functional pipelining. In: *Proceedings of ISCAS*, IEEE, May 1987.
[3] Crastes de Paulet, M. *et al.* (1989). Controller synthesis in the asyl system. In: *Logic and Architecture Synthesis for Silicon Compiler*. Elsevier Science Publisher B.V.
[4] Newton, R., Ma, H., Devadas, S. and Sangiovanni-Vincentelli, S. A.Mustang: State assignment of finite state machines for optimal multi-level logic implementations. In: *Proceedings of ICCAD 87*. IEEE, November 1987.
[5] Park, N. and Kurdahi, F.,Module assignment and interconnect sharing in register-transfer synthesis of pipelined data paths. In: *Proceedings of ICCAD -89*, IEEE Computer Society, November 1989.
[6] Kogge, P. M. (1981). *The Architecture of Pipelined Computers*. McGraw-Hill, New York, N.Y.
[7] Paulin, P. (1989). Horizontal partitioning of PLA-based finite state machine. In: *Proc. 26th Design Automation Conf.*, pp. 333–338, ACM/IEEE.
[8] Hayati, S. and Parker, A.,Automatic production of controller specifications from control and timing behavioral descriptions. In: *Proceedings of the 26th Design Automation Conference*, pp. 75–80, IEEE/ACM, June 1989.
[9] Tseng, C. *et al.*,Bridge: a versatile behavioral synthesis system. In *Design Automation Conference Proceeings no. 25*, ACM SIGDA, IEEE Computer Society – DATC, June 1988.
[10] Nagle, A., Cloutier, R. and Parker, A. (1982). Synthesis of hardware for the control of digital systems. *IEEE Transactions on Computer -Aided Design*, CAD, **1** (4), 201–212.
[11] Kramer, H. *et al.* (1989). Data path and control synthesis in the caddy system. In: *Logic and Architecture Synthesis for Silicon Compilers*, Elsevier Science Publishers B.V.
[12] Camposano, R. (1987). Structural synthesis in the yorktown silicon compiler. In: Sequin C. H., Editor, *VLSI 87*, Elsevier Science Publishers B.V.
[13] Goossens, G., Vandewalle, J. and De Man, H., Loop optimization in register-transfer scheduling for DSP-systems. In: *Proceedings of the 26th DAC*, pp. 826–831, IEEE/ACM, June 1989.
[14] Potasman, R., Lis, J., Nicolau, A. and Gajski, D. (1990). Percolation based synthesis. In: *Proc. 27th Design Automation Conf.*, pp. 444–449, ACM/IEEE.
[15] Camposano, R. and Bergamashi, R.,Synthesis using pathbased scheduling: algorithms and exercises. In: *DAC 90*, IEEE/ACM, June 1990.
[16] De Micheli, G., Brayton, R., Sangiovanni-Vincentelli, A. (1985). Optimal state assignment for finite state machines. *IEEE Transactions on Computer -Aided Design*, **4** (3), 269–285.
[17] Villa, T. and Sangiovanni-Vincentelli, A.,NOVA: state assignment of finite state machines for optimal two-level logic implementations. In: *Proceedings of the 26th Design Automation Conference*, pp. 327–332, IEEE Computer Society, June 1989.
[18] Amann, R. and Baitinger, U. (1989). Optimal state chains and state codes in finite state machines. *IEEE Transactions on Computer -Aided* Design, **8** (2), 153–170.
[19] Brayton, R. K. *et al.* (1985). *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer.
[20] Brayton, R. K. *et al.*,Multiple level logic optimization system. In: *Proceedings of ICCAD 86*, November 1986.
[21] De Micheli, G. and Santomauro, M. (1983). Smile: a computer program for partitioning of programmed logic arrays. *Computer -Aided Design*, pp. 89–97.
[22] Riordan, J. (1978). *An introduction to combinatorial analysis*. Wiley.
[23] Edwards, C. and Penney, D. (1982). *Calculus and Analytic Geometry*. Prentice Hall, Inc.
[24] Kurdahi, F. and Ramachandran, C., LAST: a Layout Area and shape function estimator for high level applications. In: *EDAC 91*, IEEE Computer Society, February 1991.
[25] Kim, J., Kurdahi, F. and Park, N.,Automatic synthesis of time-stationary controllers for pipelined data paths. In: *Proceedings of ICCAD 91*, November 1991.
[26] Chang, T.,*Applications of vertical -horizontal partitioning algorithm for PLA -based finite state machine*. Master's thesis, Dept. of Electrical and Computer Engineering, University of California, Irvine, June 1990.
[27] *L* compilers users guide.Silicon Compiler Systems, 1989.
[28] Ousterhout, J. (1985). *Using crystal for timing analysis*. Tutorial, EECS Dept. US Berkeley.
[29] Spickelmier, R. (1990). *Release notes for oct tools*. Electronics Research Laboratory, UC Berkeley.
[30] *High level synthesis benchmarks*. Microelectronic Center of North Carolina, 1991.

Author's Biographies

Jong Tae Kim received the B.E. degree in electronics engineering from SungKyunKwan University, Seoul, Korea, in 1982, and the M.S. and Ph.D. degrees in electrical and computer engineering at the University of California, Irvine, in 1987 and 1992, respectively. From 1991 to 1993 he was with the Aerospace Corporation, Elsegudo, California. He is currently an Assistant Professor in the school of electrical and computer engineering at SungKyunKwan University, since 1995. He was a Full-time Lecturer with Chunbuk National University in Korea from 1993 to 1995. His research interests include VLSI CAD, ASIC design, and Computer Architectures.

Fadi J. Kurdahi received the Bachelor of Engineering degree in Electrical Engineering from the American University of Beirut, Lebanon in 1981. He received the M.S. degree in Electrical Engineering and the Ph.D. degree in Computer Engineering from the University of Southern California, Los Angeles, CA, in 1982 and 1987, respectively.

Since 1987, he has been with the Department of Electrical and Computer Engineering at the

University of California, Irvine, where he is currently an Associate Professor. He received an NSF Research Initiation Award in 1989, and two ACM/SIGDA fellowships in 1991 and 1992. His areas of interest are high-level synthesis of digital circuits, VLSI systems design and layout, and design automation. He has served on program committees of ISSS, ED$TC, ISSS, ISCAS. Dr. Kurdahi was Associate Editor of IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing 1993 – 1995. He is a member of IEEE and ACM.

Nohbyung Park received the B.S. degree in Electronics Engineering from Seoul National University, Seuol, Korea, in 1976, the M.S. degree in Electrical Engineering from the Korea Advanced Institute of Science and Technology, Seoul, Korea, in 1978, and the Ph.D. degree in Electrical Engineering from the University of Southern California, Los Angeles, CA, in 1985. From 1976 to 1981 he was a design engineer and project manager in the Division of Industrial Equipment at Samsung Electronics Company, Seoul, Korea, where he designed computer interfaces, high-speed peripherals, and small business computer systems. From 1986 to 1990 he was an Assistant Professor in the Department of Electrical and Computer Engineering at the University of California, Irvine. He was the director of research and development at Samsung Electronics Company in Seoul, Korea. Currently he is with AST Research in Irvine, CA. His research interests are high-speed VLSI and digital system design, pipeline synthesis, and design automation.