파이프라인 방식의 ASIC 데이타 경로를 위한 시간 정지형 콘트롤러의 자동 합성

김 종 태[†]

요 약

본 논문은 파이프라인 방식의 ASIC 데이타 경로를 제어하기 위한 시간 정지형 콘트롤러의 자동합성에 관한 연구이다. 콘트롤 합성은 콘트롤 사양의 자동 생성과 유한상태기의 합성 및 최적화 단계로 구성된다. 이 유한상태기는 수평분할 방식을 응용하여 최적화되며 최소 면적의 콘트롤러가 합성된다. 실험을 통해 제안된 방식으로 생성된 콘트롤러와 기존 방식으로 합성된 유한상태기 콘트롤러를 비교하였는데 콘트롤러의 면적에서 있어서 큰 감소를 보여준다.

Automated Synthesis of Time-Stationary Controllers for Pipelined Data Path of Application Specific Integrated Circuits

Jong Tae Kim[†]

ABSTRACT

We developed an approach to automatically synthesize time-stationary controllers for a given pipelined data path of Application Specific Integrated Circuits (ASICs). This work consists of automated production of control specifications and Finite State Machine (FSM) Optimization. A FSM controller is implemented by performing horizontal partitioning so as to minimize the total controller area. We compared our approach to published work on FSM generation and optimization, and the results indicate large savings in total controller area.

1. 서 론

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

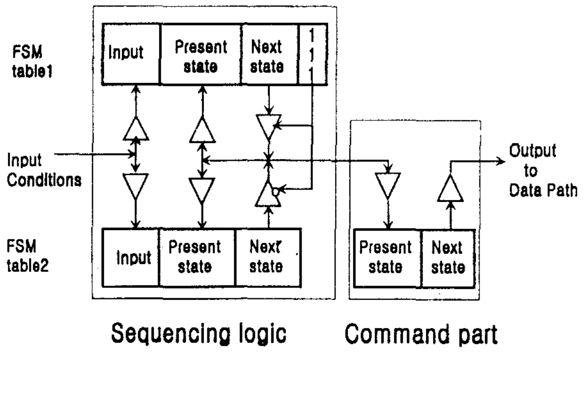
There are two types of basic control schemes for pipelined data paths[3]: data-stationary and time-stationary. A data-stationary control scheme passes the control function code along with data. This scheme

† 정 회 원: 성균관대학교 전기전자 및 컴퓨터공학부
논문접수: 1996년 12월 3일, 심사완료: 1997년 7월 3일

allows simple and straight-forward design of both the state sequencer and the data path control circuits for each stage, and thus, is usually expensive in layout area. On the other hand, a time-stationary control mechanism provides the control signals for the entire pipeline from a single source external to the pipeline. 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. In this paper we will focus on the time-stationary control scheme implementation of pipelined data paths.

The controller is modeled as a Moore style Finite State Machine (FSM) and the combinational circuits can be implemented using PLAs or random logic. The structure of a time-stationary controller is shown in (Fig. 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[4] that such a partitioning minimizes the total controller area.

In this paper, we present a method to synthesize a Moore-style FSM controller specifications given a



(Fig. 1) The Structure of a Time-Stationary Controller

pipelined data path with conditional branches. The input is a scheduled data flow graph (DFG) 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 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] 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, we define the term *latency* which will be extensively used throughout this paper. The number of time units between two consecutive initiations in a pipeline is called the ***latency***, L of the pipeline.

Section 2 presents the control specification of pipelined data paths. The partitioning and state assignment algorithms are described in Section 3. Section 4 shows some experimental results. Conclusions are drawn in Section 5.

2 Control Specification

The work in [5] dealt with the automatic production of control specifications for non-pipelined data paths. Another system, Bridge[6], performs data path and control path allocation for non-pipelined systems by applying either a local slicing or a global slicing technique. Other works for non-pipelined systems are reported in [7][8].

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

2.1 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 1: 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 2: 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 will denote by $M_{i,1}$, $M_{i,2}, \cdots$, the MES's which cover the nodes scheduled in i.

Based on Definition 2, 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 3: Let $M_{i,1}, M_{i,2}, \cdots, M_{i,n}$ 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. $P_i = \{o_1, \cdots, o_n / o_h \in M_{i,h},\ h = 1, \cdots, n\}$. We will denote by $P_{i,1}$, $P_{i,2}, \cdots$, 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 $i \le L$. Since the schedule is pipelined, time steps i, $i+L$, $i+2L, \cdots$, are overlapping and therefore, a *state* can now be defined as follows:

Definition 4: Given $1 \le i \le L$, a *state* S_i is defined as a set of PEM's corresponding to overlapping time steps i, $i+L$, $i+2L, \cdots$. $S_i = \{P_{k,l} / \forall P_{k,l}, P_{m,n} \in S_i,\ k \bmod L = m \bmod L = i\}$, and S_i is not included in any larger state S_i'.

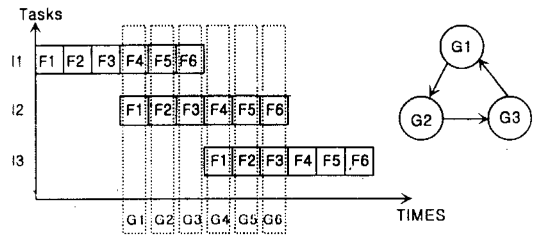
We will denote by $S_{i,1}$, $S_{i,2}, \cdots, S_{i,n_i}$ all the states that can be generated by different combinations of PEM's in i and the time steps that overlap with it, and n_i is the number of such different combinations. Since $1 \le i \le L$, we can define *groups of states* G_1, G_2, $\cdots,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

2.2 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: $G_1 \rightarrow G_2 \cdots G_i \rightarrow G_{i+1} \cdots G_L \rightarrow G_1$. This is mainly due to the pipelined nature of the scheduling and is shown in (Fig. 2). 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. 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 2.1, we 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.

Once the state table is obtained, we need to synthesize an FSM to implement the controller.



(Fig. 2) Overall timing and state transitions in a pipelined system

3 FSM Optimization

An FSM consists of two major components: a combinational circuit and a memory. The memory stores a representation of the state machine at any given time and the combinational circuit produces the primary outputs as a function of the machine state and/or the machine primary inputs. The first stage of this FSM optimization is performing partitioning and state encoding. Next stage is logic minimization. In the following subsection we briefly survey the FSM optimization schemes for partitioning and state encoding.

3.1 PLA-based FSM optimization schemes

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 FSM based on symbolic minimization of the FSM combinational component and on a related constrained encoding problem[9]. Amann presented state assignment algorithms that permit the synthesis of optimal counter-based PLA FSM's[10]. The work in [11] presents sate assignment algorithms based on the solution of face hypercube embeding and ordered face hypercube embeding. Hyper-Place using the graph embedding problem is presented in [12]. It breaks the hypercube embedding problem into two steps: (1) mapping of the adjacency graph to a grid; (2) mapping the solution on the grid to one on a minimum dimensionality hypercube with dilaton at most two. It can handle large FSM's efficiently.

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[13]. 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 [4] 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. The objectives for the horizontal partitioning are to: 1) Find a partition that does respect all boolean relations, and 2) find a partition that holds product terms (PTs) which depend on common outputs and/or common inputs.

3.2 The Partitioning Algorithm

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, this methodology can also be applied to non-PLA control structures, such as random logic, and would still result in area savings 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. 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 our model, the D nodes are scheduled in only one time step and therefore the inputs to each of the groups Gi 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 simplifications between terms because the inputs to each group are mutually exclusive. The new horizontal partitioning algorithm is presented in (Fig. 3).

We partition the groups into two subsets SP_1 and SP_2 such that the total area of the resulting partitioned FSM is minimized. As mentioned in Sec. 2.2, state transitions occur between adjacent groups of states in the following sequence: $G_1 \rightarrow G_2 \cdots G_i \rightarrow G_{i+1} \cdots \rightarrow G_L \rightarrow G_1$. And we can group the state transitions from the state is G_i to states in G_{i+1} and call them as state transition groups, TG_i.

If the controller is implemented as a PLA, in order to calculate the area we need to know the number of columns Cp_1 and Cp_2 in 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 Rp1 and Rp2 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 $= R_{p1} C_{p1} + R_{p2} C_{p2}$. If the controller is implemented in random logic, we can estimate the area of the layout by using the LAST area estimator[14] which can do so quite efficiently and within 5% accuracy for standard cell implementations. In either case, the problem reduces to finding the partitioning which results in a minimum total *Area*.

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. In the following we use basic combinatorial concepts to derive the size of the search space. In general, the size of the search space for a two-way partitioning is given by the following theorem:

Theorem 1: The number of distinct ways of partitioning L distinct objects $a_1, \ldots, a_L$ into two non-empty partitions is given by $R_{L,2} = 2^{L-1} - 1$.

Proof: Given the set of objects $A = \{a_1, \ldots, a_L\}$, we define a partition $P = \{p_1, p_2\}$ on A, where p_1 and p_2 are disjoint subsets of A. Let p and q be the sizes of p_1 and p_2, respectively, and $p + q = L$, and $1 \le p \le L-1$ and $1 \le q \le L-1$. The number of distinct ways of parititioning the objects onto p_1 and p_2 can be derived by first selecting p objects from A for inclusion into p_1 (the remaining objects will go into p_2) and summing over all possible values of p. This gives us two cases:

i) L is even.

$$\binom{L}{1}+\binom{L}{2}+\cdots+\binom{L}{\frac{4}{2}}$$

$$=\sum_{i=1}^{\frac{L}{2}-1}\binom{L}{i}+\frac{1}{2}\binom{L}{\frac{4}{2}}=\frac{1}{2}\sum_{i=1}^{L-1}\binom{L}{i}$$

ii) L is odd.

$$\binom{L}{1}+\binom{L}{2}+\cdots+\binom{L}{\frac{L-1}{2}}=\sum_{i=1}^{\frac{L-1}{2}}\binom{L}{i}+\frac{1}{2}\sum_{i=1}^{L-1}\binom{L}{i}$$

Therefore, the number of cases is $\sum_{i=1}^{L-1}\binom{L}{i}=2^{L-1}-1$.

The two way partitioning can be generalized into multi-way partitioning. In this case, the groups of states are partitioned onto n partitions by exhaustively searching all the possible configurations. The size of the search space is given by the following theorem in which can be easily derived from basic principles of combinatorics[15].

Theorem 2: 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 $R_{L,n}=\frac{1}{n!}\sum_{e}\frac{L!}{e_1!\ldots e_n!}$, where e is

$$(e_1,\ldots,e_n) \mid e_1+\ldots+e_n=L,\ e_1 \geq 1,\ldots,e_n \geq 1.$$

Proof: Given the set of objects $A=\{a_1,\ldots,a_L\}$, we define a partition $P=\{p_1,\ldots,p_n\}$ on A, where the p_i's are pairwise disjoint subsets of A. Let $e_1,\ldots,e_n$ denote the sizes of $p_1,\ldots,p_n$, repsectively. Clearly, $e_1+\ldots+e_n=L$. The number of ways of distributing the objects onto the P subsets of sizes $e_1,\ldots,e_n$ can be expressed as:

$$\binom{L}{e_1}\binom{L-e_1}{e_2}\cdots\binom{L-e_1-\cdots-e_{n-1}}{e_n}=\frac{L!}{e_1!\ldots e_n!}$$

Which can be derived by first selecting e_1 objects from A for inclusion into p_1, then e_2 object from the remaining $L-e_1$ for p_1 for p_2, and so on until all the partitions are filled. Thus, the total number of ways of partitioning A onto the ordered partitions in P can be obtained by summing (1) over all possible integral, positive solutions $(e_1,\ldots,e_n)$, of the equation $e_1+\ldots+e_n=L$ Since the partitions $p_1,\ldots,p_n$ are permutable, the final expression for $R_{L,n}$ can be obtained by dividing by the total number of permutations in P, or $n!$.

Since the values of $R_{L,n}$ are less than 100 when $L \leq 6$, $n \leq 6$, 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 that the two-way partitioning. The multiway partitioning Algorithm is a simple modificatwo-way partitioning.

3.2 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 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 states 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 $\lceil \log_2$ (*total number of states*)$\rceil / 2$. If this applies to both partitions, choose the partition that can result in a larger reduction in area. This is always possible since

```
Procedure State_Encoding

inputs: Coding Constraint Sets $CCA$ and $CCB$ in partition A & B and
total # of states, $n_s$;
outputs: encoded states;
/* code: Set of codes to be assigned to states;
 $CA_i$, $CB_i$: Coding constraints in $CCA$, $CCB$, respectively;
 $S_j$: States in $CA_i$, $CB_i$;
 $BC$: Starting number to encode $CA_i$, $CB_i$;  */
{
        code={ 0,1,···, $2^{\lceil \log 2^{n_s} \rceil}$ };
        While (( $CCA \neq 0$) and ( $CCB \neq 0$))
        {
            If ( $CCA \neq 0$) then {
                pick $CMAX = \max(\,|CA_i|\,) \in CCA$;
                $CCA = CCA - CMAX$;
                even_code = true;
            }
            else { pick $CMAX = \max(\,|CB_i|\,) \in CCB$;
                $CCB = CCB - CMAX$;
                If $N < (2^{\log_2 \lceil n_s \rceil})$
                        then even_code = false;
                        else even_code = true;
            }
            $BC = 2^{\log_2 \lceil CMAX \rceil}$;
            IF (even_code = true)
                then $N = \min\{n \mid n = k(BC)$ and $n \in code$, $k = 1, 2, \cdots\}$;
                else $N = \min\{n \mid n = k(BC) + 1$ and $n \in code$, $k = 1, 2, \cdots\}$;
            For j=1 to $|CMAX|$ {
                if $N \notin code$
                    if (even_code = true)
                        then $N = \min\{n \mid n = 2k$ and $n \in code$, $k = 1, 2, \cdots\}$;
                        else $N = \min\{n \mid n \in code\}$;
                code= code- $N$;
                $S_j = N$;
                $CMAX = CMAX - \{S_j\}$
                $N = N + 2$;
            } /* for j*/
        } /* while */
} /* procedure */
```

(Fig. 3) Horizontal partitioning algorithm

the number of next states either partitions 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 are included but their next states are set to don't cares. This allows further minimization by logic optimization tools such as Espresso[16] (for PLAs) or MIS([17] (for random logic) since it reduces the number of literals. The state encoding algorithm is shown in (Fig. 4). The encoding algorithm can be extended to the multi-way partitioning onto two blocks and assigning state codes to each as if it were a two-way partitioning.

4. Experimental Results

In this sectin, we present some experimental results which were obtained by applying our approach to two desgn 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 approach achieves better area savings compared to traditional synthesis methods.

4.1 The Sehwa example

The first example CDFG is shown in (Fig. 5). We used Sehwa to schedule this CDFG with different

```
Procedure Horizontal_Partitioning
inputs: State table { $TG_1 \cdots TG_L$ }
outputs: the partitions $SP_1$ and $SP_2$ which are the sets of $TG_i$'s such that
 $SP_1 \cap SP_2 = 0$.
/*      $PT_i$: the number of product terms in $TG_i$;
        $n_i$ : the number of coding constraints in $TG_i$
        $CC_{i,j}$: jth coding constraint in $TG_i$;
        $IC_i$ : No. of input conditions to $TG_i$;\\
        $RArea_i$ : the area to be reduced by a partition; */ .
{
      For (i=1;i≤ L;i++) {
            compute $PT_i$ and $IC_i$ in $TG_i$;
            calculate the rows $R_i = PT_i - \sum_{j=1}^{n_i}(|CC_{i,j}| - 1)$;
      } /* for */
      /* Find optimal partition by exhaustive search */
      For ( $i=1$; $i \le 2^{L-1}-1$; $i++$ ) {
            Partition $TG_1 \cdots TG_L$ to sets $SP_1$ and $SP_2$;
            $R_{p1} = \sum_{TG_m \in SP_1} R_m$;
            $C_{p1} = \sum_{TG_m \in SP_1} IC_m$;
            $R_{p2} = \sum_{TG_m \in SP_2} R_m$;
            $C_{p2} = \sum_{TG_m \in SP_2} R_m$;
            $RArea_i = R_{p1}C_{p1} + R_{p2}C_{p2}$;
      } /* for */
      Choose the partition, $SP_1$ and $SP_2$, which $\max_i(RArea_i)$;
} /* procedure */
```

(Fig. 4) State encoding algorithm

latencies. In Example Sehwa-1, the CDFG is scheduled with latency L = 3, There is a total of 16 states in this example. The array area of each can be estimated by $AREA_{PLA} = (2 \times |n_i| + |n_o|) \times n_{pt}$ gives the number of PTs. 〈Table 〉 shows the PLA areas obtained by NOVA[11], by the modified horizontal partitioning [18], and by our algorithm in PLA area units (normalized), We added an estimate of the routing and buffering areas for the two approaches which produce multiple PLAs. We ran the exact encoding strategy NOVA. The modified Horizontal Partitioning starts with the state table encoded by NOVA. To get result by 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 compred to the other two algorithms. In Example Sehwa-2, the same DFG in (Fig. 5) is now 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 supercomputer). The savings are much greater in this cases.

〈Table 1〉 Expreimental results of PLA controller for the Sehwa example

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 2〉 Expreimental results of standard cell controller for the Sehwa example

Program	Standard Cell Controller Area (μm^2)			
	latency 3		latency 2	
	area	savings	area	savings
NOVA	133,929	0%	267,246*	0%
Proposed	121,258	9.5%	72,446	72.9%
* io_hybrid encoding				

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. ⟨Table 2⟩ shows that our approach achieves savings of 9.5% and 72.9% in layout area over NOVA for Examples 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.

4.2 Reduced M6502 example

We selected the MOSTEK 6502 microprocessor as another example. The specification in ISPS was obtained rom the High Level Synthesis benchmark set[19]. Also, we reduced the instruction set to four instructions to obtain a manageable size example for Sehwa that we used as a scheduler. After we generated a pipelined RT-level implementation, we used our algorithm to synthesize several implementations of the control part using both PLAs and standard cells. Here, we could not compare against NOVA to handle, even in io_hybrid mode. In each case, we generated layouts corresponding to various n-way partitionings of the groups of states for $n = 1$, 2, 4, 6. ⟨Table 3⟩ shows area and delay data of the various implemetations. In both styles, the best area and performance were achieved using a four-way partitioning of the controller. The slightly differing technologies make it hard to compare the areas and delays of the two implemetation style. ⟨Table 4⟩ shows similar figures for a scheduling of latency $L = 4$.

⟨Table 3⟩ Experimental results for the M6502 example with L = 6 (Moore style)

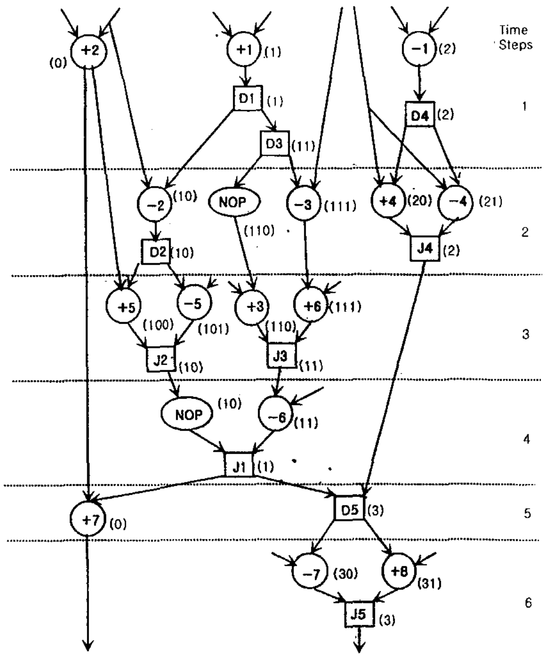
No. of Partitions	PLA		Standard Cells	
	Area (㎟)	Dealy (ns)	Area (㎟)	Delay (ns)
1	12.53	94.6	3.87	46.9
2	9.82	68.5	3.70	36.7
4	9.39	26.1	3.11	33.9
6	10.11	26.1	3.61	34.0

⟨Table 4⟩ Experimetnal results for the M6502 example with L = 4 (Moore style)

No. of Partitions	PLA		Standard Cells	
	Area (㎟)	Dealy (ns)	Area (㎟)	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

5. 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 method to produce a control specifi-



⟨Fig. 5⟩ Example sehwa-1 : scheduled CDFG with labelling L = 3 cation for a pipelined data path with conditional branches by detecting mutual exclusion between operations. A highly optimized 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 logic synthesis 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 guideline for the choice of controllers given a data path design.

참 고 문 헌

[1] N. Park and A. Parker, "Sehwa: a Software Package for Synthesis of Pipelines from Behavioral Specifications," IEEE Trans. on CAD, Vol.7, No.3, pp356-370, March 1988.

[2] N. Park and F. Kurdahi, "Module Assingment and Interconnect Sharing in Register-Transfer Synthesis of Pipelined Data Paths," In Proceedings of ICCAD89, IEEE Computer Society, November 1989.

[3] P.M. Kogge, 'The Architecture of Pipelined Computers', McGraw-Hill, N.Y., 1989.

[4] P. Paulin, "Horizontal Partitioning of PLA-based Finite State Machine," in Proc. of 26th Design Automation Conference, pp.333-338, June 1989.

[5] S. Hayati and A. Parker, "Automatic Production of Controller Specifications from Control and Timing Behavioral Descriptions," in Proc. of 26th Design Automation Conference, pp.75-80, June 1989.

[6] C. Tseng et al., "Bridge: a Versatile Behavioral Synthesis System," in Proc. of 25th Design Automation Conference, June 1988.

[7] A. Nagle, R. Cloutier, and A. Parker, "Synthesis of Hardware for the Control of Digital Systems," IEEE Trans. on CAD, Vol.1, No.4, pp201-212, 1982.

[8] H. Kramer et al., "Data Path and Control Synthesis in the Caddy System," In Logic and Architecture Synthesis for Silicon Compilers, Elsevier Science Publishers B.V., 1989.

[9] G. De Micheli, R. Brayton, and A. Sangiovanni-Vincentelli, "OPtimal State Assignment for Finite State Machine," IEEE Trans. on CAD, Vol.4, No.3, pp269-285, July 1985.

[10] R. Amann and U. Baitinger, "Optimal State Chains and State Codes in Finite State Machines," IEEE Trans. on CAD, Vol.8, No.2, pp153-170, Feb. 1989.

[11] T. Villa and A. Sangiovanni-Vincentelli, "NOVA :State Assignment of Finite State Machine for Optimal Two-Level Logic Implementations," IEEE Trans. on CAD, Vol.9, No.9, pp905-924, Septemeber 1990.

[12] S. Liu, M. Pedram, and A. Despain, "A Fast State Assignment Procedure for Large FSMs," In Proc. of 32th Design Automation Conference, June 1995.

[13] G. De Micheli and A. Sangiovnni-Vincentelli, "SMILE: a Computer Program for Partitioning of Programmed Logic Arrays," Computer-Aided Design, pp89-97, March 1983.

[14] F. Kurdahi and C. Ramachandran, "LAST: a Layout Area and shape functione STimator for high level applications," In FDAC 91, IEEE Computer Society, February 1991.

[15] C. Edwards and D. Penney, 'Calculus and Analytic Geometry,' Prentic Hall, Inc., 1982.

[16] R. K. Brayton, et al., 'Logic Minimization Algorithms for VLSI Synthesis,' Kluwer, 1985.

[17] R. K. Brayton, et al., "Multiple Level Logic Optimization System," in Proc. of ICCAD 86, November 1986.

[18] T. Chang. Application of Vertical-Horizontal Partitioning Algorithm for PLA-based Finite State Machine, Master's thesis, Dept. of ECE,

UC Irvine, June 1990.

[19] High Level Synthesis Benchmarks, Microelectronic Center of North Carolina, 1991.

김 종 태

1982년 성균관대학교 전자공학과 졸업(공학사)

1987년 미국 캘리포니아대학(Irvine) 전기 및 컴퓨터 공학과 졸업(공학석사)

1992년 미국 캘리포니아대학(Irvine) 전기 및 컴퓨터 공학과 졸업(공학박사)

1991년~1993년 미국 The Aerospace Corporation 연구원

1993년~1995년 전북대학교 컴퓨터공학과 교수

1995년~현재 성균관대학교 전기전자 및 컴퓨터공학부 교수

관심분야: VLSI CAD, ASIC 설계, 컴퓨터구조