파이프라인 방식의 ASIC 데이타 경로를 위한 무어 및 밀리식 시간 정지형 콘트롤러의 자동 합성

김 종 태†

요 약

본 논문은 파이프라인 방식의 ASIC 데이타 경로를 제어하기 위한 무어 및 밀리식의 시간 정지형 콘트롤러에 관한 연구이다. 조건분기(conditional branches)를 가진 데이타흐름도로 부터 무어 및 밀리식의 유한상태기(finite state machine) 콘트롤러를 합성하는 방법을 소개한다. 콘트롤 합성은 콘트롤 명세서의 작성과 유한상태기의 합성으로 구성된다. 콘트롤 명세서를 작성하기 위한 과정들을 통해 상태표(state table)의 형태로 표현되는 유한상태기의 내역이 작성된다. 이 유한상태기를 여러가지 다른 방식의 분할 과정과 축소화 과정을 거쳐 최소 면적을 가진 콘트롤러가 합성된다. 실험을 통해 두가지 콘트롤 방식의 특성을 비교하며 또한 두 모델의 비용과 성능의 영향 관계를 보여준다.

Automated Synthesis of Moore and Mealy-model Time-stationary Controllers for Pipelined Data Path of Application Specific Integrated Circuits

Jong Tae Kim†

ABSTRACT

In this paper we discuss Moore and Mealy-model Time-stationary control schemes of pipelined data paths of Application Specific Integrated Circuits (ASICs). We developed a method to synthesize both a Moore and a Mealy-style Finite State Machine (FSM) controller specifications given a pipelined data path with conditional branches. The control synthesis task consists of the generation of control specification and the FSM synthesis. The control specification procedure generates a FSM specification in the form of a state table. The different partitioning schemes are applied to each FSM controller so as to minimize the total area. Experimental results show the characteristics of the two different control styles and the effects of these two models on cost and performance.

1. Introduction

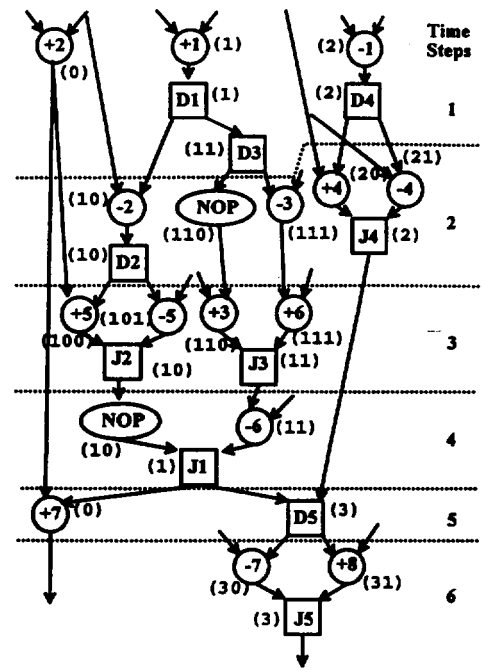
Nowadays, there is a high demand for high-speed computing with limited resources and for fast design of systems while guaranteeing their correctness. Pipelining has been a good methodology for designing fast digital, application specific integrated circuits (ASICs), but pipelined architectures become quite complex to design as circuit sizes increase. Automated design tools in high level for pipeline ASICs are necessary to cope with such complexity and explore the design space efficiently. High-level synthesis is a relatively recent branch of VLSI design automation research and the task of high-level synthesis of pipelines may be divided into data path synthesis and control synthesis. The main tasks of data path synthesis include scheduling and resource allocation[1]. Scheduling is the process of partitioning the input specification so that each partition corresponds to a single time step, or a control step in execution and assigns operations in the input graph to time steps while observing the data precedence and satisfying constraints. At this point, no concrete register transfer level (RTL) structure is implemented. It is only during RTL synthesis that operators are

† 정 회 원 : 성균관대학교 전기공학과 교수
논문접수 : 1994년 9월 8일, 심사완료 : 1994년 1월 3일

mapped onto specific hardware modules, values are mapped to specific registers and multiplexors are assigned at inputs of shared resources. RTL synthesis outputs a complete RTL design implementation. Control synthesis can be done in parallel with data path synthesis or afterwards. In this paper we are focusing on the synthesis of controllers for pipelined data paths, especially on control mechanisms which provide control signals for the entire pipeline from a single source external to the pipeline, as usually known as time-stationary control schemes[2].

The controller can be modeled as both a Moore-style or Mealy-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. One type of FSM whose output is a function of present state is called a Moore machine [3]. If the output of FSM is a function of both the input and the current state, then it is called a Mealy machine[4]. We present a method to synthesize both a Moore-style and a Mealy-style FSM controller specifications given a pipelined data path with conditional branches. The input is a scheduled data flow graph (DFG) which shows operator-to-time step assignments and interconnects 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. 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. We assume that loops are handled by unrolling them and the CDFG schedule is pipelined with a latency (or initiation interval) of L.

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 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. (Fig. 1) shows the scheduled CDFG with node label.



(Fig. 1) Scheduled CDFG with Node Labelling : L=3 (Moore Model)

2. Related Work

Early works[5, 6] in pipeline design focused on either scheduling and controlling existing pipelines or physically construction of stages with fixed functions. Pipeline stages are physically separated (structural pipelining). This may force non-optimal use or sharing of resources. The data path synthesis program we use, called Sehwa[1], presents some theoretical foundations of pipelined synthesis. In Sehwa, the logical-stage concept (functional pipelining) was used. A logical stage corresponds to the set of operators, registers, and multiplexors which are activated during the same clock cycle. Two logical stages may share the same resource, which makes more complex and efficient sharing of resources possible.

Most of the previous work done in the control synthesis at the register transfer level was for non-pipeline systems. Automatic synthesis of microprogrammable control hardware was addressed in [7] using the two optimization algorithms. The optimization techniques can be used to reduce the number of branches, to shorten conditional branching time, and to reduce the number of micro cycles which leads to increase the performance of the micro engine. 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 sytem developed at AT&T Bell Laboratory and performs data path and control path allocation for non-pipeline systems. There have been numerous reported works on FSM optimization. These include algorithms for state encoding[10], counter embedding and logic partitioning[11, 12, 13], and logic minimization[14]. Vertical partitioning[12] separates the set of output functions into two or more PLA's while minimizing the number of redundant product terms in all the PLAs. Horizontal partitioning [13] allows the reduction of the number of input and/or output columns in the PLAs resulting from the partition. As a part of our approach to FSM optimization, we use a variation of the horizontal partitioning technique.

3. Control Synthesis

The control synthesis system consists of two parts : the automated generation of control specification and FSM optimization. The control specification procedure consists of three major steps: preprocessing, state decision, and state transition. In preprocessing, to reduce the number of states and simplify the control synthesis tasks we modify the input CDFG by rearranging the D and J nodes, inserting NOP nodes. In addition, we need to insert latches for keeping input conditions if the life time of conditional branches are longer than L for Mealy model. In Moore model the conditional branches occur before the next time step, in other words the conditional branches decide the next states. In the case of Mealy model the conditional branches decide the output functions. Edges with no operation nodes increase the number of states in the FSM and our solution to this problem is to keep the D and J nodes as close as possible. NOP nodes are also 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). Through the state decision process the states are bound with operation. After identifying the states, the state transitions are determined. The state outputs are determined by binding the control signals available

from the RTL data path synthesis for each state. Once the state table is obtained, we need to synthesize an FSM to implement the controller. FSMs can be implemented with either PLAs or standard cells. The first stage of the FSM optimization is performing both partitioning and state encoding or state encoding alone, depending on the user's choice. Next stage is logic minimization with Espresso[14] for two level or MIS[15] for multi-level logic.

3.1 Moore Style Controller Synthesis

3.1.1 Control Specification

First the *mutual exclusion set* (MES) is identified for given time step i. A set M of nodes is said to be a MES if all the nodes in M are pairwise-mutually exclusive and M is not included in any larger MES M'. MESs are the maximal groups of mutually exclusive operations within a given time step. 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. $Pi=\{o_1,\cdots,o_n \mid o_h \in M_{i,h},\ h=1,\cdots, n\}$. We will denote by $P_{i,1}$, $P_{i,2}$,$\cdots$, the different PEM's in time step i. Next, we find sets of operations $P_{i,j}$, PEM, which can be executed concurrently in each time step by picking one operation from each MES and combining them. Thus, each P_{ij} represents a subset of nodes that can be executed in parallel during time step i. 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: Given $1\le i\le$L, a state Si is defined as a set of PEM's corresponding to overlapping time steps i, $i+L$, $i+2L$,$\cdots$. $Si=\{P_{kl} \mid \forall P_{kl}, P_{m,n}\in S_i,\ k \bmod L = m \bmod L = i\}$, and Si is not included in any larger state Si'. We will denote by $S_{i,1}$, $S_{i,2}$,$\cdots S_{i,ni}$ 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 $G_i=\{S_{i,j},\ 1\le j\le n_i\}$. As an example, we took the CDFG shown in (Fig. 1). We used Sehwa to schedule this CDFG with latency L=3. Preprocessing adds two NOP nodes to the CDFG. One, (110), is added to time step 2 and other, (10), to time step 4, so that the procedure deals with only operation nodes. <Table 1> shows the state identification procedure. The mutual exclusion sets are $M_{2,1}=(10,110,111)$, $M_{2,2}=(20,21)$ in time step 2, $M_{3,1}=(100,101,110,111)$ in time step 3, $M_{4,1}=(10,11)$ in time 4, and $M_{5,1}=(30,31)$ in time step 6. There are 6 different PEMs in time step 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 $P_{2,6}=(111,21)$. In time steps 3, 4, 5, and 6 there are 4, 2, 1, and 2 different PEM sets, respectively. Since L=3, time steps 2 and 5 are overlapped. From these PEMs we can bind operation nodes and states. We can find 6 states in G2 which are $S_{2,1}=(P_{2,1},P_{5,1})$, $S_{2,2}=(P_{2,2},P_{5,1})$, $S_{2,3}=(P_{2,3},P_{5,1})$, $S_{2,4}=(P_{2,4},P_{5,1})$, $S_{2,5}=(P_{2,5},P_{5,1})$, and $S_{2,6}=(P_{2,6},P_{5,1})$. There is a total of 16 states in this example.

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

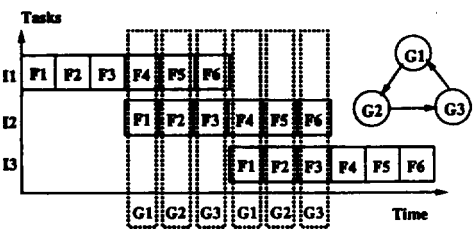
〈Table 1〉 State decisions for the example

1. Mutual Exclusion Sets		$M_{2,1}=(10,110,111)$, $M_{2,2}=(20,21)$, $M_{3,1}=(100,101,110,111)$, $M_{4,1}=(10,11)$, $M_{5,1}=(30,31)$
2. Sets of Possible Execution Modes		$P_{1,1}=(0,1,2)$, $P_{2,1}=(10,21)$, $P_{2,2}=(10,21)$, $P_{2,3}=(110,20)$, $P_{2,4}=(110,21)$, $P_{2,5}=(111,20)$, $P_{2,6}=(111,21)$, $P_{3,1}=(100)$, $P_{3,2}=(101)$, $P_{3,3}=(110)$, $P_{3,4}=(111)$, $P_{4,1}=(10)$, $P_{4,2}=(11)$, $P_{5,1}=(0,3)$, $P_{6,1}=(30)$, $P_{6,2}=(31)$
3. States	G_1	$S_{11}=(P_{11},P_{41})$, $S_{12}=(P_{11},P_{42})$
	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	$S_{3,1}=(P_{3,1},P_{6,1})$, $S_{3,2}=(P_{3,1},P_{6,2})$, $S_{3,3}=(P_{3,2},P_{6,1})$, $S_{3,4}=(P_{3,2},P_{6,2})$, $S_{3,5}=(P_{3,3},P_{6,1})$, $S_{3,6}=(P_{3,3},P_{6,2})$, $S_{3,7}=(P_{3,4},P_{6,1})$, $S_{3,8}=(P_{3,4},P_{6,2})$

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, the next state is the one which has all the compatible (i.e. not mutually exclusive) nodes of the present state. We find compatible nodes by searching only the PEMs corresponding to the next time step within states in the next group. The state transition procedure is presented in (Fig. 3).

3.1.2 FSM Optimization

The controller is vertically partitioned into a sequencing part and a command part. The sequencing logic is partitioned horizontally



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

```
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
   Sij: jth state in Gi //

begin
      For i=1 to L
      begin
         While (Gi≠0)
         begin
            Choose a state Sij ∈ Gi and set Gi=Gi-{Sij};
            Check segment k in Sij to identify input condition Dx;
            If there are such Dx
               then For (all possible input conditions)
                        Decide next state Sr,j ∈ Gr;
               else find the next state Sr,j;
               // where r=(i+1)modulo L //
            Put state transition information in TGi;
         end while
      end for
end procedure
```

(Fig. 3) State transition procedure

into two parts[13]. In our Moore model, the inputs to each of the groups G_i are mutually exclusive since the D nodes are scheduled in only one time step. Thus, grouping overlapped stages in a pipelined data path has the great advantage that input/output relations do not block any binary simplifications 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 know the number of columns CP_1 and CP_2 in SP_1 and SP_2, the number of product terms (PTs) n_{pt1} and n_{pt2}, and the number of binary relations in each partition. We can estimate the number of rows RP_1 and RP_2 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 $Area = RP_1CP_1 + RP_2CP_2$. The problem reduces to finding the partitioning which results in a minimum total $Area$. Since we deal with only L groups instead of a much larger number of states, we can find the optimal partitioning by exhaustive search (which is otherwise not feasible).

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 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). The detailed descriptions of FSM optimization can be found in [16].

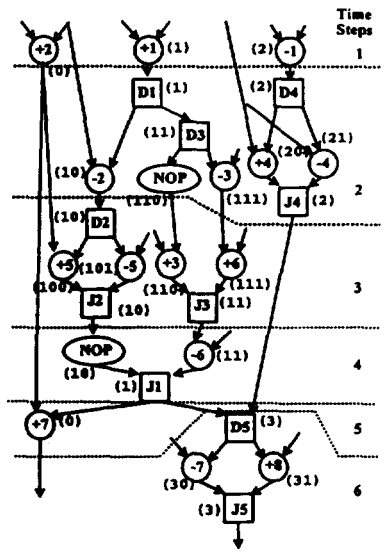
3.2 Mealy Style Controller Synthesis

3.2.1 Control Specification

In our Mealy model, a state is defined as a set of nodes in overlapping time steps i, $i+L$, $i+2L$,···. The number of states doesn't depend on the conditional branches by the definition of state and the number of states is the same as latency L which is the number of groups in Moore model. We can see that the number of states in a Mealy-style controller is much less than in a corresponding Moore-style controller. Thus the state decision is less complicated than Moore model one and quite straight forward.

The state transitions are the same as group transition concepts. The state transitions occur in circular fashion, i.e., $S_1 \rightarrow S_2 \cdots S_i \rightarrow S_{i+1} \cdots S_L \rightarrow S_1$ where $1 \leq i \leq L$. If there are no conditional branches, the number of states in both controllers is the same. The number of PTs in a Mealy-style FSM is the same as the number of states in a Moore-style FSM. The number of PTs in Mealy model depends on the binding of the input conditions and the current states. The number of states in each group of Moore-style FSM is the same as the number of PTs in the corresponding states in Mealy-style FSM. The output generation is quite complex compared to state sequencing synthesis, since the output is dependent on both current state and input conditions. The output binding at each state starts by identifying the dependency of nodes on conditional branches in each time step and deciding on the outputs for all the possible combinations of input conditions on nodes in that state. In this case the input conditions are not mutually exclusive between states, rather, they depend on the life times of conditional branches. If the length of a conditional branch is longer than the latency L we need latches to keep the input conditions every L time steps.

As an example of Mealy-style controller we take the same CDFG scheduled with latency $L=3$. (Fig. 4) shows the CDFG of Mealy model. Preprocessing adds two NOP nodes as in the Moore-style controller and moves the conditional nodes after time step boundary. For example, D1, D3, D4 nodes are scheduled in time step 2 for the Mealy-stype preprocessing, but these conditional nodes are scheduled in time step 1 for Moore-style preprocessing. There are 3 states, S_1, S_2, and S_3, which are same as the latency. S_1 governs nodes (0,1,2) in time step 1 and nodes (10,11) in time step 4. S_2 controls nodes (10,110,111,20,21) in time step 2 and a node (0) in time step 5. And nodes (100, 101,110,111) in time step 3 and nodes (30, 31) in time step 6 are the set of nodes in S_3.



(Fig. 4) Scheduled CDFG with Node Labelling : L=3 (Mealy Model)

Condition nodes (D1), (D1,D3,D4), and (D2, D5) are the input conditions for S_1, S_2, and S_3, respectively. The output binding starts with state S_1 and there are 16 PTs in this example as shown in 〈Table 2〉.

3.2.2 FSM Optimization

Since the output is both the present state and input conditions, i.e., the partitioning of sequencing part and command part is not feasible. Only horizontal partitioning is performed. We use the same methodology of the horizontal partitioning that the Moore machine synthesis is used and the only difference is the command part is included to do partitioning. But, in this case the column reduction has more effect than the row reduction. The reason is there are not many coding constraints groups since the command signals should be considered to check the coding constraints.

4. Experimental Results

In this section, we present some experimental results which were obtained by applying our approaches to two design examples. The first example is from [1] and called Sehwa example. We used Sehwa to schedule this CDFG with different latencies. Using the control synthesis techniques and the FSM optimization schemes of Section 3, the controllers are built with PLAs. 〈Table 3〉 shows the PLA areas and delays obtained by Moore machine synthesis and by Mealy machine synthesis in PLA area units (normalized), and FSM information. We added an estimate of the routing and buffering areas. The delays are computed as the worst case delay and the PLA delay figures are obtained using Crystal[17]. In this particular example, Mealy machine is smaller than Moore machine in area but the worst case delay of Mealy machine is longer than the delay of counter part.

The second example is the MOSTEK 6502 microprocessor. The specifications in ISPS were obtained from the High Level Synthesis benchmark set[18]. In order to obtain a manageable size example, which can be handled by Sehwa, we reduced the instruction set to four instructions. Also, since the original specifications were based on a non-pipelined scheduling, in order to enable us to perform pipelined scheduling with high throughput, we made some modifications on the data path and the CDFG. We used Sehwa to schedule the CDFG with latencies $L = 4$ and 6. For each scheduling, we generated a pipelined RTL implementation of the data path. State tables information for both Moore and Mealy

〈Table 2〉 Mealy-style state table for the example

Input DDDDD 12345	Present State	Next State	Output
0----	S1	S2	00--0-00000-00------0--
1----	S1	S2	00--0-00000-00-0-0--1--
0--0-	S2	S3	10--1011-10-10-------0-
0--1-	S2	S3	------11010-10-1010--1-
1-00-	S2	S3	10--10110----------0-
1-01-	S2	S3	------110-10-101010--1-
1-10-	S2	S3	10--10110-10-10------0-
1-11-	S2	S3	------110-10-101010--1-
-00-0	S3	S1	-11110---------11110--0
-00-1	S3	S1	-111101-1----------0--1
-01-0	S3	S1	----------11-1111111--0
-01-1	S3	S1	------1-1-11-11----1--1
-10-0	S3	S1	-10-11---------1111---0
-10-1	S3	S1	-10-111-1----------1
-11-0	S3	S1	-11010---------1111---0
-11-1	S3	S1	-110101-1----------1

〈Table 3〉 Experimental results of PLA controllers for both Moore and Mealy-style

Control Style		Moore	Mealy
L=2	area	397	361
	delay (ns)	4.11	4.15
	No. of states	16	3
	No. of PTs	40	16
L=3	area	370	338
	delay (ns)	3.18	3.30
	No. of states	32	2
	No. of PTs	128	32

〈Table 4〉 State table information of the M6502 example with L=4 and L=6

Control Style	Moore		Mealy	
L	4	6	4	6
No. of inputs	13	13	13	13
No. of states	664	196	4	6
No. of PTs	1,368	305	664	196

〈Table 5〉 Experimental results for the M6502 example with L=6 (Moore style)

No. of Partitions	PLA		Standard Cells	
	Area (mm^2)	Dealy (ns)	Area (mm^2)	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 6〉 Experimental results for the M6502 example with L=4 (Moore style)

No. of Partitions	PLA		Standard Cells	
	Area (mm^2)	Dealy (ns)	Area (mm^2)	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

〈Table 7〉 Experimental results for the M6502 example with L=6 (Mealy style)

No. of Partitions	PLA		Standard Cells	
	Area (mm^2)	Dealy (ns)	Area (mm^2)	Delay (ns)
1	2.50	32.69	0.70	31.40
2	2.65	15.75	0.98	26.80
4	2.94	11.93	1.02	16.90
6	3.13	11.93	1.13	16.90

〈Table 8〉 Experimental results for the M6502 example with L=4 (Mealy style)

No. of Partitions	PLA		Standard Cells	
	Area (mm^2)	Dealy (ns)	Area (mm^2)	Delay (ns)
1	2.16	16.51	0.64	23.30
2	2.11	14.42	0.69	16.80
4	2.32	12.35	0.85	14.30

model is shown in 〈Table 4〉. We used our algorithm to synthesize several implementations of the control part using both PLAs and standard cells. In each case, we generated layouts corresponding to various n-way partitioning of the groups of states for n = 1, 2, 4, 6. 〈Table 5〉 through 〈Table 8〉 shows area and delay data of the various implementations. The total area figures for the PLA implementations include 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[19] in SCMOS 3 technology whereas the Standard Cells were laid out using the GDT[20] CMOS 3 technology. The PLA delay figures were obtained using Crystal. The Standard Cell delay figures were estimated by MIS. As shown in 〈Table 5〉, the best area and performance were achieved using a four-way partitioning of the controller in both PLA and standard cell implementations for Moore model. In the case of Mealy model controller, unpartitioned implementations have best areas except the PLA implementation for a latency L=4. The areas are increased while the number of partitions goes up. The main reason is that the column and row reductions of the state table are not enough to compensate the extra routing area due to multiple partition. Delays are always getting smaller while the number of partitions is increased.

5. Conculsion

Pipelining has been a good methodology for designing fast digital ASICs, but pipeline architectures become quite complex to design as circuit sizes increase. Automated design tools in high level for pipeline ASICs are necessary

to cope with such complexity and explore the design space efficiently. In this paper we are focusing on the synthesis of controllers for pipelined data paths, especially on control mechanisms which provide control signals for the entire pipeline from a single source external to the pipeline, as usually known as time-stationary control schemes. The pipelined data path is generated by using the logical-stage concept (functional pipelining), not structural pipelining. we compare the synthesis schemes of Moore and Mealy-model controllers for pipelined data paths with conditional branches. First, the control specification is generated in the form of state table. Then FSM is synthesized and optimized by performing partitioning, state encoding, and logic minimization. Experimental results show the characteristics of the two different control styles and the effects of these two models on cost and performance.

References

[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, pp. 356-370, March 1988.

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

[3] E. F. Moore, 'Gedanken Experiments on Sequential Machines', Princeton University Press, Princeton, N.J., 1956.

[4] G. H. Mealy, "A Method for Synthesizing Sequential Circuits," Bell System Technical Journal, Vol. 34, No. 5, pp. 1045-1080, 1955.

[5] E. Davidson, "The Design and Control of Pipelined Function Generators," in Proc. of 1971 International IEEE Conference on Systems, Networks, and Computers, pp. 19-21, January 1971.

[6] C. Ramamoorthy and H. Li, "Pipeline Architecture," ACM Computing Surveys, Vol. 9, No. 1, pp. 61-102, March 1977.

[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, pp. 201-212, October 1982.

[8] 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.

[9] C. Tseng et al., "Bridge: a Versatile Behavioral Synthesis System," in Proc. of 25th Design Automation Conference, pp. 415-420, June 1988.

[10] G. De Micheli, R. Brayton, and A. Sangiovanni-Vincentelli, "Optimal State Assignment for Finite State Machine," IEEE Trans. on CAD, Vol. 4, No. 3, pp. 269-285, July 1985.

[11] R. Amann and U. Baitinger, "Optimal State Chains and State Codes in Finite State Machines," IEEE Trans. on CAD, Vol. 8, No. 2, pp. 153-170, February 1989.

[12] G. De Micheli and A. Sangiovanni-Vincentelli, "SMILE: a Computer Program for Partitioning of Programmed Logic Arrays," Computer-Aided Design, pp. 89-97, March 1983.

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

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

[15] R. K. Brayton, et al., "MIS:A Multiple Level Logic Optimization System," IEEE Trans. on CAD, Vol. 6, pp. 1062-1081, November 1987.

[16] J. Kim and F. Kurdahi, "Finite State Machine Optimization Algorithms for Pipelined Data Path Controllers," in Proc. of 4th Annual IEEE International ASIC Conference and Exhibit, pp. 18:7. 1-18:7.4, September 1991.

[17] J. Ousterhout, "Using Crystal for Timing Analysis," Tutorial, EECS Dept., UC Berkeley, 1985.

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

[19] R. Sickelmier, 'Release Notes for Oct Tools', Electronics Research Laboratory, UC Berkely, 1990.

[20] L Compilers Users Guide, Silicon Compiler Systems, 1989.

김 종 태

1982년 성균관대학교 전자공학과졸업(공학사)
1987년 미국 캘리포니아대학(Irvine) 전기및 컴퓨터공학과졸업(공학석사)
1992년 미국 캘리포니아대학(Irvine) 전기및 컴퓨터공학과졸업(공학박사)
1991년~93년 미국 The Aerospace Corp. 연구원
1993년~95년 전북대학교 컴퓨터공학과 교수
1995년~현재 성균관대학교 전기공학과 교수
관심분야: VLSI CAD, ASIC 설계, 컴퓨터구조