Automatic Synthesis of Pipeline Structures with Variable Data Initiation Intervals

Hong Shin Jun, Sun Young Hwang
CAD & Computer Systems Lab., Sogang University,
C.P.O. Box 1142, Seoul, Korea

Abstract - In this paper, we propose a novel algorithm for synthesizing the pipeline structures with variable data initiation intervals (DIIs). Compared to previous researches where the pipeline synthesis is confined to those with fixed DIIs, the proposed algorithm tries to optimize the pipeline latency by fully utilizing hardware resources to which abstract operations in high-level design descriptions are assigned.

Determining time-overlapping of pipeline stages, the proposed algorithm performs scheduling and module allocation using the time-overlapping information for the proper control of pipelines with variable DIIs. Experimental results on benchmarks show that significant improvement can be achieved both in speed and in area.

1. Introduction

High level synthesis is a design automation process that generates RTL (Register Transfer Level) hardware from behavioral descriptions. The process consists of two major steps: scheduling and module allocation. In the scheduling process, a specific time step is determined for each of the operations in input descriptions such that the design constraints (area, delay, power consumption, etc.) are satisfied while the other factors are optimized. In the module allocation process, operations and variables are assigned to FUs (Functional Units) and memory units, respectively, and interconnections among those modules are constructed by buses and/or multiplexors.

The higher is the abstraction level of design automation, the wider is the design space to explore. [6] It is very difficult to devise an efficient high-level synthesis algorithm that allows exhaustive search possibilities in the design space. To overcome this problem, application area of a high-level synthesis system is specified to a certain target architecture so that the system can synthesize the datapath optimized in the frame of the target architecture.[1][2]

Pipeline structures have been popularly employed for high-performance system designs. Pipelining is a powerful technique used for achieving high throughput in digital system.[5] The synthesis of pipelined datapaths has been addressed in a number of systems; Sehwa[7], PISYN[3], HAL[9]. each system has its own scheduling and module allocation algorithm. Sehwa and PISYN systems employ a list scheduling algorithm using the priority functions based on urgency and freedom, respectively. HAL system uses a force directed scheduling algorithm, which constructs a schedule for one operation in each iteration to make equi-distribution of operations among control steps for maximal hardware utilization. Target architecture supported in those systems is confined to the pipeline structures with fixed DII (Data Initiation Interval).

Pipeline structures with fixed DII have been widely employed for their simplicity of control and design. Pipeline structures with variable DIIs are the generalized form. It is well known that the high throughput can be obtained with variable DIIs. However, the complexity of the synthesis algorithm and pipeline control prevent the pipeline structures with variable DIIs from being utilized in digital system design. By automating the design of pipeline structures with variable DIIs, we can obtain hardware designs with higher-performance in throughput. In this paper, we propose novel algorithms for synthesizing the pipeline structures with variable DIIs. In Section 2, pipelining concepts are described. Scheduling, module allocation for pipeline structures with variable DIIs are presented in Section 3. Section 4 presents the experimental results for benchmarks and conclusions are drawn in Section 5.

2. Pipelining

Pipelining takes the approach of splitting the function to be performed into smaller pieces and allocating separate hardware to each piece, termed a *stage*. Pipelining provides a way to start a new task before an old one has been completed. Hence the completion rate is not a function of the total processing time, but rather of how soon a new process can be introduced, i.e., data initiation interval.[5][10] Pipeline technique is essential for repetitive computations for consecutive data as in digital signal processing.

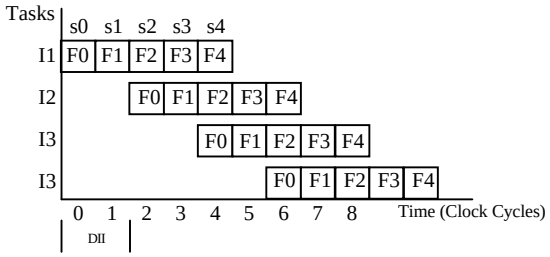


Fig. 1. Space-time diagram of pipeline with DII = 2.

Fig. 1 shows the space time-diagram for five-stage pipeline with DII = 2, where Ij represents the j-th task input and Fi represents the function executed in stage Si. At time 0, task I1 is entered into the pipeline and functions F0 and F1 are performed at stages S0 and S1 on I1 in two clock cycles. At time 2, while functions F2 and F3 are performed on I1, a new task I2 is initiated and functions F0 and F1 are performed on I2. At time 4, another task is entered into the

pipeline, while I1 and I2 are served in functional units F3/F4 and F1/F2, respectively.

In Fig. 1, we can find that functions F0, F2, F4 performed at the same time, as are functions F1 and F3. We can group the functions into two clusters; { F0, F2, F4 } and { F1, F3 }. Any functions belonging to the same cluster cannot be performed at the same FU, while functions belonging to different clusters can be. *Pipeline partitions* are defined to represent the sets of time-overlapping stages that are executed concurrently on consecutive data. In Fig. 1, sets {S0, S2, S4} and {S1, S3} are two pipeline partitions. It is noticeable that the number of pipeline partitions is equal to DII value. It is impossible for operations in a pipeline partition to share hardware resources, while operations in different pipeline partitions are allowed to do so. Most of pipeline synthesis systems employ scheduling and module allocation algorithms exploiting this fact.

3. Pipeline Structures with Variable DIIs

A. Background

Fig. 2 shows the hardware usage for pipeline structures supporting fixed and variable DIIs. For the 6-stage pipeline whose DII is 2, three adders and three subtractors are required as shown in Fig. 2 (a). For efficient utilization of hardware, delay insertion method has been proposed and used popularly[7]. A pipeline cannot sustain initiations at the maximum possible rate due to the collisions within the pipeline that prevent new operations from being initiated at certain crucial instants of time. By adding delays within the pipeline, it is always possible to attain maximum performance. Although the delays lengthen the total transit time through the pipeline, they increase the rate at which operations terminate if they remove collision that restricts the initiation rate.[10] By inserting a delay element in the pipeline of Fig. 2 (a), efficient hardware sharing can be obtained as shown in Fig. 2 (b). This pipeline structure requires two adders and two subtractors. Fig. 2 (c) shows the pipeline structure with DIIs alternating between 1 and 2. In the pipeline, two adders and two subtractors are required, each of which is fully utilized at all partitions. It is an efficient structure both in area and in performance. The throughput is increased from 1/2 to 1/1.5 without much hardware overhead.

B. Definitions

In the pipeline structures with fixed DII, the concept of *pipeline partition* is important for understanding hardware sharing scheme and for the design of synthesis algorithms. Several terms are formally defined, which are needed to describe the proposed algorithm for synthesizing the pipeline structures with variable DIIs.

Definition 1. (Initiation interval Sequence, IS)

Initiation interval sequence is a sequence which represents varying DIIs. In the pipeline structure with variable DIIs represented by initiation interval sequence IS = (I_0, I_1, ..., I_{L-1}), the i-th task is initiated at I_{i-1} time unit after the (i-1)st task has been initiated. The size of IS is represented by $|IS|$. In the sequence of IS = (I_0, I_1, ..., I_{L-1}), $|IS|$ is L.

Definition 2. (Acyclic IS, AIS)

Initiation interval sequence IS = (I_0, I_1, ..., I_{L-1}) is an acyclic IS, if and only if there exists no integer t satisfying

$$(I_0, I_1, ..., I_{L-1}) = (I_{t\%L}, I_{(t+1)\%L}, ..., I_{(t+L-1)\%L}),\ 0 < t < L.$$

For example, IS = (1,2,1,2) is not an AIS, because there exists an integer t = 2, which makes the same sequence. Given initiation sequence can be written IS = (1,2), which is acyclic.

Definition 3. (Initiation Time sequence, IT)

For a given initiation sequence IS = (I_0, I_1, ..., I_{L-1}), the initiation time sequence IT is defined as equation (1).

$$IT = (t_0, t_1, ..., t_L),\ t_0 = 0,\ t_i = \sum_{j=0}^{i-1} I_j \quad (i \geq 1) \qquad (1)$$

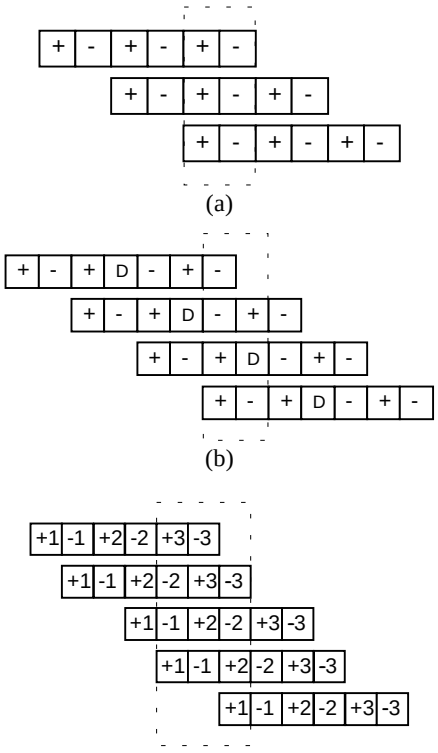


Fig. 2. Pipeline structures and hardware utilization. (a) Pipeline structure with a fixed DII. (b) After delay insertion. (c) Pipeline structure with variable DIIs.

Data initiations in the pipeline whose IS = (1, 2) and IT = (0, 1, 3) are shown in Fig. 3. The number below the absolute time k (in the second row) represents the total number of data initiations by time k. The number of data initiations during the time interval corresponding to a horizontal line segment is equal to the size of IS. As the number of horizontal line segments by time k, excluding the segment to which k belongs, is $\lfloor k/t_2 \rfloor$, the number of data initiations amounts to $\lfloor k/t_2 \rfloor * 2$. In the horizontal line segment to which time k belongs, two

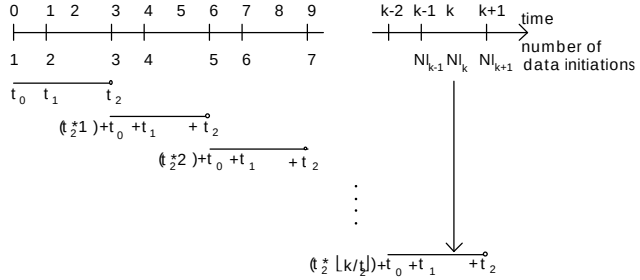


Fig. 3. Data initiations in the pipeline whose IS = (1,2).

data initiations occur. This number is obtained from j + 1, where j is an integer satisfying $t_j \le k \bmod t_2 < t_{j+1}$. In the Figure, we can find that $t_1 \le k \bmod t_2 < t_2$ is satisfied and determine j to 1. From this argument, the number of data initiations can be generalized as follows.

For a given initiation sequence IS = (I_0, I_1, ..., I_{L-1}), and/or initiation time sequence IT = (t_0, t_1, ..., t_L), the number of data initiations by time k is obtained by equation (2).

$$NI_k = L * \lfloor k/t_L \rfloor + (j + 1) \quad (2)$$

where j is an integer satisfying $t_j \le k \bmod t_L < t_{j+1}$.

We can represent the stages activated at time k by using the minimum stage index and the incremental indices. The space-time diagram for the pipeline of Fig. 3 is shown in Fig. 4. From the figure, the stages activated at time k are s1, s2, s4, s5. The minimum stage index is 1 and the incremental indices are 1, 2, 1.

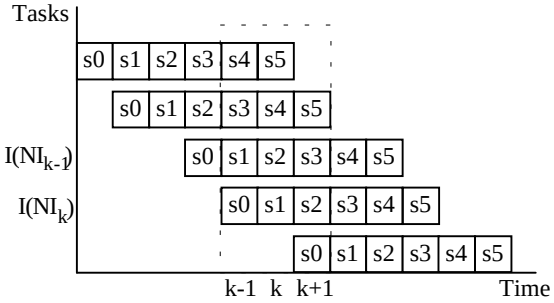


Fig. 4. Space-time diagram for the pipeline in Fig. 3.

The minimum stage index is determined by the time difference between the current time and the last data initiation time. The minimum index among the stages activated at time k becomes $k - (t_2 * \lfloor k/t_2 \rfloor + t_1)$, because the last data initiation occurs at $(t_2 * \lfloor k/t_2 \rfloor + t_1)$. This argument can be generalized as follows:

For a given initiation sequence IS = (I_0, I_1, ..., I_{L-1}), and/or initiation time sequence IT = (t_0, t_1,..., t_L), the minimum index of the stages activated at time k, $s_{min,k}$, is given by equation (3).

$$s_{min,k} = k - (t_L * \lfloor k/t_L \rfloor + t_j) \quad (3)$$

where j is an integer satisfying $t_j \le k \bmod t_L < t_{j+1}$.

The other stages activated at time k are also determined from the minimum stage index. For a given initiation sequence IS = (I_0, I_1, ..., I_{L-1}), there exists an execution time overlap between stages whose time difference equals to an initiation interval I_i. The set of stages activated at time k (pipeline partition) is recursively defined as

$$P_k = \{ \text{ stages whose index is } s_n, \text{ s.t. } s_n = s_{n-1} + I_{(j-n) \bmod L},\ s_0 = s_{min,k},\ 1 \le s_n \le N_{stage} \} \quad (4)$$

where j is an integer satisfying $t_j \le k \bmod t_L < t_{j+1}$, and N_{stage} is the number of stages

Theorem 1. (Number of pipeline partitions)

In a pipeline structure with the acyclic IS = (I_0, I_1, ..., I_{L-1}), there are ΣI_i different pipeline partitions.

Proof> From equation (1), T_L is equal to ΣI_i. The minimum stage index at time $k+t_L$, $s_{min,k+t_L}$ is equal to $s_{min,k}$ as follows.

$$s_{min,k+t_L} = (k+t_L) - (t_L * \lfloor (k+t_L)/t_L \rfloor + t_j) = k - (t_L * \lfloor k/t_L \rfloor + t_j) = s_{min,k}$$

where j is an integer satisfying $t_j \le k \bmod t_L < t_{j+1}$.

Thus, the set of stages activated at time $k+t_L$, P_{k+t_L}, is equal to P_k. By restricting k to $0 \le k < t_L$, equations (2), (3), (4) can be simplified and given by equations (5), (6), (7).

$$NI_k = j + 1, \quad \text{s.t. } t_j \le k < t_{j+1} \quad (5)$$

$$s_{min,k} = k - t_j, \quad \text{s.t. } t_j \le k < t_{j+1} \quad (6)$$

$$P_k = \{ \text{ stages whose index is } s_n, \text{ s.t. } s_n = s_{n-1} + I_{(j-n) \bmod L},\ s_0 = s_{min,k},\ 1 \le s_n \le N_{stage} \} \quad (7)$$

For k1, k2 such that $0 \le k1 < k2 < t_L$, if there exists j satisfying $t_j \le k1$, $k2 < t_{j+1}$, P_{k1} is different from P_{k2}. It is due to the fact that their minimum stage indices $s_{min,k1}$ and $s_{min,k2}$ are different. If not, which means $j1 \ne j2$, there is no same sequence starting from different I_i. Hence P_{k1} is different from P_{k2}. By this argument, it is concluded that there are N_p different pipeline partitions.

Fig. 4 also shows the stage overlapping in the 6-stage pipeline with IS = (I_0, I_1) = (1,2). The initiation time sequence, IT, is (0, 1, 3) by equation (1). The number of partitions is 3, which is equal to t_L by Theorem 1. The minimum stage index for each partition can be calculated by equation (6); $s_{min,0} = 0 - 0 = 0$, $s_{min,1} = 1 - 1 = 0$, $s_{min,2} = 2 - 1 = 1$. The indices of the stages belonging to the pipeline partition P_0 are determined by equation (7). $s_0 = s_{min,0} = 0$, $s_1 = s_0 + I_{(0-1) \bmod 2} = 0 + I_1 = 2$, $s_2 = s_1 + I_{(0-2) \bmod 2} = 2 + I_0 = 3$, $s_3 = s_2 + I_{(0-3) \bmod 2} = 3 + I_1 = 5$. Thus, P_0 becomes { s0, s2, s3, s5 }. In the same way, pipeline partitions P_1 and P_2 are calculated, resulting in P_1 = { s0, s1, s3, s4 } and P_2 = { s1, s2, s4, s5 }.

C. Synthesis of pipeline structures with variable DIIs

1) Pipeline Scheduling

Synthesis of the pipeline structures with variable DIIs consists of pipeline scheduling, module allocation, and control generation. In the pipeline scheduling process, the stage is determined for each operation so that maximal sharing of FU can be achieved. To support variable DIIs in pipeline scheduling, the pipeline partition formally defined in section 3.B is derived and utilized for possible resource sharing. Throughout the scheduling process, the time frame interval for each abstract operation is calculated and maintained as a *scheduling state*. At an intermediate state of the pipeline scheduling process, each operation has its time frame interval, $[b_{opn}, e_{opn}]$. The objective function of the pipeline scheduling is defined as the measure of equi-distribution of operations to pipeline partitions and can be calculated by the time frame intervals. The probability that operations of type 'OP' belong to stage i, $p_{OP}(i)$, is the normalized form of the distribution graph[9] and is given by equation (8), where N_{OP} is the number of operations of type OP and Prob(opn, i) is the probability that an operation 'opn' is scheduled at stage i.

$$p_{OP}(i) = \sum_{opn \in OP} Prob(opn,i) / N_{OP}, \quad i = 1,...,max\_stage \quad (8)$$

$$\text{where } Prob(opn,i) = \begin{cases} 1 / (e_{opn} - b_{opn} + 1) & \text{for } e_{opn} \le i \le b_{opn} \\ 0 & \text{otherwise} \end{cases}$$

The probability that operations of type OP belong to pipeline partition P_k is given in equation (9), where the sum is taken over all the stages in the pipeline partition.

$$P_{OP}(k) = \sum_{\substack{\text{for all } i, \\ \text{s.t. } s_i \in P_k}} p_{OP}(i), \quad k = 0, ..., N_p - 1 \quad (9)$$

For efficient hardware utilization, a measure of equi-distribution for each type of operations is defined by an entropy function and given by equation (10).

$$H(OP) = - \sum_{k=0}^{N_p - 1} P_{OP}(k) \log P_{OP}(k) / \log N_p \quad (10)$$

The value of H(OP) lies between 0 and 1. When all the pipeline partitions have the same probability, H(OP) becomes 1. If the probability that operations of a given type belong to a pipeline partition is 1, i.e., when all the operations are concentrated on a certain pipeline partition, the H(OP) becomes 0. The objective function for a scheduling state S is the form of weighted sum of the entropy functions for all operation types and is given by equation (11), where the weight w(OP) for operation type OP is defined by the area and the number of appearances of the OP type operations in the behavioral description. The maximal sharing of a functional unit can be achieved by maximizing the objective function.

$$OF(S) = \sum H(OP)w(OP) \quad (11)$$

Neighbor states are new states which are introduced from present state by reducing the time frame interval of an operation by 1. The gain for a neighbor state is proportional to the derivative of the objective function. The priority function of the proposed iterative/ constructive algorithm is the linear approximation of the derivative of the objective function in time frame interval and is given by equation (12).

$$PF(opn) = abs(OF(Sb_{opn}) - OF(Se_{opn})) / (e_{opn} - b_{opn}) \quad (12)$$

where, $[b_{opn},e_{opn}]$ is the time frame interval of operation opn, and Sb_{opn}, Se_{opn} are neighbor states for operation opn whose time frame interval is changed to $[b_{opn}+1,e_{opn}]$ and $[b_{opn},e_{opn}-1]$, respectively.

The proposed scheduling algorithm generates a schedule that uses the minimum number of FUs while meeting given constraints of the number of stages and IS. The algorithm is of iterative/constructive nature in that it constructs schedule incrementally. From the initial state in which the time frame interval is set by ASAP and ALAP schedules, a neighbor state with the highest priority function is selected in each iteration. The scheduling process under the time constraint in DII and stage number is summarized in Fig. 5.

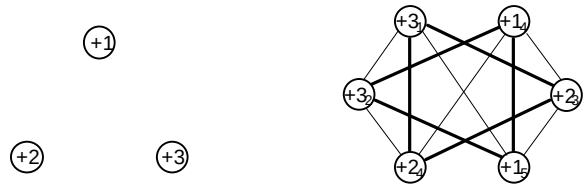
Step 1: Set the initial scheduling state where the time frame interval of each operation is determined by ASAP and ALAP schedulings.
Step 2: Calculate the priority function for each neighbor state.
For each unfixed operation opn do
Calculate objective function for 2 neighbor states, $OF(Sb_{opn})$ and $OF(Se_{opn})$.
Calculate priority function PF(opn).
Step 3: Make transition to the neighbor state with the highest priority function.
Step 4: If there remain unfixed nodes, go to Step 2.

Fig. 5. Pipeline scheduling algorithm.

2) Module Allocation of Virtual Operation

Module allocation process generates datapaths after assigning a FU for each operation, and providing interconnections among FUs using multiplexors and/or latches. In the process, efforts are made to allow maximal sharing of FUs. Sharing of a FU is not possible among the operations belonging to the same pipeline partition. The possibility of sharing can be modeled in compatibility graph, where vertices denote operations and edges represent the possibility the possibility of FU sharing.

The graph of Fig. 6 (a) shows the compatibility graph for the pipeline of Fig. 2 (c). No edges are drawn in the graph. For pipeline implementation, three adders would be required. However, this could be improved as follows. For the design of a pipeline with minimal hardware, virtual operations are introduced. In pipeline, an operation is multiply instantiated as a virtual operation for each data initiation. An operation for each data initiation is distinguished by attaching subscripts for each operation instance. In Fig. 2 (c), the virtual add operations in each pipeline partition are $(+3_1, +1_4)$, $(+3_2, +2_3)$, $(+2_4, +1_5)$. The compatibility graph for these virtual operations is presented in Fig. 6 (b). From the graph, a pipeline structure requiring two adders can be designed. One adder performs the virtual operations $(+3_1, +2_3, +2_4)$ and the other one performs $(+1_4, +1_5, +3_2)$.

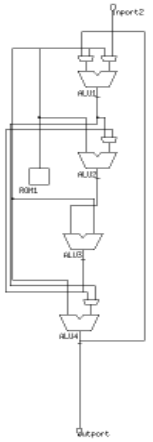


(a) (b)

Fig. 6. Compatibility graphs. (a) When an operation is executed on an operator. (b) When virtual operations are used.

4. Experimental Results

Fig. 7 shows the synthesized hardware for the pipelined execution of the tasks shown in Fig. 2 (c). The datapath is presented in Fig. 7 (a) and the controller description in TT format is shown in Fig. 7 (b). Allocation results for the virtual operations are presented in Table 1.



```
.i 2
.o 10
.fb 2
.p 3

11 0111111110
01 0011111001
00 1111110011
.e
```

(a) (b)

Fig. 7. Synthesized hardware for the pipelined execution of tasks shown in Fig. 2 (c). (a) Datapath (b) Controller description.

Table 1. Results of functional unit allocation.

FU	Function	Operations
ALU1	+	$+3_2$, $+1_4$, $+1_5$
ALU2	-	-1_3, -1_4, -2_3
ALU3	+	$+2_4$, $+2_3$, $+3_1$
ALU4	-	-3_2, -2_2, -3_1

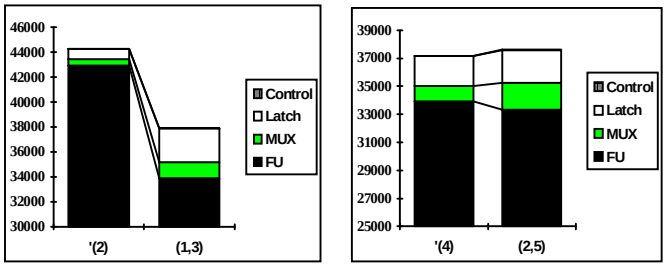
For experiments, synthesis has been performed for the high-level synthesis benchmark circuits; 16-point FIR filter, 5th order elliptic filter, and AR filter. Table 2 shows the synthesis results for a 16-point FIR filter. These results are obtained with the design constraints of 6-stage pipeline. When fixing DII to 5, synthesized datapath consists of 4 adders, 2 multipliers, 23 two-input MUXes, and 46 latches. The gate count of its controller amounts to 32. Pipelined datapath with initiation sequence of (4,5) needs the same number of FUs; 4 adders and 2 multipliers. Total gate count has been increased by 4.6 % due to the overhead in interconnection and controller area. However, performance gain of 10 % over the pipeline with fixed DII is obtained. The 5th order elliptic wave filter is synthesized in pipeline structure with 9 stages. Table 3 shows the

synthesis results for the pipelines with various initiation sequences together with the pipelines with fixed DII values. The pipeline with initiation sequence of (4, 6) requires less area than that of the pipeline with fixed DII of 5. Results in the other rows demonstrate the throughput enhancement with small amounts of hardware overhead. Synthesis results for the AR filter[4] are presented in table 4. New points in design space that could not be obtained with fixed DIIs are synthesized in the pipeline structures with variable DIIs. In the pipelines with IS = (1, 3) and (3, 5), 14.4 % and 22.7 % of area could be saved without throughput loss, respectively. When IS = (5, 6), performance improvement both in throughput and in area was achieved.

Fig. 8 shows the area overhead in the pipelines with variable DIIs. This figure compares the area of the pipelines in the first and second rows of table 4. In Fig. 8 (a), the total number of gates for FUs is significantly reduced for the pipeline with variable DIIs (from 42964 to 33904). On the contrary, the number of gates for control and interconnection is increased from 1317 to 4009. Even though the percentage of control and interconnection is increased from 3 % to 10.5 %, the total area is significantly reduced in the pipeline with variable DIIs. In Fig. 8 (b), the average DII for the pipeline structure with IS =(2,5) is 3.5 clock cycles, thus 12.5 % of performance gain is obtained. The number of gates for FUs is reduced from 33904 to 33320 (2 adders), while the total number of gates for control and interconnection is increased from 3260 to 4283. The total number of gates is increased by 1.2 % (37164 to 37603), because the amount of overhead in control and interconnection area is greater than that of benefits obtained in FUs. 12.5 % of throughput improvement is achieved with only 1.2 % of area overhead in the pipeline with variable DIIs.

5. Conclusions

Design of a synthesis system for the pipeline structures with variable DIIs has been presented. By automatic synthesis of the pipelines with variable DIIs, a larger design space can be explored. In this paper, a novel algorithm for automatic synthesis of the pipelines with variable DIIs is proposed. After determining time-overlapping



(a) (b)

Fig. 8. Comparison of gate counts for the AR filter.
(a) Area efficient design. (b) Higher throughput design.

of pipeline stages and pipeline partitions, the algorithm performs scheduling and allocation for maximal sharing of hardware modules. Comparing the synthesis results with the pipelines with fixed DIIs, significant improvement can be achieved both in speed and in area. Research is being continued for finding a method of selecting the initiation interval sequence that generates the pipeline structure with the best performance.

References

[1] J. Allen, F. Catthoor, "Architecture driven synthesis technique for VLSI implementation of DSP algorithms," *IEEE Proceedings*, Vol. 78, No. 2, pp. 319-335, Feb. 1990.

[2] B. S. Haroun, M. I. Elmasry, "Architectural synthesis for DSP silicon compilers," *IEEE Trans. CAD*, Vol. 8, No. 4, pp. 431-447, April 1989.

[3] K. Hwang, A. E. Casavant, "Scheduling and hardware sharing in pipelined data paths," in *Proc. ICCAD*, pp. 24-27, Nov. 1989.

[4] R. Jain, A. C. Parker, "Predicting area-time tradeoffs for pipelined design," in *Proc. 24th DAC*, pp. 35-40. June 1987.

[5] P. M. Kogge, *The Architecture of Pipelined computers*, McGraw-Hill, 1982.

[6] M. C. McFarland, A. C. Parker, "The high level synthesis of digital systems," *IEEE Proceedings*, Vol. 78, No. 2, pp. 301-318, Feb. 1990.

[7] N. Park, A. C. Parker, "Sehwa: A software package for synthesis of pipelines from behavioral specification," *IEEE Trans. CAD*, Vol. 7, No. 3, pp. 356-370, March 1988.

[8] N. Park, *Synthesis of High-Speed Digital Systems*, PhD thesis, University of Southern California, Oct. 1985.

[9] P. Paulin, "Force directed scheduling for the behavioral synthesis of ASIC's," *IEEE Trans. CAD*, Vol. 8, No. 6, pp. 661-679, June 1989.

[10] H. S. Stone, *High-Performance Computer Architecture*, Addison-Wesley: Reading, Mass., 1987.

Table 2. Synthesis results for the 16-point FIR filter.

DII	+	*	MUX	Latch	Cont.	Tot.	AIS	+	*	MUX	Latch	Cont.	Tot.	ΔS(%)	ΔA(%)
5	4	2	23	46	32	14244	(4,5)	4	2	29	49	59	14859	10	4.6

Table 3. Synthesis results for the 5th order elliptic filter.

DII	+	*	MUX	Latch	Cont.	Tot.	AIS	+	*	MUX	Latch	Cont.	Tot.	ΔS(%)	ΔA(%)
5	8	4	29	52	25	24161	(4,6)	7	3	43	54	106	21060	0	-12.8
							(3,6)	6	8	42	57	60	25458	10	5.2
7	6	2	29	46	49	15229	(6,7)	6	2	36	48	99	15887	7.1	4.3
8	6	2	29	46	50	15230	(6,9)	5	2	35	48	67	15499	6.3	1.8

Table 4. Synthesis results for the AR filter.

DII	+	*	MUX	Latch	Cont.	Tot.	AIS	+	*	MUX	Latch	Cont.	Tot.	ΔS(%)	ΔA(%)
2	12	10	8	10	5	44281	(1,3)	8	8	20	34	9	37913	0	-14.4
4	8	8	17	27	12	37164	(2,5)	6	8	30	29	43	37603	12.5	1.2
							(3,5)	4	6	30	24	41	28725	0	-22.7
6	4	4	28	2	25	21089	(5,6)	4	4	28	23	42	20625	8.3	-2.2