상위수준합성을 위한 배정가능범위 축소 스케줄링
(Mobility Reduction Scheduling for High-Level Synthesis)

유 희 진 † 유 희 용 ††
(Heejin Yoo) (Heeyong Yoo)

요 약 본 논문은 자원제약 조건하에서 파이프라인 데이타패스 합성을 위한 스케줄링 방법을 제안한다. 제안 방법은 연산의 배정 가능한 제어단계들 중에서 처음과 마지막 제어단계에 임시로 연산을 배정하여 스케줄링 해가 존재하는지를 평가한다. 만약 해를 발견할 수 없다면 이는 자원제약 위반에 의해 연산을 그 제어단계에 배정하는 것이 불가능함을 의미하기 때문에 그 제어단계를 배정 가능한 제어단계 후보에서 제거한다. 제안 알고리즘은 점진적 배정가능범위 축소에 기초하여 스케줄하고 자원 배정에 대한 영향을 고려하여 성능개선을 위한 해를 찾는다. 벤치마크에 대한 실험결과는 기존 방법들과 비교하여 개선된 실험결과를 보였다.
키워드 : 스케줄링, 상위수준합성, 데이타패스, 레지스터 배정, 파이프라인

Abstract This paper presents a scheduling approach for synthesizing pipelined datapaths under resource constraints. The proposed approach evaluates whether or not a scheduling solution can exist in case an operation temporarily is assigned to the earliest or latest control step among the assignable steps for the operation. If a solution cannot be found, it is impossible to assign the operation to that control step due to a violation against resource constraints, and so we can eliminate that control step among candidate assignable control steps. The proposed algorithm builds up a schedule based on gradual mobility reduction and finds a solution that yields high performance by evaluating on the impact on register assignment. Experiments on benchmarks show that this approach gains a considerable improvement over previous approaches.
Key words : scheduling, high level synthesis, datapath, register assignment, pipeline

1. Introduction

The scheduling in high level synthesis assigns operations in a Control Data Flow Graph(CDFG) to the control steps in which they will be executed. A scheduling problem is known to be NP-complete[1] in general, and so the existence of a polynomial time algorithm of solving that problem is highly unlikely. Lee et al.[2] formulate the constraints and objective function for scheduling and then find the optimal results of scheduling to satisfy the constraints. Unfortunately, for large problems, this technique is impractical in time and space. To solve this NP-complete problem, near-optimally heuristic methods that use the values of a defined priority function have been proposed. Current scheduling algorithms can be classified based on the basic techniques used, namely, integer programming based, list scheduling based, probability based, randomized searching based, and transformation based. Sehwa[3] is based on the list scheduling technique. The ready operations are sorted to a list in accordance to a non-increasing order of urgency and are scheduled into the control steps according to this sorted list. When the number of operations to be scheduled in a control step exceeds the number of resources, the remaining operations are delayed. PLS[4,5] uses a combination of forward and backward scheduling to schedule a loop under resource constraint. The previously scheduled operations are iteratively moved up and down by using the priority functions to accomodate the

† 정 회 원 : 순천제일대학 컴퓨터과학과 교수
hjyoo@suncheon.ac.kr
†† 정 회 원 : 동국대학교 컴퓨터공학과
yooheeyong@nate.com
논문접수 : 2004년 10월 7일
심사완료 : 2005년 5월 14일

ready-yet unscheduled operations. Different from Sehwa, in which operations are not allowed to be relocated, PLS reassigned scheduled operations to reduce the turn-around time of a loop. Choi[6] checks the critical paths to see whether operations on the critical paths can be scheduled with the resource constraints, and inserts delay operators into the critical paths in case of a failure. In each iteration, an operation type is selected first by the use of a priority function. One of the operations with the smallest force in the least dense partition of the distribution graph of the selected operation type is then scheduled. Hwang et al.[7] have shown a scheduling algorithm for distributing operations equally among partitions to maximize hardware sharing. The algorithm picks the least dense partition and then picks the least free operation in that partition. Paulin and Knight[8] introduced an approximation algorithm known as force-directed scheduling, which is an algorithm that tries to find minimal resource schedules under given time constraints. The intent of force-directed scheduling is to reduce the number of functional units, registers, and buses required by balancing the concurrency of the operations assigned to them but without lengthening the total execution time. The algorithm is iterative, with one operation scheduled in each iteration. The selection of the control step in which it will be placed is based on achieving a balanced distribution of operations in each control step. Given hardware constraints, the force-directed list scheduling kernel[8] schedules one of operations into a control step if the given modules are still available. The scheduling priority is measured by the total force of assigning an operation in a specific possible step. The smaller the total force, the higher priority of scheduling.

In the algorithms of previous researches, an operation being scheduled is selected by depending on a value of priority function. Those algorithms schedule all the operations by assigning it to a control step among all assignable control steps. As a result, those algorithms cannot always guarantee the optimal solution.

In this paper, we proposes a scheduling approach for synthesizing pipelined datapaths under resource constraints. A scheduling problem can be defined as follows: *given a fixed amount of resources, find the fastest schedule that satisfies the given set of constraints*. We present a technique that builds up a schedule based on gradual mobility reduction. It finds a solution by evaluating on assigning operations to each control step of the reduced mobility and its impact on register assignment.

This paper is organized as follows. Section 2 explains the proposed scheduling algorithm, Section 3 presents the violation evaluation algorithm for detecting any violation against resource constraints, and Section 4 considers assigning operations to each control step of the reduced mobility and its impact on register assignment. In section 5, we describe the time complexity of the proposed algorithm. Experimental results are shown in Section 6. Finally, concluding remark are made in section 7.

2. Scheduling algorithm

Consecutive tasks in the pipelined datapaths are initiated at latency[6]. The minimum achievable latency can be obtained by considering the resource constraints and analyzing the precedence constraints. The lower bound of latency for a DFG is given by

$$\max_{1 \leq k \leq m} \lceil \frac{N_k}{R_k} \rceil * D_k$$

where N_k, R_k, and D_k are the number of operations of type k, the number of functional units of type k, and the delay of each unit, respectively[5]. The initial set of assignable control steps for all operations is obtained by using both As Soon As Possible (ASAP) scheduling and As Late As Possible (ALAP) scheduling. Operations on the critical paths are checked so as to determine whether they can be scheduled with the available functional units and latency. If the operations on the critical paths cannot be scheduled due to resource conflicts, a new set of assignable control steps should be recalculated after increasing the number of control steps by one.

Verhaegh et al.[9] have shown an incremental way for computing the changes in the distribution functions, based on gradual time frame reduction. Instead of reducing in each iteration the time frame $\{a_v, ..., b_v\}$ of an operation v to one time t, they decide to reduce the time frame with only one time point, i.e., to $\{a_v+1, ..., b_v\}$ or $\{a_v, ..., b_v+1\}$. They consider the difference of force between the earliest and latest time frame among the assignable time frame of each operation as force gain by reducing the time frame of an operation and keeps on scheduling by selecting an operation with maximum force gain and reducing the worst side time frame of the operation.

Different from Verhaegh's method, Our work focuses on evaluating whether or not a scheduling solution can exist in case an operation temporarily is assigned to the earliest control step or the latest control step among the assignable steps for the operation. Let S_{E_k} and S_{L_k} be the control steps into which operation op_k is scheduled by the ASAP and ALAP algorithms. Clearly, $E_k \leq L_k$. In a feasible schedule, op_k must begin its execution in a control step no sooner than S_{E_k} and no later than S_{L_k}. The number of control steps between S_{E_k} and S_{L_k} is called the mobility of operation op_k(i.e. $mobility(op_k) = \{S_j \mid E_k \leq j \leq L_k\}$)[10]. Figure 1 shows the principle of scheduling by the proposed method. Figure 1(a) shows the mobility of every operation. Let us assume that the number of available resources for five operations of the same type is two. In figure 1(a), the mobility of operation 2 range from control step 0 to control step 2. If this operation is assigned to the earliest control step 0 or the latest control step 2, it is impossible to schedule other operations due to resource constraints, and so we must eliminate the control steps for scheduling operations as shown in

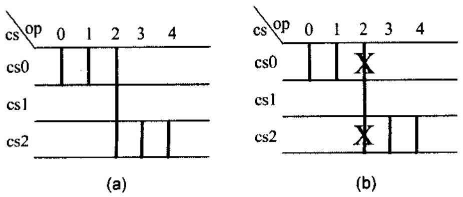


Figure 1 the principle of scheduling (a) the mobility of operations (b) the result of scheduling

in figure 1(b).

The scheduling process shown in Figure 2 is as follows. First, the scheduling algorithm begins by temporarily assigning an operation to the earliest control step, i.e., ASAP value, or the latest control step, i.e., ALAP value, among the assignable steps of the operation. Once an operation is assigned to the control step, the violation evaluation algorithm determines whether or not the assignment of the operation to the control step violates resource constraints. If it is impossible to assign the operation to the control step due to resource conflict, the scheduling algorithm eliminates that control step by reducing the mobility of the operation. If the ASAP value of an operation is greater than the ALAP value after the reduction of mobility, operations cannot be scheduled within the current number of control steps. Thus, the algorithm repeats the above process after increasing the number of control steps by one. The scheduling algorithm is reiterated until reduction of mobility cannot be obtained. When there is no further mobility reduction and the modified ASAP value of an operation is the same as the modified ALAP value for all operations, the scheduling is complete. If the values differ, assigning operations to each control step of the reduced mobility and its effectiveness on register assignment leads to the schedule.

3. Violation evaluation

Lee et al.[11] have shown an algorithm for scheduling a loop in a pipelined fashion such that the iteration time is minimized. The algorithm returns a feasible schedule of DFG if any one has been found. Whenever a violation against the resource constraint is detected in a certain state, some operations scheduled into that state are rescheduled to several other states. The selection of candidate operations for rescheduling is based on a priority function, called as *variability*. Variability of an operation is defined as the difference between the average hardware cost required to implement operations scheduled into a state in the look-ahead schedule and the resource constraint. The look-ahead schedule is obtained by delaying operation as little as possible. Since the smaller the variability

```
(1) Determine the latency.
(2) Calculate the initial set of assignable control steps
(3) Find the critical path in the DFG
        if ( resource conflict on the critical path )
          then recalculate the initial set after increasing the number of control steps by 1.
(4) Repeat following sub-steps for all the operations, $op_i$
 (4-1) Assign an operation to the earliest control step of the operation temporarily.
 (4-2) if ( violation_evaluation( ) )
        then Assign an operation to ASAP control step of the operation.
        else  eliminate the control step.
            if ( ASAP( $op_i$) > ALAP( $op_i$) )
              then Go to step 4 after increasing the number of control steps by 1.
 (4-3) Assign an operation to the latest control step of the operation temporarily.
 (4-4) if ( violation_evaluation( ) )
        then Assign an operation to ALAP control step of the operation.
        else  eliminate the control step.
            if ( ASAP( $op_i$) > ALAP( $op_i$) )
            then Go to step 4 after increasing the number of control steps by 1.
(5) if ( ASAP( $op_i$ ) = ALAP( $op_i$ ) )
      then scheduling copmplete
      else register_minimization( )
```

Figure 2 scheduling algorithm

value is, the better the corresponding candidate operation is. The violation resolving for the resource constraint relies on variability for choosing operations for rescheduling.

Different from Lee's method, Our proposed algorithm relies on the freedom of an operation to assign its resource for detecting any violation against resource constraint. The scheduling of operations in a DFG entails placing the operations in one of l partitions $P_0, P_1, \ldots, P_{l-1}$, where l is the latency. The operations scheduled into control steps $k+ml(m=0,1,2,\ldots)$ run concurrently, as they are in the same partition P_k. The violation evaluation algorithm determines whether an assignment of the operation to one of l partitions violates resource constraints within the assignable control steps while preserving the dependency between operations. The violation evaluation principle is as follows. If a set of assignable partitions for operation n is M_n and an expanded set of its partitions is P_n, then $P_n = M_n$ in the initial state of scheduling. In the process of obtaining a schedule, if the partitions belonging to M_n are $i,\ldots,j,\ldots,k$, the operations already assigned to the partitions $i,\ldots,j,\ldots,k$ are $op_{i1}, op_{i2}, \ldots, op_{j1}, op_{j2}, \ldots, op_{k1}, op_{k2}, \ldots$, respectively, and the sets of assignable partitions for those operations are $M_{i1}, M_{i2}, \ldots, M_{j1}, M_{j2}, \ldots, M_{k1}, M_{k2}, \ldots$, respectively, then $M_{i1}, M_{i2}, \ldots, M_{j1}, M_{j2}, \ldots, M_{k1}, M_{k2}, \ldots$ means the partitions are to be expanded to resolve a violation against the resource constraints. Therefore $P_n = M_n \cup M_{i1} \cup M_{i2} \cup \cdots \cup M_{j1} \cup M_{j2} \cup \cdots \cup M_{k1} \cup M_{k2} \cdots$. If there exists at least one partition among the elements of P_n to which the resources are not assigned, the violation should be resolved. However, if all of the resources were assigned to those partitions that correspond to all elements of P_n, the violation cannot be resolved.

Table 1 shows an example of the evaluation of a violation against resource constraints. Table 1(a) represents the mobilities of five operations that use resources of the same type. Table 1(b) means that four operations are already assigned to two resources. Then, if operation op_4 demands a resource assignment to partition P_0, a set of assignable partitions for operation op_4 is $\{P_0\}$ and an expanded set of assignable partitions for operation op_4 in the initial state of scheduling is $\{P_0\}$. Because a set of operations already assigned to partition P_0 are $\{op_0, op_1\}$ and the set of assignable partitions for those operations are $\{P_0, P_1\}$, partition P_4 is expanded into $\{P_0, P_1\}$ to resolve a violation against the resource constraints. As a result of this process, an expanded set of assignable partitions for op_4 is $\{P_0, P_1, P_2\}$. Therefore, we can assign operation op_4 to partition P_0 after moving op_2 to partition P_2 and op_0 to partition P_1.

Table 1 evaluation of a violation against resource constraints

(a) mobility of operations

operation / scheduling	op_0	op_1	op_2	op_3	op_4
ASAP	0	0	1	1	0
ALAP	1	0	2	1	0

(b) resource assignment for operations

resource / partition	R_1	R_2
P_0	op_0	op_1
P_1	op_2	op_3
P_2		

The violation evaluation algorithm tries to assign resources to the partitions corresponding to the ASAP value of all operations. If there is resource conflict in any partition due to resource constraints, one operation out of the operations in the relevant partition is selected to be reassigned. In selecting an operation that should be moved to the next partition due to resource conflict in a partition, the violation evaluation algorithm selects the most free operation to assign its resource. Here, we define a novel function, freedom, which is based on probability. This function can increase the possibility of resolving resource conflicts even in the next partition. The *freedom* of an operation is defined as

$$M(op_i) - Average(M(successors(op_i)))$$

where M denotes the mobility of an operation and $successors()$ denotes the set of all successors of the same type of op_i.

Figure 3 shows the proposed violation evaluation algorithm for detecting any violation against resource constraints. The algorithm tries to assign resources to the partitions corresponding to the ASAP control step of operations. If there are any resource conflicts in that partition due to resource constraints, an operation with the largest freedom among operations in that partition is selected to yield. It can also increase the possibility of resolving any violation against resource constraints at the continuing resource conflict.

4. Register Minimization

In order to illustrate how the function used by scheduling can capture the requirement of the register assignment, we use the concepts were taken from forward-looking objective functions [12]. We assume that a variable is alive at the end of the control step where it's created until the beginning of the control step where it's consumed. The number on the right side of each scheduling instance in figure 4 indicates the number of variables that are alive in each control step. A *variable cut* refers to the variables that are alive in a scheduled CDFG between control steps. The scheduling instances in figure 4 illustrate the observation that moving a group of two operations can impact the maximum number of registers by two variables. The key observation in figure 4 is that the effectiveness of the register assignment is directly impacted by function proportional to the sum of lifetimes of variable. Generally, variables that are alive for shorter lifetime interval are better for minimizing the number of registers since the likelihood that they will overlap with other variables is reduced. Due to the nature of register assignment problem, in addition to the lifetime of a variable, we must also consider the interplay among all variables. In order to address this, we use the concept of *weighted lifetime* to characterize individual variable lifetimes. The notion of weighted lifetime is to model the observation that variables that are alive while many others are also alive are more likely to adversely impact the ultimate number of required registers than otherwise.

```
violation_evaluation_algorithm( )
{
for ( $op_i$)
  {
  if (there are available resources in a partition, ASAP( $op_i$)%Latency)
    then return success;
    else if ( yield(an operation which has the largest freedom of operations in that partition) )
          then return success;
          else return fail;
    }
}

yield( $op_i$)
{
  yield(all successors);
  if (ASAP( $op_i$) = ALAP( $op_i$)) return fail;
  ASAP( $op_i$)++;
  if (there are available resources in a partition, ASAP( $op_i$)%Latency)
    then return success;
    else if ( yield(an operation which has the largest freedom of operations in that partition) )
          then return success;
          else return fail;
    }
}
```

Figure 3 violation evaluation algorithm

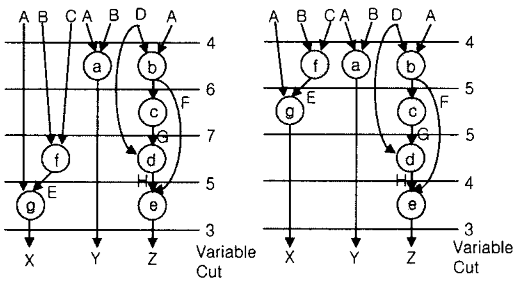


Figure 4 scheduling and variable-cut tradeoffs

The following steps describe a method to compute the longest possible variable lifetimes and worst-case the number of the variables that are alive between control steps. The cardinality of variable cuts play an important role in determining how many registers are required for a given scheduled computation. In general, if a variable belongs to several large cuts, then it's more likely to remain in large cuts. Ultimately, the size of the largest variable cut will determine the minimum number of required registers for a given schedule. Since we are interested in reducing lifetime of all variables, we select the tentative schedule with the most negative combined weighted lifetime.

step 1. Calculate the as-soon-as-possible(ASAP) creation and as-late-as-possible(ALAP) consumption for each variable. Additionally, determine the lifetime of each variable $L(x)$:

$$L(x) = ALAP(x) - ASAP(x)$$

where $ASAP(\,)$ indicates the earliest and $ALAP(\,)$ indicates the latest control step at which a variable can be alive.

step 2. For each variable create a vector $V_{cs}(x)$, which denotes the number of variables which are alive in each control step that variable x is alive. We define the set $V_{cs}(x)$ for a variable x:

$$V_{cs}(x) = \{V_1, V_2, .., V_t\}$$

where V_i is the cardinality of the variable-cut in control step i, if variable x is alive in control step

i and 0 otherwise.

step 3. Calculate *weighted lifetime* of each variable.

$$WL(x) = max\,(V_{cs}(x)) \cdot L(x)$$

step 4. Calculate the combined weighted lifetime cWL:

$$cWL = \sum_{i=1}^{n} WL(x)$$

where x_i represents the i-th variable in the given computation CDFG, and n is the total number of variables.

Note when there is no further mobility reduction in the scheduling algorithm shown in figure 2, the proposed algorithm finds a solution by evaluating on assigning operations to each control step of the reduced mobility and its combined weighted lifetime on register assignment.

5. Time complexity

The scheduling time in the worst case is $O(n^3m)$, where n and m are the number of operations in a DFG and the average mobility per operation, respectively. The time complexity can be derived as follows. The violation evaluation algorithm for detecting any violation against resource constraints executes in the worst case as many times as the number of assignable control steps for all operations. Therefore, it has the execution time of $O(n^2m)$. The proposed scheduling algorithm calls the violation evaluation algorithm as many times as the number of operations, resulting in an overall execution time of $O(n^3m)$.

6. Experimental Results

We have implemented the proposed algorithm in a C++ program running on a PC environment. To evaluate the performance of the algorithm, we have conducted experiments on a set of applications available from the literature[3,13]. The experiments are repeated on the benchmarks for different resource constraints. The scheduled results are compared with that by ALPS[2], Sehwa[3], PLS[5], Choi[6], HAL[8] and Lee[11].

A. A 16-point FIR filter (chaining model)

In a pipelined 16-point digital FIR filter example borrowed from [3], the clock cycle is lengthened so that a multiplication takes only one cycle while two additions can be executed within a cycle (chaining). Table 2 shows the scheduling results for the FIR filter using both Sehwa and our proposed method. Since the proposed algorithm does not yet perform register allocation or mapping, a complete comparison between it and the whole of HAL system is not possible. However, to provide some form of yardstick, manual register assignments were made for each version of the design. Comparing the results of proposed method and Sehwa one can see that for the examples, designs with the same register count and fewer or same mux inputs were found. Table 3 shows comparative statistics on vari-

Table 2 Comparision of scheduling results for the FIR filetr

scheduling / resources	Sehwa (Greedy)	Sehwa (Optimized)	Proposed method
# of adders	5	5	5
# of multipliers	3	3	3
# of registers	18	18	18
# of mux inputs	27	23	23
delay time (# of control steps)	6	6	6

Table 3 The pipelined synthesis of the 16-point digital FIR filter

# of multipliers		8	4	3	3	2	2	1	1
# of adders		15	8	6	5	4	3	2	1
Latency		1	2	3	3	4	5	8	15
delay time (# of control steps)	Sehwa			7	6				
	HAL				6				
	Choi		6		6	6	7		
	Hwang	6	6		6	6			
	Lee	6	6	6	6	6	7	10	16
	Proposed method	6	6	6	6	6	6	10	15

ous design components. As shown in Table 3, for each combination of resource constraints, our scheduler accomplishes a schedule with the minimum delay time.

B. the fifth-order digital wave filter

This example has a relatively large number of loop carried data dependencies as well as intra-loop dependencies. We assume that there are no data dependencies between iteration; i.e. the outputs of the data flow graph will not feed back into the inputs. We have achieved the minimal number of resources for each latency and we have also minimized the delay time. As most systems do, we suppose a multiplication takes two cycles (multi-cycling) while an addition takes one cycle to complete. The critical path length is 17 cycles. Under the resource constraints, we have near optimally minimized both the latency and the delay for most of the case. Table 4 shows the scheduling results for the elliptic filter using both HAL and our proposed method. Since the proposed algorithm does not yet perform register allocation or mapping, a complete comparison between it and the whole of HAL system is not possible. However, to provide some form of yardstick, manual register assignments were made for each version of the design. Comparing the results of proposed method and HAL one can see that for the examples, designs with the same register count and fewer mux inputs were found. Table 5 shows comparative statistics on various design components. As shown in Table 5, with the exception of the resource constraint of only one adder and one multiplier, we observed that the proposed scheduler achieves the same results as the ILP.

Table 4 Comparision of scheduling results for the elliptic filter

scheduling / resources	HAL			Proposed method		
# of adders	3	3	2	3	3	2
# of multipliers	2	1	1	2	1	1
# of registers	12	12	12	12	12	12
# of mux inputs	31	34	26	31	30	21
delay time (# of control steps)	17	18	19	17	18	19

7. Conclusion

A scheduling algorithm for synthesizing pipelined datapaths under resource constraints has been presented. The algorithm builds up a schedule based on gradual mobility reduction and finds a solution that yields high performance by evaluating on the impact on register assignment. The success of an optimization algorithm often depends on carefully designed and tuned objective function. Experimental results have shown that our methodology is indeed very effective.

For most real designs, the saving in register requirements can be substantial relative to the extra overhead required for additional functional units. So we need to obtain more information in order to improve the schedule.

Reference

[1] Narasimhan, M. and Ramanujam, J., "Improving the computational performance of ILP-based problems," Proc. of Int. Conf. on Computer-Aided Design, pp. 593-596, 1998.

[2] C. T. Hwang, Y. C. Hsu and Y. L. Lin "Optimum and Heuristic Data Path Scheduling under Resource Constraints," Proc. of the 27th Design Automation Conference, pp. 65-70 July 1990.

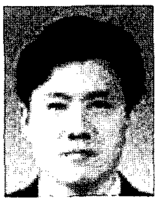
[3] N. Park and A. C. Parker, "Sehwa: A software package for synthesis of pipelines from behavioral specification," IEEE Trans. on Computer-Aided Design, vol. 7, pp. 356-370, March 1988.

Table 5 The pipelined synthesis of the fifth-order digital wave filter

# of multipliers		8	4	3	2	2	2	2	1	1	1	1
# of adders		26	13	9	7	6	5	4	4	3	2	1
Latency		1	2	3	4	5	6	7	8	9	13	26
delay time (# of control steps)	Optimum[2]	17	17	18	19	19	17	18	20	22	23	33
	PLS	17	17	18	19	19	17	18	20	22	23	33
	Sehwa	17	17	18	19	20	21	20	22	23	24	33
	Proposed method	17	17	18	19	19	17	18	20	22	23	34

[4] Hwang, C.T., Hsu, Y.C., and Lin, Y.L., "Scheduling for functional Pipelining and Loop Winding," Proc. of the 28th Design Automation Conference, pp. 764-769, 1991.
[5] Hwang, C.T., Hsu, Y.C., and Lin, Y.L., "PLS: A scheduler for pipeline synthesis," IEEE Trans. on CAD/ICAS, vol. 12, no. 9, pp. 1279-1286, Sept. 1993.
[6] Choi, Y.H., "Synthesis of pipelined data paths," Proc. of Int. Conf. on Computer-Aided Design, pp. 36-40, Jan. 1992.
[7] Hwang K.S., Casavant A.E., Chang C.T. and Manuel A. d'Abreu, "Scheduling and Hardware Sharing in Pipelined Data Path," Proc. of Int. Conf. Computer-Aided Design, pp. 24-27, 1989.
[8] Paulin P.G. and Knight J.P., "Force-directed scheduling for behavioral synthesis of ASIC's," IEEE Trans. on Computer-Aided Design, vol. 8, pp. 661-679, March 1989.
[9] Verhaegh W.F.J., Lippens P.E.R., Aarts E.H.L., Korst J.H.M., A. van der Werf and J.L. van Meerbergen, "Efficiency Improvements for Force-Directed Scheduling," Proc. of Int. Conf. Computer-Aided Design, pp. 286-291, 1992.
[10] D. Gajski, A. Wu, N. Dutt and S. Lin, HIGH-LEVEL SYNTHESIS Introduction to Chip and Syatem Design, pp.213-258, Kluwer Academic Publishers, 1992.
[11] Lee T.F., Wu A.C., Gajski D.D. and Lin Y.L., "An effective methodology for functional pipelining," Proc. of Int. Conf. Computer-Aided Design, pp. 230-233, 1992.
[12] J. L. Wong, S. Megerian, and M. Potkonjak, "Forward-Looking Objective Function: Concept & Applications in High Level Synthesis," Proc. of the 39th Design Automation Conference, June 2002.
[13] S. Kung, H. Whitehouse and T. Kailath, VLSI and Modern Signal Processing,, pp.258-264, Prentice Hall, 1985.

유 희 진
1990년 2월 원광대학교 전자계산공학과(학사). 1992년 2월 홍익대학교 전자계산학과(석사). 2000년 8월 홍익대학교 전자계산학과(박사). 2002년 3월~현재 순천제일대학 컴퓨터과학과 조교수. 관심분야는 설계자동화, 상위수준합성, 임베디드 시스템


유 희 용
1996년 2월 원광대학교 컴퓨터공학과(학사). 1998년 2월 동국대학교 컴퓨터공학과(석사). 2000년 2월 동국대학교 컴퓨터공학과(박사수료). 2000년 2월~2003년 8월 (주)창해소프트 근무. 2003년 8월~2005년 4월 (주)네오엠텔 근무. 관심분야는 사용자 인터페이스, 임베디드 시스템, 모바일 컴퓨팅