A Scheduling Algorithm for Multiport Memory Minimization in Datapath Synthesis

Hae-Dong Lee Sun-Young Hwang

CAD/Computer Systems Lab.
Dept. of Electronic Engineering, Sogang University
C.P.O. Box 1142, Seoul, 100-611, Korea
Tel: +82-2-705-8469
Fax: +82-2-3272-3220
e-mail: hwang@ccs.sogang.ac.kr

Abstract - In this paper, we present a new scheduling algorithms that generates area-efficient register transfer level datapaths with multiport memories. The proposed scheduling algorithm assigns an operation to a specific control step such that maximal sharing of functional units can be achieved with minimal number of memory ports, while satisfying given constraints. We propose a measure of multiport memory cost, *MAV* (Multiple Access Variable) which is defined as a variable accessed at several control steps , and overall memory cost is reduced by equally distributing the MAVs throughout all the control steps. When compared with previous approaches for several benchmarks available from the literature, the proposed algorithm generates the datapaths with less memory modules and interconnection structures by reflecting the memory cost in the scheduling process.

. INTRODUCTION

Due to the advance of VLSI circuit fabrication and design techniques, it has been become feasible to realize a large-scale system in a single chip. For the productivity enhancement of design engineers, researchers in the CAD community have been attempting to automate design process at higher levels of abstraction. In the past few years, synthesis from behavioral descriptions has become an active field of research[1]. A high-level synthesizer generates RTL (Register Transfer Level) datapaths according to the behavioral specifications described in a hardware description language. High-level synthesis consists of two major tasks: scheduling and allocation. Scheduling determines the number of control steps (clock cycles) needed to execute input behavior and the operations performed in each control step. Allocation consists of three subtasks: functional unit allocation, register allocation, and interconnection binding. Each operation is assigned to a FU in functional unit allocation process, and variables are bound to registers or register files during register allocation, and connection structures are constructed between FUs and registers in interconnection binding process.

In designing complicated VLSI chips, registers are usually grouped into a register file for efficient implementation. Multiport memories provide an effective way for such an implementation, and the advantages of enhancing system performance. Designs utilizing multiport memories are more structured and modular, hence require less chip area, as compared to random logic. Furthermore, the generated design can be tested easily due to the reduced number of hardware modules. The availability of high-density and high-speed multiport memories motivates the use of multiport memories in datapath synthesis.

Most of the previous approaches allocate variables either to isolated registers or to register files [2-5]. However, they do not fully utilize the advantages of multiport memories for variable mapping. A handful of systems reported the use of multiport memories for datapath designs. Balakrishnan et al.[6] have reported a technique to minimize the number of memory modules. Grouping a maximal number of registers into a cluster, their algorithm assigns a cluster of registers to a multiport memory at a time. The left-over registers are either allocated to isolated registers or grouped into multiport memories by repeatedly applying the same procedure. The algorithm often leads to non-optimal register allocations in the number of memory modules and interconnection cost. Wilson et. al.[7] presented a heuristic algorithm in which registers are allocated to available multiport memories one-by-one. Due to the local nature of greedy search, the algorithm does not guarantee an optimal solution in the number of memory modules and the number of registers in each memory module. Ahmad et al.[8] formulated the 0-1 integer linear programming (ILP) to generate the minimum number of multiport memory modules, and tried to reduce the number of registers in each memory module. However, by not considering the connections between multiport memory modules and FUs, the algorithm incurs larger inter-connection cost in the final implementation of datapaths. Kim et al.[9] also used the 0-1 ILP to group variables into multiport memory modules, but they did not try to minimize the number of registers in each memory module.

In this paper, we propose new scheduling algorithm that utilizes multiport memories in RT-level datapath synthesis. Most of the systems utilizing multiport memories described above generate datapaths by taking a scheduled code sequence as input. In these systems, multiport memories are not fully utilized because the cost of memory modules is considered only in allocation process. The number of memory modules is related to the number of variables simultaneously accessed at a control step. More efficient utilization of multiport memories can be achieved by considering the cost of memory modules in scheduling process under time and/or resource constraints. The time-constrained scheduling tries to minimize the costs of FUs and memory modules, while satisfying the given time

constraints. In conventional resource-constrained scheduling, constraints are usually given by the available number of FUs. In addition to the constraints on FUs, constraints on memory modules, such as the maximum number of memory ports and the type of each port, can be given as constraints in scheduling process. The proposed resource-constrained scheduling algorithm accepts the constraints on the numbers of FUs and memory modules.

This paper is organized as follows: In the section to follow, the overall picture of the proposed system and our target architecture is presented, and the scheduling algorithm to utilize multiport memories is described in Section 3. In Section 4, experimental results and comparisons with other approaches are presented. Conclusions are drawn in Section 5.

. System Description & Target Architecture

Figure 1 shows the overall configuration and synthesis methodology of the proposed system, SODAS (SOgang Design Automation System). Taking VHDL behavior descriptions as input, *C/DFG* (Control/Data Flow Graph) containing data and control informations on the input behavior is generated as an intermediate format, and is used for both simulations of input design descriptions and synthesis of datapaths. Design constraints with respect to resources and/or execution delays are refined into synthesis constraints more suitable for synthesis process, such as number of control steps and available hardware modules. Under those constraints, scheduling is performed with *C/DFG*s which are verified through simulation.

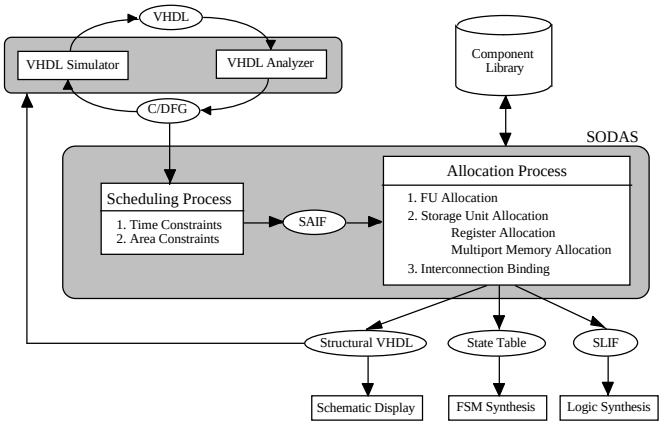


Figure 1: Overall picture of SODAS.

The scheduling result is saved in *SAIF* (Scheduling/Allocation Intermediate Format). Taking *SAIF* as input, allocation process assigns variables to storage units, operations to FUs, and connections between storage units and FUs to interconnection structures such as muxes and buses. During synthesis process, the component library is used to provide information on area and delay of each hardware module. Synthesis results are generated in forms of structural VHDL description and state table. Structural VHDL of the generated datapath is used for schematic display, and controllers to drive the generated datapaths are synthesized by FSM generator using state table.

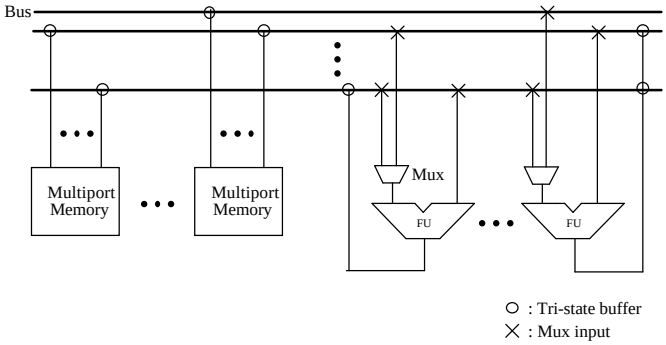


Figure 2. Target architecture of SODAS.

Target architecture of SODAS is similar to that in MAP system [8], as shown in Figure 2. Supporting linear topology, the architecture is flexible enough to be employed for general digital systems. It consists of a set of FUs, a set of multiport memories, and a set of interconnection units between memory modules and FUs for data communication. Each multiport memory consists of several ports, each of which may be read-only, write-only, or read-write type. A bus is connected to each port. Tri-state buffers are used to drive a specific bus, and output of a FU is connected to specific buses through the tri-state buffer. SODAS employs two-phase clocking schemes to drive generated datapaths. In the two-phase clocking scheme, data is stored into a memory module at phase-1, while data is loaded from a memory module to a FU at phase-2.

. Scheduling Reflecting Multiport Memory Usage

Determination of a proper objective function in scheduling determines overall performance of the synthesizer. Efficient usage of FUs can be achieved in scheduling process by equally distributing operations over the control steps[3]. Most of the scheduling algorithms previously published try to minimize the numbers of FUs and control steps. In contrast to the previous approaches, the proposed scheduling algorithm tries to minimize not only the number of FUs but also the number of memory ports through which variables are accessed.

Each operation executed at a control step is divided into three phases: operand read, execution, and write-back phases. The operands used in FUs are loaded from memory modules at read phase, and the result of execution is stored at memory modules at write phase. The number of memory ports is proportional to the maximum number of data transfer operations (reads and writes) executed at a control step. By considering the number of data transfers in the scheduling process, the requirement of memory ports can be reduced. Two different schedulings under the time constraints of five control steps are presented in Figure 3 to show the benefits obtained by considering data transfers in scheduling process.

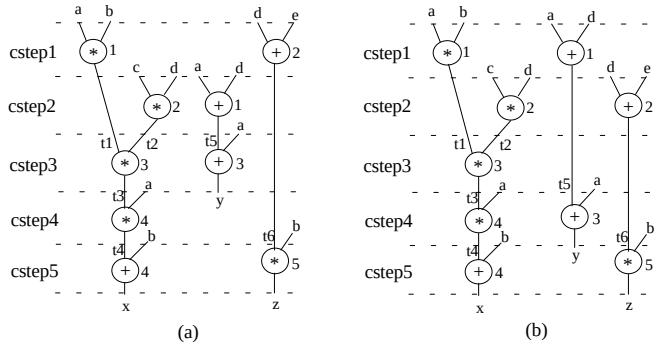


Figure 3. Two different schedules obtained by
(a) not considering data transfers (b) considering data transfers.

Optimal solution requires one multiplier and one ALU, as shown in both schedules. However, Figure 3 (b) shows less number of data transfer operations than that of Figure 3 (a). For the schedule in Figure 3 (a), the maximum number of operand reads is determined to be four, due to the variables accessed at cstep 1 (a, b, d, and e). In contrast to Figure 3 (a), maximum number of reads for the schedule in Figure 3 (b) is three: at cstep1 (due to a, b, and d), cstep2 (due to c, d, and e), and cstep5 (due to b, t4, and t6). Number of writes is two for both schedules. Assuming that available memory modules have one read-only port and two read/write ports and that the two-phase clocking scheme is employed, two memory modules are required for the schedule in Figure 3 (a). In Figure 3 (b), one memory module is sufficient for data operations by assigning $+_1$ and $+_3$ operations to cstep1 and cstep 4, respectively. In the proposed scheduling algorithm, efforts are made to reduce the number of concurrent data transfers as well as the number of FUs.

A. Priority Function

The time frame interval for each operation in C/DFG can be obtained by the ASAP and ALAP schedules. Let b_{opn} and e_{opn} be the ASAP and ALAP schedules for an operation *opn*, respectively. The priority function of the proposed scheduling consists of two major factors. The first factor is defined as the measure of equi-distribution of operations to control steps, and can be calculated by the time frame intervals for the operations in C/DFG. The probability for the operations of type 'OP' of belonging to control step i, $P_{OP}(i)$, is the normalized form of the distribution graph[4] and is given by equation (1), where N_{OP} is the number of operations of type 'OP' and Prob(opn,i) is the probability of an operation *opn* scheduled at control step i.

$$P_{OP}(i) = \sum_{opn \in OP} Prob(opn, i) \;/\; N_{OP} \qquad (1)$$

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

For the maximal utilization of hardware resources, a measure of equi-distribution for each type of operations is defined by an entropy function[10] and given by equation (2). The value of H(OP) lies between 0 and 1. H(OP) becomes 1, when all the control steps have the same probability. If the probability that the operations of a type belong to a certain control step is 1, i.e., when all the operations are concentrated at a certain control step, the H(op) becomes 0.

$$H(OP) = - \sum_{i=1}^{\#csteps} P_{OP}(i) \cdot \log(P_{OP}(i)) / \log(\#csteps) \qquad (2)$$

The second factor of our priority function reflects the distribution of data transfers over all control steps. The number of data transfers at a control step determines the number of memory ports required to access the operands. To reduce the number of memory ports, we try to evenly distribute data transfers over all the control steps. A variable accessed at several control steps, called as *multiple access variable (MAV)*, is treated with special care. For example, variable *a* is accessed at cstep1, cstep2, cstep3, and cstep4 in Figure 3 (a), and is accessed at cstep1 and cstep4 in Figure 3 (b). It means that less ports are required for Figure 3 (b). The situation of Figure 3 (a) can be improved utilizing MAVs. If operation requiring an MAV is not on critical path, memory port requirement can be reduced by equally distributing MAVs over all the control steps. We define *VarSet*(*opn*) as the set of variables accessed by an operation *opn*. For example, *VarSet*(*1) in Figure 3 is {a, b, t1}. The probability for a variable *v* to be accessed at control step i, Prob(*v*, i), is obtained by multiplying the fractions of time frame intervals for the operations which access variable *v*, as shown in equation (3). The probability for variables of being simultaneously accessed at control step i, $P_V(i)$, is given by equation (4), where *V* is the set of MAVs. A measure of equi-distribution for these variables is defined by an entropy function and given by equation (5). Lager value of H(V) indicates that MAVs are more equally distributed throughout the control steps.

$$Prob(v, i) = \prod_{\text{for all opn s.t. } v \in VarSet(opn)} \frac{1}{b_{opn}(v) - e_{opn}(v) + 1} \qquad (3)$$

where $b_{opn}(v)$ and $e_{opn}(v)$ are ASAP and ALAP schedules of operation *opn* which uses an MAV *v* as operand.

$$P_V(i) = \sum_{v \in V} Prob(v, i) \qquad (4)$$

$$H(V) = - \sum_{i=1}^{\#csteps} P_V(i) \cdot \log(P_V(i)) / \log(\#csteps) \qquad (5)$$

Figure 4 presents the initial scheduling state for the C/DFG in Figure 3, and the time frame intervals for operations are shown in Figure 4 (a). The MAV set is determined to be {a, b, d} from the C/DFG. Lifetime for an MAV is determined as the union of the time frame intervals of operations which use the MAV as operands, as shown in Figure 4 (b). Lifetime for MAV "a" is obtained by the time frame intervals of operations $*_1$, $*_4$, $+_1$ and $+_3$ which use the MAV as operand. Figure 4 (c) shows the probabilities for operations and MAVs in each control step. From Figure 4 (c), the measures of equi-distribution for operations and for MAVs are calculated by substituting the probabilities into equation (2) and (5), respectively. For example, the entropy value for MAVs is determined by (0.13*log0.13 + 0.19*log0.19 + 0.19*log0.19 +

0.3*log0.3 + 0.19*log0.19)/log5 = 0.914. Similarly, H(+) becomes 0.914 and H(*) is 0.977. It can be observed that multiplication operations are distributed more evenly than those of addition operations.

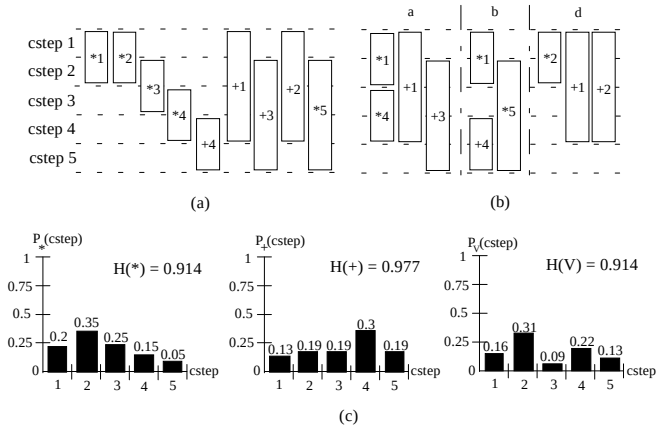


Figure 4. Initial scheduling state for the C/DFG in Figure 3
(a) Time frame intervals for operations (b) Lifetimes for MAVs.
(c) Probabilities for operations and MAVs in each control step.

Through the scheduling process, the time frame interval for each operation is calculated and maintained as a *scheduling state*. We define the objective function for a scheduling state S as the weighted sum of the entropy functions for operations and variables, as shown in (6).

$$OF(S) = \sum_{\text{for all OP type}} H(OP) \cdot W_{OP} + H(V) \cdot W_p \qquad (6)$$

The weight W_{OP} for operation type 'OP' is defined to be the cost of the corresponding module in system library, and the weight W_p is defined as the cost of memory ports. The maximal sharing of a functional unit and efficient utilization of multiport memories can be achieved by maximizing the objective function.

As scheduling progresses, the time frame interval for each operation is getting tighter. A pair of neighbor states, S_b and S_e, for a scheduling state S are defined as the scheduling states obtained by reducing the time frame interval of an operation *opn* from $[b_{opn}, e_{opn}]$ to $[b_{opn}+1, e_{opn}]$ and $[b_{opn}, e_{opn}-1]$, respectively. The gain for the transition to a neighbor state is proportional to the derivative of the objective function. The priority function of our scheduling process is the linear approximation of the derivative of the objective function in the time frame interval and is given by equation (7).

$$PF(opn) = \frac{| OF(Sb_{opn}) - OF(Se_{opn}) |}{e_{opn} - b_{opn}} \qquad (7)$$

In Figure 4 (a), the time frame interval of operation $+_1$ is found to be [1, 4]. Two neighbor states for $+_1$, Sb_{+1} and Se_{+1}, are obtained by changing time frame intervals of $+_1$, as shown in Figure 5 (a) and (b), respectively. The lifetime for MAVs in each state is also presented, and change of the time frame interval is indicated in bold line. As described earlier, Sb_{+1} and Se_{+1} are obtained by changing the time frame interval [1, 4] into [2, 4] and [1, 4] into [1, 3], respectively. The time frame interval of $+_3$ is also changed because $+_3$ is successor of $+_1$, as shown in Figure 5 (a).

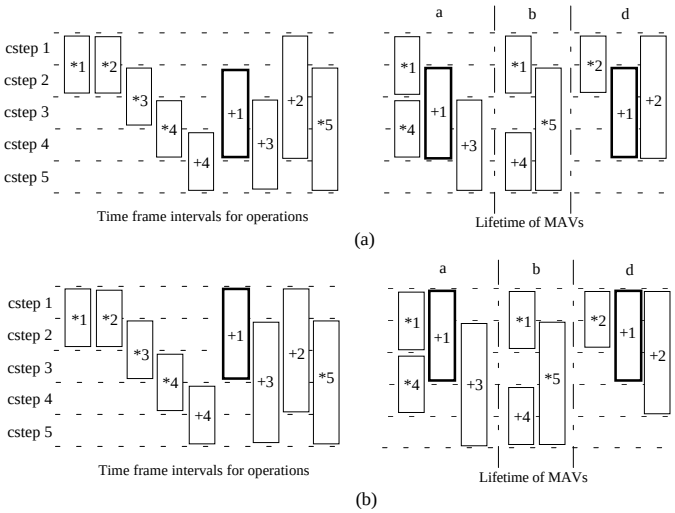


Figure 5. Two neighbor states for operation $+_1$, (a) Sb_{+1} , (b) Se_{+1}.

Probabilities for operations and MAVs at a control step of the neighbor states can be obtained by equations (1) and (4), as indicated in Figure 6. Entropy values for all operation types and MAVs are also presented in the figure. Comparing the entropy values of the neighbor states with those of the initial state of Figure 4 (c), probabilities for addition operation and MAVs become more balanced, which means that the numbers of adders and memory ports can be reduced in the neighbor states. The objective function for each neighbor state is calculated by equation (6) using the entropy values of operations and MAVs, and state transition is made to the neighbor state with larger priority function. The entropy values for multiplications in the neighbor states are the same as those of initial state, because the time frame intervals of multiplication operations have not been changed. However, the entropy values for addition operation and MAVs are increased by state transition to Se_{+1}, resulting in less number of adders and/or memory ports. Therefore, Se_{+1} in Figure 5 (b) is selected as next state.

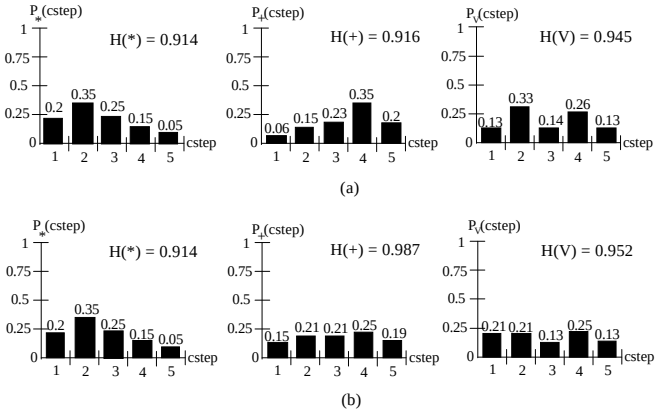


Figure 6. Probabilities for operations and MAVs (a) in Sb_{+1}, (b) in Se_{+1}.

B. Scheduling under Time Constraints

Figure 7 describes the overall time-constrained scheduling algorithm. The algorithm is of iterative/constructive nature in that it constructs a schedule incrementally. Time constraints can be specified in the number of control steps by the user. The scheduling algorithm generates a schedule that uses minimal number of FUs and memory ports, while meeting the given time constraints. From the initial state in which the time frame intervals are set by the ASAP and ALAP schedules, a neighbor state with the highest priority function is selected.

```
Time_Constrained_Scheduling ()
begin
 Find the initial scheduling state using ASAP and ALAP;
 while there remain unscheduled nodes do
  begin
   for each unscheduled operation opn do
    begin
      Calculate objective function for OF(Sb_opn), OF(Se_opn);
      Calculate priority function PF(opn);
    end;
    Make transition to the neighbor state with the highest PF(opn);
  end;
end;
```

Figure 7. Time-constrained scheduling algorithm.

C. Scheduling under Resource Constraints

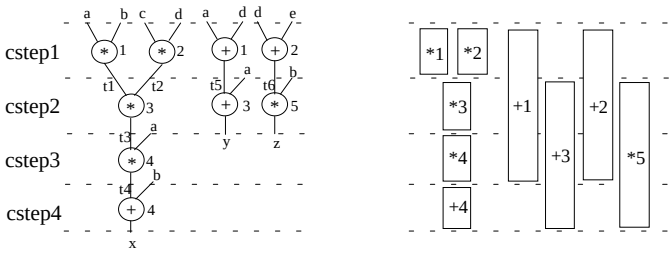
List scheduling is one of the most popular scheduling methods under resource constraints[1]. The constraints are usually given in terms of the number of FUs. SODAS employs a variation of the list scheduling algorithm for the synthesis of datapaths with both FU and memory port constraints. The goal of resource-constrained scheduling is that each operation is assigned to a control step such that maximal sharing of functional units can be achieved with minimal number of memory ports, while satisfying the given resource constraints on both memory modules and FUs. Resource constraints consist of two factors: available numbers of FUs and multiport memory modules. Design constraints on memory modules, such as number of memory modules, are refined into synthesis constraints more suitable for synthesis process. M_{OP} is defined as available number of 'OP' type operations, and M_r and M_w as the synthesis constraints in the number of reads and writes which can be concurrently executed in a control step, respectively. Overall resource-constrained scheduling algorithm is described in Figure 8.

In the proposed resource-constrained scheduling algorithm, operations which can be assigned to a control step *cs* are inserted into ready list denoted by *RL*(*cs*). N_{OP} represents the number of 'OP' type operations in the ready list, and the total numbers of reads and writes in the ready list is defined as N_r and N_w, respectively. During scheduling process, execution of the operation with the highest value of priority function is deferred to next control step when the given synthesis constraints are not met.

```
Resource_Constrained_Scheduling ()
begin
  for each operation opn do
  begin
     Set b_opn to the result of ASAP(opn);
     Set e_opn to the result of ALAP(opn);
  end;
  M_r = maximum allowed number of read operations for a cstep;
  M_w = maximum allowed number of write operations for a cstep;
  cstep = 1;
  while (constraints are met in all control steps) do
    begin
      Construct the ready list RL(cstep);
      N_OP = number of 'OP' type operations in RL(cstep);
      N_r = number of reads in RL(cstep);
      N_w = number of writes in RL(cstep);
      while ( N_OP>M_OP or N_r>M_r or N_w>M_w ) do
       begin
         if (all operations in ready list are in the critical path) then
           for (each operation in ready list) do e_opn = e_opn + 1;
         Calculate PF of Se_opn for all operations in ready list;
         Take Se with highest priority as the next scheduling state;
       end;
       cstep = cstep + 1;
    end;
end;
```

Figure 8. Resource-constrained scheduling algorithm.

Figure 9 (a) shows an initial scheduling state and the ASAP schedule for the C/DFG in Figure 3, and the time frame interval of each operation is shown in Figure 9 (b). Now, consider the case where FU constraints of two multipliers and one adder are given, and memory port constraints of four reads and three writes are derived from the design constraints specified by users. The ready list for cstep 1 is constructed by the ASAP schedule. The operations scheduled at cstep 1 is set to $\{*_1, *_2, +_1, +_2\}$. $*_1$ and $*_2$ can be scheduled to cstep1 because two multipliers are available. However, both $+_1$ and $+_2$ cannot be scheduled to control step1 due to the resource constraints. To satisfy the resource constraints, one of two addition operations must be deferred to a later control step. Priority function in equation (7) is used to determine the operation to be deferred. Figure 10 shows the neighbor states of $+_1$ and $+_2$ and probability at each control step.



(a) (b)

Figure 9. Initial scheduling state
(a) ASAP scheduling, (b) time frame intervals.

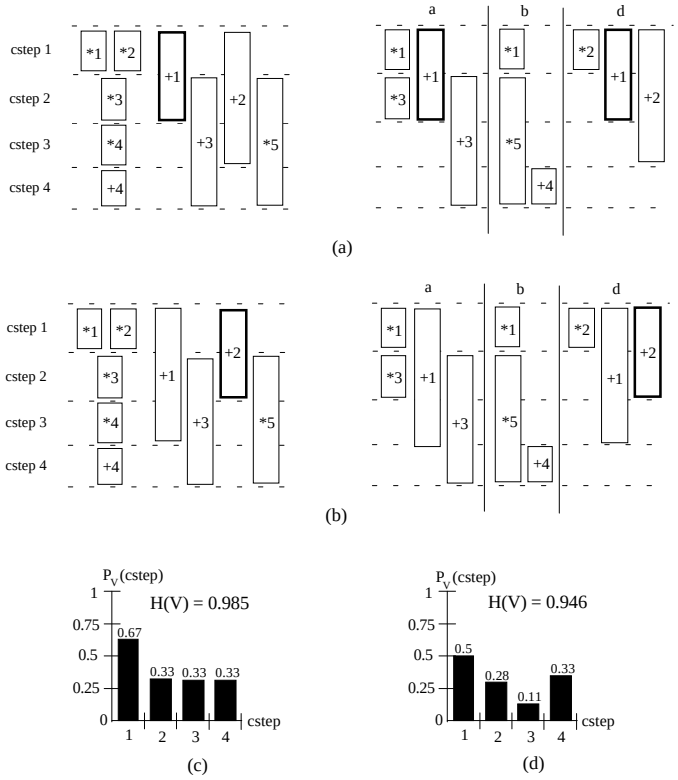


Figure 10. Neighbor states S_e for $+_1$ and $+_2$ in Figure 9.
(a) neighbor state Se_{+1}, (b) neighbor state Se_{+2}, (c) P_V for (a), (d) P_V for (b).

Figure 10 (a) and (b) show the neighbor states Se_{+1} and Se_{+2}, respectively. The entropy values of operations are same for both states. However, the probabilities for MAVs are changed from the initial state, as shown in Figure 11 (c) and (d). The entropy value for the MAVs in Se_{+1} is found to be 0.985, which is greater than 0.946 of Se_{+2}. Se_{+1} is chosen as next state. The distribution of MAVs is more balanced in the state.

D. Multiport Memory Allocation

Allocation process in SODAS consists of three phases: *FU allocation, multiport memory allocation, and interconnection binding*. FU allocation is performed by assigning each operation in C/DFG to a physical FU. The overall procedure for FU allocation consists of two phases: weighted compatibility graph construction and application of the weight-directed clique partitioning algorithm[2]. Edge weight in compatibility graph reflects the number of common variables used as sources or destinations of operator pair, thus represents the savings in area by sharing a FU for the operator pair.

Multiport memory allocation process is performed in two phases; *initialization* and *variable-to-memory port assignment.* In the first phase, lifetime analysis for the variables in *C/DFG* is performed and the number of memory ports is determined by analyzing all the control steps in code sequences. In the second phase, variable assignment to memory port is performed for each control step. In each control step, multiport memories are constructed and updated by using the results of the current assignment of variables to memory ports. For the assignment of variables to multiport memories, we employ a graph-theoretic approach using the weighted bipartite matching algorithm. Each of the variables accessed in control step *cs* is assigned to a specific memory port. A weighted bipartite graph, G(*cs*), is constructed for each control step *cs*, where variables accessed in the control step form a partition and the memory ports form the other partition. The edge weights are calculated to obtain the minimal cost assignment for the bipartite graph G(*cs*). They are determined by the internal and the external costs. Internal cost represents the numbers of registers and memory ports in each memory module, while the external cost reflects the number of multiplexers (and/or multiplexer inputs) in front of input ports of a functional unit and the number of tri-state buffers (TSB) required at the output port. After the edge weights in the bipartite graph G(*cs*) have been determined, each of the variables accessed in the control step *cs* is assigned to a specific memory port by using the bipartite matching algorithm. We denote N_V and N_p as number of variables in cs and number of memory ports, respectively. Let x_{ij} be 0-1 integer variable, where i = 1,...,N_V and j = 1,...,N_p. Variable x_{ij} is 1 when variable v_i is assigned to port p_j, and is 0 otherwise. Then, the variable-to-memory port assignment problem is transformed into a complete bipartite matching problem as shown in equation (8), where w_{ij} represents the edge weight between v_i and p_j.

$$\text{Minimize} \sum_{i,\,j} w_{ij} \cdot x_{ij} \qquad (8)$$

$$\text{subject to} \sum_{i} x_{ij} = 1 \text{ for } i = 1, \ldots, N_V$$

$$\sum_{j} x_{ij} = 1 \text{ for } j = 1, \ldots, N_p$$

In the formulation, constraints represent the requirements that there must be only one matching for variable v_i and port p_j. As a final step, each of the variables accessed in cs is assigned to a specific memory port. Multiport memory modules are constructed using the results of variable-to-port assignment. A variable must be assigned to only one memory module not to maintain multiple copies. By clustering the ports through which a variable is accessed into a memory module, memory modules are determined.

In interconnection binding phase, interconnection costs are reduced in number of multiplexer inputs by exchanging buses between memory modules and FUs performing commutative operations. The overall flow of Multiport memory allocation algorithm and detailed descriptions for cost function can be found in [11].

. EXPERIMENTAL RESULTS

The proposed algorithm has been implemented in C programming language on SUN SPARC workstation running UNIX™ operating system. Experiments have been performed for three MCNC benchmark programs; differential equation solver, sixth-order elliptic bandpass filter[5] and fifth-order elliptic wave filter[3]. In the experiments for filter benchmark programs, all the coefficients are assumed to be stored in a single port ROM. Results are compared with those synthesized by existing systems.

TABLE

SCHEDULING RESULTS UNDER SEVERAL TIME CONSTRAINTS FOR THE DIFFERENTIAL EQUATION SOLVER, WHEN THE MEMORY PORT COSTS ARE CONSIDERED DURING SCHEDULING.

#csteps		4	5	6	7	8	9	10	≥ 11
Scheduling with port costs considered	#read ops	6	4	4	4	4	3	3	2
	#write ops	4	3	3	2	2	2	2	1
	#RAMs	2	2	2	2	2	1	1	1
	#ALUs	2	2	2	1	1	1	1	1
	#Mults	2	1	1	1	1	1	1	1
Scheduling without port costs considered	#read ops	6	6	6	4	4	4	4	2
	#write ops	4	3	3	2	2	2	2	1
	#RAMs	2	2	2	2	2	2	2	1
	#ALUs	2	2	2	1	1	1	1	1
	#Mults	2	1	1	1	1	1	1	1

A. Synthesis of Differential Equation Solver

Table 1 compares the scheduling results for the differential equation solver under several time constraints, when the memory port costs are taken into account in the scheduling phase. Scheduling results without port costs considered are obtained by setting W_p in equation (6) to zero. The number of FUs is same for both cases, but more efficient schedules in terms of number of data transfers can be obtained by taking memory port costs into account during scheduling. Under time constraints of 5 and 6 control steps, the number of reads is reduced by two. When time constraints are set to 9 or 10 control steps, the number of reads is reduced by one.

From the experiments, it can be observed that the schedule requiring less memory ports is generated by taking the memory port costs into account. Reduction in the number of memory ports incurs the reduction in memory module costs. When it is assumed that the available memory modules have one read-only port and two read/write ports, the number of memory modules is reduced by one for both time constraints of 9 and 10 control steps, compared with the conventional scheduling methods.

TABLE

SCHEDULING RESULTS UNDER VARIOUS FU CONSTRAINTS FOR THE DIFFERENTIAL EQUATION SOLVER.

		(1,1)	(1,2)	(2,1)	(2,2)
Scheduling with port costs considered	#csteps	7	6	5	4
	#read ops	3	5	4	6
	#write ops	2	3	3	4
	#RAMs**	1	2	2	2
	#ALU's	1	2	1	2
	#Mults	1	1	2	2
Scheduling without port costs considered	#csteps	7	6	5	4
	#read ops	4	6	6	6
	#write ops	2	3	3	4
	#RAMs**	2	2	2	2
	#ALU's	1	2	1	2
	#Mults	1	1	2	2

* FU constraints: ($M_{mult.}$, M_{ALU}) ** RAM with 1 R port and 2 R/Wports

Table 2 shows the scheduling results for the differential equation solver under various resource constraints. The goal of resource-constrained scheduling is to minimize the number of control steps, while satisfying the given constraints. The results show that the schedule requiring the minimal number of control steps is achieved under FU constraints. It can be observed that the number of data transfers is also reduced by taking memory port costs into account during scheduling. For the resource constraints of two multipliers and one ALU, the number of reads is reduced by two when using the memory modules with one read port and two read/write ports.

B. Synthesis of 6-th Order Bandpass Filter

Table 3 shows the synthesis results for the 6-th order elliptic bandpass filter with the time constraints of 11 control steps. We compare the results with those produced by the system proposed in reference [9], MAP, and ADPS[5] systems. The number of registers is the same for all the systems. However, SODAS generates the datapath with fewer multiplexer inputs and tri-state buffers than those by the other systems. Both SODAS and Ref[9] generate the datapath requiring two memory modules. However, port costs (number of R/W ports) of SODAS is less than those of Ref[9].

TABLE

SYNTHESIS RESULTS FOR THE 6-TH ORDER BANDPASS FILTER

	SODAS	Ref[9]	MAP	ADPS
#Csteps	11	11	11	11
#ALUs	2	2	2	2
#Multipliers	1	1	1	1
#Mux Inputs	9	10	12	27
#TSBs	6	6	7	N/A
#Buses	5	5	5	N/A
#Registers	11	11	11	11
#RAMs	2^a	2^b	2^c	N/A
#ROMs	1	1	1	N/A
CPU Sec.	0.76	N/A	1.01	197*

a : a RAM with 1 R port & 1 W port, and a RAM with 1 R port & 2 R/W ports

b : a RAM with 2 R/W ports, and a RAM with 3 R/W ports c: Unknown

TABLE

SYNTHESIS RESULTS FOR THE 5-TH ORDER ELLIPTIC WAVE FILTER

	SODAS	Ref[9]	MAP	GMD	STAR	SPAID	HAL
#Csteps	17	17	17	17	17	17	17
#Adders	2	2	2	2	2	2	2
#Multipliers	1	1	1	1	1	1	1
#MUX Inputs	8	9	10	N/A	11	17	26
#TSBs	5	5	5	N/A	12	N/A	N/A
#BUSes	5	5	5	N/A	4	5	6
#Registers	11	12	14	11	13	19	12
#RAMs	2[a]	2[b]	2[c]	2[c]	5[d]	5[e]	N/A
#ROMs	1	1	1	1	1	1	1
CPU Time(sec.)	0.97	N/A	1.33	N/A	N/A	N/A	360*

a : a RAM with 2 R/W write ports and a RAM with 2 read ports & 1 read/write port b : a RAM with 2 R/Wports and a RAM with 3 R/Wports
c : Unknown d : Register files with 1 R port & 1 W port e : RAMs with 1 R/W port * Includes scheduling time

C. Synthesis of 5-th Order Elliptical Wave Filter

The synthesis results for the 5-th order elliptic wave filter with the time constraints of 17 control steps are shown in Table 4. The results are compared with Ref [9], GMD[12], MAP, STAR[4], and SPAID[13]. Both SODAS and GMD generate the datapaths requiring the least number of registers. GMD system also tries to minimize the number of registers in a memory module when clustering and assigning variables to memory modules. However, the comparison with GMD in terms of interconnection costs is not possible. GMD system does not take into account interconnection costs. The hardware generated by SODAS requires the least amounts of registers and multiplexer inputs comparing with those generated by the other systems. Comparisons with STAR and SPAID which supports memory modules with only one port, show that the number of multiplexer inputs is significantly reduced.

. CONCLUDING REMARKS

In this paper, we presented a heuristic scheduling algorithm for utilization of the multiport memories in datapath synthesis. Most of previous approaches utilizing multiport memories had taken into account the memory cost in allocation, which may lead to inefficient results. The proposed scheduling tries to assign each operation in input behavioral description to a specific control step, such that minimal number of FUs and data transfers can be achieved under time and resource constraints. We defined *MAV* as a variable accessed at several control steps to denote a measure of multiport memory cost, and overall multiport memory cost can be reduced by equally distributing the MAVs throughout all the control steps. Experimental results for benchmark programs show that SODAS generates efficient synthesis results utilizing multiport memories under time and/or resource constraints by reflecting the memory cost in the scheduling process.

ACKNOWLEDGEMENT

This work was supported by Korea Science and Engineering Foundation under Grant No. 9101-0009.

REFERENCES

[1] M. McFarland, A. Parker, R. Camposano, "The High-Level Synthesis of Digital Systems," Proc. of IEEE, Vol. 78, No. 2, pp. 301-318, Feb. 1990.

[2] C. Tseng, D. Siewiorek, "Automated Synthesis of Data Path in Digital Systems," IEEE Trans. on CAD, Vol. 5, No. 3, pp. 379-395, July 1986.

[3] P. Paulin, J. Knight, "Scheduling and Binding Algorithms for High-Level Synthesis," in Proc. 23rd ACM/IEEE Design Automation Conference, pp. 1-6, June 1989.

[4] F. Tsai, Y. Hsu, "Data Path Construction and Refinement," in Proc. IEEE International Conference on Computer-Aided Design, ICCAD-90, pp. 308-311, Nov. 1990.

[5] C. Papachristou, H. Konuk, "A Linear Program Driven Scheduling and Allocation Method Followed by an Interconnect Optimization Algorithm," in Proc. 27th ACM/IEEE Design Automation Conference, pp. 77-83, June 1990.

[6] M. Balakrishnan, A. Majmudar, D. Banerji, J. Linders, J. Majithia, "Allocation of Multiport Memories in Data Path Synthesis," IEEE Trans. on CAD, Vol. 7, No. 4, pp. 536-540, Apr. 1988.

[7] T. Wilson, D. Banerji, J. Majithia, A. Majumdar, "Optimal Allocation of Multiport Memories in Data Path Synthesis," in Proc. 32nd Midwest Symposium on Circuits and Systems, Urbana, IL, pp. 1070-1073, Aug. 1989.

[8] I. Ahmad, C. Chen, "Post-Processor for Data Path Synthesis Using Multiport Memories," in Proc. IEEE International Conference on Computer-Aided Design, ICCAD-91, pp. 418-421, Nov. 1991.

[9] T. Kim, C. Liu, "Utilization of Multiport Memories in Data Path Synthesis," in Proc. 30th ACM/IEEE Design Automation Conference, pp. 298-302, June 1993.

[10] H. S. Jun, S. Y. Hwang, "Design of a Pipelined Datapath Synthesis System for Digital Signal Processing," IEEE Trans. VLSI Systems, Vol. 2, No. 3, pp. 292-303, Sept. 1994.

[11] H. D. Lee, S. Y. Hwang, "Design of an Area-Efficient Allocation Agorithm for Datapaths with Multiport Memories," Journal of Microelectronic Systems Integration, Vol. 3, No. 1, pp. 109-121, 1995.

[12] I. Ahmad, C. Chen, "Grouping Variables into Multiport Memories for ASIC Data Path Synthesis," in Proc. IEEE International ASIC Conference, pp. 162-165, Sept. 1992.

[13] B. Haroun, M. Elmasry, "Architecture Synthesis for DSP Silicon Compiler," IEEE Trans. on CAD, Vol. 8, No. 4, pp. 431-477, Apr. 1989.