INTEGRATED SCHEDULING AND BINDING : A SYNTHESIS APPROACH FOR DESIGN SPACE EXPLORATION

M. Balakrishnan
Comp. Sci. & Engg. Dept.
I.I.T. Delhi, New Delhi
INDIA 110016

P. Marwedel
Institut fur Informatik
University of Kiel, Kiel
W.Germany, D-2300

Abstract

Synthesis of digital systems, involves a number of tasks ranging from scheduling to generating interconnections. The interrelationship between these tasks implies that good designs can only be generated by considering the overall impact of a design decision. The approach presented in this paper provides a framework for integrating scheduling decisions with binding decisions. The methodology supports allocation of a wider mix of operator modules and covers the design space more effectively. The process itself can be described as incremental synthesis and is thus well-suited for applications involving partial pre-synthesized structures.

1. INTRODUCTION

Synthesis, in the context of design automation of digital systems, usually refers to the process of transforming a behavioral design description into a structural design. This design process is equivalent to one to many mapping and thus the notion of a design space. Though a number of constraints on different parameters are to be satisfied for realizing the synthesized design, maximum emphasis is placed on the delay time (execution time of the behavioral description by the synthesized design) and the cost (considered proportional to the design area).

Both the delay time and the cost are closely related to scheduling, resource allocation and binding. Initial synthesis approaches involved performing these tasks in a sequence: now referred to as vertical synthesis by some authors. Tseng's [1] approach is typical where the synthesis is mapped to three steps, each modelled as a clique partitioning problem. The steps are :

a) Storage allocation and binding,
b) Operator allocation and binding and
c) Interconnection allocation and binding

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

Another example is MIMOLA [2,3] which starts by generating a maximum parallel schedule (as late as possible) by analyzing the data dependency among the operations. The global module allocation problem is modelled as an integer programming problem. The system is capable of handling complex multi-function modules and the clock time is determined by the maximum propagation delay path. Synthesis in many other systems follow a similar approach with some variations.

A major limitation of the above approaches is their inability to explore the design space. The only way a new design can be generated is by modifying the input program or the schedule. Further, by performing the optimization in a strict sequence, interaction between these steps is ignored resulting in poor designs.

This led to various new approaches to synthesis which either perform the tasks together (global optimization), iteratively or as an expert system. The intent is the better exploration of the design space. HAL[4,5] has a force-directed scheduling algorithm which performs scheduling within a time-constraint along with the minimization of required number of operators. Pfahler[6] describes a model based on a two-dimensional arrangement of operations for scheduling, processor mapping and register assignment. MAHA[7] performs scheduling of operations and allocation and binding of resources iteratively by first considering the operations in the critical path and then the other operations. The cost model is essentially that of operators with storage locations and interconnections ignored. SPLICER[8] presents a heuristic method for binding operations to operators and intermediate values to registers with a view to keep interconnections low. The results indicate that a look-ahead of a large number of subsequent control steps do not significantly affect the result. Peng [9] presents an iterative synthesis methodology (he calls it horizontal synthesis) based on a Petri-net model for design representation. The synthesis is carried out by a sequence of semantic preserving transformations on this representation. A key difference is the integration of a simple floor-planner within the design iteration.

Some results have been reported for predicting the design-space or more specifically to explore the area-time tradeoff [10-12]. Jain et al.[11] describe a model for

© 1989 ACM 0-89791-310-8/89/0006/0068 $1.50

predicting the area-time curves based on operator utilization. The approach presently makes a number of simplifying assumptions on module types and considers the cost as the sum of individual operator costs. On the other hand, McFarland [12] makes startling observations about area-time curves. His results point to two significant conclusions:

i) The area-time curve shape changes dramatically when one considers lower level details (like multiplexers, interconnections and layout).
ii) The design points do not lie on a smooth curve.

The synthesis approach presented in this paper is intended to overcome some of the limitations of the approaches discussed in [4-9]. We describe the key features below.

a) All synthesis systems are capable of handling designs with multiple operator modules, though only some of them can handle multifunction operators. On the other hand, nobody tackles the designs with a mix of different speed operators for the same operation. Jain et al.[13] present a scheme for module selection for pipelined designs but do not consider the possibility that a mix (like one high speed multiplier and one slow speed multiplier) could constitute a good design point. In that sense, present synthesis systems do not explore the potential design space spanned by the module library.

b) In many real-life design projects a partial pre-synthesized structure may need to be integrated efficiently i.e. to use the existing resources as much as possible for realizing the behavioral description while synthesizing only the extra structures needed. We refer to this as incremental synthesis and our methodology can handle such situations very naturally.

c) Area-time tradeoff seems to be a key to exploring the design space but none of the current synthesis systems seem capable of handling design decisions based on such a tradeoff. That is to say, A decision X is acceptable as it results in an expected extra area of only y whereas not(X) would result in an expected extra delay time of z. It is claimed that our approach provides a preliminary framework which can be extended for handling such tradeoffs. At present, design decisions are evaluated to minimize interconnections but it is intended to integrate a floorplanner to handle more accurate area projections (see Section 5). This would enable the design decisions to be based on a single parameter representing interconnection cost, multiplexer cost and operator cost.

2. SYSTEM OVERVIEW

The input to the synthesis program is a data flow graph with nodes representing the operations and arcs representing the values i.e. inputs, intermediate values and outputs. The operator modules which implement the operations and storage elements which store the values are the resources. Designers can specify the resources or let them be instantiated at an associated cost. This option is necessary for treating all operators uniformly as low cost operators (like gates) are more likely to be allocated instead of being shared. This is especially true if sharing involves creating additional paths and multiplexers. A mix of operator modules implementing the same operation at different speeds is allowed e.g. different speed multipliers with the same interface can be allocated together. The present implementation restricts all operations implemented by a multifunction operator to have the same associated propagation delay. Overall design constraints on area and time can be specified but this information is used only to reject design points.

Figure 1 shows the overview of the design process. A set of operations called candidate operations are prepared based on the data dependency and operator availability. These operations are bound to the available operators optimally i.e. to minimize extra interconnections. The values generated by the scheduled operations are bound to the available registers, again with a view to minimize interconnections. This is followed by updating the structure based on the current bindings. As the updated structure is used in the subsequent iterations, the process of incremental synthesis is obvious. Thus it is possible to start with a partially synthesized structure and use it effectively to complete the synthesis.

3. SCHEDULING AND BINDING

The data flow graph is analyzed and the longest pathlength for each operation is computed. In case of operations being implemented by operators with different delay times, the allocated operator with the least delay time is chosen as the associated delay of the operation node. The scheduling can be performed in three different modes:

i) Forward scheduling (based on as soon as possible)
ii) Backward scheduling (based on as late as possible)
iii) Double headed scheduling (based on alternating between the two)

3.1 Selection of Candidate Operations

The set of candidate operations are chosen based on data available operations and available operators. For forward scheduling, data available operations are those all of whose predecessors have been scheduled whereas for backward scheduling, it means those all of whose successors have been scheduled. The available operators are those which have been free at least for the duration of their delay time. Contention among operations is resolved by choosing the one with the longer unscheduled path-length. The marking of an operator as available only after its delay-time prevents an operation with a long path-length getting scheduled on a slow operator.

3.2 Zero-one Integer Programming Model

For binding both operations to operators and values to storage locations, a very similar zero-one integer programming model is used. Binding is equivalent to generating a specific mapping from a set of candidates $(a_i), i=1,..,n$ to a set of resources $(b_j), j=1,..,m$.

$$\text{BINDING: } \{ a_i \} \text{ ----> } \{ b_j \}$$

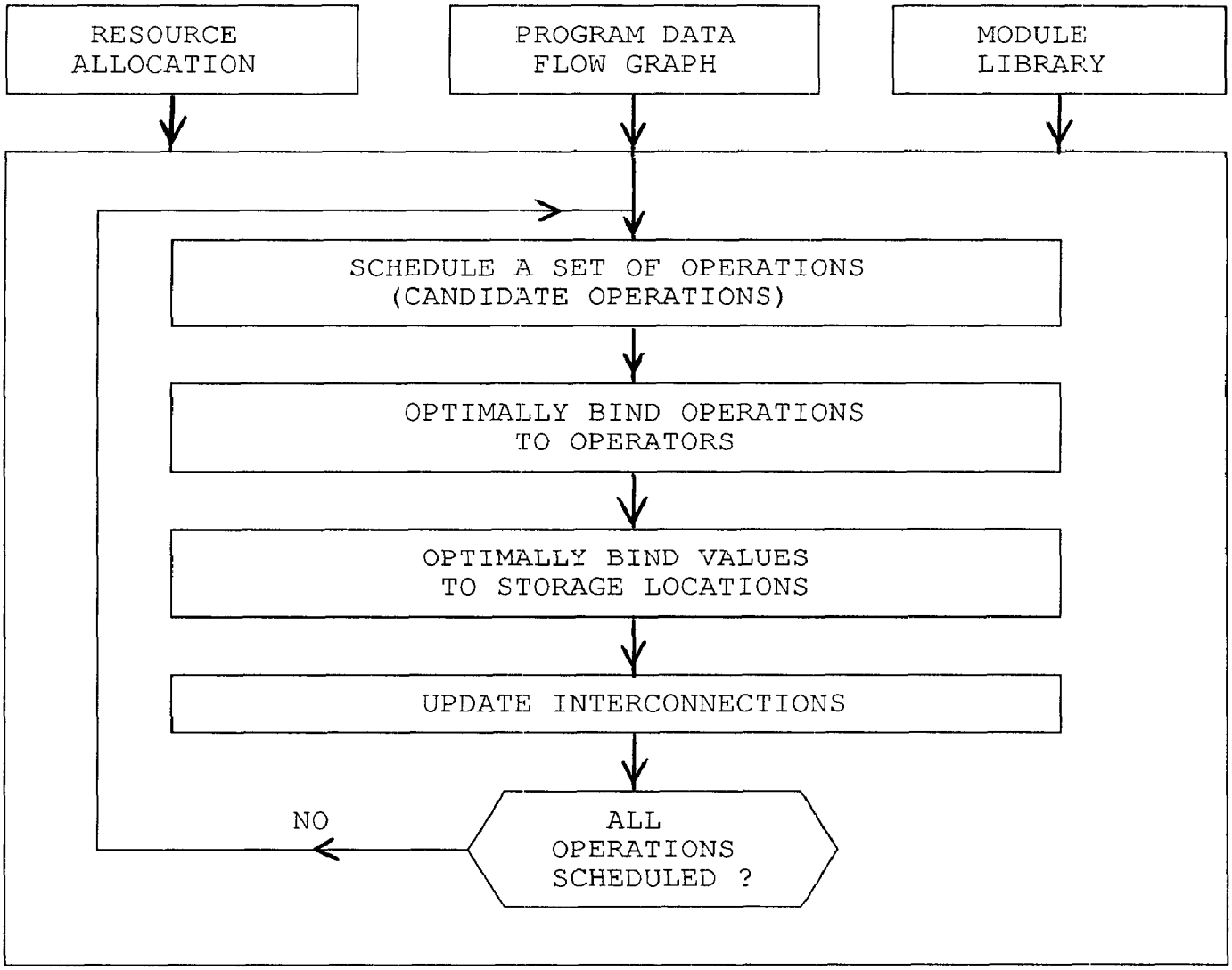

FIGURE 1. OVERVIEW OF THE SYNTHESIS PROCESS

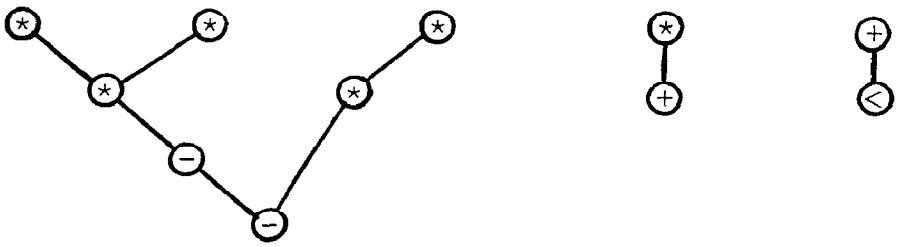

FIGURE 2 : Example Data Flow Graph[4]

We define :

x_{ij}: equal to 1 if a_i is bound to b_j else equal to 0.
: zero-one integer variable which defines the mapping

f_{ij}: defines feasibility of mapping a_i to b_j
: equal to 1 if the mapping is feasible else 0.

w_{ij}: cost of binding a_i to b_j
: computed only if the mapping is feasible.

In the current implementation, cost of a binding is decided by the number of extra interconnections required due to the binding. A simple procedure to compute this cost is as follows :

```
w_ij <- 0;

for (ALL CASES directly related to binding a_i to b_j) do

begin
     cond1 : w_ij <- w_ij + 2;
     (a definite extra interconnection
     for this case)
     cond2 : w_ij <- w_ij + 1;
     (a probable extra interconnection
     for this case)
     cond3 : w_ij <- w_ij + 0;
     (definitely no extra interconnection
     for this case)
end;
```

Thus the function to be minimized is

$$\sum_{i=1}^{n} \sum_{j=1}^{m} w_{ij}\, x_{ij}$$

The constraints originate from two types of restrictions on x_{ij}'s

i) Not more than one candidate can be mapped to a resource during the time under consideration.

$$\sum_{i=1}^{n} x_{ij} \leq 1 \qquad j=1,..,m$$

ii) All (or some) of the candidates have to be bound to at least one of the resources.

$$\sum_{j=1}^{m} f_{ij}\, x_{ij} = 1 \quad i=1,..,n$$

3.3 Operation-Operator Binding

Operation-operator binding involves mapping the set of candidate operations (p_i) to available operator modules (m_j).

P-M BINDING: (p_i) ------> (m_j)

We define the related terms to use the model described in Section 3.2

x_{ij} feasible: m_j performs the operation specified by p_i AND p_i has been available from data dependency considerations for at least the delay time of m_j.

For computing the weight w_{ij}

ALL CASES: corresponds to examining all ports (t_k) of module m_j.

COND1: t_k is not connected at all OR it is not connected to the storage to which the value (associated with p_i) is bound.

COND2: The value (associated with p_i) is not bound AND t_k is connected to at least one free location.

COND3: The value (associated with p_i) is bound AND t_k is connected to the location to which it is bound.

The computation of weight takes care of commutative operations. The solution to the above optimization results in a low cost binding of operations to operators.

3.4 Value-Storage Location Binding

Value-storage binding involves mapping the set of values (v_i) generated by the candidate operations to the free storage locations (s_j).

V-S BINDING: (v_i) ------> (s_j)

We define the related terms to use the model described in Section 3.2

x_{ij} feasible: If s_j is a free location when v_i is generated. A location is free when the previous value stored in it is dead. Though this is rather simple to compute for forward scheduling, it is more complicated in case of backward and double headed scheduling.

For computing the weight w_{ij}

ALL CASES: corresponds to examining all operations (p_k) which use v_i

COND1: s_j (appropriate port) is not connected at all OR it is not connected to the module to which p_k is bound.

COND2: p_k is not bound AND s_k (appropriate port) is connected to at least one operator to which p_k can be bound.

COND3: p_k is bound AND s_k (appropriate port) is connected to the module to which it is bound.

Again, the computation of weight takes care of commutative operations. A solution to the above optimization results in a low cost binding of values to storage locations while instantiating more storage units if required.

4. EXPERIMENTAL RESULTS

All the results in this section are based on data flow graphs available in the literature. The first example is from Paulin et al.[4] which they have used to illustrate their force-directed scheduling. Figure 2 shows the data flow graph taken from their paper. Table 1 presents the results of our scheduling along with HAL's. The module set for this example was :

MF : Fast Multiplier <time = 2, area = 40>
MS : Slow Multiplier <time = 4, area = 10>
AF : Fast Accumulator<time = 1, area = 4>
AS : Slow Accumulator<time = 2, area = 1>

Table 1 illustrates an important advantage of our design procedure. More effective coverage of the design space is realized due to inclusion of operators with different speeds. This is seen in not only covering up of the time gaps but also in reaching the minimal possible schedule time of 6 with an operator set with smaller cost (considering only the operator area). The cpu time for scheduling this small example was less than 0.6 seconds on an APOLLO workstation (DN 4000 with 4 MB memory) for all cases.

The second example we present is the digital filter discussed in SEHWA[15]. The module set is same as the one for Example 1 except that the accumulators can be replaced by adders. The results from our synthesis are presented in Table 2. Interconnections (col 4), number of multiplexer inputs (col 5) and storage locations required (col 6) along with the operator set(col 1) collectively represent the design area. The storage for multiplication constants and paths for routing it to multipliers were assumed to be available. The cpu time on the APOLLO workstation ranged between 1 and 5 seconds for all cases.

It is interesting to note that the schedule time was independent of the scheduling option used. In fact, experiments on a number of other large flow graphs indicated only small differences (if at all) in schedule time. Double headed scheduling (DHS) matched the faster schedule except for few exceptions. On the other hand, the effect of scheduling option on area parameters was large and varied. First two sets indicate Forward scheduling to be superior whereas the next three indicate Backward scheduling to be superior. DHS was always poor in terms of storage locations as short paths in the data flow graph scheduled from both sides locked up a number of locations as unavailable. Surprisingly, small routing area for backward scheduling in the 4th and 5th set seem to confirm the observation in [12] that design points do not lie on a smooth curve.

The last example we present is the fifth-order digital elliptic filter discussed in [5] and adopted as a benchmark example by the synthesis workshop group. The flow-graph is relatively large with 34 operations (additions and multiplications). The module set was the same as used in the last example. Table 3 presents the best result(s) for each of the eight operator sets. In two cases more than one choice is presented as the one with the lower schedule time appeared to have higher path cost. The cpu time on the APOLLO workstation for all cases presented below ranged from 2 to 8 seconds.

The results of scheduling are easily comparable to those reported in [5]. Our best case schedule time always matched that produced by force-directed scheduling and at least for one case (set 3 Backward option) it was better. Further, some results produced by a mix of operators seem to be very attractive design points, e.g. set 4.

5. LIMITATIONS AND FUTURE WORK

As mentioned in the introduction, the present system provides a framework for a synthesis based on area-time tradeoff. The key to this is the adjustment of optimization weights which govern the binding decisions based on both area and time. In the absence of a floor-planner, the set of candidate operations are so chosen that all of them can be scheduled on the available operators. But once an area estimate for binding an operation is obtainable from a floor-planner, the schedule-bind decision would be based on weighing the expected extra area against the expected extra time. The extra time of not binding an operation can be estimated from the unscheduled portion of the longest data flow path associated with the operation. In this case, the candidate operations would be a superset of the operations to be scheduled. Similarly, the weights for value-binding would more closely represent actual area. The designs generated are expected to reflect area-time tradeoff.

Synthesis of multi-port memories[15] has been incorporated as a post-processor for further area optimization. The number of interconnections are further reduced as the selection of registers to be merged in the memory as well as port allocation for locations is based on optimally sharing interconnections.

Presently, the system accepts only a subset of MIMOLA[16] for program input specification. Work is going on for extending it to accept the full MIMOLA syntax including hardware module declarations.

6. CONCLUSION

This paper presents a synthesis tool based on integrating scheduling with binding decisions. The two binding decisions considered are: operations to operators and values to storage locations. The approach supports a more flexible mix of operator modules which results in a more extensive cover of the design space than previously reported techniques. The methodology is naturally suited to projects involving partial pre-synthesized structures. Further, it is indicated how the system could be extended for taking schedule-bind decisions based on area-time tradeoff.The approach is illustrated with examples. The reported cpu time for synthesis is small enough to permit design space exploration by allocating different operator sets.

7. ACKNOWLEDGEMENT

The authors would like to thank Mr. R.Johnk for his varied help through different stages of this work.

OPERATOR SET	SCHEDULE TIME			
	DHS	FORWARD	BACKWARD	HAL
{1 MF, 1 AF}	13	13	13	13
{1 MF,1 MS, 1 AF}	10	10	11	—
{1 MF,2 MS, 1 AF}	9	9	9	—
{ 2 MF, 1 AF}	8	8	8	8-12
{ 2 MF, 2 AF}	7	7	7	7
{ 3 MF, 1 AF}	7	7	7	7
{3 MF,1 AF, 1 AS}	6	6	6	—
{ 3 MF, 2 AF}	6	6	6	6
{ 4 MF, 1 AF}	6	6	6	6

TABLE 1 : Schedule Time as a Function of Operator Allocation (Example Figure 2)

OPERATOR SET	SCHED. OPT.	SCHED. TIME	INTER-CONNECTS	MUX INPUTS	STORAGE LOCATIONS
{1 MF,1 AF}	DHS	18	18	15	6
	FORW.	18	11	4	5
	BACK.	18	19	11	8
{2 MF,1 AF}	DHS	15	15	5	6
	FORW.	15	11	6	4
	BACK.	15	18	8	8
{1MF,2MS,3AF}	DHS	12	30	23	8
	FORW.	12	24	14	8
	BACK.	12	21	15	6
{2 MF,2 AF}	DHS	11	23	15	7
	FORW.	11	19	11	7
	BACK.	11	12	6	4
{3 MF,2 AF}	DHS	10	24	15	6
	FORW.	10	21	12	6
	BACK.	10	13	6	4

TABLE 2 : Results from Scheduling and Binding the Digital Filter Data Flow Graph [15].

OPERATOR SET	SCHED. OPT.	SCHED. TIME	INTER-CONNECTS	MUX INPUTS	STORAGE LOCATIONS
{3 MF,3 AF}	FORW.	17	38	32	8
{2 MF,3 AF}	FORW.	18	35	30	7
{2 MF,2 AF}	FORW. BACK.	19 18	33 37	30 27	8 11
{1MF,1MS,2AF}	FORW.	20	30	28	6
{1 MF,1 MS, 1 AF,1 AS}	DHS	22	35	26	10
{1 MF,2 AF}	FORW.	22	28	26	6
{1MF,1AF,1AS}	FORW.	23	33	30	8
{1 MF,1 AF}	DHS BACK.	28 29	30 25	23 17	10 9

TABLE 3 : Results from Scheduling and Binding the Elliptic Filter Data Flow Graph [5].

8. REFERENCES

1. C-J. Tseng and D.P.Siewiorek,"Automated Synthesis of DataPaths in Digital Systems", IEEE Trans. Computer-Aided Design, Vol. CAD-5, No.3, pp. 379-395, July 1986.
2. P. Marwedel,"The MIMOLA Design System: Tools for the Design of Digital Processors", Proc. DAC-21,pp.378-384,1984.
3. P.Marwedel,"The MIMOLA Design System: Tools for the Design of Digital Processors", Proc. DAC-23,1986,pp.587-593.
4. P.G.Paulin,J.P.Knight and G.E.Girczyc, "HAL: A Multi- Paradigm Approach to Automatic Data Path Synthesis", Proc. DAC-23,1986, pp.267-277.
5. P.G.Paulin and J.P. Knight, " Force-Directed Scheduling in Automated Data Path Synthesis", Proc. DAC-24, 1987, 195-202.
6. P. Pfahler, "Automated Data Path Synthesis : A Compilation Approach", Euromicro Journal of Microprocessing and Microprogramming, Vol.21, 1987, pp.577-584.
7. A.C.Parker,J.T.Pizarro and M.Mlinar, "MAHA: A Program for Data Path Synthesis", Proc. DAC-23, 1986, pp.461-466.
8. B.M.Pangle, "Splicer: A Heuristic Approach to Connectivity Binding", Proc. DAC-25, 1988, pp.536-541.
9. Z. Peng, "A Formal methodology for Automated Synthesis of VLSI Systems", Ph.D. Dissertation, Linkoping University, Linkoping, Sweden, 1987.
10. J.J.Granadi and A.C.Parker, "The Effect of Register-Transfer Design Tradeoffs on Chip-Area and Performance", Proc. DAC-20, pp. 419-424, 1983.
11. R.Jain,M.J.Mlinar and A.C.Parker, "Area-Time Model for Synthesis of Non-Pipelined Designs", ICCAD-88,Nov.1988.
12. M.C.McFarland, "Reevaluating the Design Space for Register-Transfer Hardware Synthesis", ICCAD-87, 1987.
13. R.Jain, A.C.Parker and N.Park, "Module Selection for Pipelined Synthesis" Proc., DAC-25, 1988, pp.542-547.
14. M.Balakrishnan et al., "Allocation of Multi-port Memories in Data Path Synthesis", IEEE Trans. Computer-Aided Design, Vol. 7, No. 4, April 1988, pp. 536-547.
15. N.Park and A.C.Parker, "SEHWA: A Program for Synthesis of Pipelines", Proc. DAC-23, 1986, pp.454-460.
16. R.Johnk and P.Marwedel, MIMOLA Reference Manual, Version 3.5,Institut fur Informatik, Univ. of Kiel, Kiel, W.Germany.