RT-LEVEL SYNTHESIS BASED ON INTEGRATED SCHEDULING AND BINDING

M. Balakrishnan*

Bericht Nr. 8813

December 1988

Institut für Informatik

der Universität Kiel

Olshausenstr. 40-60

D-2300, Kiel

W. Germany

INDEX

List of Figures

List of Tables

Acknowledgements

First of all I would like to thank Dr. Peter Marwedel for making it possible for me to visit Kiel. The research was supported by a contract (NT 2850 6) from The German Ministry of Research and Technology (BMFT). I have very pleasant memories from my seven months stay in Kiel and that is primarily due to the efforts of Peter, Reinhard and their families. I sincerely thank them for all the technical and personal help they provided.

I acknowledge Mr. E.Lerch for interfacing the synthesis system to MIMOLA. The interface as reported in Appendix B is primarily his work. I acknowledge the timely help of other group members especially Lothar, Wolfgang, Detlef and Jürgen.

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 report 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 presynthesised structures.

Specifically, the report deals with the tasks of scheduling, binding operations to operators and intermediate values to storage units. Further, the generated structure is optimized by examining the feasibility of merging storage units to form memories. All the optimization tasks are modelled as constrained 0-1 integer programming problems with the objective of reducing interconnections.

Keywords

Zero-one **integer programming,** Design automation, Data path, Areatime tradeoff, Design space, Data flow graph, Scheduling, As soon as possible scheduling,

1. INTRODUCTION

1.1 A Brief Survey

Synthesis, in the context of design automation of digital systems, usually refers to the process of transforming a behavioural 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 an different parameters are to be satisfied for realizing the synthesised design, maximum emphasis is placed an the delay time (execution time of the behavioural description by the synthesised 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. Traditional synthesis approaches perform these tasks in a sequence (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

i) Storage allocation and binding,
ii) Operator allocation and binding and
iii) Interconnection allocation and binding

Another example is MIMOLA [2,3] which starts by generating a maximum parallel schedule (as late as possible) by analysing 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 clocktime is determined by the maximum propagation delay path. The interaction between the different tasks is limited.

Synthesis in many other systems follows a similar approach with some variations. There have been various attempts to perform the tasks together (global optimization), iteratively or as an expert system. HAI. [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 an 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 an a Petri-net model for design representation. The synthesis is carried out by a sequence of semantic preserving transformations an 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 predicting the area-time **curves** based an operator utilization. The approach presently makes a number of simplifying assumptions an 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 an a smooth curve.

A number of synthesis systems have been proposed which deal with a specific application area or a specific target architecture. Parker and Park [13-15] have reported extensively an designing pipelined systems. De Man et al. [16] describe a CAD system with a synthesis interface for signal processing applications. Kowalski[17] presents a knowledge based approach for processor design. The present state of the synthesis systems is summed up in two papers presented at the 25th Design automation conference [18-19].

1.2 objectives of Our Approach

The synthesis approach presented in this report 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 multi-function operators. On the other hand, nobody tackles the designs with a mix of different speed operators for the same operation. Jain et al.[14] 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-synthesised structure may need to be integrated efficiently i.e. to use the existing resources as much as possible for realising the behavioural description while synthesising only the extra struc

tures 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 an 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 Chapter 6) .

1.3 Summary

The report is organised into six chapters and two appendices. Chapter 2 presents a brief overview of the synthesis system along with its salient features and limitations. Chapter 3 discusses the scheduling approach and the options available. Optimizations needed at different steps are modelled as zero-one integer programming. The zero-one integer programming model, the binding of operations to operators and the binding of values to storage elements are discussed in Chapter 4. The discrete storage locations can be merged to form multi-port memories as presented in Chapter 5. A simple running example in Chapters 3, 4 and 5 illustrates the approach while a more comprehensive example is presented in Appendix A. The results are discussed in Chapter 6. The extension of the synthesis approach by integration of a floor-planner along with other proposed future work is also discussed in Chapter 6. The approach is being implemented as a tool within the MIMOLA system. The present state of the interface along with its integration to the MIMOLA software system is discussed in Appendix B.

z. SYSTEM OVERVIEW

2.1 A Brief Description

Figure 1 shows an overview of the synthesis system. 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. The inputs are

a) Program Data Flow Graph : The data dependency of the operations in the source program is analysed to generate a data flow graph.

b) Module Library : The definition of the resources needed to synthesize the structure constitutes the module library. At present it only includes the definition of operators, registers and memories declared in the source.

c) Allocated Resources : The module library elements declared as parts by the designer are instantiated as preallocated resources. Apart from the elements of the module library, the predeclared interconnections also constitutes this set.

For the present state of the input interface and the MIMOLA constructs supported by the implementation, refer to Appendix B.

A set of operations called candidate operations are prepared based an 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 an the current bindings. Once all operations are scheduled and bound, multi-port memory synthesis [21] is attempted for allocated memories. First the maximum number of registers,

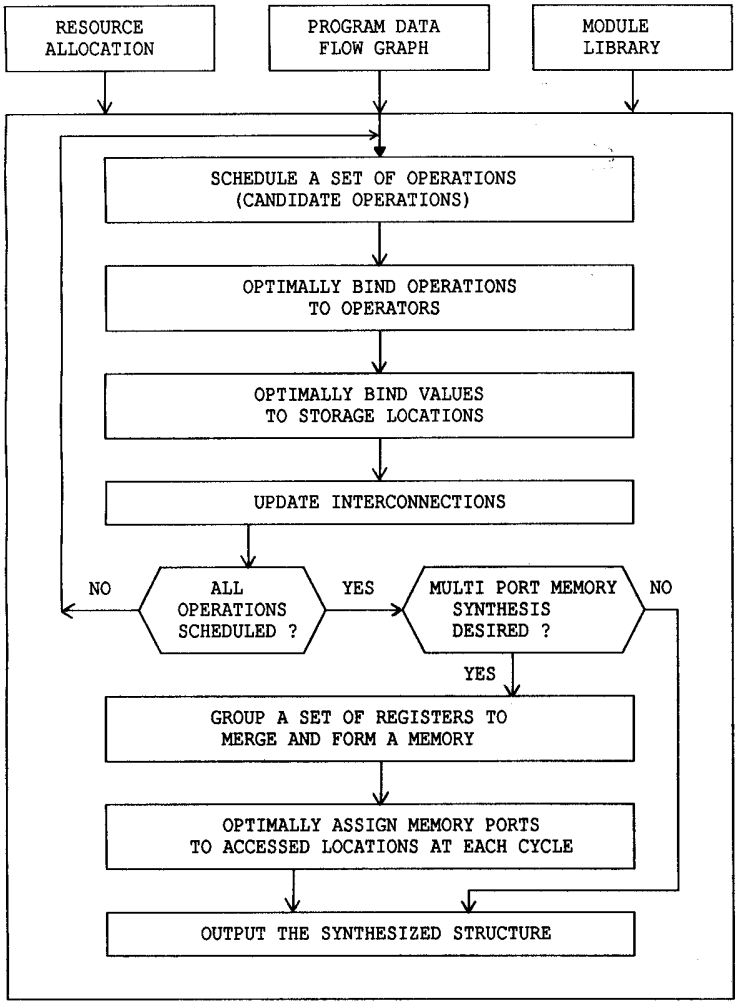

FIGURE 1. Overview of the Synthesis Process

within the specified memory size and satisfying the simultaneous access constraints of the memory ports are grouped to form the memory. This is followed by assigning ports to the accessed locations for each cycle so that 'minimal' extra interconnections are generated.

Before we take up the details in the next chapters, we describe the key features and some constraints of our synthesis system. Some of the constraints are related only to the present implementation and Chapter 6 an future work describes the proposed enhancements.

2.2 Key Features

a) A mix of operator modules implementing the same operation at different speeds can be allocated together e.g. the synthesis program could schedule operations an a fast and slow multiplier simultaneously. This feature allows us to explore the design space more extensively.

b) At every synthesis step extra interconnections are generated only if no binding of operations and values is feasible with the 'present' structure. Thus, the algorithm is inherently suitable for incremental synthesis. It is possible to specify a partial structure (i.e. predeclared parts with some interconnections) and utilise it effectively for further synthesis.

c) All 'optimizations' are mapped to a 'Zero-one' Integer programming model. The Pascal version of a standard algorithm[21] is used for four different tasks

- Binding scheduled operations to available operators
- Binding generated values to available registers
- Identifying registers which can be grouped into a memory
- Assigning ports to accessed locations for each cycle

2.3 Some Constraints

a) All operators have an unique associated delay time. Thus a multi-function operator is assumed to implement all functions/ operations with the same delay time.

b) Designers can specify the resources or let them be instantiated at an associated cost during synthesis. This option is necessary for treating all operators uniformly as low cost operators (like gates) are more likely to be allocated afresh instead of being shared. This is especially true if sharing involves creating additional paths and multiplexers. Presently, however all resources(operators and storage elements) need to be preallocated to generate a point in the design-space.

c) All the operations included in the set of candidate operations are scheduled. Thus, the candidate set is so generated that it is feasible to schedule all the operations in the set. Refer to Chapter 6 for a detailed discussion of the changes required and for the effect of removing this constraint .

3. SCHEDULING

3.1 Scheduling Modes

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

a) Forward scheduling (FS) : This is based an scheduling operations as soon as their source operands are available.

b) Backward scheduling (BS) : This involves scheduling operations based an as late as possible technique.

c) Double headed scheduling (DHS) : In this mode scheduling is done by alternating between forward and backward scheduling. As long as the operations being scheduled lie an different paths, scheduling is continued in forward (backward) mode. A switch from forward (backward) to backward (forward) is performed so that no two operations lying an any one path are scheduled between two mode changes.

3.2 Selection of Candidate Operation

The set of candidate operations are chosen based an 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. In the present implementation, the selection of operations to the candidate set is performed by choosing at most one

operation from the data available set for each available operator. Contention among operations is resolved by choosing the one with the longer unscheduled delay path. In case of DHS, this involves keeping track of operations scheduled from either direction for all the paths. This is simplified by creating a temporary structure from the dfg called the path graph. The path graph contains a linked list of operations for each of the paths in the dfg. This is created by traversing all the paths (inputs to outputs) in the dfg. Thus each dfg operation has as many entries in the path graph as the paths it lies on. The scheduling is performed by using the path graph. This facilitates computation of unscheduled maximum delay path for each operation. Further, a flag associated with each path signals the need to change the scheduling direction in case of DHS.

In case of operators implementing the same operation with significantly different delay times,it is possible that an operation lying on a long delay path gets scheduled on a rather slow operator with undesirable effects. This is prevented by marking an operator as available only after its delay-time. Thus, the choice of the most eligible operation (i.e. longest unscheduled delay path) is made at the time the operation would finish rather than begin. We illustrate the same with a small example.

Assume the available operation set consists of two elements x and y. They are characterised as x = ('@',28) and y = ('@',27). The first symbol denotes the operation to be performed and the second element its associated maximum unscheduled delay path. Consider, there are two operators A and B capable of performing the operation '@' and with associated delay times $d_A = 2$ and $d_B = 5$. Including both A and B as available at T = 0 produces the partial schedule shown in Figure 2(a). On the other hand to generate the schedule of Figure 2(b), operator A becomes available at time T = 2 to schedule x at T = 0 and again becomes available at T = 4 to schedule y at T = 2. The slow operator B is declared available only at time T = 5. The only condition under which y can be scheduled on B is if there is a contending available operation for A with a longer unscheduled delay path at time T = 4.

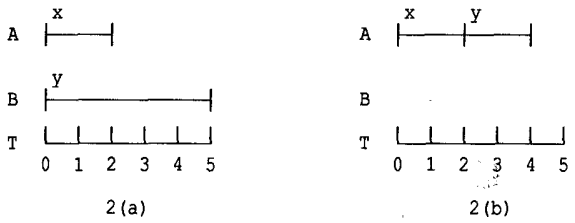

FIGURE z. Example Illustrating Scheduling Options

3.3 **Scheduling Example**

The example dfg in Figure 3 is used to illustrate scheduling. The dfg is taken from Paulin's paper an force directed scheduling (4) .

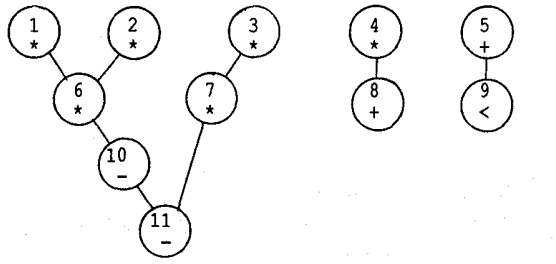

FIGURE 3. Example Data Flow Graph[4]

The module library assumed for the following discussion consists of four Operator types

MF : Fast Multiplier, delay time = 2, Operation = ('*')

MS : Slow Multiplier, delay time = 4, Operation = ('*')

AF : Fast Accumulator, delay time = 1, operations = ('+','-','<')

AS : Slow Accumulator, delay time = 2, operations = ('+','-','<')

The result of scheduling the dfg of figure 3 with an allocated Operator set of {1MF, 1MS, 1AF} is shown in Figure 4. Table 1 lists the results of scheduling an different Operator sets. The last column lists the results from [4].

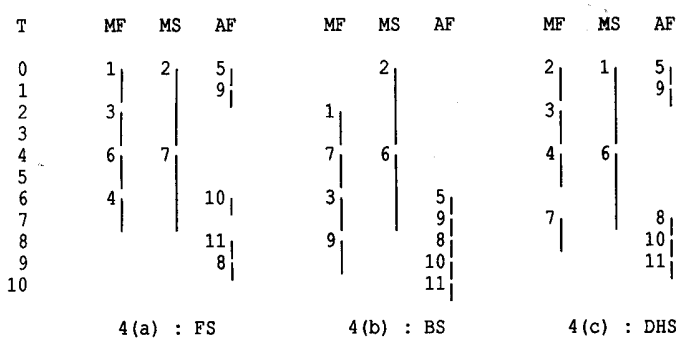

FIGURE 4. Scheduling Results with Operator Set {1MF, 1MS, 1AF}

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 AF1	8	8	8	8-12
{ 2 MF, 2 AF}	7	7	7	7
{ 3 MF, 1 AF1	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 3)

4. BINDING OPERATIONS AND VALUES

4.1 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 $\{b_j\}, j=1,..,m$.

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

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 tost 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.

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

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

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

4.2 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 4.1

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.

4.3 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 4.1

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.

4.4 Discussion and Example

The binding of operations to operators as well as values to storage units is performed with a view to minimize extra interconnections on an existing structure. The results in this section are based on the following assumptions :

a) All values generated by the operations are to be latched.
b) The latches for storing these values are trailing edge triggered. i.e. a fresh value can be latched in the last 'use cycle' of a currently latched value.
c) The interconnections required for routing the inputs and constants are not considered.

Table 2 presents the result of scheduling and binding the dfg of figure 3 on a operator set {2MF, 2AF} with Forward scheduling. The table lists the binding decisions taken in each cycle along with the number of existing interconnections used and new interconnections generated. The value v_i is the result of operation p_i.

Table 3 presents the results of scheduling and binding with different operator sets. 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. Though, the schedule time is almost the same for the three modes (except in one case), the designs produced are different. In some cases it is easy to choose but others would require an actual area estimate from a floor-planner.

TIME ⟶	0	1	2	3	4	5	6
Operator Binding							
MF1	p1		p6		p7		
MF2	p2		p3		p4		
AF1							p8
AF2	p5	p9			p10		p11
Storage Binding							
R1	v5	v9			v10		v11
R2		v1		v6		v7	v8
R3		v2		v3		v4	
Interconnections							
Existing Used	0	1	0	2	2	2	3
New Created	1	3	2	0	1	0	2

TABLE 2 : Results from Scheduling and Binding the dfg in Figu
Operator Set : {2MF, 2AF) and Scheduling Mode : FS

OPERATOR SET	SCHED. OPT.	SCHED. TIME	INTER-CONNECTS	MUX INPUTS	STORAGE LOCATIONS
{1 MF,1 AF1	DHS	13	12	6	5
	FORW.	13	9	9	3
	BACK.	13	13	6	6
(1MF,1MS,1AF}	DHS	10	13	6	5
	FORW.	10	11	6	3
	BACK.	11	12	6	5
{1MF,2MS,1AF}	DHS	9	11	4	5
	FORW.	9	10	2	4
	BACK.	9	12	6	5
{2 MF,1 AF)	DHS	B	12	4	4
	FORW.	8	8	2	3
	BACK.	8	14	6	6
{2 MF,2 AF)	DHS	7	11	2	4
	FORW.	7	9	2	3
	BACK.	7	10	0	9
{3MF,1AF,1AS}	DHS	6	12	2	5
	FORW.	6	12	4	4
	BACK.	6	13	4	5

TABLE 3 : Results from Scheduling and Binding the dfg of Figure 3

5. MULTI-PORT MEMORY SYNTHESIS*

5.1 Memory Synthesis Problem

The approach presented till now produces designs with isolated discrete storage units. We refer to these storage units as registers in this section while remembering that the term is used in a general context i.e. they Gould be registers or latches. On the other hand, merging these registers to form memories has some potential advantages. The generated design is likely to be more Compact due to reduction in number of interconnections, multiplexers and multiplexer Inputs. This results from sharing of interconnections present between Operators and isolated registers. Further, the generated design is better 'testable' or at least the area overhead for testing is expected to be smaller. Of course, we do make an assumption that additional area required for address paths and address decoders are Small compared to the reduction from shared data paths.

The general strategy is to consider multi-port memories because Single port memories cannot satisfy the simultaneous access requirements even for a few merged registers. The present implementation Supports **four types of specialised memory ports**

i) **Read** only ports : rd
ii) Write only ports : wr
iii) Read or write ports : row
iv) Read and write ports : raw

A point about the 'raw' type ports needs to be clarified. We assume such a port can read and write at an address specified by the Port address in a Single control step/cycle. Further, the write is

*

The discussion in this chapter is based an work reported elsewhere [20].

performed at the end of the control step. This is to support merging of registers with a similar characteristic i.e. read during the cycle and Write at the trailing edge. Only if such a port is present, the registers read and written in the saure cycle would be considered for merging.

5.2 Approach

All the declared memory units are considered as potential units for replacing registers which can be merged into them. In the present implementation one memory at a time is tried and the corresponding structure generated. It is intended to augment it with a strategy for automatic selection of 'best' design. Another possibility in case of designs with a large number of registers, is to repeat the process after removing the merged registers.

The process itself is performed in two steps

i) a set of registers is identified which can be merged into a specific memory unit.

ü) a port assignment is made for every accessed location.

The grouping of registers (step i) is performed globally whereas the assignment of ports to accessed locations (step ü) is performed considering one control step at a time.

5.3 Grouping Registers

The problem of grouping registers can be defined as follows
For an allocated memory,identify a maximal set of registers (maximal is in terms of number of elements) whose access requirements at any control step can be satisfied by the availability of ports in the memory.

It is easy to visualise the modelling of the above problem as a 0-1 integer programming problem.

We define :

x_i : equal to 1 if register r_i is included in memory M
: zero-one integer variable which defines the grouping

w_i : benefit of grouping a_i in M
: computed only if the grouping is feasible.

The function to be maximized is : $\sum_{i=1}^{N} w_i x_i$

For the simple model presently being employed, the weight is taken to be 1 for all registers. A better heuristic is to Base the weight (at least partially) an the number of connections to the register. It is more advantageous to group registers with larger number of interconnections as the potential for sharing paths and thus area reduction is larger.

The constraints originate from the simaultaneous access requirement of registers. For access at each control step, upto four constraints can be defined.

i) Number of locations being read should not exceed the number of read ports (i.e. ports of type rd/row/raw).

ü) Number of locations being written should not exceed the number of write Ports (i.e. Ports of type wr/row/raw).

iii) Number of locations being read/written should not exceed the number of read/write Ports (i.e. Ports of type rd/wr/row/raw).

iv) Number of locations being 'read and written' should not exceed the number of 'read and write' Ports (i.e. Ports of type raw).

One global constraint results from the size of the memory i.e. no more than m registers can be grouped in a m word memory. Though, the number of constraints appear large (4k + 1 for a k step schedule)

but most of them are trivially satisfied and are removed by a prunner. The above Problem is also mapped to the minimization Problem of Section 4.1 and solved.

The supply of source operands from memory units through Ports introduces a major complication. As a source Operand supplied from a Port at the start of a multi-cycle Operation has to be maintained at the saure Port throughout the execution of the Operation, Port availability at different steps cannot be considered completely independently.

5.4 Assigning Ports

The Problem of assigning Ports to accessed locations globally, while minimizing the number of interconnections, leads to a formulation with a large number of variables [20]. Thus the Problem is broken down and the Ports are assigned to accessed locations by considering one control step at a time. Further, the optimization criteria is modified to minimizing extra interconnections required over and above the existing structure.

The problem at each control step can be defined as the location-port binding problem. The set of accessed locations $\{r_i\}$ is to be mapped to the set of free ports $\{t_i\}$.

R-T BINDING: $\{ r_i \}$ ------> $\{ t_j \}$

We define the related terms to use the model described in Section 4.1

x_{ij} feasible: If t_j is a free port of the type suitable to assign r_i based on the type of access required for r_i in the current control step. A port is not free if it is bound by a previous assignment to a location which is a source for an unfinished operation.

For computing the weight w_{ij} only conditions 1 and 3 are relevant.

ALL CASES: corresponds to examining all operations $\{p_k\}$ which use r_i in the current control step.

COND1 : t_j is not connected to the appropriate port of the module to which p_k is bound.

COND2 : not relevant

COND3 : t_j is connected to the appropriate port of the module to which p_k 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 locations to ports while generating the memory-port/ Operator interconnections required. Further, the solution of this integer programming problem is also required for control Synthesis as the memory addresses for all ports in each control step are assumed to be supplied by a Controller.

5.5 Example

The dfg of Figure 3 is again used as the example. The effect of multi-port Synthesis is presented in Table 4 and Figures 5 and 6.

Table 4 contains an overview of the result produced by allocating memory modules differing in number of ports and port definitions. The Input for memory Synthesis was the structure and schedule produced by allocating an Operator set of {1MF, 1AF} with DHS as the scheduling mode. (Refer to row 1 of Table 3 in Section 4.9). The set of registers that can be merged into the memory (col 2) along with the interconnections saved (col 3) give an indication of the benefits of allocating a particular memory module.

Figure 5 is essentially included for comparison with Figure 6 and is the structure before memory synthesis. The case presented again corresponds to an Operator set allocation of {1MF,1AF} with DHS scheduling. The Output of memory synthesis is shown in Figure 6 and corresponds to row 4 of Table 4. Along with the reduction in the number of interconnections, one multiplexer has,, also been removed. For a more complex example refer to Appendix A.

MEMORY PORTS	REGISTER SET	INTERCONNECTS SAVED
M1 = (1 row)	{R2,R4}	1
M2 = {1 raw)	{R3,R4}	2
M3 = {1 rd,1 raw)	{R2, R5}	2
M4 = {1 row,1 raw)	(R2, R4, R5)	2
M5 = {1 row,2 raw}	{R2, R3, R4, R5}	4

TABLE 4 : Results from Memory Synthesis
Operator Set : [IMF, 1AF} and Scheduling Mode : DHS

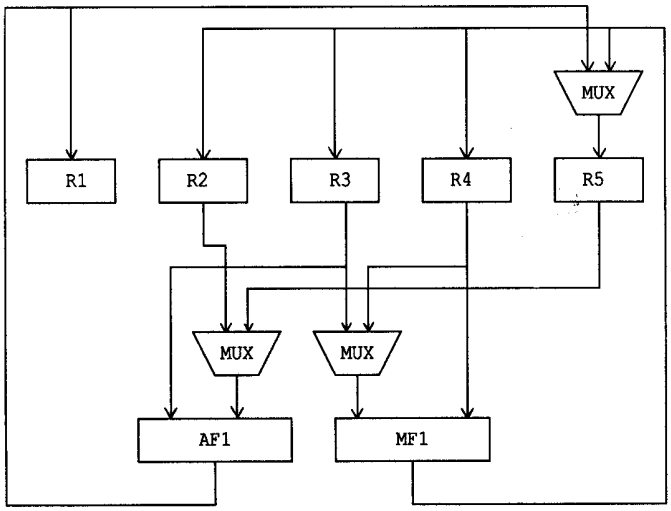

FIGURE 5 Structure Before Memory Synthesis
Operator Set {1MF,1AF}, Scheduling Mode DHS, Connects 12, Mux 3* 2:1

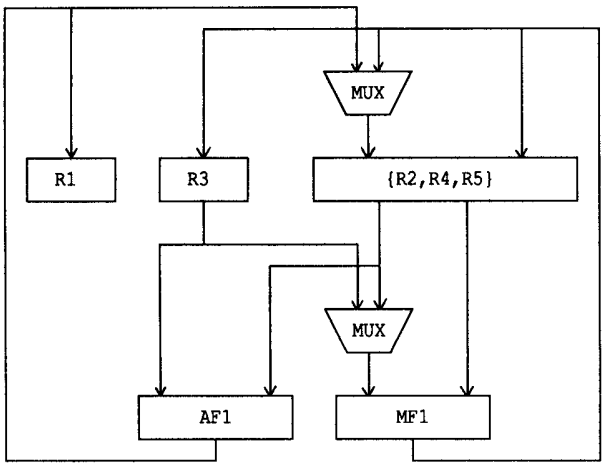

FIGURE 6 Structure After Memory Synthesis; Memory with {1raw,1row}
Operator Set {1MF,1AF}, Scheduling Mode DHS, connects 10, Mux 2* 2:1

6. CONCLUSION

6.1 Experimental Results

All the results in this section are based an data flow graphs available in the literature. The module Set for these examples are the saure as the one Chosen for the example in Chapters 3,4 and 5. The cpu time refers to the time an a DN 4000 APOLLO Workstation with 4 MB memory.

The second example we present is the digital filter discussed in SEHWA[15]. The accumulators in the module Set can be replaced by adders for this example. The results from our Synthesis are presented in Table 5. 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 an the APOLLO Workstation ranged between 1 and 5 seconds for all Gases.

Another 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 saure as used in the last example. Table 6 presents the Best result(s) for each of the eight Operator Sets. In two Gases more than one choice is presented as the one with the lower schedule time appeared to haue higher path cost. The cpu time an the APOLLO Workstation for all Gases presented below ranged from 2 to 8 seconds.

OPERATOR SET	SCHED. OPT.	SCHED. TIME	INTER-CONNECTS	MUX INPUTS	STORAGE LOCATIONS
	DHS	18	18	15	6
11 MF,1 AF)	FS	18	11	4	5
	BS	18	19	11	8
	DHS	15	15	5	6
{2 MF,1 AF)	FS	15	11	6	4
	BS	15	18	8	8
	DHS	12	30	23	8
{1MF,2MS,3AF}	FS	12	24	14	B
	BS	12	21	15	6
	DHS	il	23	15	7
{2 MF,2 AF}	FS	11	19	11	7
	BS	11	12	6	4
	DHS	10	24	15	6
{3 MF,2 AF}	FS	10	21	12	6
	BS	10	13	6	9

TABLE 5 : Results from Scheduling and Einding the FIR Filter [15)

OPERATOR SET	SCHED. *OPT.*	SCHED. ***TIME***	INTER-CONNECTS	MUX INPUTS	STORAGE LOCATIONS
(3 MF,3 AF)	FS	17	38	32	8
{2 MF,3 AN	FS	18	35	30	7
{2 MF,2 AF)	FS	19	33	30	8
	BS	18	37	27	11
{1MF,1MS,2AF}	FS	20	28	24	6
{1 MF, 1 MS, 1 AF,1 AS}	DHS	22	35	26	10
{1 MF,2 AF)	FS	22	28	26	6
{1MF,1AF,1AS}	FS	23	33	30	8
{1 MF,1 AF}	DHS	28	23	17	11
	BS	29	26	17	10

TABLE 6 : Results from Scheduling and Binding the Elliptic Filter[5]

Table 7 presents the results from allocating a two port memory for four different Operator allocations representing a range of time schedules.Columns 4 and 5 list the number of interconnections before and after memory synthesis while column 7 specifies the number of locations that can be merged. Again the cpu time was of the Order of 6 to 10 seconds.

OPERATOR SET	SCHED. OPT.	SCHED. TIME	CONNECTS		LOCATIONS	
			BEFORE	AFTER	TOTAL	MEMORY
12 MF,2 AF}	DHS	19	40	35	10	4
	FS	19	33	34	8	3
	BS	18	37	34	11	4
{1MF,1MS,2AF}	DHS	21	35	33	10	4
	FS	20	28	26	6	3
	BS	21	38	31	11	9
{1MF,1AF,1AS}	DHS	23	34	28	11	5
	FS	23	33	28	8	4
	BS	23	39	27	10	5
11 MF,1 AF}	DHS	28	23	17	11	4
	FS	28	33	22	10	6
	BS	29	26	19	10	6

TABLE 7 : Results from Memory Synthesis for the Elliptic Filter Data Flow Graph; Allocated Memory Ports {1 **raw,1 row)**

6.2 Analysis of Results

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 an the APOLLO workstation for all Gases.

Results from scheduling the FIR filter (Table 5) show hardly any difference in schedule time for different modes. In fact experiments an 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 an Small flow graphs. On the other hand, the effect of scheduling Option an area Parameters was large and varied. First two sets in Table 5 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 represent abrupt area changes against regular changes in schedule time. This seems to confirm the Observation by McFarland [12] that design points an a area-time graph do not lie an a smooth curve.

The results of scheduling are easily comparable to those reported by Paulin et al. using force-directed scheduling[5]. Table 6 provides one basis for comparison. Our Best Gase schedule time always matched that produced by force-directed scheduling and at least for one Gase (Set 3 Backward Option) it was better. They haue reported that their schedule times are optimal except for one Gase (not specified). Further, some results produced by a mix of Operators seem to be very attractive design points, e.g. Set 4 in Table 6.

The Table 7 Shows the effect of memory Synthesis. on an average, the interconnections saved as well as the locations that can be merged decrease with increase in the number of allocated Operators. The increase in number of Operators represent less potential for sharing interconnections by merging registers. This is similar to the effect an forming buses. Increase in number of Operators also result in an increase in the frequency of register accesses and thus reduce the number of registers that can be merged. Though, these trends are important, but more important is to generate and pick out exceptional Gases. For example, in set 4, the Forward scheduling

produced a rather poor alternative with 10 registers and 33 interconnects. But, after memory synthesis it gets transformed into a good alternative with a reduction of 11 interconnects by merging 6 registers to form a two port memory.

6.3 Future Work

As mentioned in the introduction, the present system provides a framework for a Synthesis based an area-time tradeoff. The key to this is the adjustment of optimization weights which govern the binding decisions based an 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 an the available Operators. But once an area estimate for binding an Operation is obtainable from a floorplanner, the schedule-bind decision would be based an weighting the expected 'extra' area against the expected 'extra' time. The extra area can be estimated from the need to instantiate extra Operators as well as interconnections. The 'extra' time of not binding an Operation can be estimated by comparing the unscheduled portion of the longest data flow path associated with a candidate Operation with other candidate operations. 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.

The incorporation of multi-port memory Synthesis as a postprocessor provides a further means for area optimization. This also means an explosion in the number of design points if the module library contains a **large** variety of memories. A strategy to automatically select a few 'good' designs would be highly desirable. The scope for forming Buses is rather limited after memory Synthesis but it is possible to merge interconnections to form Buses based an the Same model as memory Synthesis as indicated in [20].

Presently, the System accepts only a Subset of MIMOLA[16] for program Input specification and hardware module declaration. We are working to extend it so that a complete Synthesis including microcode generation Gould be performed under the MIMOLA environment. The major improvements required are in handling the interconnections to input/output ports and to modules storing constants. For a more detailed specification of the current Status of the implemetation refer to Appendix B.

6.4 Concluding Remarks

This report presents a Synthesis tool based an integrating scheduling with binding decisions. The two binding decisions considered are: operations to Operators and values to storage locations. The designs generated can be further optimized by merging registers. 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-synthesised structures. Further, it is indicated how the system Gould be extended for taking schedulebind decisions based an 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. The System is being integrated into an existing CAD environment and thus would provide an alternative to the present vertical synthesis strategy.

REFERENCES

1. C-J. Tseng and D.P.Siewiorek,"Automated Synthesis of Data Paths in Digital Systems", IEEE Trans. Computer-Aided Design, Vol. CAD-5, No.3, pp. 379-395, July 1986.

z. P. Marwedel,"The MIMOLA Design System: Tools for the Design of Digital Processors", DAC-21,pp.378-384,1984.

3. P.Marwedel,"A New Synthesis Algorithm for the MIMOLA Software System", Proc. DAC-23,1986, pp.271-277.

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 Auto mated 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, "MAHR: A Program for DataPath Synthesis", Proc. DAC-23, 1986, pp.461-966.

B. 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, Linköping University, Linköping, Sweden, 1987.

10. J.J.Granadi and A.C.Parker, "The Effect of Register-Transfer Design Tradeoffs an Chip-Area and Performance", 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", Preprint, To appear in ICCAD-88, Nov.1988.

12. M.C.MCFarland, "Reevaluating the Design Space for RegisterTransfer Hardware Synthesis", ICCAD-87, 1987.

13. N.Park and A.C.Parker, "SEHWA: A Program for Synthesis of Pipelines", Proc. DAC-23, 1986, pp.454-460.

14. R.Jain, A.C.Parker and N.Park, "Predicting Area-Time Tradeoffs for Pipelined Design" Proc., DAC-24, 1987, pp.35-41.

15. R.Jain, A.C.Parker and N.Park, "Module Selection for Pipelined Synthesis" Proc., DAC-25, 1988, pp.542-547.

16. H.De Man et al.,"CATHEDRAL - II - a Computer-aided Synthesis system for Digital Signal Processing VLSI Systems", ComputerAided Engg. Journal, April 1988, pp. 55-66.

17. T.J.Kowalski, An Artificial Intelligence Approach to VLSI Design, Kluwer Academic Publishers, 1985.

18. G.Borriello and E.Detjens, "High-level Synthesis : Current Status and Future Directions", DAC-25, 1988, p477-482.

19. M.C.McFarland, A.C.Parker and R.Camposano, "Tutorial an Highlevel Synthesis", DAC-25, 1988, p330-336.

20. 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.

21. J.L.Byrne and L.G.Proll, "Solution of linear programs in 0-1 variables by implicit enumeration algorithm 341", Commun. Ass. Comp. Mach., Vol. 11, no. 11, p.782, Nov. 1968.

22. R.Jöhnk and P.Marwedel, MIMOLA Reference Manual, Version 3.45, Institut für Informatik, Univ. of Kiel, Kiel, W.Germany.

APPENDIX A : A comprehensive Example

We illustrate the methodology presented in this report with a complete example. The dataflow graph shown in figure A.1 corresponds to the Elliptic filter referred to in [5]. The definition and allocation part of the MIMOLA description is shown in figure A.2. Theo Operator set (1 MF, 1 MS, 2 AF) is allocated and forward scheduling mode is used. The generated schedule is shown in figure A.3 and requires 20 time units. Figure A.4 shows the schematic with 28 interconnects between 6 storage units and the allocated Operators.

Multi-port memory synthesis is illustrated with an allocation of (1 MF, 1 AF) as the Operator set and forward scheduling. Figure A.5 shows the schematic before the memory synthesis; 33 interconnects with 10 locations. Six of these locations can be merged to form a two port (1 row, 1 raw) memory reducing the interconnects to 22. The final schematic is shown in figure A.6.

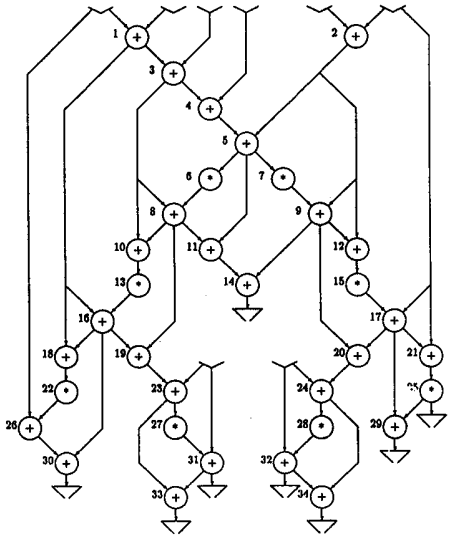

Figure A.1 Dataflow Graph : Elliptic Filter [5]

```
MODULE Mul
 (IN I1, I2: WORD; OUT O: WORD);
 BEHAVIOUR Fast OF Mul IS
  ATTRIBUTE COST IS 40;
  CONBEGIN
   O <- I1 * I2 AFTER 2;
  CONEND;
 BEHAVIOUR Slow OF Mul IS
  ATTRIBUTE COST IS 10;
  CONBEGIN
   O <- I1 * I2 AFTER 4;
  CONEND;

MODULE Alu
 (IN I1, I2: WORD; FCT F: BIT; OUT O: WORD);
 BEHAVIOUR Fast OF Alu IS
  ATTRIBUTE COST IS 7;
  CONBEGIN
   CASE F OF
   CASE F OF
    %0 : O <- I1 + I2 AFTER 1;
    %1 : O <- I1 - I2 AFTER 1;
   END;
  CONEND;

MODULE Mem8Two
 PORT Port1
  (IN Wr1: WORD; OUT Rd1: WORD; ADR ADDRESS: (2:0); FCT f: BIT; CLK C: BIT);
 PORT Port2
  (IN Wr2: WORD; OUT Rd2: WORD; ADR ADDRESS: (2:0); FCT f: BIT; CLK C: BIT);
BEHAVIOUR Simple OF Mem8Two IS
 VAR
  Cells : ARRAY[0..7] OF WORD;
 CONBEGIN
  WITH Port1 DO
   CONBEGIN
    AT C DO
     CASE f OF
      1 : Cells [ADDRESS] := Wr1 AFTER 1;
      0 : Rd1 <- Cells [ADDRESS] AFTER 1;
     END;
   CONEND;
  WITH port2 DO
   CONBEGIN
    Rd2 <- Cells [ADDRESS] AFTER 1;
    AT C DO
     CASE f OF
      1 : Cells [ADDRESS] := Wr2 AFTER 1;
      0 : NOLOAD AFTER 0;
     END;
   CONEND;
 CONEND;

PARTSET ForSynthesis IS
 BEGIN
  AF1,
  AF2  : MODULE Alu BEHAVIOUR Fast;
  MS1  : MODULE Mul BEHAVIOUR Slow;
  MF1  : MODULE Mul BEHAVIOUR Fast;
  M821 : MODULE Mem8Two BEHAVIOUR Simple;
 END;
```

Figure A.2 Example MIMOLA Hardware Desription (Version 4.0)

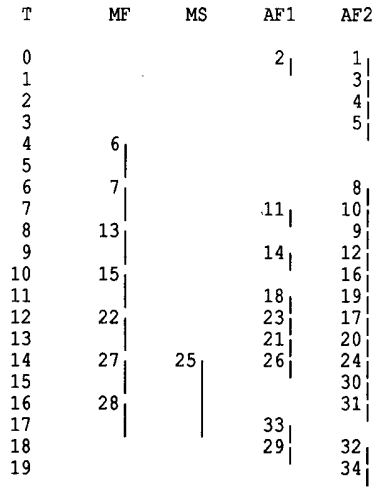

Figure A.3 Schedule with Operator Set (1 MF, 1 MS, 2 AF}

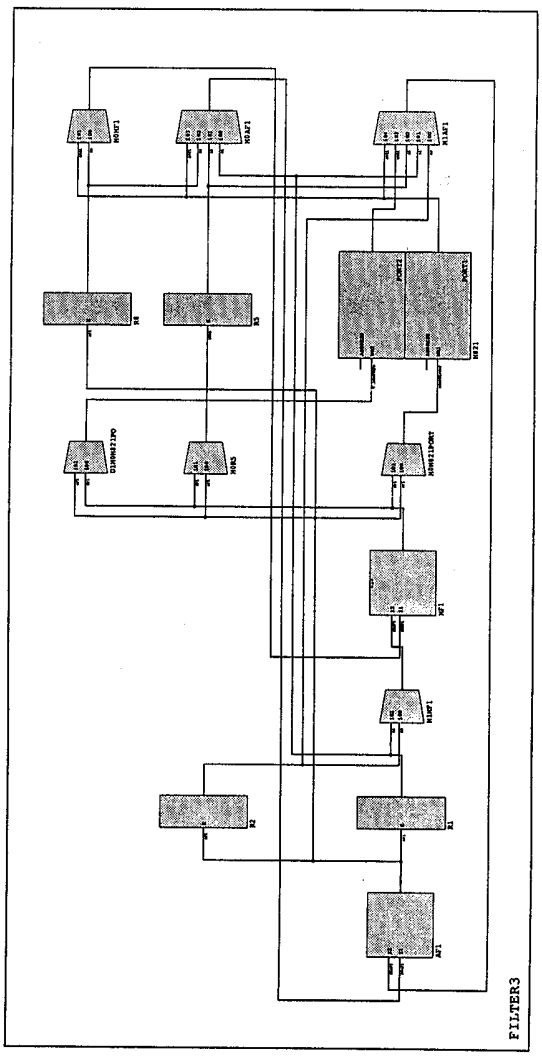

Figure A.4 Synthesized Data Path with Operator Set
{1 MF, 1 MS, 2 AF}

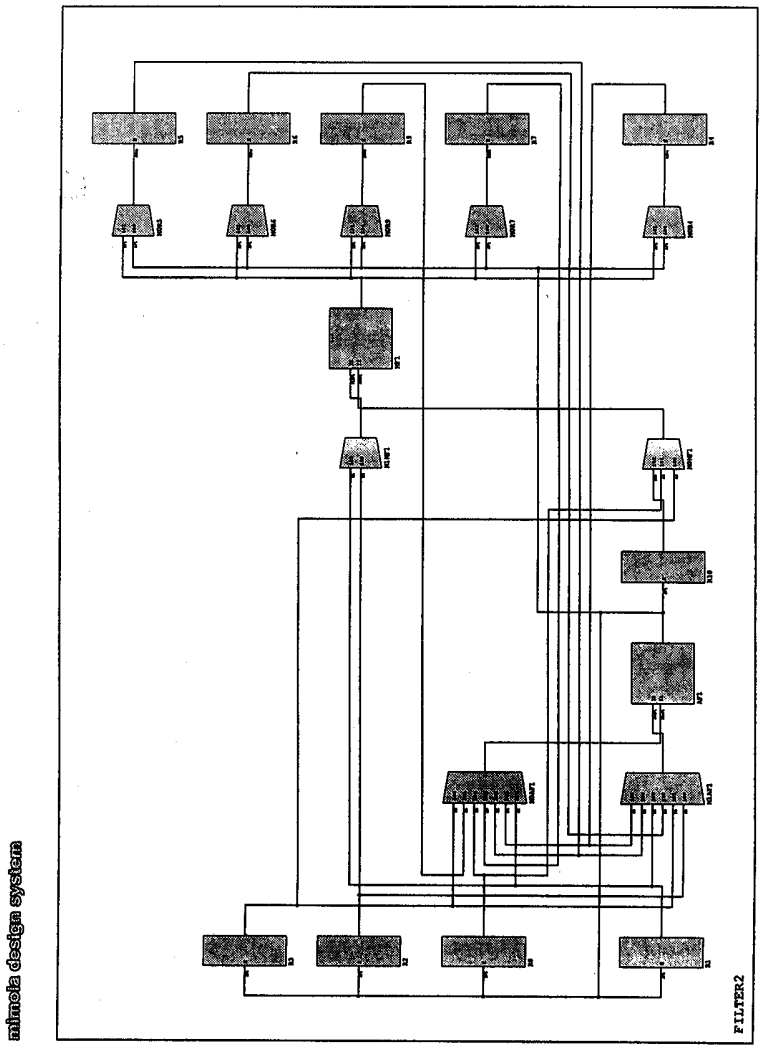

Figure A.5 Synthesized Data Path with Operator Set {1 MF, 1 AF}; Before Memory Synthesis

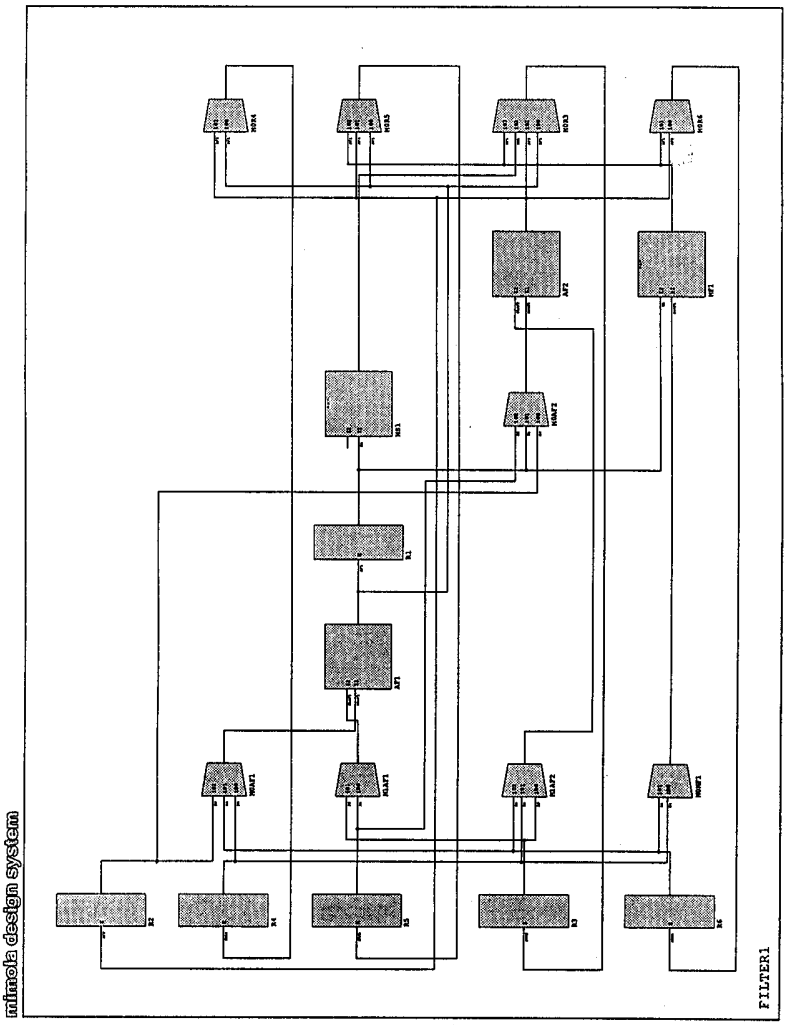

Figure A.6 Synthesized Data Path with Operator Set
{1 MF, 1 AF}; After Memory Synthesis

APPENDIX B : Interface to MIMOLA

MIMOLA is a procedural hardware description language [22] and Supports definition of hardware modules as well as algorithms to be realized in hardware. A number of design tools haue been developed within a design environment named MDS [3]. A common intermediate representation called TREEMOLA acts as the link. A frontend translator (MSSF) converts the MIMOLA source to the TREEMOLA format. The datapath Synthesis technique described in this report accepts a TREEMOLA Input. A number of restrictions are placed an the MIMOLA source which essentially relate to present state of the implementation.

i) The Synthesis program accepts only 3-address Statements for generating the dataflow graph. Due to this the synthesis is capable to handle only scalar variables and dyadic operations.

ü) The connections from the input/output ports as well as modules generating constants are not yet considered during Synthesis (this restriction can be easily removed).

iii) The program handles only simple straight line Segments in the source program. An existing tool (MSSI) translates all control constructs in the source to a version with conditional jumps intersparsed with straight line Blocks. It is intended to develop an Interface such that the present microprogram control Synthesis can be utilized along with the generated datapath. This can be achieved by traversing the TREEMOLA structure generated by MSSI and invoking the datapath Synthesis for every simple block between two consecutive jump Statements.

iv) Modules during one design process are assumed to be of the saure width. Hence different bitwidths and catenations of Bitranges are not supported.

The above mentioned restrictions would be removed with the development of a new frontend to the program, which is under progress. This will include an intraprocedural dataflow analysis, decomposition of complex expressions and generation of the flowgraph.