DATA PATH DESIGN TRADEOFFS USING MABAL

Kayhan Kucukcakar and Alice C. Parker

Technical Report CENG 89-21

*This research was supported in part by the Defense Advanced Research Projects Agency and monitored by the Office of Naval Research under contract No. N00014-87-K-0861 and in part by the Department of Air Force, The Department of Army and the Department of Navy, Contract No. N00039-87-C-0194.

Data Path Design Tradeoffs using MABAL

Kayhan Küçükçakar and Alice C. Parker

Department of Electrical Engineering – Systems

University of Southern California

University Park

Los Angeles, CA 90089-0781

April 27, 1989

* This research was supported in part by the Defense Advanced Research Projects Agency and monitored by the Office of Naval Research under contract No. N00014-87-K-0861 and in part by the Department of Air Force, the Department of Army and the Department of Navy, Contract No. N00039-87-C-0194.

Data Path Design Tradeoffs using MABAL

Abstract

This paper describes a set of novel tradeoff experiments using MABAL, a Module And Bus ALlocation program. MABAL uses a simple heuristic algorithm to concurrently perform functional unit allocation, register allocation, interconnect allocation and module binding, while minimizing overall cost. MABAL can build on partial designs and it corrects designers' input errors. MABAL was used to produce over 3000 RTL designs from a specification which had been previously scheduled. Tradeoffs between buses and multiplexers and between data steering logic and functional logic were investigated. The results indicate data path tradeoffs are sensitive to the characteristics of the module library used, and illustrate the difficulty of integrating module generation or logic synthesis with high-level synthesis. This tradeoff study highlighted MABAL's capabilities and is unlike any other reported in the literature.

1 Introduction

In a digital design, the buses and multiplexers required to interconnect functional units and registers may have a first-order effect on hardware cost (silicon area). Furthermore, the number and type of interconnections required are heavily dependent on the number of functional units and registers in the design, as well as the assignment of operations and values to these units. Thus, interconnect costs must be considered at the same time that other high-level design decisions are being made.

Synthesis of data steering structures has been demonstrated by many systems. However, a number of issues remain to be addressed.

A synthesis program should be comprehensive. It should not be limited to a single style of interconnect. This includes the ability to trade off between buses and multiplexers and the ability to trade off between data steering logic and functional logic. Data path synthesis researchers are often confronted with the comment that "it may be advisable to duplicate functional logic because routing and switching costs outweigh the savings when functional resources are shared". If this is true, a synthesis program should somehow take into account actual wiring costs when trading off interconnect complexity for functional complexity. Furthermore, in order to support tradeoff studies it should be fast enough to support multiple iterations with designer-imposed decisions and constraints, and should be able to search different parts of the design space with different user constraints. Finally, it should provide summary information to the designer as output data so that tedious details about the design do not have to be digested and evaluated. This paper describes a tradeoff study using such a synthesis program (MABAL) to address the above issues.

MABAL[4], concurrently performs the functions known as resource, register and interconnect allocation, and module binding and addresses all the above issues, producing correct designs quickly and minimizing area (cost). MABAL is a part of the ADAM[2] Advanced Design AutoMation synthesis subsystem. The model used by MABAL is capable of creating highly complicated interconnect structures. To illustrate the model used by MABAL, a possible design MABAL might produce is shown in Figure 1.

MABAL provides the design manager program or designer with the ability to influence the design

process, achieving different results for each iteration of MABAL. Different parts of the design space can be explored depending upon the human designer's or design manager's inputs. Very short run times allow designers to iterate frequently, spending more time on higher-level decisions which further improve the overall quality of the final design.

MABAL handles both pipelined and non-pipelined designs, including operator chaining. It allows designer intervention with specification of partial structure and constraints on input and output data availability. Finally, a summary of the resource usage and a cost breakdown are output by MABAL to the designer, freeing the designer from dealing with the design details.

2 Related Research

There are a variety of systems and approaches to data path synthesis, but very few tradeoff studies have been performed. None of the current systems described in the literature have been used to support comprehensive tradeoff studies of the type described here.

BUD[5] has performed studies showing the effects of data steering, register, interconnect and unused area on total area. SAW[11] allows a designer to make tradeoffs interactively. EMUCS[1] has many of the capabilities of MABAL, but these has not been used for tradeoff studies. IMBSL[3] is a system which also accepts partial structures, allows designer intervention, and constructs a layout model for detailed analysis of timing and area, allowing some tradeoff studies.

3 Overview of MABAL

3.1 Basic Operation

The basic problem MABAL solves includes allocation of hardware units and module binding, given a scheduled dataflow graph. The function performed can be modeled as a mapping from the domain consisting of all scheduled operations and values to a range consisting of functional units, registers and their interconnect. The interconnect includes multiplexers, bus drivers and wires. The general mapping can not be fully defined before the RTL design is complete. During allocation and binding,

not only are the best bindings from domain to range to be found, but also the set of range elements is to be determined simultaneously.

Although the minimum number of functional units can be calculated using existing theory [7], prior to RTL synthesis there is very little known about the interconnect for a given design, whose cost is highly dependent on the module binding. A complication of the problem is that a design with the minimum allocation of functional units and registers does not guarantee a small interconnect cost. In fact, a design with non-optimal allocation of registers and functional units may turn out to be cheaper overall than the design with optimal resource allocation, since interconnect costs can have a first-order effect on total cost. The heavy resource sharing required when functional resources are scarce can cause multiplexing and bussing costs to dominate the total cost.

As a result, the range of the mapping has to be totally unconstrained to get optimal results, which causes the complexity of finding the best mapping to be computationally intensive.

3.2 How MABAL performs synthesis

MABAL performs synthesis using the following approach. A greedy, incremental algorithm is used by MABAL with some decisions reversible later. Although the approach used does not allow backtracking, the reversible decisions provide a similar, but more limited effect of exploring a larger portion of the design space without actual searching. In this way, the negative consequences resulting from decisions which are not suitable for the final design are reduced. MABAL begins with a scheduled DFG and optionally, full or partial structure. Resource allocation is incrementally performed as needed. The idea behind the main algorithm is to perform incremental binding by delaying interconnect style decisions until all bindings are set. The decisions made about the interconnect are tentative unless they are forced by the user. The connections between hardware modules are implicitly set by bindings but the way multiplexers and bus drivers are used is not finalized until all operations and values are bound to hardware modules. During incremental binding decisions, MABAL calculates the cost of the possible bindings for an operation and chooses the best binding with respect to the partial design and tentative interconnect already in place.

3.3 Features of MABAL

MABAL performs functional unit allocation, register allocation, operation/value binding and data steering allocation concurrently, with the objective of minimizing total cost by selecting between allocating more functional units or data steering logic. MABAL handles both pipelined and non-pipelined designs. Information about conditionals, constants and commutativity of operations is effectively used to minimize interconnect cost and share resources. ALUs are supported. MABAL handles designs with outer loops in which values are fed back from primary outputs to primary inputs. MABAL allows the user to constrain inputs and outputs or even partially specify a design, while checking for errors in designer's inputs, and constructing interconnect which is not restricted to a single style.

Variations in input and output signals are supported by MABAL. Any input to the target design can be selectively latched or left unlatched upon user request. The input values can be constants which do not require storage, variables used for multiple iterations, variables sampled once at the beginning of the hardware operation and stored until the end, and finally variables which are sampled at each iteration of a loop body. Any output from the target design can be selectively latched for a user-specified time duration. Since MABAL allows the user to decide how and how long to latch inputs and outputs, the problem of interfacing several pieces of a larger design becomes simpler.

Data steering allocation consists of allocating multiplexers, bus drivers and carriers. There is no enforced bus style. The resulting interconnect is decided entirely by making tradeoffs between using more multiplexers, bus drivers, functional units or registers subject to bindings and restrictions requested by the user. Unless the user restricts the use of tri-state drivers, multiplexers and tri-state drivers may be mixed depending on the relative costs of these modules.

One of the key features of MABAL is the ability to allow the designer to influence the design process, while automating the tasks that the designer does not want to control[1]. If the designer intervenes, a variety of items can be specified such as: the binding of any operation or value to functional units or registers, and the type of interconnect. The degree of manual intervention is not

[1]MABAL is able to produce good results without requiring designer intervention.

fixed. The program can, to some extent, take advantage of other approaches to the RTL design process (such as clustering or clique partitioning) building onto partially (even imperfectly) specified designs.

Depending on user-supplied structural information and parameters (e.g. resource allocation, bindings and restriction of bussed connections for specified ports) different parts of the design space can be explored. Since an incremental algorithm which builds on the existing design is used, every different starting point in the design space may yield a different design.

Since designer intervention is allowed, there is always a chance for designers' errors to be introduced. The algorithm is designed to detect and correct these errors. The user-supplied structural information is checked and, in case there are errors, corrections are made by the program during execution. For example, there may be time conflicts in user-supplied module binding information. If such conflicts exist, MABAL will relax these user decisions one at a time to resolve the conflict. The user is also informed with a warning. If a complete design is given to MABAL, the program can be used to check the validity of the design and correct it if necessary.

As stated earlier, it might be desirable to duplicate some small hardware modules to save routing area. To accomplish this, a parameter which indicates a degree of tendency for MABAL to use more hardware modules or more interconnect at individual decisions can be set by the user to override a MABAL decision. This number can be derived from some statistical data, the characteristics of the library, and the technology used and can be effectively used to make tradeoffs between using more functional area or interconnect area.

4 Experimental Data and Results

4.1 The Example and Experiments

A fairly small dataflow graph for a controller has been chosen as an example. This example, which has been scheduled by MAHA [8] is shown in Figure 3. The non-pipelined schedule produced by MAHA will be used as a starting point to demonstrate some design tradeoffs. The dataflow graph

has feedback from its outputs to its inputs. All inputs to the dataflow graph need to be stored and the output signals are stored until the end of each iteration. The values *xs* and *ys* are to be stored until the end of an undetermined number of loop iterations while new *xi* and *yi* values are sampled at each iteration. There is a single-ported memory with memory read operation *m*. Three closely related libraries (Table 2) have been used during the experiment. The original library (Library 1) was created from the Texas Instruments 2 micron CMOS standard cell library [10]. Library 2 is the same as the first library except that the cost of the tri-state driver was decreased. Library 3 has the same costs for data steering modules as the first library but it has some smaller, slower functional units. It would be unreasonable to assume that slower modules could be used without changing the delay characteristics of the design. It is obvious that the design will be slower if library 3 is used instead of library 1, if the same schedule is used. Our goal here is to demonstrate the concept of how well variations in libraries can be utilized by a data path synthesis algorithm which is sensitive to the technology.

MABAL was run on these three libraries with combinations of following user preferences/parameters:

- allowing/disallowing the use of tri-state drivers,
- allowing/disallowing trading off between the steering logic area and the functional area, and
- starting MABAL with different functional resource allocation seeds.

4.2 Results

The specific purpose of the experiments was to observe how varying the relative costs of modules affected tradeoffs involved in their usage in a data path implementation. Two particular studies focused on trading off between buses and multiplexers and trading off between functional and bit-steering logic. The schedule of operations was assumed to be fixed, but even minor variations in module sets might have produced different schedules. Also, wiring space and unused chip area are not included here, but will be included in future studies. Finally, controller area is a factor, and will be taken into account in the future. In general, more complex tradeoff studies will be possible once the ADAM synthesis system is fully integrated.

In general, for designs which have little sharing, multiplexing is most cost effective, while for heavily shared designs, buses are cheaper. However, the point where the tradeoff between multiplexers and buses is made shifts depending on the relative costs of both types of modules.

A lower bound on functional resource allocation is achieved when all resources are shared to the fullest extent possible, given the schedule. An upper bound is achieved by assigning a resource for every data flow graph operation, independent of potential sharing. Between these two bounds, a number of designs exist, with varying bit steering logic, depending on the amount of sharing.

The capability of trading between multiplexers and tri-state drivers, and between the interconnection units and functional units may be very important when the costs of multiplexers, tri-state drivers and some functional units are comparable. For example, the designs encountered in some communication applications have coding and non-arithmetic processing, and include simpler modules.

For library 3, the costs of modules were adjusted such that costs of additional adders, subtractors and registers were slightly higher than the worst-case multiplexer costs to share them. While this particular situation is not realistic for this example, this shows the capability of making tradeoffs between functional logic and steering logic.

A total of 3310 designs have been generated by using eight major user parameter combinations and changing the resource allocation seed to the program. Selected designs are shown in Table 3 - Table 5. Not all the designs generated are unique; some designs have the same RTL structure although the bindings for operations and values differ. It took less than a day to generate the designs and to extract the data in the tables[2]. This would not be possible to do if MABAL were not this flexible and fast.

Results obtained indicate that the decision to trade off between tri-state buses and multiplexer steering logic is quite sensitive to the relative costs of modules employed. A 20% decrease in costs of tri-state drivers produced a 50% decrease in the use of multiplexing logic.

As is seen from Table 7, tri-state drivers are hardly used in the designs generated from library 1. But, when a 20% cheaper tri-state driver in library 2 is substituted, tri-state driver use increases drastically. It's seen from Table 7 that the use of buses is also related to the resource sharing. If

[2]Human analysis of the results, of course, took somewhat longer.

maximum resource sharing (T=0) is used, buses are more likely to be created.

Results obtained from tradeoffs between bit-steering and functional resources show that there are literally thousands of good designs possible, as shown in Table 6. In fact, ignoring routing and controller area, there are 76 equally good "best" designs.

Although, it may seem plausible to use many functional units in the designs using this library, the resulting designs actually have a functional area which is less than the upper-bound functional area (Table 5). It is also seen that there are no tri-state drivers used with library 3 in any of 1140 designs generated (Table 7) because resource sharing obviously tends to be lower when multiplexers and bus drivers cost as much as the shared resource, making buses less desirable. The best automatically generated design from Table 3 was reevaluated using library 3 and was shown to be comparatively cheaper than the others in Table 5. Further decreases in the ratios of functional module costs to data steering module costs, however might result in cheaper designs.

From the designs generated from library 3, we observed some general characteristics. We generated several designs keeping functional resource allocation constant and then varied functional resource allocation and generated several more. Repeating this process, we obtained a distribution which is roughly similar to Figure 2 when we plotted the designs points corresponding to minimum steering logic area for each functional resource allocation (non-inferior designs).

Figure 2 tells us a number of important things. The line segment between points 1 and 2 corresponds to designs with the minimum possible steering logic area (which can be non-zero in cases like having to share memory address lines, a value conditionally coming from multiple sources, etc). All designs with minimum functional resource allocation lie on the line segment from point 3 and point 4. We are not generally interested any designs to the right of point 3 or above point 1 because they are more likely to be inferior. The design points between points 2 and 3 represent designs with roughly equal total area but different architectures. The number of designs on this line segment can vary greatly depending on the characteristics of the library, the dataflow graph and the schedule. If no tradeoff is feasible between the functional units and the steering logic, this line segment shrinks to a single point. The larger the number of *tradable*[3] library modules and, the larger the number of

[3]a *tradable* module means its area is close to the area of steering logic modules

operations implemented by those library modules, the higher the likelihood of having more designs in this region (ignoring the fact that some potential designs will coincide). The final differences between areas of designs will be due to floorplanning and routing differences which are highly sensitive to the architectures generated. Although, floorplanning is ignored by most synthesis programs (an exception is [5]), it is the only criterion which can be used to determine the best design among the designs lying on the line from point 2 to point 3. The general approach in the synthesis community has been to generate designs in close proximity of point 3 (minimum functional resources), although the designs towards point 3 might generally have a higher number of nets to be routed and longer wires per net, which we postulate may produce larger designs.

We cannot yet draw conclusions about the routing and unused area for each design. However, an examination of the designs in Table 5 shows an almost-constant number of two-point nets. Even when the schedule produced by MAHA is changed, the number of nets varies by only $\pm 5\%$.

An illustration of the flexibility of MABAL for designer intervention is shown in Table 3. A simple examination of the best automatically produced design in Table 3 has revealed that the inputs to division operations and consecutive memory-read operations come from two different places. Simply moving $d1$ to stage 7 without changing the delay characteristics of the design, results in binding all input values of division operations to the same register, which saves one 8-bit-wide 2-to-1 multiplexer. The new binding patterns for other values and operations are slightly changed by MABAL, given this change. If there were a clustering algorithm available in ADAM, the whole set of bindings produced by the clusterer could be input to the MABAL program. The designer could also affect the design process by specifying the initial number of resources to MABAL which, in turn, might generate a different pattern of bindings resulting in a better design. But, even when the user does not intervene, MABAL generates designs within a few percent of the cost of the best design which can be generated by manipulating the initial resource allocation.

In order to draw conclusions about the design tradeoffs, the quality of designs produced by MABAL was investigated. To illustrate the quality of MABAL results we have synthesized other dataflow graphs known in the literature. The results for a fifth-order elliptic wave filter example and differential equation solver example are shown in Tables 8 and 9. A different library which does not allow any

tradeoff between the functional units and the steering logic was used for these examples. For more information about the results of these examples and more details about MABAL, refer to [4].

5 Conclusions

A number of conclusions have been arrived at as a result of the experiments described here. Broadly speaking, the results indicate that data-path designs are sensitive to variations in the module library. This leads us to believe module selection should precede data-path synthesis. Furthermore, if module generators, logic synthesis or logic optimization are employed, their interaction with data-path synthesis procedures must be carefully studied. Synthesis programs with a rigid design style (using multiplexers or buses exclusively) will not provide the tradeoffs shown here, and synthesis programs which do not tradeoff functional costs for data steering costs might miss some better designs.

We expect more definitive conclusions to be drawn when routing area and control area are included, when layouts are produced, and when more examples have been studied.

Routing area is not explicitly considered by MABAL, but the effect of routing area can be taken into account to some extent by incorporating the routing area per net (area taken by the average length wire) into functional units and steering logic modules. Since MABAL minimizes the number of interconnections as a result of trying to minimize interconnect area and can be forced to duplicate functional modules rather than including steering logic, the deficiency of not considering the routing area explicitly can be partially overcome. Functions like unsigned-shift or concatenation can be input to the program with some cost corresponding to the estimate of the area taken by wiring.

Since MABAL uses a greedy method in a simple and effective way, the run time of the program is negligible compared to other activities like scheduling, without sacrificing design quality. The speed up obtained here over comparable programs can effectively enable more design explorations at higher levels followed by a final extensive optimization on the interconnect.

MABAL provides a flexible method of specifying partial designs and supports some user preferences for the point to point connection model [6].

The capability of MABAL to trade off between multiplexers and tri-state-drivers and between

interconnect area and functional unit area has been shown to be important in designs which include less-complicated functional units.

References

[1] C. Y. Hitchcock, "Automated Synthesis of Data Paths", Master's Thesis, Department of Electrical Engineering, Carnegie-Mellon University, January 1983.

[2] R. Jain, K. Küçükçakar, M. J. Mlinar and A. C. Parker, "Experience with the ADAM Synthesis System", to appear in Proc. of 1989 Design Automation Conf., Las Vegas, June 1989.

[3] David W. Knapp, "Synthesis from Partial Structure", Proc. of IFIP TC-10 Conference, Pisa, September 1988.

[4] Kayhan Küçükçakar and Alice C. Parker, "MABAL : A software package for Module And Bus ALlocation", to appear in International Journal of Computer Aided VLSI Design, June 1989.

[5] M. C. Farland, "Using Bottom-Up Design Techniques in the Synthesis of Digital Hardware from Abstract Behavioral Descriptions", Proc. of 1986 Design Automation Conf., Las Vegas, June 1986.

[6] Barry M. Pangrle, "Splicer: A Heuristic Approach to Connectivity Binding", Proc. of 1988 Design Automation Conf., Anaheim, June 1988.

[7] Nohbyung Park, "Sehwa: A Software Package for Synthesis of Pipelines from Behavioral Specifications", IEEE Transactions on Computer-Aided Design, Vol 7, No 3, March 1988.

[8] Alice C. Parker, Jorge "T" Pizarro and Mitch Mlinar, "MAHA: A Program for Datapath Synthesis", Proc. of 1986 Design Automation Conf., Las Vegas, June 1986.

[9] P.G. Paulin, J.P. Knight, E.F. Girczyc, "HAL : A Multi-Paradigm Approach to Automatic Data Path Synthesis", Proc. of 1986 Design Automation Conf., Las Vegas, June 1986.

[10] Texas Instruments, "2 micrometer CMOS Standard Cell Data Book", 1986.

[11] D.E. Thomas, et. al. , "The System Architect's Workbench", Proc. of 1988 Design Automation Conf., Anaheim, June 1988.

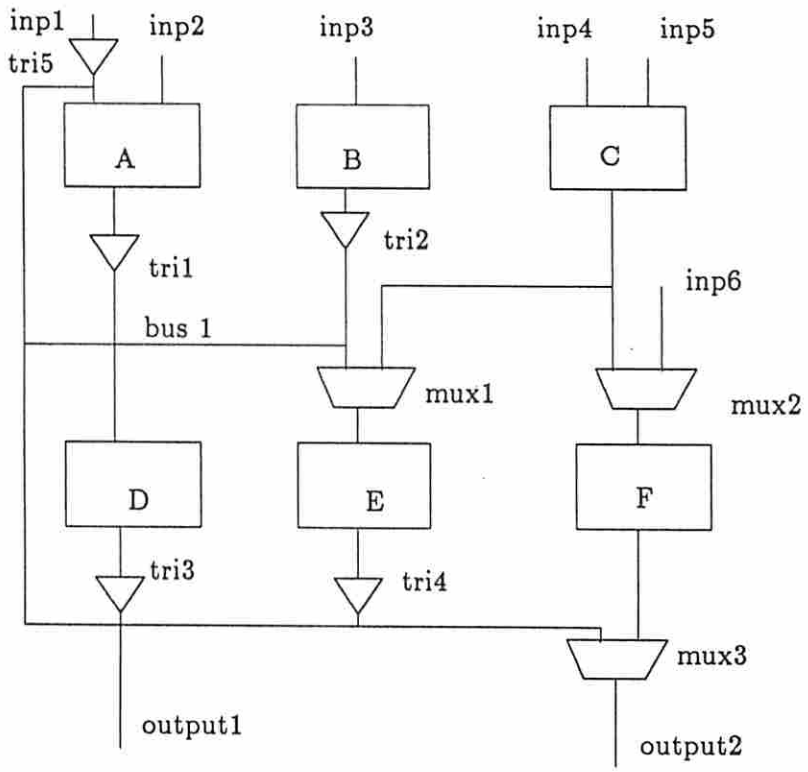


Figure 1: A possible design which might be generated by MABAL

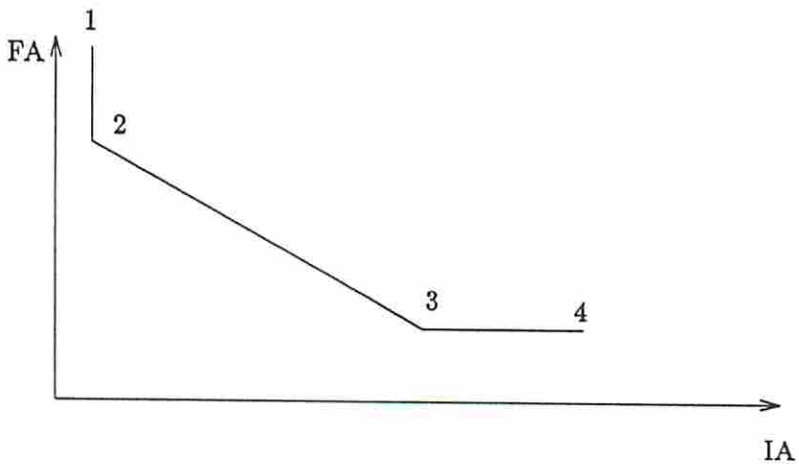
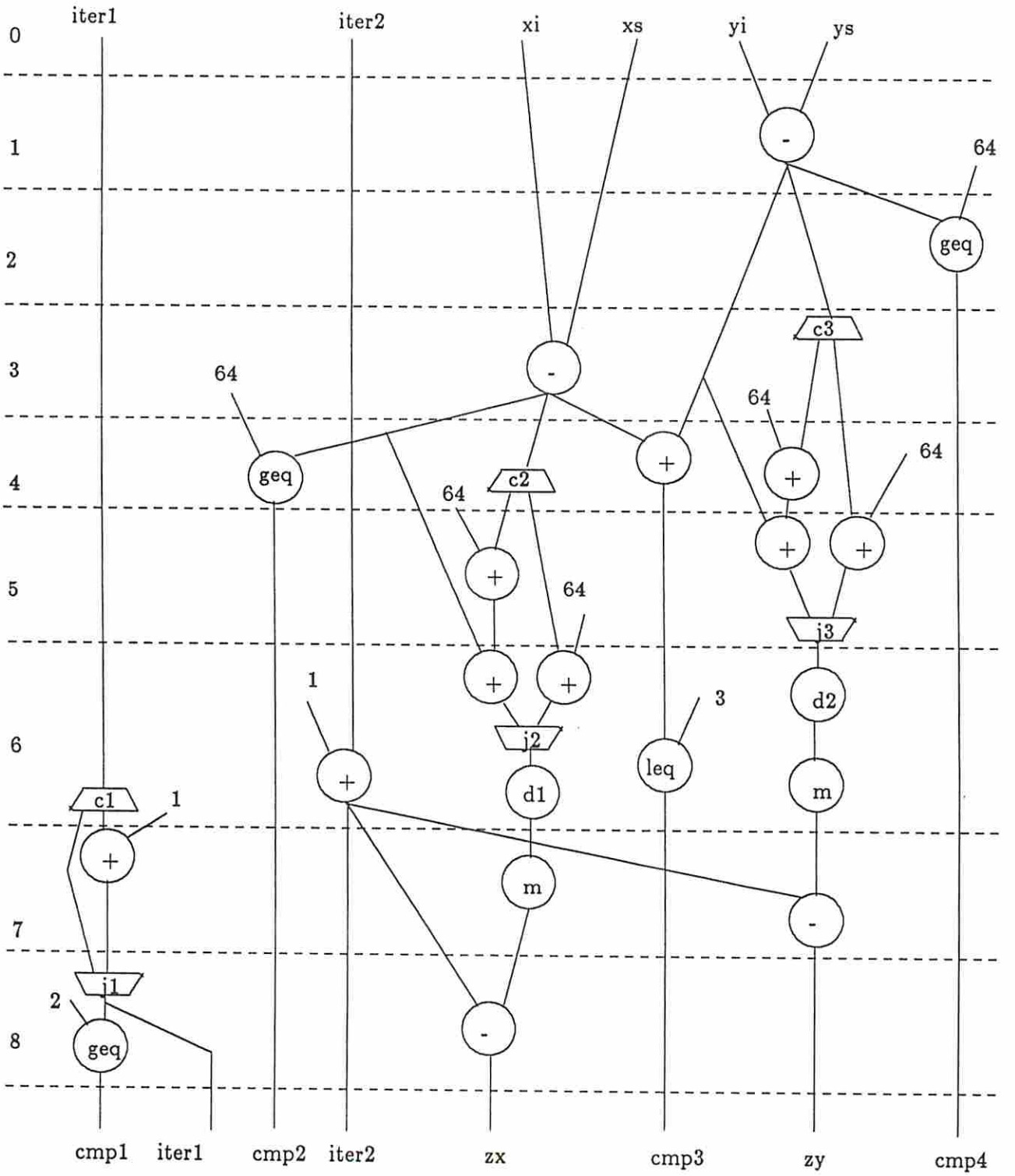


Figure 2: Distribution of designs by functional module area versus steering logic area tradeoffs



C-J : Conditional Block
All values are 8 bits except outputs of comparison operations

Figure 3: The dataflow graph used for experimentation

MC	Multiplexer (tree) count
MI	The number of multiplexer (tree) inputs
MA	Multiplexer area calculated as the number of 2-to-1 multiplexers needed to create the tree
B/D	The number of buses and number of 8-bit tri-state drivers
IA	Steering logic area (tri-state drivers and multiplexers)
TA	Normalized total active area within a table
FA	Functional active area
TAA	Total active chip area
T	1 if trading off between functional and steering logic is allowed, 0 otherwise
B	0 if only multiplexers are used, 1 if both multiplexers and tri-state drivers are allowed to be used
A	The number of adders
S	The number of subtractors
Ri	The number of i-bit registers
G	The number of greater-than-or-equal-to operators
L	The number of less-than-or-equal-to operators
x	Don't care
†	Comments in Paulin's dissertation indicate a run time of close to ten minutes
††	The best design generated from this library which used tri-state drivers
N/A	Not Available
‡	Minimum steering logic
£	one module per operation
§	Manual intervention to reduce the number of multiplexers
	CPU Time is on a Sun-3/160 for MABAL, Sun-3/260 for Splicer and Xerox 1108 for HAL.

Table 1: Nomenclature used in tables and figures

Library		1		2	3
Module	Bitwidth	Area	Delay(ns)	Area	Area
adder	8	13350	17.0	13350	5000
subtractor	8	13350	17.0	13350	5000
multiplexer	1	370	3.7	370	370
tri-state driver	1	250	2.4	200	250
register	1	1106	5.0	1106	350
register	8	8850	5.0	8850	2800
G	8	8710	16.3	8710	8710
L	8	8885	18.3	8885	8885
divide	8	0	0.0	0.0	0.0
memoryread	8	∞*	17.0	∞	∞

* Set to force the selection of only one memory unit

Table 2: Libraries used for experiments

T	B	A	S	G	L	R1	R8	MC	MI	MA	B/D	IA	FA	TAA	
x	x	2	1	1	1	3	8	12	32	20	-	59200	131763	190963	
1	0	2	1	1	1	3	8	11	30	19	-	56240	131763	188003	§
0	1	4	2	1	1	3	8	12	29	17	1/8	60320	171813	232133	††

Table 3: Selected designs generated from library 1

T	B	A	S	G	L	R1	R8	MC	MI	MA	B/D	IA	FA	TAA
1	1	2	1	2	1	3	9	9	20	11	3/9	46960	149323	196283
0	1	2	1	1	1	3	8	8	18	10	5/16	55200	131763	186963
0	1	2	1	1	1	3	8	7	17	10	6/20	61600	131763	193363

Table 4: Selected designs generated from library 2

T	B	A	S	G	L	R1	R8	MC	MI	MA	B/D	IA	FA	TAA	
1	0	4	3	2	1	3	16	6	12	6	-	17760	107155	124915	
1	1	4	3	1	1	3	16	7	16	9	-	26640	98445	125085	
1	0	3	3	2	1	3	16	7	15	8	-	23680	102155	125835	
1	1	3	3	1	1	3	16	8	19	11	-	32560	93445	126005	
1	0	5	3	2	1	3	17	5	10	5	-	14800	114955	129755	
The best automatically-produced design reevaluated from Table 3 using library 3															
0	0	2	1	1	1	3	8	12	32	20	-	59200	56045	115245	
Designs with upper bound resource allocations															
1	0	7	3	3	1	3	17	2	4	2	-	5920	133665	139585	‡
-	-	9	4	3	1	3	19	2	4	2	-	5920	154265	160185	£

Table 5: Selected designs generated from library 3

T	Area	Normalized Area	Number of designs with same or smaller area	
1	135505	1.09	2280	100.0%
1	131585	1.05	1672	73.0%
1	126005	1.01	608	26.7%
1	124915	1.00	76	3.3%

Table 6: Distribution of designs generated from library 3

T	Percentage of designs with buses	Total number of designs	Library
0	1.25	160	1
1	0.00	240	1
0	96.67	150	2
1	73.75	160	2
0	0.00	1140	3

Table 7: Statistical Data about Bus(tri-state driver) Use

Non-pipelined elliptic filter							
2 Adders			1 Multiplier				
HAL Schedule					19 Stages		
	Reg	MC	MI	MA	B/D	TA	CPU Time
MABAL	10	12	42	30	-	1.04	10.52
MABAL	10	10	32	22	2/11	1.05	14.46
HAL	12	6	26	20	-	1.00	~600 †
Splicer					21 Stages		
	Reg	MC	MI	MA	B/D	TA	CPU Time
Splicer	N/A	9	44	35	-	N/A	20257
Splicer	N/A	9	43	34	-	N/A	55
Splicer	N/A	9	45	36	-	N/A	300

Table 8: Non-pipelined elliptic filter, Case 1

Differential Equation						
Same Schedule for all Programs						
1 Adder	1 Subtractor			2 Multipliers		1 comparator
	Reg	MC	MI	MA	TA	CPU Time
MABAL	5	8	17	9	1.01	0.74
MABAL	5	6	13	7	1.01	0.74
MABAL	6	7	15	8	1.02	0.82
HAL	5	6	13	7	1.01	140
Splicer	6	5	11	6	1.00	1245
Splicer	6	5	12	7	1.01	291
Splicer	6	6	16	10	1.02	6.40

Table 9: Differential Equation

DATA PATH DESIGN TRADEOFFS USING MABAL

Kayhan Kucukcakar and Alice C. Parker

Technical Report CENG 89-21

*This research was supported in part by the Defense Advanced Research Projects Agency and monitored by the Office of Naval Research under contract No. N00014-87-K-0861 and in part by the Department of Air Force, The Department of Army and the Department of Navy, Contract No. N00039-87-C-0194.

Data Path Design Tradeoffs using MABAL

Kayhan Küçükçakar and Alice C. Parker

Department of Electrical Engineering – Systems

University of Southern California

University Park

Los Angeles, CA 90089-0781

April 27, 1989

* This research was supported in part by the Defense Advanced Research Projects Agency and monitored by the Office of Naval Research under contract No. N00014-87-K-0861 and in part by the Department of Air Force, the Department of Army and the Department of Navy, Contract No. N00039-87-C-0194.

Data Path Design Tradeoffs using MABAL

Abstract

This paper describes a set of novel tradeoff experiments using MABAL, a Module And Bus ALlocation program. MABAL uses a simple heuristic algorithm to concurrently perform functional unit allocation, register allocation, interconnect allocation and module binding, while minimizing overall cost. MABAL can build on partial designs and it corrects designers' input errors. MABAL was used to produce over 3000 RTL designs from a specification which had been previously scheduled. Tradeoffs between buses and multiplexers and between data steering logic and functional logic were investigated. The results indicate data path tradeoffs are sensitive to the characteristics of the module library used, and illustrate the difficulty of integrating module generation or logic synthesis with high-level synthesis. This tradeoff study highlighted MABAL's capabilities and is unlike any other reported in the literature.

1 Introduction

In a digital design, the buses and multiplexers required to interconnect functional units and registers may have a first-order effect on hardware cost (silicon area). Furthermore, the number and type of interconnections required are heavily dependent on the number of functional units and registers in the design, as well as the assignment of operations and values to these units. Thus, interconnect costs must be considered at the same time that other high-level design decisions are being made.

Synthesis of data steering structures has been demonstrated by many systems. However, a number of issues remain to be addressed.

A synthesis program should be comprehensive. It should not be limited to a single style of interconnect. This includes the ability to trade off between buses and multiplexers and the ability to trade off between data steering logic and functional logic. Data path synthesis researchers are often confronted with the comment that "it may be advisable to duplicate functional logic because routing and switching costs outweigh the savings when functional resources are shared". If this is true, a synthesis program should somehow take into account actual wiring costs when trading off interconnect complexity for functional complexity. Furthermore, in order to support tradeoff studies it should be fast enough to support multiple iterations with designer-imposed decisions and constraints, and should be able to search different parts of the design space with different user constraints. Finally, it should provide summary information to the designer as output data so that tedious details about the design do not have to be digested and evaluated. This paper describes a tradeoff study using such a synthesis program (MABAL) to address the above issues.

MABAL[4], concurrently performs the functions known as resource, register and interconnect allocation, and module binding and addresses all the above issues, producing correct designs quickly and minimizing area (cost). MABAL is a part of the ADAM[2] Advanced Design AutoMation synthesis subsystem. The model used by MABAL is capable of creating highly complicated interconnect structures. To illustrate the model used by MABAL, a possible design MABAL might produce is shown in Figure 1.

MABAL provides the design manager program or designer with the ability to influence the design

process, achieving different results for each iteration of MABAL. Different parts of the design space can be explored depending upon the human designer's or design manager's inputs. Very short run times allow designers to iterate frequently, spending more time on higher-level decisions which further improve the overall quality of the final design.

MABAL handles both pipelined and non-pipelined designs, including operator chaining. It allows designer intervention with specification of partial structure and constraints on input and output data availability. Finally, a summary of the resource usage and a cost breakdown are output by MABAL to the designer, freeing the designer from dealing with the design details.

2 Related Research

There are a variety of systems and approaches to data path synthesis, but very few tradeoff studies have been performed. None of the current systems described in the literature have been used to support comprehensive tradeoff studies of the type described here.

BUD[5] has performed studies showing the effects of data steering, register, interconnect and unused area on total area. SAW[11] allows a designer to make tradeoffs interactively. EMUCS[1] has many of the capabilities of MABAL, but these has not been used for tradeoff studies. IMBSL[3] is a system which also accepts partial structures, allows designer intervention, and constructs a layout model for detailed analysis of timing and area, allowing some tradeoff studies.

3 Overview of MABAL

3.1 Basic Operation

The basic problem MABAL solves includes allocation of hardware units and module binding, given a scheduled dataflow graph. The function performed can be modeled as a mapping from the domain consisting of all scheduled operations and values to a range consisting of functional units, registers and their interconnect. The interconnect includes multiplexers, bus drivers and wires. The general mapping can not be fully defined before the RTL design is complete. During allocation and binding,

not only are the best bindings from domain to range to be found, but also the set of range elements is to be determined simultaneously.

Although the minimum number of functional units can be calculated using existing theory [7], prior to RTL synthesis there is very little known about the interconnect for a given design, whose cost is highly dependent on the module binding. A complication of the problem is that a design with the minimum allocation of functional units and registers does not guarantee a small interconnect cost. In fact, a design with non-optimal allocation of registers and functional units may turn out to be cheaper overall than the design with optimal resource allocation, since interconnect costs can have a first-order effect on total cost. The heavy resource sharing required when functional resources are scarce can cause multiplexing and bussing costs to dominate the total cost.

As a result, the range of the mapping has to be totally unconstrained to get optimal results, which causes the complexity of finding the best mapping to be computationally intensive.

3.2 How MABAL performs synthesis

MABAL performs synthesis using the following approach. A greedy, incremental algorithm is used by MABAL with some decisions reversible later. Although the approach used does not allow backtracking, the reversible decisions provide a similar, but more limited effect of exploring a larger portion of the design space without actual searching. In this way, the negative consequences resulting from decisions which are not suitable for the final design are reduced. MABAL begins with a scheduled DFG and optionally, full or partial structure. Resource allocation is incrementally performed as needed. The idea behind the main algorithm is to perform incremental binding by delaying interconnect style decisions until all bindings are set. The decisions made about the interconnect are tentative unless they are forced by the user. The connections between hardware modules are implicitly set by bindings but the way multiplexers and bus drivers are used is not finalized until all operations and values are bound to hardware modules. During incremental binding decisions, MABAL calculates the cost of the possible bindings for an operation and chooses the best binding with respect to the partial design and tentative interconnect already in place.

3.3 Features of MABAL

MABAL performs functional unit allocation, register allocation, operation/value binding and data steering allocation concurrently, with the objective of minimizing total cost by selecting between allocating more functional units or data steering logic. MABAL handles both pipelined and non-pipelined designs. Information about conditionals, constants and commutativity of operations is effectively used to minimize interconnect cost and share resources. ALUs are supported. MABAL handles designs with outer loops in which values are fed back from primary outputs to primary inputs. MABAL allows the user to constrain inputs and outputs or even partially specify a design, while checking for errors in designer's inputs, and constructing interconnect which is not restricted to a single style.

Variations in input and output signals are supported by MABAL. Any input to the target design can be selectively latched or left unlatched upon user request. The input values can be constants which do not require storage, variables used for multiple iterations, variables sampled once at the beginning of the hardware operation and stored until the end, and finally variables which are sampled at each iteration of a loop body. Any output from the target design can be selectively latched for a user-specified time duration. Since MABAL allows the user to decide how and how long to latch inputs and outputs, the problem of interfacing several pieces of a larger design becomes simpler.

Data steering allocation consists of allocating multiplexers, bus drivers and carriers. There is no enforced bus style. The resulting interconnect is decided entirely by making tradeoffs between using more multiplexers, bus drivers, functional units or registers subject to bindings and restrictions requested by the user. Unless the user restricts the use of tri-state drivers, multiplexers and tri-state drivers may be mixed depending on the relative costs of these modules.

One of the key features of MABAL is the ability to allow the designer to influence the design process, while automating the tasks that the designer does not want to control[1]. If the designer intervenes, a variety of items can be specified such as: the binding of any operation or value to functional units or registers, and the type of interconnect. The degree of manual intervention is not

[1]MABAL is able to produce good results without requiring designer intervention.

fixed. The program can, to some extent, take advantage of other approaches to the RTL design process (such as clustering or clique partitioning) building onto partially (even imperfectly) specified designs.

Depending on user-supplied structural information and parameters (e.g. resource allocation, bindings and restriction of bussed connections for specified ports) different parts of the design space can be explored. Since an incremental algorithm which builds on the existing design is used, every different starting point in the design space may yield a different design.

Since designer intervention is allowed, there is always a chance for designers' errors to be introduced. The algorithm is designed to detect and correct these errors. The user-supplied structural information is checked and, in case there are errors, corrections are made by the program during execution. For example, there may be time conflicts in user-supplied module binding information. If such conflicts exist, MABAL will relax these user decisions one at a time to resolve the conflict. The user is also informed with a warning. If a complete design is given to MABAL, the program can be used to check the validity of the design and correct it if necessary.

As stated earlier, it might be desirable to duplicate some small hardware modules to save routing area. To accomplish this, a parameter which indicates a degree of tendency for MABAL to use more hardware modules or more interconnect at individual decisions can be set by the user to override a MABAL decision. This number can be derived from some statistical data, the characteristics of the library, and the technology used and can be effectively used to make tradeoffs between using more functional area or interconnect area.

4 Experimental Data and Results

4.1 The Example and Experiments

A fairly small dataflow graph for a controller has been chosen as an example. This example, which has been scheduled by MAHA [8] is shown in Figure 3. The non-pipelined schedule produced by MAHA will be used as a starting point to demonstrate some design tradeoffs. The dataflow graph

has feedback from its outputs to its inputs. All inputs to the dataflow graph need to be stored and the output signals are stored until the end of each iteration. The values *xs* and *ys* are to be stored until the end of an undetermined number of loop iterations while new *xi* and *yi* values are sampled at each iteration. There is a single-ported memory with memory read operation *m*. Three closely related libraries (Table 2) have been used during the experiment. The original library (Library 1) was created from the Texas Instruments 2 micron CMOS standard cell library [10]. Library 2 is the same as the first library except that the cost of the tri-state driver was decreased. Library 3 has the same costs for data steering modules as the first library but it has some smaller, slower functional units. It would be unreasonable to assume that slower modules could be used without changing the delay characteristics of the design. It is obvious that the design will be slower if library 3 is used instead of library 1, if the same schedule is used. Our goal here is to demonstrate the concept of how well variations in libraries can be utilized by a data path synthesis algorithm which is sensitive to the technology.

MABAL was run on these three libraries with combinations of following user preferences/parameters:

- allowing/disallowing the use of tri-state drivers,
- allowing/disallowing trading off between the steering logic area and the functional area, and
- starting MABAL with different functional resource allocation seeds.

4.2 Results

The specific purpose of the experiments was to observe how varying the relative costs of modules affected tradeoffs involved in their usage in a data path implementation. Two particular studies focused on trading off between buses and multiplexers and trading off between functional and bit-steering logic. The schedule of operations was assumed to be fixed, but even minor variations in module sets might have produced different schedules. Also, wiring space and unused chip area are not included here, but will be included in future studies. Finally, controller area is a factor, and will be taken into account in the future. In general, more complex tradeoff studies will be possible once the ADAM synthesis system is fully integrated.

In general, for designs which have little sharing, multiplexing is most cost effective, while for heavily shared designs, buses are cheaper. However, the point where the tradeoff between multiplexers and buses is made shifts depending on the relative costs of both types of modules.

A lower bound on functional resource allocation is achieved when all resources are shared to the fullest extent possible, given the schedule. An upper bound is achieved by assigning a resource for every data flow graph operation, independent of potential sharing. Between these two bounds, a number of designs exist, with varying bit steering logic, depending on the amount of sharing.

The capability of trading between multiplexers and tri-state drivers, and between the interconnection units and functional units may be very important when the costs of multiplexers, tri-state drivers and some functional units are comparable. For example, the designs encountered in some communication applications have coding and non-arithmetic processing, and include simpler modules.

For library 3, the costs of modules were adjusted such that costs of additional adders, subtractors and registers were slightly higher than the worst-case multiplexer costs to share them. While this particular situation is not realistic for this example, this shows the capability of making tradeoffs between functional logic and steering logic.

A total of 3310 designs have been generated by using eight major user parameter combinations and changing the resource allocation seed to the program. Selected designs are shown in Table 3 - Table 5. Not all the designs generated are unique; some designs have the same RTL structure although the bindings for operations and values differ. It took less than a day to generate the designs and to extract the data in the tables[2]. This would not be possible to do if MABAL were not this flexible and fast.

Results obtained indicate that the decision to trade off between tri-state buses and multiplexer steering logic is quite sensitive to the relative costs of modules employed. A 20% decrease in costs of tri-state drivers produced a 50% decrease in the use of multiplexing logic.

As is seen from Table 7, tri-state drivers are hardly used in the designs generated from library 1. But, when a 20% cheaper tri-state driver in library 2 is substituted, tri-state driver use increases drastically. It's seen from Table 7 that the use of buses is also related to the resource sharing. If

[2]Human analysis of the results, of course, took somewhat longer.

maximum resource sharing (T=0) is used, buses are more likely to be created.

Results obtained from tradeoffs between bit-steering and functional resources show that there are literally thousands of good designs possible, as shown in Table 6. In fact, ignoring routing and controller area, there are 76 equally good "best" designs.

Although, it may seem plausible to use many functional units in the designs using this library, the resulting designs actually have a functional area which is less than the upper-bound functional area (Table 5). It is also seen that there are no tri-state drivers used with library 3 in any of 1140 designs generated (Table 7) because resource sharing obviously tends to be lower when multiplexers and bus drivers cost as much as the shared resource, making buses less desirable. The best automatically generated design from Table 3 was reevaluated using library 3 and was shown to be comparatively cheaper than the others in Table 5. Further decreases in the ratios of functional module costs to data steering module costs, however might result in cheaper designs.

From the designs generated from library 3, we observed some general characteristics. We generated several designs keeping functional resource allocation constant and then varied functional resource allocation and generated several more. Repeating this process, we obtained a distribution which is roughly similar to Figure 2 when we plotted the designs points corresponding to minimum steering logic area for each functional resource allocation (non-inferior designs).

Figure 2 tells us a number of important things. The line segment between points 1 and 2 corresponds to designs with the minimum possible steering logic area (which can be non-zero in cases like having to share memory address lines, a value conditionally coming from multiple sources, etc). All designs with minimum functional resource allocation lie on the line segment from point 3 and point 4. We are not generally interested any designs to the right of point 3 or above point 1 because they are more likely to be inferior. The design points between points 2 and 3 represent designs with roughly equal total area but different architectures. The number of designs on this line segment can vary greatly depending on the characteristics of the library, the dataflow graph and the schedule. If no tradeoff is feasible between the functional units and the steering logic, this line segment shrinks to a single point. The larger the number of *tradable*[3] library modules and, the larger the number of

[3] a *tradable* module means its area is close to the area of steering logic modules

operations implemented by those library modules, the higher the likelihood of having more designs in this region (ignoring the fact that some potential designs will coincide). The final differences between areas of designs will be due to floorplanning and routing differences which are highly sensitive to the architectures generated. Although, floorplanning is ignored by most synthesis programs (an exception is [5]), it is the only criterion which can be used to determine the best design among the designs lying on the line from point 2 to point 3. The general approach in the synthesis community has been to generate designs in close proximity of point 3 (minimum functional resources), although the designs towards point 3 might generally have a higher number of nets to be routed and longer wires per net, which we postulate may produce larger designs.

We cannot yet draw conclusions about the routing and unused area for each design. However, an examination of the designs in Table 5 shows an almost-constant number of two-point nets. Even when the schedule produced by MAHA is changed, the number of nets varies by only $\pm 5\%$.

An illustration of the flexibility of MABAL for designer intervention is shown in Table 3. A simple examination of the best automatically produced design in Table 3 has revealed that the inputs to division operations and consecutive memory-read operations come from two different places. Simply moving $d1$ to stage 7 without changing the delay characteristics of the design, results in binding all input values of division operations to the same register, which saves one 8-bit-wide 2-to-1 multiplexer. The new binding patterns for other values and operations are slightly changed by MABAL, given this change. If there were a clustering algorithm available in ADAM, the whole set of bindings produced by the clusterer could be input to the MABAL program. The designer could also affect the design process by specifying the initial number of resources to MABAL which, in turn, might generate a different pattern of bindings resulting in a better design. But, even when the user does not intervene, MABAL generates designs within a few percent of the cost of the best design which can be generated by manipulating the initial resource allocation.

In order to draw conclusions about the design tradeoffs, the quality of designs produced by MABAL was investigated. To illustrate the quality of MABAL results we have synthesized other dataflow graphs known in the literature. The results for a fifth-order elliptic wave filter example and differential equation solver example are shown in Tables 8 and 9. A different library which does not allow any

tradeoff between the functional units and the steering logic was used for these examples. For more information about the results of these examples and more details about MABAL, refer to [4].

5 Conclusions

A number of conclusions have been arrived at as a result of the experiments described here. Broadly speaking, the results indicate that data-path designs are sensitive to variations in the module library. This leads us to believe module selection should precede data-path synthesis. Furthermore, if module generators, logic synthesis or logic optimization are employed, their interaction with data-path synthesis procedures must be carefully studied. Synthesis programs with a rigid design style (using multiplexers or buses exclusively) will not provide the tradeoffs shown here, and synthesis programs which do not tradeoff functional costs for data steering costs might miss some better designs.

We expect more definitive conclusions to be drawn when routing area and control area are included, when layouts are produced, and when more examples have been studied.

Routing area is not explicitly considered by MABAL, but the effect of routing area can be taken into account to some extent by incorporating the routing area per net (area taken by the average length wire) into functional units and steering logic modules. Since MABAL minimizes the number of interconnections as a result of trying to minimize interconnect area and can be forced to duplicate functional modules rather than including steering logic, the deficiency of not considering the routing area explicitly can be partially overcome. Functions like unsigned-shift or concatenation can be input to the program with some cost corresponding to the estimate of the area taken by wiring.

Since MABAL uses a greedy method in a simple and effective way, the run time of the program is negligible compared to other activities like scheduling, without sacrificing design quality. The speed up obtained here over comparable programs can effectively enable more design explorations at higher levels followed by a final extensive optimization on the interconnect.

MABAL provides a flexible method of specifying partial designs and supports some user preferences for the point to point connection model [6].

The capability of MABAL to trade off between multiplexers and tri-state-drivers and between

interconnect area and functional unit area has been shown to be important in designs which include less-complicated functional units.

References

[1] C. Y. Hitchcock, "Automated Synthesis of Data Paths", Master's Thesis, Department of Electrical Engineering, Carnegie-Mellon University, January 1983.

[2] R. Jain, K. Küçükçakar, M. J. Mlinar and A. C. Parker, "Experience with the ADAM Synthesis System", to appear in Proc. of 1989 Design Automation Conf., Las Vegas, June 1989.

[3] David W. Knapp, "Synthesis from Partial Structure", Proc. of IFIP TC-10 Conference, Pisa, September 1988.

[4] Kayhan Küçükçakar and Alice C. Parker, "MABAL : A software package for Module And Bus ALlocation", to appear in International Journal of Computer Aided VLSI Design, June 1989.

[5] M. C. Farland, "Using Bottom-Up Design Techniques in the Synthesis of Digital Hardware from Abstract Behavioral Descriptions", Proc. of 1986 Design Automation Conf., Las Vegas, June 1986.

[6] Barry M. Pangrle, "Splicer: A Heuristic Approach to Connectivity Binding", Proc. of 1988 Design Automation Conf., Anaheim, June 1988.

[7] Nohbyung Park, "Sehwa: A Software Package for Synthesis of Pipelines from Behavioral Specifications", IEEE Transactions on Computer-Aided Design, Vol 7, No 3, March 1988.

[8] Alice C. Parker, Jorge "T" Pizarro and Mitch Mlinar, "MAHA: A Program for Datapath Synthesis", Proc. of 1986 Design Automation Conf., Las Vegas, June 1986.

[9] P.G. Paulin, J.P. Knight, E.F. Girczyc, "HAL : A Multi-Paradigm Approach to Automatic Data Path Synthesis", Proc. of 1986 Design Automation Conf., Las Vegas, June 1986.

[10] Texas Instruments, "2 micrometer CMOS Standard Cell Data Book", 1986.

[11] D.E. Thomas, et. al. , "The System Architect's Workbench", Proc. of 1988 Design Automation Conf., Anaheim, June 1988.

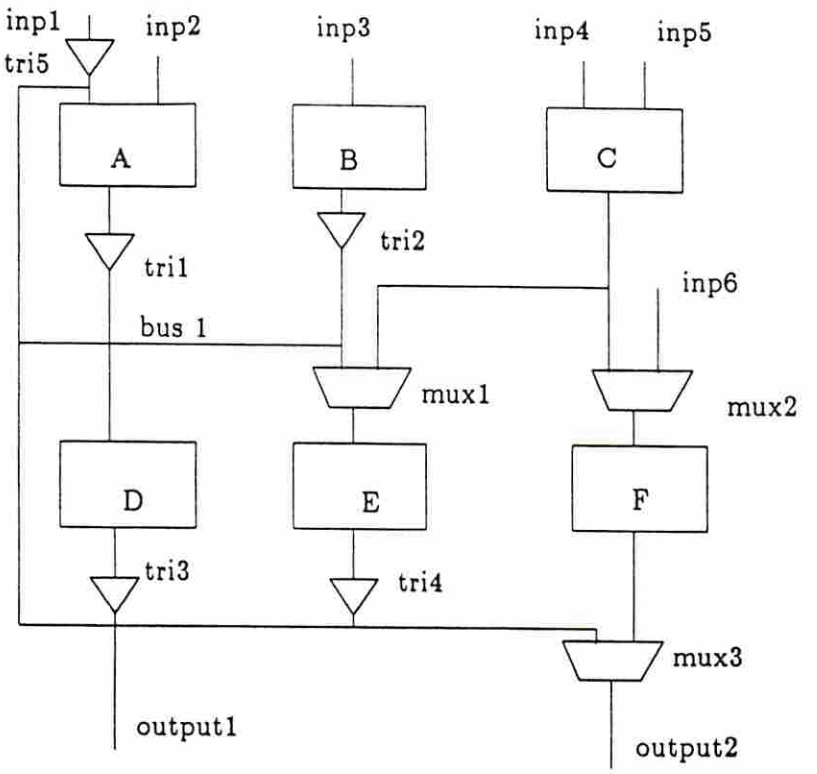


Figure 1: A possible design which might be generated by MABAL

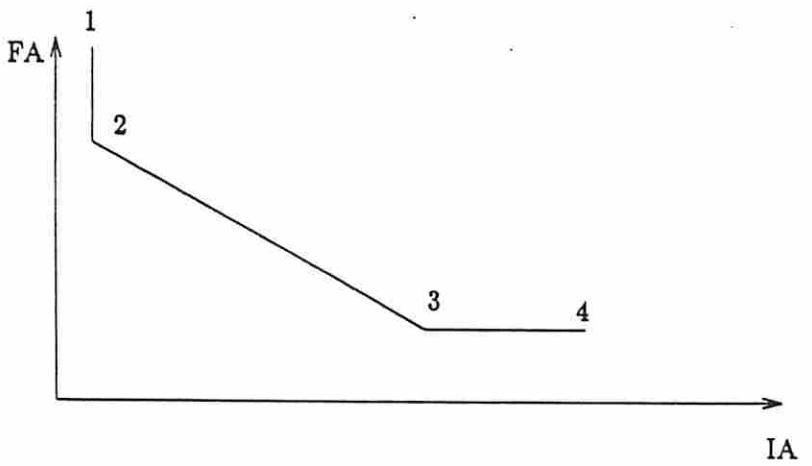
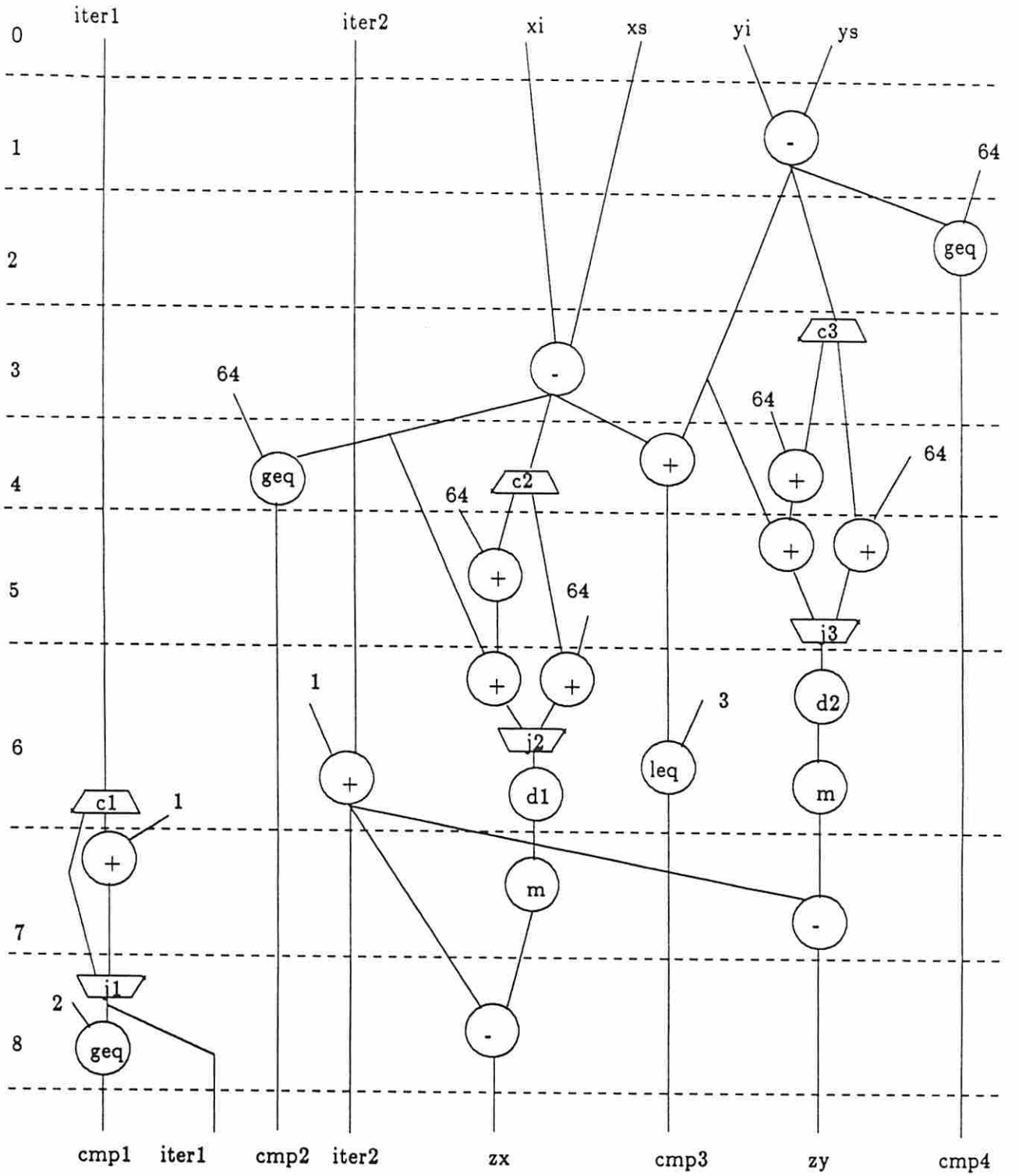


Figure 2: Distribution of designs by functional module area versus steering logic area tradeoffs



C-J : Conditional Block
All values are 8 bits except outputs of comparison operations

Figure 3: The dataflow graph used for experimentation

MC	Multiplexer (tree) count
MI	The number of multiplexer (tree) inputs
MA	Multiplexer area calculated as the number of 2-to-1 multiplexers needed to create the tree
B/D	The number of buses and number of 8-bit tri-state drivers
IA	Steering logic area (tri-state drivers and multiplexers)
TA	Normalized total active area within a table
FA	Functional active area
TAA	Total active chip area
T	1 if trading off between functional and steering logic is allowed, 0 otherwise
B	0 if only multiplexers are used, 1 if both multiplexers and tri-state drivers are allowed to be used
A	The number of adders
S	The number of subtractors
Ri	The number of i-bit registers
G	The number of greater-than-or-equal-to operators
L	The number of less-than-or-equal-to operators
x	Don't care
†	Comments in Paulin's dissertation indicate a run time of close to ten minutes
††	The best design generated from this library which used tri-state drivers
N/A	Not Available
‡	Minimum steering logic
£	one module per operation
§	Manual intervention to reduce the number of multiplexers
	CPU Time is on a Sun-3/160 for MABAL, Sun-3/260 for Splicer and Xerox 1108 for HAL.

Table 1: Nomenclature used in tables and figures

Library		1		2	3
Module	Bitwidth	Area	Delay(ns)	Area	Area
adder	8	13350	17.0	13350	5000
subtractor	8	13350	17.0	13350	5000
multiplexer	1	370	3.7	370	370
tri-state driver	1	250	2.4	200	250
register	1	1106	5.0	1106	350
register	8	8850	5.0	8850	2800
G	8	8710	16.3	8710	8710
L	8	8885	18.3	8885	8885
divide	8	0	0.0	0.0	0.0
memoryread	8	∞*	17.0	∞	∞

* Set to force the selection of only one memory unit

Table 2: Libraries used for experiments

T	B	A	S	G	L	R1	R8	MC	MI	MA	B/D	IA	FA	TAA	
x	x	2	1	1	1	3	8	12	32	20	-	59200	131763	190963	
1	0	2	1	1	1	3	8	11	30	19	-	56240	131763	188003	§
0	1	4	2	1	1	3	8	12	29	17	1/8	60320	171813	232133	††

Table 3: Selected designs generated from library 1

T	B	A	S	G	L	R1	R8	MC	MI	MA	B/D	IA	FA	TAA
1	1	2	1	2	1	3	9	9	20	11	3/9	46960	149323	196283
0	1	2	1	1	1	3	8	8	18	10	5/16	55200	131763	186963
0	1	2	1	1	1	3	8	7	17	10	6/20	61600	131763	193363

Table 4: Selected designs generated from library 2

T	B	A	S	G	L	R1	R8	MC	MI	MA	B/D	IA	FA	TAA	
1	0	4	3	2	1	3	16	6	12	6	-	17760	107155	124915	
1	1	4	3	1	1	3	16	7	16	9	-	26640	98445	125085	
1	0	3	3	2	1	3	16	7	15	8	-	23680	102155	125835	
1	1	3	3	1	1	3	16	8	19	11	-	32560	93445	126005	
1	0	5	3	2	1	3	17	5	10	5	-	14800	114955	129755	
The best automatically-produced design reevaluated from Table 3 using library 3															
0	0	2	1	1	1	3	8	12	32	20	-	59200	56045	115245	
Designs with upper bound resource allocations															
1	0	7	3	3	1	3	17	2	4	2	-	5920	133665	139585	‡
-	-	9	4	3	1	3	19	2	4	2	-	5920	154265	160185	£

Table 5: Selected designs generated from library 3

T	Area	Normalized Area	Number of designs with same or smaller area	
1	135505	1.09	2280	100.0%
1	131585	1.05	1672	73.0%
1	126005	1.01	608	26.7%
1	124915	1.00	76	3.3%

Table 6: Distribution of designs generated from library 3

T	Percentage of designs with buses	Total number of designs	Library
0	1.25	160	1
1	0.00	240	1
0	96.67	150	2
1	73.75	160	2
0	0.00	1140	3

Table 7: Statistical Data about Bus(tri-state driver) Use

Non-pipelined elliptic filter							
2 Adders			1 Multiplier				
HAL Schedule					19 Stages		
	Reg	MC	MI	MA	B/D	TA	CPU Time
MABAL	10	12	42	30	-	1.04	10.52
MABAL	10	10	32	22	2/11	1.05	14.46
HAL	12	6	26	20	-	1.00	~600 †
Splicer					21 Stages		
	Reg	MC	MI	MA	B/D	TA	CPU Time
Splicer	N/A	9	44	35	-	N/A	20257
Splicer	N/A	9	43	34	-	N/A	55
Splicer	N/A	9	45	36	-	N/A	300

Table 8: Non-pipelined elliptic filter, Case 1

Differential Equation						
Same Schedule for all Programs						
1 Adder	1 Subtractor			2 Multipliers		1 comparator
	Reg	MC	MI	MA	TA	CPU Time
MABAL	5	8	17	9	1.01	0.74
MABAL	5	6	13	7	1.01	0.74
MABAL	6	7	15	8	1.02	0.82
HAL	5	6	13	7	1.01	140
Splicer	6	5	11	6	1.00	1245
Splicer	6	5	12	7	1.01	291
Splicer	6	6	16	10	1.02	6.40

Table 9: Differential Equation