MABAL: A software package for Module and Bus ALlocation

Technical Report No. CRI-88-61

MABAL : A software package for Module And Bus ALlocation

Kayhan Küçükçakar and Alice C. Parker
Department of Electrical Engineering – Systems
University of Southern California
University Park
Los Angeles, CA 90089-0781

March 3, 1989

* This research was supported by the Department of Air Force, the Department of Army and the Department of Navy, Contract No. N00039-87-C-0194.

MABAL : A software package for Module And Bus ALlocation

Abstract

This paper describes MABAL, a Module And Bus ALlocation program. MABAL uses a simple heuristic algorithm to concurrently perform the RTL (register-transfer-level) synthesis subtasks known as functional unit allocation, register allocation, interconnect allocation and module binding. The objective of the current version of the program is to minimize the overall cost (area) of the design. MABAL is capable of building on partially specified pipelined or non-pipelined designs and it checks and corrects designers' errors which might be input to the program. MABAL is a part of the USC ADAM system and is used to produce RTL designs from behavioral specifications which have been scheduled by Sehwa or MAHA. MABAL is at least an order of magnitude faster than existing programs, while supporting a variety of novel constraints which can be imposed on the design by the user.

Key words: high-level synthesis, RTL synthesis, module binding, resource allocation, interconnect allocation, partial design, designer intervention.

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 interconnect structures has been demonstrated by early systems like the program by Hafer[2] to more recent packages with marked improvement over initial attempts. However, in order for such a synthesis tool to become a practical reality, it should possess certain attributes:

- It should be comprehensive. It should not be limited to a single style of interconnect.
- It should address constraints on input and output data availability.
- It should handle variations in scheduling of operations, including pipelining and operator chaining.
- It should allow the designer to make design decisions, either to support interfacing the design with other fragments, for testing purposes, or to allow designer intuition to intervene.
- It should be fast enough to support multiple iterations with designer-imposed decisions and constraints.
- 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.
- It should somehow take into account actual wiring costs when trading off interconnect complexity for functional complexity.
- It should be able to search different parts of the design space with different user constraints.

MABAL, a register-transfer synthesis program which concurrently performs the functions known as resource, register and interconnect allocation, and module binding, addresses all the above issues. The current objective of MABAL to produce correct designs quickly and to minimize area (cost). Performance optimization is a future MABAL objective.

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.

An important approach to design synthesis is automation and control of the design process by a design manager. This approach has been used by Knapp[10] in the ADAM system. The use of a design manager program entails that some of the design information produced by the programs be made global and easily processable by the design manager. MABAL is well suited for this purpose. It also provides the design manager with the ability to influence MABAL's 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 us to iterate frequently, spending more time on higher-level decisions which further improve the overall quality of the final design.

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 The USC ADAM System

MABAL is a part of the ADAM Advanced Design AutoMation synthesis subsystem. This subsystem consists of two sets of tools which are arranged around a multilevel design representation called the DDS (Design Data Structure) [9]. The first set of tools, which are called synthesis tools, transform purely behavioral input descriptions to RTL (Register-Transfer Level) structures which include the data path and control. The second set of tools (prediction tools) predict the area overall area-delay trade-off curve of a design without actually synthesizing the design. Three sets of inputs are required by the tools: a behavioral specification in the form of a dataflow graph, a design library and a set of constraints in the form of absolute values of area or performance. The synthesis tool set, which includes five major tools and execution order are shown shown in Figure 2.

1. An interface (future) which will read VHDL [21] behavioral input into the database;
2. SLIMOS [4] which selects the best set of functional module types to be used;
3. Sehwa[17] which performs pipelined scheduling;
4. MAHA [18] which performs non-pipelined scheduling;
5. MABAL which performs resource, register and interconnect allocation and module binding.

The prediction tools collaboratively estimate operator area [5], [6], register and multiplexer area, PLA control area (PASTA) and finally wiring area (PLEST [12]). The prediction tool set and execution order are shown in Figure 3.

The ADAM synthesis subsystem is geared to minimize the overall cost subject to a performance constraint or maximize the performance subject to a cost constraint. Because of the high complexity of synthesizing a design as a whole is not feasible, the tasks in the synthesis process (high level transformations, design style selection, module selection, scheduling and allocation) are performed in a sequential manner.

This sequential decision making can be modeled as a search tree whose nodes correspond to the decisions made at each step to drive the system. When a path in the search tree is taken, at some point, it may be discovered that the path will not lead to an acceptable design. Then, appropriate actions (generating alternative solutions at the same level or backtracking one level up to search other parts of the tree) have to be taken. Traditional synthesis systems do not provide enough flexibility to generate alternative designs based on global goals and constraints. Furthermore, in traditional systems, detecting that a path does not lead to a an acceptable solution often happens at the leaf nodes of the search tree, i.e. a complete design has to be produced. The ADAM system gives the user a flexible, fast way of pruning these infeasible decision paths, while on the other hand, is able to generate a wide range of good-quality alternative solutions at any decision level without requiring extensive effort on the part of the designer.

As a result of fast synthesis and prediction, ADAM allows the designer to iterate on the synthesis process much faster without wasting time in infeasible parts of the design space, which demonstrate that the ADAM system is more than the sum of its parts. More detailed information about the advantages of the ADAM system can be found in [7].

After a design style (pipelined or non-pipelined) has been selected, module selection is performed to decide which set of library modules to implement the operations in the dataflow graph best meets constraints. Then, scheduling is performed. At this point in the design process, operations in the dataflow graph have been assigned to time steps and it has been decided which operation type is to be implemented with which functional module type. MABAL decides on the allocation of functional modules, registers and interconnect and also the module binding to produce the complete RTL design.

At the register-transfer level, the design is in terms of floating interconnected building blocks but, the placement of blocks and the exact routing which are necessary to get detailed cost and performance figures are yet to be determined. Therefore, at the register-transfer level, if all constraints are not satisfied, the designer should continue iterating the design process. If all constraints are met, then the RTL design could be forwarded to a lower-level logic synthesis system, silicon compiler or placement and routing.

3 Related Research

There are a variety of systems and approaches to data path synthesis. Some of these systems implement general-purpose synthesis methods, whereas some specialize on specific applications like microprocessor or digital filter design. None of the current systems satisfies all of the requirements posed in Section 1. Since it is not the intention of this paper to do a wide literature survey, only the most comparable approaches will be mentioned.

The approaches for solving the allocation and binding subtasks of the data path synthesis process can be grouped into two major categories: rule-based and algorithmic methods. But, the boundaries are not distinct as some systems fall into both categories.

The algorithmic category can be further decomposed into other sub-categories based on the specific approach used. Some approaches use search-based algorithms or heuristics, e.g. Splicer [15] and some use clique-partitioning (e.g. Emerald, Facet [20]) while most others use heuristics and do not search for alternative designs.

Some approaches try to minimize only area while others try to minimize area subject to time constraints. Others perform some optimization of cost or performance with or without constraints. Backtracking of decisions is not usually supported. Eitrher bus interconnect or point-to-point interconnect styles are generally used.

Elf [1] and DAA [8] are rule-based systems. HAL [19] performs operator allocation with a heuristic rule-based system and uses clique partitioning for register allocation. The binding phase and interconnect generation are also rule-based. Emerald and Facet use clique-partitioning methods for solving all allocation and binding problems. EMUCS [3] [14] is based on McFarland's greedy, heuristic algorithm which binds operations and values one at a time to minimize a cost function. The intuitive idea in EMUCS is to give priority to decisions which may have the worst impact on design if delayed in the incremental binding process. In other words, the idea is to minimize the blocking of future important decisions by less important current ones. At each step, an update of a cost table containing the costs of all bindings is made. This process of updating is time consuming; however a much faster local update method can be used at the user's option, at the expense of missing some of the good designs which could be found otherwise. IMBSL [11] is a system which also accepts partial structures, allows designer intervention, and constructs a layout model for detailed analysis of area and timing.

4 Overview of MABAL

4.1 Problem Statement

The basic problem MABAL solves includes allocation of hardware units and module binding. 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 optimal number of functional units can be calculated using existing theory [16], prior to RTL synthesis there is very little known about the interconnect for a given design, whose cost is highly dependent on the way in which module binding is performed. 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.

4.2 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 actually searching that larger portion of the design space. In this way, the negative consequences resulting from decisions which are not suitable for the final design are reduced. MABAL might start with no structural design or a given structural design in place. 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.

4.3 Features of MABAL

MABAL concurrently performs functional unit allocation, register allocation, operation/value binding and interconnect allocation. Module binding is performed with the objective of minimizing total cost by selecting the least-cost choice between allocating more hardware units or interconnect.

MABAL contains a number of features. 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 also 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 constructing interconnect which is not restricted to a single style and checking for errors in designer's inputs.

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 one of four types: constants which do not require storage, variables which are used for multiple iterations, variables which are 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.

Interconnect allocation consists of allocating multiplexers, bus drivers and carriers. There is no enforced bus style. The resulting interconnect is decided entirely by making trade-offs between using more multiplexers, bus drivers, functional hardware units or registers subject to bindings and restrictions requested by the user.

One of the key features of MABAL is the ability to allow the designer to intervene and influence the design process to some degree, correcting designers' errors which are introduced, while automating the tasks that the designer does not want to control. MABAL is able to produce good results without requiring designer intervention. If the designer intervenes, a variety of items can be specified such as: the number of functional units and registers, the binding of any operation or value to functional units or registers, and the type of interconnect. The degree of manual intervention is not fixed and the method used is able to cope with varying degrees of designer intervention. The program can, to some extent, take advantage of other approaches to the RTL design process with the capability of 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 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 trade-offs between using more functional area or interconnect area.

5 MABAL Implementation and Use

5.1 Assumptions

In our approach to this synthesis problem, the following assumptions hold in order to be consistent with the remainder of the USC ADAM system. In practice, these assumptions do not severely restrict the design space explored.

- Any operation has to be completed within one clock cycle. Slow operations are assumed to be partitioned into multiple single-cycle operations prior to synthesis.
- Scheduling has to performed prior to running MABAL.
- Module style selection has to be done prior to both MABAL and the scheduling task.

- Operations the designer desires to implement with different hardware module types must have been given different type names even if they perform identical functions. Usually, there is more than one hardware module type which can implement an operation type; the module selection program which runs prior to MABAL has decided which one is better to use. Any number of operation types can be implemented by an ALU.
- The final design is synchronous with a single clock.

5.2 Inputs

The required data input by MABAL consists of three types :

- A scheduled data flow graph with information about conditional branches,
- a set of library components which can implement all operations, and
- optional structural information in addition to the required data:
  - an initial resource allocation which might be changed by the program,
  - bindings of operations and values to hardware modules,
  - restrictions on the use of bussed connections at given inputs of given modules, or throughout the design, and
  - a degree of tendency to use more hardware modules or more interconnect at individual decisions even though the other choice is cheaper.

Although an entire RTL design can be specified as an input to MABAL, the designer must use the restricted interconnect model used by MABAL. Multi-level interconnections, buses, arbitrary bindings of values to time steps, multi-phase clocking schemes and complicated interconnect elements (e.g. inverting bus drivers) are not allowed. Although specifying buses to MABAL is easy, it has not been implemented yet because of the fact that the global effects of predefined buses on the algorithm used by MABAL are not predictable.

5.3 Outputs

MABAL completes data path synthesis and generates an RTL design without the controller. It specifies all functional units, registers, multiplexers, bus drivers, inputs, outputs, connections between them and the bindings. Each operation is bound to a single module, whereas each value can be bound to a number of wires and registers if necessary. The input arrangements of multiplexers are arbitrary and can be optimized by a controller synthesis program.

5.4 Overview of MABAL Operation

The algorithm used by MABAL does not differentiate between operations and values, between functional units and registers, or between resource and register allocation. E.g., a value needing storage will be assigned a storage-operation. Therefore, the terms operation, hardware module and resource allocation will be used in a broader sense covering both terms in each case.

MABAL implements switching hardware using multiplexers and bus drivers along with wire connections. Multiplexer outputs are not shared and multiplexer-tree outputs are connected solely to the inputs of hardware modules. When it is cheaper to use buses and there is no time conflict, buses are created and connected to the module ports. A module may drive more than one bus and those buses may be isolated from each other. The inputs to the modules can be directly connected to the buses. The outputs of modules can be tri-stated or a simple wired fanout tree can be used.

Buses are dynamically updated as the design progresses. Each binding may add new drivers to buses or remove some of the existing drivers. Buses can be combined or divided into smaller buses, if necessary. If having a specific bus is not feasible because of a bus conflict or the resulting cost is higher than using multiplexed connections, then the bus is completely removed from the design. All of these decisions are done locally and incrementally during the design process. Parts of the design which are unrelated to the decision being made are not processed. Connections, buses and bindings are used to access the parts of the design which may undergo a change or must be examined to make a decision.

6 The MABAL Algorithm

A pseudo language has been used below to illustrate the fundamentals of the MABAL algorithm where

- i is an operation node
- $m(i)$ is a hardware module type which will be used to implement operation i.
- $m(i)^j$ is the j^{th} hardware module of module type $m(i)$.
- $m(i)^j_*$ is the set of bindings for hardware module j of module type $m(i)$.
- $old(x)$ is the cost x existing prior to the binding currently considered.
- $muxcost$ is the cost of having a multiplexed connection to a hardware module port.
- $buscost$ is the cost of having a bussed connection to a hardware module port.
- ic is the incremental cost of an individual decision, i.e a binding.
- generosity factor is the degree to which MABAL uses more hardware modules rather than interconnect.

The main algorithm tries to minimize total cost by trading-off between module cost and interconnect cost for each binding decision. In order to make this decision, the algorithm trades off between multiplexers and bus drivers and calculates the interconnect cost introduced at each binding step. The algorithm can be described in an intuitive and simplified way as follows:

```
Bindtominimizecost()
Begin
Sort all $i$ operations according to their partition numbers and within that order,
   according to precedence order in the data flow graph.
For all $i$ in the sorted order
   For all modules $m(i)^j$ already existing in the design
      If $i$ is bindable to $m(i)^j$
         Calculate the binding cost of $i$ to $m(i)^j$
         If $Bindingcostof(i\, to\, m(i)^j) \geq$ (module cost / generosity factor)
            or $i$ is not bindable to any $m(i)^j$ /* due to schedule conflicts */
               Allocate $m(i)^{j'}$          where $j' = max(j) + 1 \quad \forall j$
               Bind $i$ to $m(i)^{j'}$
         else
```

```
            Make a tentative decision about local interconnect
            Make a permanent decision to bind $i$ to $m(i)^j$
          endif
        endif
    For all $m(i)^j$
      Make final decisions about the interconnect, given the bindings of all nodes

    Try to merge buses into larger buses
    Discard unused pre-allocated modules and unnecessary tri-state-drivers
    End.
```

The binding cost of an operation to a module can be calculated by summing the cost of additional interconnect which must exist for each port of the module. The part of the algorithm which decides how the module should be connected to the rest of the design given some already existing bindings and a new candidate for binding can be outlined as follows:

```
Bindingcostof($i$ to $m(i)^j$)
Begin
$ic = 0$
If $m(i)^j_*$ is nil
then
  Choose direct wire connection for all ports
  /* There is no incremental interconnect cost for this case */
else
  For all input ports of $m(i)^j$
    case { the existing connection type at the current input port }
      Single wire connection :
        calculate $muxcost$, $buscost$
        if $muxcost < buscost$
          then
            choose multiplexed connection
            $ic = ic + muxcost$
          else
            choose bussed connection
            $ic = ic + buscost$
        endif
      Multiplexed connection :
        calculate $muxcost$, $buscost$
        if $muxcost - old(muxcost) < buscost - old(muxcost)$
          then
            choose multiplexed connection
            $ic = ic + muxcost - old(muxcost)$
          else
            choose bussed connection
            $ic = ic + buscost - old(muxcost)$
        endif
      Bussed connection :
        calculate $muxcost$, $buscost$
```

```
            if $muxcost$ - $old(buscost) < buscost$ - $old(buscost)$
              then
                choose multiplexed connection
                $ic = ic + muxcost$ - $old(buscost)$
              else
                choose bussed connection
                $ic = ic + buscost$ - $old(buscost)$
            endif
   End Case
   For all output ports of $m(i)^j$
     {A similar algorithm applies}
endif
End
```

7 Experimental Data and Results

In a single paper, there is not adequate space to provide examples which illustrate all MABAL's capabilities. For this paper, MABAL's output designs were compared to others. Results obtained were comparable, in spite of the many additional features supported by MABAL and MABAL's rapid run times (at least an order of magnitude faster than HAL, Splicer and EMUCS). The elliptic wave filter and differential equation designs are too restricted to exploit the capabilities of MABAL but they are used in this section to compare against results obtained by other programs.

Exploiting the designers' ability using MABAL to control I/O latching and timing and to specify partial structures a priori would have been resulted in designs different from those described here.

In this section, two examples are used to illustrate the results obtained by MABAL. The first example is a fifth-order elliptical wave filter used in several publications. A two-stage multiplier is used in non-pipelined designs generated by all programs.

The pipelined designs generated here are designed to operate on a number of different interleaved sets of data since it is not possible to have pipelined designs with output values required for initiation of the next task in the sequence. The pipelined schedule has a 13 clock-cycle delay between successive initiations. The pipelined schedule has been selected such that the functional resource allocation is the same as other non-pipelined examples for easier comparison. Two designs are given in Table 2, one with buses, one without.

Multiple designs from MABAL are obtained by giving different initial resource allocations to the program and/or by restricting the use of buses. The resource allocation in Table 4 is known to be inferior but is included here to make a comparison with EMUCS, which used this resource allocation.

Interconnect and total cost calculations for HAL and Splicer are made using the same library MABAL used. This library has a moderately-sized adder and subtractor and a multiplier twice as large as the adder. The sizes of these functional units do not dominate the total cost. If larger, faster modules were used, the differences between the areas of designs generated by the different programs would decrease. This would not be a fair comparison in the sense that programs may produce relativelt better or worse results with different libraries. Also a specific schedule can result in good quality designs for one datapath allocator but not for another datapath allocator which is not designed to work in harmony with that specific scheduling style. In spite of this, in order to compare to existing programs, we have tried to use their schedules whenever it is possible. Unfortunately, this is the only possible method of comparison for the time being until enough number of benchmarks are available to have more statistically significant data.

The CPU times are measured on a Sun 3/160 for MABAL. The CPU times for Splicer are also measured on a Sun 3/260. HAL's CPU times are on a XEROX 1108. EMUCS' CPU times are on a DEC VAX II/GPX.

The second example is the differential equation description taken from the literature. This description is fairly small but it is included here to have comparison with other programs. The schedule and resulting functional resource allocation are the same for all programs. Detailed comparisons with other programs are given in Tables 1 through 5 on pp 18 - 22.

8 Analysis of MABAL's Performance

MABAL is guaranteed to obtain an optimal resource allocation for operators. If functional unit costs are comparable to interconnect costs and the user does not require the minimum resource allocation, MABAL has yielded designs in which the resource allocation is not optimal. However, the total cost for these designs is less than designs with optimal resource allocation. Depending on the cost of a hardware module, buying another module can be cheaper than buying more interconnect.

The algorithm is reduced to the REAL [13] algorithm for register allocation if the designer decides to start with no registers in place, and requires that modules should not be traded off with the interconnect. This does not necessarily result in globally better designs. The register allocation algorithm used in REAL is known to be optimal only with respect to the number of registers needed, if the data flow graph is unconditional.

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 into multiplexer and bus driver costs. Since MABAL minimizes the number of interconnections as a result of trying to minimize interconnect area and can be forced to duplicate functional modules, 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 estimation 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 approximate complexity of the algorithm is

$$O(\sum_i n_i * o_i * p_i)$$

where i is the number of operation types, n_i, o_i and p_i are the number of operations, number of modules and number of ports of module type i, respectively. MABAL's complexity is less than HAL, Splicer or EMUCS.

9 Conclusions

MABAL provides a flexible method of specifying partial designs and supports some user preferences for the point to point connection model [15]. This capability allows importing other design approaches or partial designs. For example, clustering or clique partitioning can be used for module binding and the results can be input to MABAL.

The capability of MABAL to trade-off between multiplexers and tri-state-drivers and between interconnect area and functional unit area is quite important in designs which have less-complicated functional units.

The speed up obtained here over existing programs, can effectively enable more design explorations at higher levels followed by a final extensive optimization on the interconnect.

The model used by MABAL does not have explicit information about the routing area, although the model can still be used to some extent by incorporating the average area corresponding to a two-point net into costs of modules.

10 Future research

Although good results have been achieved, there are a few more enhancements to be made in order to get results comparable to or better than the human designer.

MABAL currently does not share multiplexers outputs. A post processing algorithm to share multiplexer outputs will be written to enhance the quality of designs generated.

The minimization of cost subject to time constraints or vice versa are also subjects to be examined carefully to get better designs which have characteristics closer to the designer's specification.

References

[1] Emil F. Girczyc, "Automated Generation of Microsequenced Data Paths to Realize ADA Circuit Descriptions", Doctoral Thesis, Department of Electronics, Carleton University, Canada, 1984.

[2] L. Hafer and A. Parker, "Register-Transfer Level Digital Design Automation: The allocation process", Proc. of 1978 Design Automation Conf., Las Vegas, June 1978.

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

[4] R. Jain, Alice C. Parker and N. Park, "Module Selection for Pipelined Designs", Proc. of 1988 Design Automation Conf., Anaheim, June 1988.

[5] R. Jain, M. J. Mlinar and A. C. Parker, "Area-Time Model for Synthesis of Non-Pipelined Designs", Proc. of International Conf. on Computer-Aided-Design, Santa Clara, November 1988.

[6] R. Jain, A. C. Parker and N. Park, "Area-Time Model for Synthesis of Pipelined Designs", Proc. of 1987 Design Automation Conf., Miami Beach, June 1987.

[7] Rajiv Jain, Kayhan Küçükçakar, Mitchell J. Mlinar and Alice C. Parker, "Experience with the ADAM Synthesis System", to appear in Proc. of 1989 Design Automation Conf., Las Vegas, June 1989.

[8] T. J. Kowalski and D. E. Thomas, "The VLSI Design Automation Assistant: Prototype System", Proc. of 1983 Design Automation Conf., Miami Beach, June 1983.

[9] D. Knapp and Alice C. Parker, "A Unified Representation for Design Information", Proc. of CHDL-85 Conf., North-Holland, August, 1985.

[10] David W. Knapp, "A Planning Model of the Design Process", Doctoral Thesis, Department of Electrical Engineering - Systems, University of Southern California, December 1986.

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

[12] Fadi J. Kurdahi and Alice C. Parker, "PLEST: A Program for Area Estimation of VLSI Integrated Circuits", Proc. of 1986 Design Automation Conf., Las Vegas, June 1986.

[13] Fadi J. Kurdahi and Alice C. Parker, "REAL: A Program for REgister ALlocation", Proc. of 1987 Design Automation Conf., Miami Beach, June 1987.

[14] John Anthony Nestor, "Specification and Synthesis of Digital Systems with Interfaces", Doctoral Thesis, Department of Electrical Engineering, Carnegie-Mellon University, April 1987.

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

[16] Nohbyung Park, "Synthesis of High Speed Digital Systems", Doctoral Thesis, Department of Electrical Engineering - Systems, University of Southern California, December, 1985.

[17] Nohbyung Park and Alice C. Parker, "SEHWA: A Program for Synthesis of Pipelines", Proc. of 1986 Design Automation Conf., Las Vegas, June 1986.

[18] 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.

[19] 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.

[20] Chia-Jeng Tseng and D. P. Siewiorek, "Automated Synthesis of Data Paths in Digital Systems", IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems, Volume CAD-5, Number 3, July 1986.

[21] "IEEE Standard VHDL Language Reference Manual", The Institute of Electrical and Electronics Engineers Inc, March 31, 1988.

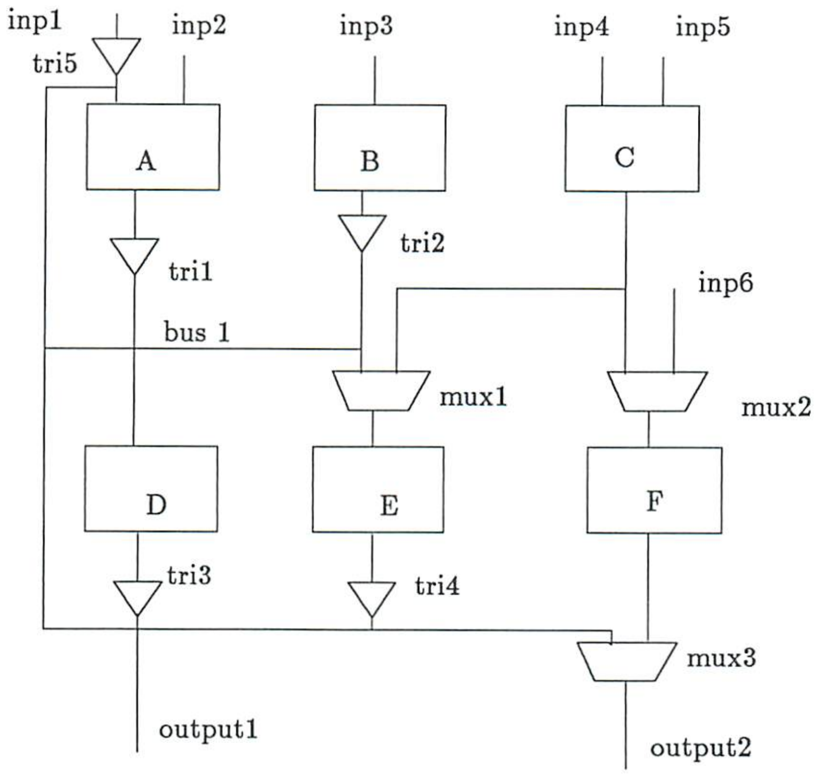

Figure 1: A possible design which might be generated by MABAL

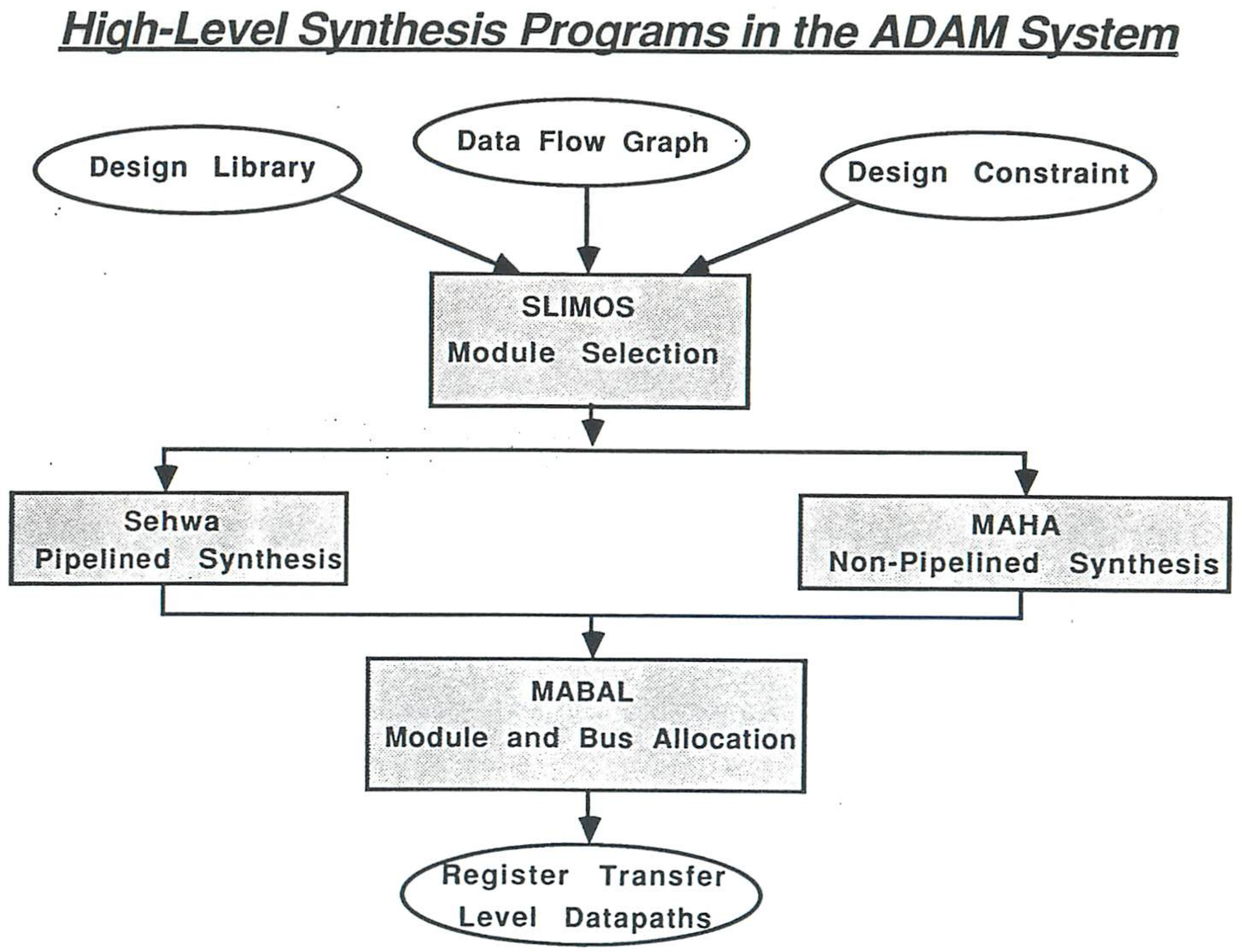

Figure 2: High Level Synthesis Tools in ADAM System

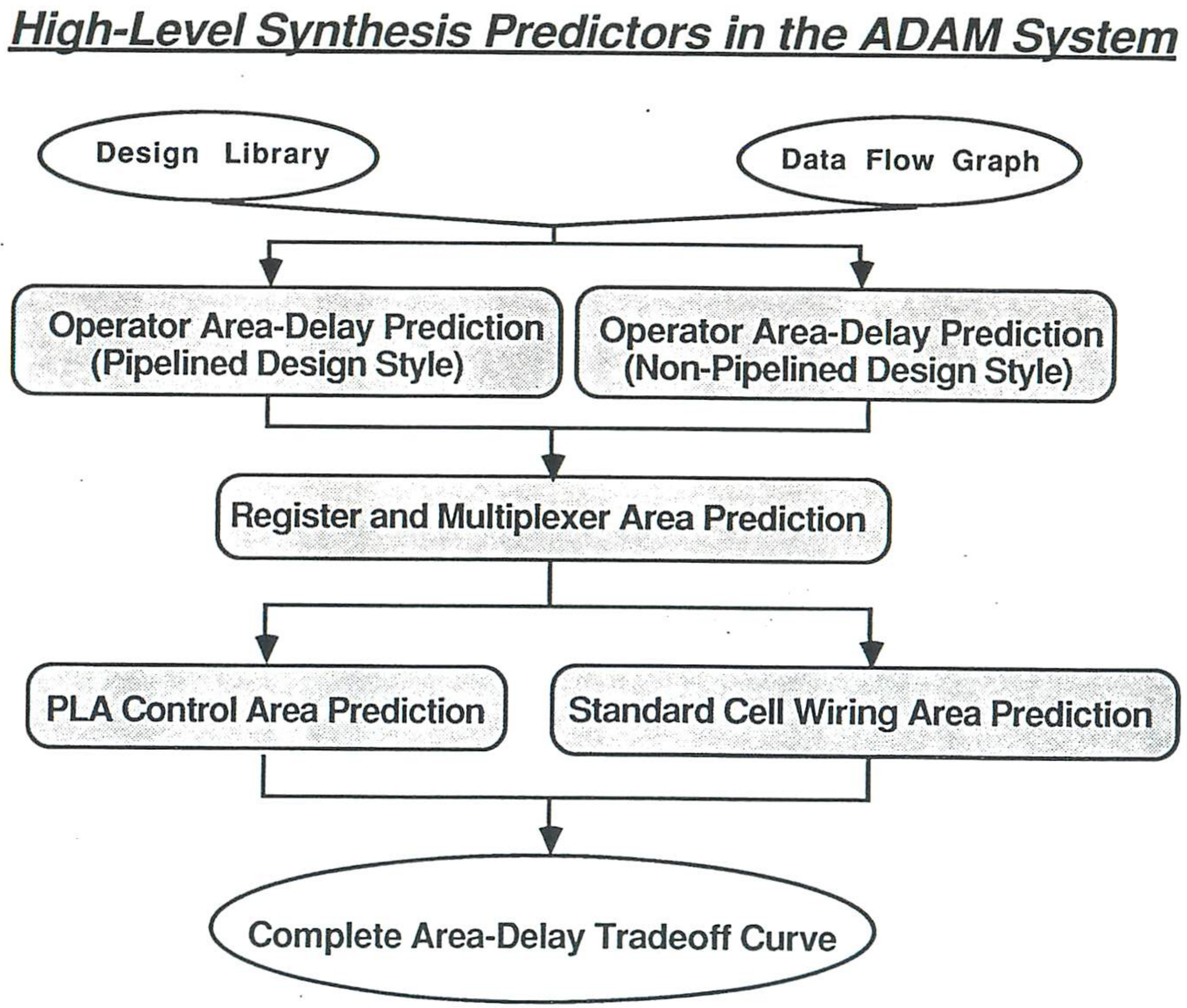
High-Level Synthesis Predictors in the ADAM System
Design Library
Data Flow Graph
Operator Area-Delay Prediction (Pipelined Design Style)
Operator Area-Delay Prediction (Non-Pipelined Design Style)
Register and Multiplexer Area Prediction
PLA Control Area Prediction
Standard Cell Wiring Area Prediction
Complete Area-Delay Tradeoff Curve

Figure 3: Prediction Tools in ADAM System

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 tri-state drivers
IA:	Interconnection active area
TA:	Normalized total active area within a table
TAA:	Total active chip area
CPU Time is in seconds.	
All of the bus drivers and multiplexers are 16-bit wide	
†:	Comments in Paulin's dissertation indicate a run time of close to ten minutes
††:	Semi manually pre-partitioned design
N/A:	Not Available

Table 1: Definition of terms used in the following tables

MABAL						Pipelined elliptic filter		
2 Adders			1 Multiplier			17 Stages		
	Reg	MC	MI	MA	B/D	IA	TAA	CPU time
1	16	8	17	9	5/28	4432	26632	10.76
2	16	13	45	32	-	4608	26808	7.48

Table 2: MABAL - Pipelined elliptic filter

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 3: Non-pipelined elliptic filter, Case 1

Non-pipelined elliptic filter							
2 Adders			2 Multipliers				
EMUCS					19 Stages		
	Reg	MC	MI	MA	B/D	TA	CPU Time
EMUCS	12	14	50	36	-	1.06	N/A
EMUCS ††	12	12	34	22	-	1.00	127
MAHA Schedule					21 Stages		
	Reg	MC	MI	MA	B/D	TA	CPU Time
MABAL	11	11	33	22	2/14	1.04	16.70
MABAL	11	13	43	30	-	1.03	10.22

Table 4: Non-pipelined elliptic filter, Case 2

Differential Equation						
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 5: Differential Equation