MAHA: A Program for Datapath Synthesis

Alice C. Parker Jorge "T" Pizarro Mitch Mlinar

Department of Electrical Engineering-Systems
University of Southern California

Abstract

MAHA is a program which implements an algorithm for register level synthesis of data paths from a data flow specification. The algorithm is based on a linear hardware assignment to critical path nodes, followed by a cost-based assignment using the concept of the **freedom** of a node to be scheduled. Functions with the least scheduling freedom are scheduled first. The program either minimizes cost, subject to a time constraint, or maximizes speed subject to a cost constraint. The implementation of this algorithm is presented using examples from the literature.

MAHA is written in Franz LISP, and executes within minutes for problems of practical size on a VAX 11/780.

1 Introduction

Register-level synthesis is composed of several tasks, including allocation of values to registers and operations to operators, scheduling when operations can or must occur, composition or construction of operators and registers from primitive components, and optimization by application of transformations and exploration of alternative designs. There are several different constraints associated with synthesis involving area, power and time minimization. For each operator, allocated area includes not only functional area, but also the associated interconnect, power and ground routing, control hardware and routing of control signals. Another important consideration in many cases is the amount of power available for the design. Finally, the overall delay allowable for completion of the operations may be highly constrained.

Flexibility is important for synthesis programs to be practical. Such programs must adapt to changing constraints as applications change, and must meet either cost or speed constraints, or sometimes both, depending on the application. Since most synthesis problems are NP-complete, programs cannot investigate all alternatives. Thus, they must either make decisions in the "best" order, or must be able to backtrack or restart.

Current datapath synthesis programs are experimental - they each meet some requirements, but no one existing program meets all of the above. Two related approaches which have been taken to design synthesis are EMUCS by Hitchcock [3] and an ADA to standard cell algorithm by Girczyc [1]. EMUCS is based on an algorithm by McFarland [6] . It starts with a Value Trace representation from which tables reflecting the need to use, create, or modify a processing element in order to bind an operation to that element are created. EMUCS selects a binding which will have the smallest impact on the nodes which have not been chosen for hardware asssignment. EMUCS does not consider the critical path, and has as a single, fixed goal; the minimization of cost. The approach by Girczyc takes an ADA control/data flow graph (similar to the Value Trace) which has been functionally optimized and makes hardware assignments by adding cells composed of a register, multiplexer and operator. This approach does consider the critical path, but always attempts to minimize cost, subject to timing constraints. Thus, both of these approach the problem by using cost-based greedy algorithms.

The work described in this paper is similar in intent to the algorithm designed by McFarland, but actually has its roots in a control synthesis algorithm developed by Nagle [7]. Nagle's notion of **freedoms** (attraction weights) has been directly applied to this research.

In order for a program to meet the requirements on flexibility, a program has to make decisions in some order such that earlier decisions do not overly constrain later decisions. It must also have some method for computing the effect of a single design decision on the overall cost and speed of the resulting hardware.

Thus, the synthesis problem we describe is as follows: The program must input a data flow description of the hardware behavior, and must output a datapath structure, consisting of registers, operators, and required interconnections, along with a time schedule giving the ordering of operations. The program must make the most constrained decisions first, so that the ordering of decisions does not greatly affect the optimality of the resulting design. The program should adjust either to cost or speed constraints, and it should be able to measure the impact of each design decision, to avoid large amounts of searching of the design space. Finally, the program should be able to restart or

0738-100X/86/0000/0461$01.00 ©1986 IEEE

backtrack when it is clear that constraints will not be met with the current strategy.

The next section of this paper describes the operation of the Modified Automatic Hardware Allocator (MAHA). MAHA is part of the ADAM (Advanced Design AutoMation) system [2]. The detailed algorithm is then described, followed by two small examples MAHA has synthesized. Finally, conclusions are drawn.

2 Overview of the MAHA Algorithm

MAHA carries out the synthesis tasks described above in the following manner: First a list of the nodes is read by MAHA, followed by a list of edges connecting those nodes. It locates the critical path and divides the path into n time steps of equal duration. Each time step now represents one minor cycle or register transfer. MAHA allocates operators for the critical path in a first-come first-served fashion, with multiple operations sharing resources as long as the operations do not occur in the same time-step partition.

For the off-critical-path nodes, we introduce the notion of **freedom**. The freedom of a node is defined as the difference between the time when the input values are needed by a given operator, and the time when the result of that operation is required, less the propagation delay of the hardware. Once the critical path has been assigned, the remaining nodes are assigned by computing the freedoms of each node and assigning the node with the least freedom.

Thus, the first operations scheduled, which may have scheduling difficulties, get the first chance to share resources. Operations scheduled later may find most resources fully utilized; however they have the greatest scheduling flexibility, and thus are more likely to find a free resource in some time slot. MAHA adds resources as necessary when it schedules the operations.

At some point, MAHA may run out of resources due to a cost constraint. At this point, it repartitions the critical path into more time steps, and begins allocation over. Adding more time steps allows scheduling the resources for more operations than before; thus fewer resources may be required. As time steps are added, the timing constraint is checked to insure that it is being met. If cost is to be minimized, MAHA will add time steps as long as it can without violating the timing constraint; if speed is to be maximized, MAHA will add resources as long as it can without violating the cost constraint.

Each node represents an operation to be performed by a piece of hardware. The critical path nodes, which includes the nodes on multiple critical paths, are removed from the set of all nodes and are assigned to hardware first. Since the critical path nodes are sequential, there are no timing conflicts. Also, since the critical path determines the overall speed, doing the critical path assignment first ensures the fastest possible resulting design. Further, once the critical path has been assigned, we have a lower bound on the total cost. Thus, the approach used in our algorithm allows us to partition the problem.

The method for computing the critical path is a program by N. Park called the Clocking Scheme Synthesis Package (CSSP) [8] . This program takes a set of nodes and edges and computes optimal clocking schemes, the critical path, and clock cycle times for single and multi-phase clocks. Given this information, hardware can be assigned to clock periods; specifically, critical path nodes are assigned to slots and bound to hardware modules for that clock cycle. Nodes which are not on the critical path could share this hardware during other clock cycles.

A complete description of the algorithm follows:

1. Take the data flow graph and assign delay values to each node (operator).

2. Use CSSP to find the critical path.

3. Given the overall timing constraints, use CSSP to find the optimal number of steps in the critical path.

4. Allocate and bind hardware resources to each critical path operation.

5. If there are no more nodes to be assigned, the algorithm terminates.

6. Compute the freedoms of all non-critical path nodes.

7. Assign the node with the smallest freedom.

   - If hardware can be shared, we allocate the hardware to that node and go to step 5.

   - If there is no hardware that can be shared, we determine whether to add another hardware resource and go to step 5, or add another time slot and go to step 3.

   - If both constraints have been exceeded, we start over at step 1 with a new set of constraints.

Several advantages are apparent in this algorithm. First it ensures that the hardware for the critical path has priority, which will improve the speed of the design. Next, the critical path assignment is completed in linear time. This removes a subset of the total nodes from the computationally more expensive parts of the algorithm. The assignment of off-critical-path nodes is done in n^4 time in the worst case, but on a smaller set of nodes since the critical path nodes have already been assigned.

3 The MAHA Program Structure

The MAHA program is written in Franz LISP. MAHA is divided into four sections: library generation, data flow graph partitioner, critical path analyzer and allocator, and off-critical path analyzer and allocator. The database MAHA uses is the node-list and edge-list description of the data flow graph as well as the hardware library selected by the user. Allocation is bounded by maximum time and/or area (cost) for the data flow graph entered by the user; if both are given, the user must choose which of time and area is higher priority.

The first task MAHA performs is generating a list of "average component parameters". After selecting the hardware library, MAHA analyzes each type of operation performed at a node and generates a list of all components in the library which can perform it. During synthesis, there is no checking for either cost or speed of individual components; rather, each list is reduced to one entry which has as its properties the average speed and cost of all components in the original list which perform the same operation. Having an average library simplifies hardware assignment as each node has only a single component in the library which can implement its function. It also allows us to center the design at the chosen point in the design space.

Next, MAHA determines the critical path or paths through the data flow graph by using CSSP [9] . Using the average library, each node is assigned a propagation delay equal to its component delay. All possible paths from the root node(s) to the output(s) are followed and the maximum delay at each node is retained. Each node is marked as it is visited to prevent traversing any portion of the graph more than once except to propagate the new maximum delay. Hence, the critical path is found in linear time.

After finding the critical path(s), the clock-cycle time is chosen. The maximum delay of any node in the critical path(s) is usually chosen as the cycle time. The data flow graph is then optimally partitioned into **n** stages where n is less than or equal to the number of nodes in the critical path, using the optimal clocking synthesis algorithm (CSSP) developed by N. Park [9] . After partitioning the data flow graph into **n** stages, the total path delay is analyzed. If the critical path delay exceeds the time bound set by the user, an attempt is made to partition using **n-1** stages. When the trivial case of one stage is reached, MAHA exits with the appropriate message:

Number of stages equals 1.

Re-partitioning continues until either a partition is found which meets the maximum time constraint or the trivial case where the number of stages equals one occurs.

Allocation of the critical path can now be accomplished. MAHA sequentially assigns hardware to each node in the critical path, keeping track of hardware usage and cost. An "assigned hardware" list is generated which contains all "purchased" hardware and the time range(s) currently assigned. Since each node has only a single average component type which can be used, allocating hardware to a node operation only requires analyzing the usage of previously allocated hardware. If an average component of the right type has been previously assigned, the corresponding time ranges are examined to determine if that particular instantiation can be reused. If previously purchased hardware can not be shared, a new piece of hardware is purchased (a new average component is instantiated). This maximal sharing algorithm is performed on the data flow graph in linear time.

If the maximal cost is exceeded during critical path allocation, MAHA checks if partitioning the data flow graph into more stages is allowed. The decision is based upon the partitioning history. If MAHA had not previously re-partitioned with less stages to meet the time constraint, partitioning is performed with more stages and the critical path delay is checked against the constraint. If MAHA had previously partitioned with less stages, partitioning into more stages will occur only if the cost constraint has priority over the time constraint. In either case, a message is sent to the user allowing him/her to make the decision instead.

After completing the critical path allocation, off-critical path nodes are assigned hardware. Initially, each node which has not been allocated hardware is analyzed to determine its freedom. Essentially, freedom is the time range in which an operation can be performed without lengthening the critical path. All parents and children of the node being analyzed are analyzed and marked similiar to the critical path algorithm; the worst case parent and child delays are retained. A list of freedoms for all non-allocated nodes is formed in linear time.

The off-critical path node with the smallest freedom (tightest constraint) is chosen for allocation. If the freedom is so small that the operation must occur during one specific time stage, then that stage is assigned to the node. However, it is more common to see freedoms which allow a node to be assigned to any one of a number of consecutive stages. In this case, the hardware allocated in the allowed stages is sequentially examined and the first stage where hardware sharing can occur is assigned to the node. If none of the stages allows resource sharing, the earliest stage is arbitrarily assigned to the node.

Once the stage has been assigned to the chosen off-critical path node, allocation occurs identical to that described for the critical path nodes. Exceeding the maximum cost may result in re-partitioning of the data flow graph, forcing re-allocation of all hardware. After allocating the chosen off-critical path node, MAHA starts again by calculating new freedoms for all remaining nodes. This process continues until all nodes have been allocated hardware.

Upon completion, MAHA lists the component types used and all instantiations of them including the time ranges where they were utilized. In addition, MAHA computes the utilization factor for each element as well as the

overall hardware usage. As an aid to other portions of the ADAM system, MAHA generates a list of the time range assigned to each node.

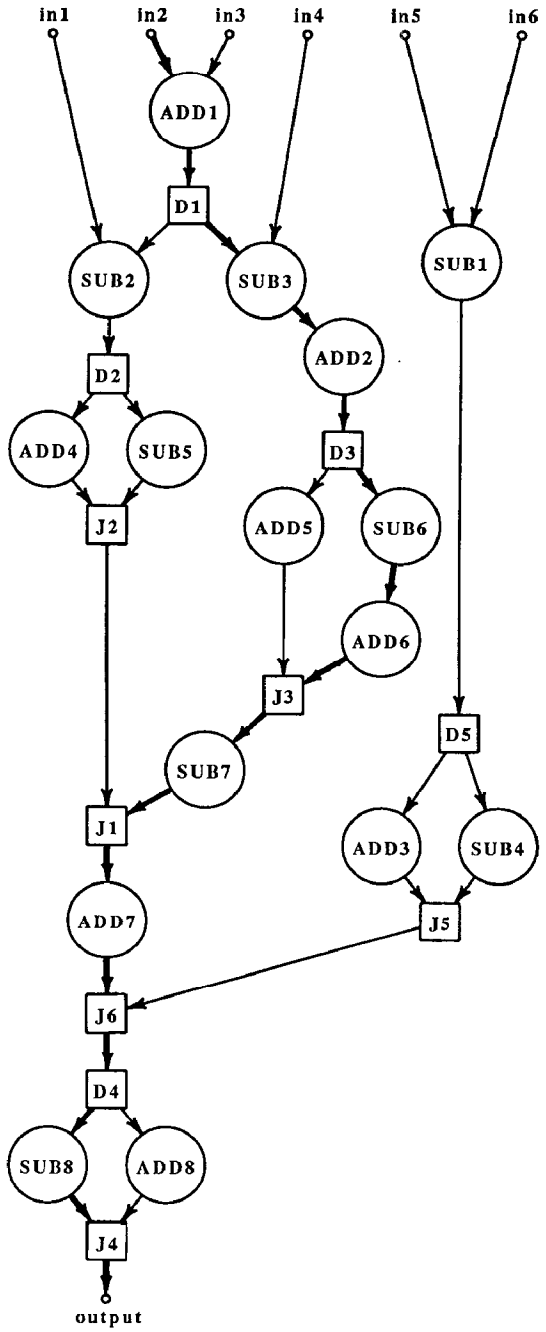

Figure 1: data flow graph for the first example

4 Two examples

To test the program, two examples were chosen from current literature. The first example chosen is from Park [9] and is shown in Figure 1; it was selected due to its complexity and potential for resource sharing. The overall constraints of cost and speed determine the probable number of clock cycles for partitioning the data flow graph. For this example, two extremes were forced: minimum delay and then minimum cost. The **D** and **J** nodes indicate parallel branches.

The minimum delay is forced by decrementing the number of clock cycles (stages) allowed and determining the clock cycle time required for each stage count. An overall delay is calculated and the fastest is chosen. If two different stage counts produce the same delay, the alternative with the highest cost is rejected. Similiarly, the stage count with the minimum cost can be found where equal cost alternatives differ in their maximum propagation delay (the fastest being chosen).

The cheapest design occurs with the maximum number of stages possible (8) since extensive resource sharing is possible. Eight is the upper limit for this example since the critical path, which is shaded in Figure 1, has only eight operation nodes. (Distribute and join nodes are ignored.) Only a single adder and subtracter were purchased for this design. Two fast designs were computed (stage counts of 2 and 4) which had the same overall delay; the stage count of 4 was chosen since it required 37% less hardware (2 versus 4 adders and 3 versus 4 subtracters). The allocation of resources performed by MAHA is shown in Tables 1 and 2.

Time / Resource	1	2	3	4
add	add1	add2	add6	add7
add	add4	add5	add3	add8
sub	sub3	sub6	sub7	sub8
sub	sub2	sub5	sub4	--
sub	sub1	--	--	--

Table 1: The Fastest Allocation for the First Example

Time / Resource	1	2	3	4
add	add1	add4	add2	add5
sub	sub2	sub3	sub5	sub6

Time / Resource	5	6	7	8
add	add6	add3	add7	add8
sub	sub1	sub7	sub4	sub8

Table 2: The Cheapest Allocation for the First Example

The second example chosen is the temperature controller described in [1] and shown in Figure 2. To demonstrate the power of MAHA, an average cost and speed design was synthesized initially. From this middle point, the cost was increasingly constrained, producing a range of cost-efficient designs. A second set of fast designs was produced by reducing the overall delay allowed for completion. The results from a design with four stages is depicted in Figure 2 with both the critical path and stages delineated; a summary of all results is plotted in Figure 3. This example clearly shows an underlying problem of

design synthesis: although the cost and delay curve has a somewhat "linear" shape, the space is discrete rather than continuous and a chosen point on the "line" may not be near a feasible design.

For the example above, MAHA allocated 4 adders, 2 subtracters, 2 comparators, 3 buffers (out1-3), and a single divider.

MAHA was written in LISP and executed on a VAX 11/750. For both examples, the initializer and critical path finder took 13 seconds (real-time, not CPU seconds). Re-partitioning of the first example averaged 2.7 minutes; the second, 2.2 minutes. Allocation of hardware to the critical path and off-critical path took approximately 30 seconds in both cases.

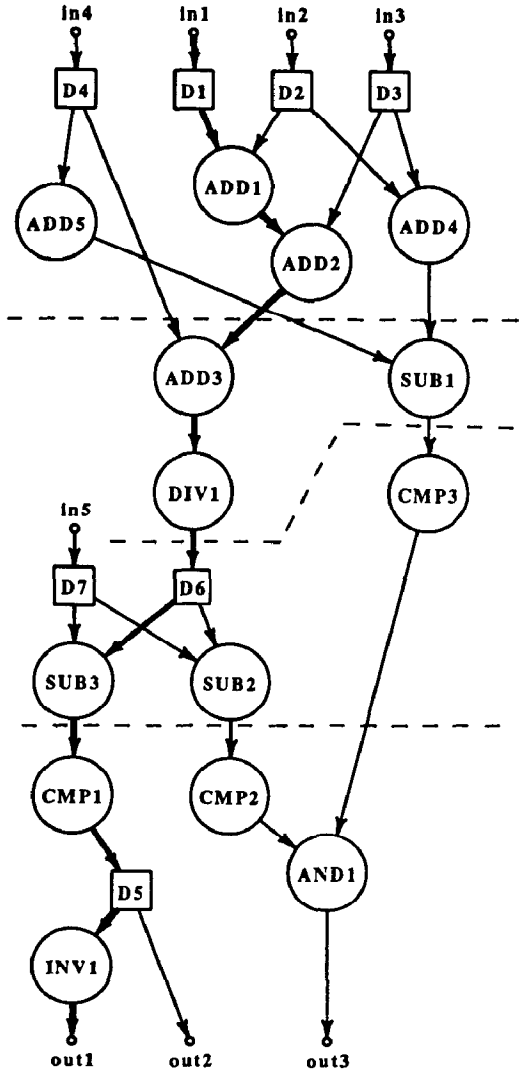

Figure 2: data flow graph for the second example

5 Conclusions

In conjunction with CSSP, this program does an assignment of operations in a data flow representation to hardware operators. The resulting bindings can then be passed to a placement and routing program to produce silicon. As an example, using the MP2D cell library [10] as the hardware module library, a data flow representation can be taken to silicon using CSSP and MAHA to generate the module descriptions for the data flow nodes, which are then fed to MP2D to produce the final design. Module binding must still be performed manually, however, and multiplexers have not yet been allocated.

MAHA illustrates a flexibility not found with other synthesizers, including its adaptability to either cost or speed constraints, depending on the application. MAHA currently assigns operations to operators, schedules when the operations should or must occur, and allows exploration of the design space given the constraints of the user.

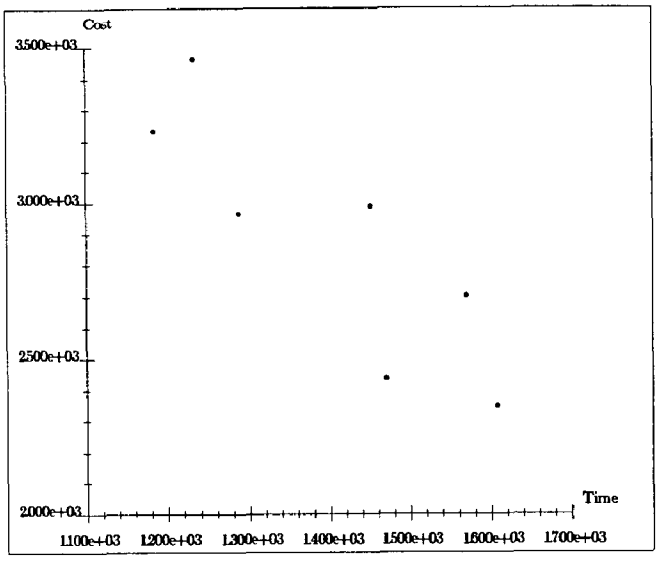

Figure 3: Cost speed curve for the second example

CSSP assigns registers, as necessary, to the data paths, but does not attempt to share registers during free clock cycles. Currently, multiplexers are indicated where needed, with no consideration of when it would be advantageous to use a bus or other types of control for signal paths. MAHA currently does not consider sharing ALUs or sharing different bit-width operators. These problems are being addressed, and solutions are being added to the basic MAHA program.

One of the most powerful features which can be easily added to MAHA is the capability of backtracking since the bindings to hardware can be broken [4] and/or modified at any point in the program. This will allow MAHA to make assignments, determine approximate area using an area estimator [5] and backtrack when it becomes apparent that some design constraint will be violated. Along with hardware/software tradeoffs and allocation of busses and register arrays to conserve area, these are features that will be added to MAHA to make the system a powerful tool for design synthesis.

6 Acknowledgements

The authors wish to thank Nohbyung Park for his help and assistance with CSSP. We also wish to thank Esther Brotoatmodjo for her help with this paper.

References

1. Girczyc, E. F. and Knight, J. P. An ADA to Standard Cell Hardware Compiler Based on Graph Grammers and Scheduling. Proceedings of the ICCD '84, IEEE Computer Society, October, 1984.

2. Granacki, J., Knapp, D., and Parker, A. The ADAM Advanced Design Automation System: Overview, Planner and Natural Language Interface. Proceedings of the 22nd Design Automation Conference, ACM-IEEE, June, 1985.

3. Hitchcock, C.Y. Automated Synthesis of Data Paths. Master Th., Carnegie-Mellon University,1983.

4. Knapp, D. and Parker, A. A Data Structure for VLSI Synthesis and Verification. Digital Integrated Systems Center, Dept. of EE-Systems, University of Southern California, October, 1983.

5. Kurdahi, F. and Parker, A. Area Estimation of VLSI Integrated Circuits. CRI-85-05, EE-Systems Dept. USC, 1985.

6. McFarland, M.C. Allocating Registers, Processors and Connections. Internal Carnegie-Mellon University Report.

7. Nagle, A., Cloutier, R., and Parker, A. "Synthesis of Hardware for the Control of Digital Systems". *IEEE Transactions on Computer-Aided Design CAD-1*, 4 (1982), 201-212.

8. Park, N. and Parker, A. Synthesis of Optimal Clocking Schemes. Proceedings of the 22nd Design Automation Conference, ACM IEEE, June, 1985.

9. Park, N. and Parker, A. Synthesis of Optimal Pipeline Clocking Schemes. DISC/85-1, Dept. of EE-Systems, University of Southern California, January, 1985.

10. RCA. *Integrated Computer Aided Design and Design Automation System.* Second edition, RCA, Advanced Technology Labs., Moorestown,N.J., 1985. 3-Micron Standard Cell Library.