VLSI DESIGN
1997, Vol. 5, No. 2, pp. 167–182
Reprints available directly from the publisher
Photocopying permitted by license only

Module Selection in Microarchitectural Synthesis for Multiple Critical Constraint Satisfaction

IAN G. HARRIS and ALEX ORAILOĞLU*

Department of Computer Science and Engineering, University of California, San Diego, La Jolla, CA 92093

Accurate design descriptions during synthesis allow efficient use of resources. The appropriate use of distinct implementations of RTL operators helps generate optimal VLSI designs. The system presented here utilizes libraries composed of multiple modules with identical functionality, but distinct performance and area characteristics. Such libraries allow the generation of an accurate estimate of the area and delay of the final design during synthesis. Full use of the module selection capability is possible by allowing the user to specify a total area limit rather than a detailed allocation. Consequently, tradeoffs between different allocations can be fully explored. Scheduling, module selection, and allocation are performed simultaneously to achieve optimal use of area and delay, and to facilitate the incorporation of lower level design considerations into behavioral synthesis. Synthesis decisions are made in a time-constrained *and* area-constrained fashion, by using both constraints to identify and avoid infeasible design possibilities. Module selection, scheduling, and allocation for pipelined designs is also implemented. Experimental results show that the use of module selection and time-and-area-constrained synthesis results in an area/delay design curve which is superior to the results of traditional systems.

Keywords: module selection, behavioral synthesis, scheduling, allocation, constraint satisfaction, estimation

1. INTRODUCTION

There are many parameters of a final chip design, such as area, power, and performance, which must be addressed to make the chip useful, and its manufacture profitable. At each step of the design process, these parameters must be accurately estimated and used to guide the progress of the design towards a high quality chip. Behavioral synthesis typically suffers from inaccurate estimates because there is little information about the physical layout at the algorithmic level of description. Recently, attempts have been made to connect behavioral synthesis to lower levels of design (floorplanning [31], placement and routing [26]) to achieve better estimates during synthesis. Since design decisions made at the behavioral synthesis level have a great impact on the quality of the final design, it is crucial that good estimates be used to direct synthesis.

The goal of behavioral synthesis is to generate a datapath from an algorithmic description of the de-

*Corresponding author.

sired functionality of the chip. The first step in the process is usually the generation of an intermediate algorithmic representation [24], composed of a control flowgraph describing conditional branching and looping constructs in the behavioral description, and a dataflow graph describing the dataflow dependencies between operations in the algorithm. The task of deciding at which clock cycle each dataflow operation will be performed is called scheduling. The allocation task allots hardware modules to perform the operations in the dataflow graph, and the operator binding task assigns each dataflow operation to a particular allocated hardware module which will perform it. Additionally, an RTL description of the control unit must be generated to sequence the operations as described in the schedule. A layout can then be generated from the RTL description of the entire chip using standard physical design tools.

In the design of a chip with any reasonable degree of complexity, it is likely that more than one implementation of an operator will be utilized. For instance, a slow ripple carry adder could be used as well as a carry-lookahead or carry-select adder. These three different types of adders will have different area and delay characteristics which should be considered during behavioral synthesis while the datapath is being created. The characteristics of the modules used during synthesis should be close to the characteristics of the modules which will be used in the physical design so that accurate estimates of delay and area consumption can be made. Delaying the resolution of abstract modules to actual components until after microarchitectural synthesis may result in inefficient area usage and reduced throughput.

In this paper, we propose integration of module selection into the scheduling and allocation tasks of behavioral synthesis. Traditional hardware allocation is performed under the assumption that only one module type of each functionality exists in a design library. Allocation with module selection allows us to obtain a better estimate of the area of the final design by using area information from a full library rather than the rough approximation afforded by only a single module type. Performing scheduling with module selection allows us to better estimate the delay of the final dataflow graph schedule by using timing information from a full library. Module selection facilitates the efficient use of area and delay resources by allowing non-critical path nodes to be executed by small, low speed modules. Performing module selection during behavioral synthesis facilitates better use of chip area by allowing operator binding to be performed to the full range of library modules.

2. RESEARCH CONTRIBUTIONS

Design decisions during behavioral synthesis are driven by the need to meet a set of constraints on the properties of the physical design. Performance constraints are critical in many applications where processing must be performed in real time, such as DSP. At the same time, new design requirements such as design-for-test [4, 32, 1], reliability [25], and fault tolerance [19], have made the optimization of area overhead more imperative. For time-critical and area-critical designs, both constraints should be enforced simultaneously during high-level synthesis. Efficient use of resources is necessary to meet user-imposed design constraints. The use of module selection provides a more accurate description of the area and delay characteristics of the design during behavioral synthesis, allowing area and delay resources to be used efficiently. The tradeoff between area and delay can be explored fully when no user-imposed limits on the allocations of each module type are specified.

The primary research contributions of our work can be summarized as follows:

- **Module Selection** Conventional behavioral synthesis methods are extended to accommodate libraries containing modules with identical functionality but different area and delay characteristics.
- **Time-*and*-Area Constrained Synthesis** Time and area constraints are simultaneously satisfied by using the area estimation to determine which design decisions will cause the area limit to be violated

and using the module delays in the module library to determine which decisions will violate the delay constraint.

- **Allocation Freedom** Conventional behavioral synthesis algorithms require that the user specify a limit on the allocation of each hardware type. In contrast, we propose the use of a total chip area constraint to enable thorough exploration of the solution space for the appropriate allocation.

3. PREVIOUS WORK

Many scheduling algorithms which are either time-constrained such as Force-Directed Scheduling [28], or area-constrained such as List Scheduling [16], have been explored [5]. To our knowledge, no approach besides the ILP formulations ([6, 22]) performs synthesis under both an area and a delay constraint. Although promising execution time results have been shown, the ILP problem is NP-complete and remains intractable for challenging synthesis problems.

Various forms of the module selection problem have been explored previously. The problem of module set selection through resolution to a restricted library containing a single module type for each operator functionality prior to microarchitectural synthesis has been explored in [14, 15, 17]. This approach selects a single module type for each functionality which will be used in the allocation by generating an area/delay design curve for each possible subset of module types. The selection of a module set and a corresponding clock period has been studied in [3]. The selection of a single module type for each functionality negatively impacts the scheduling of flowgraphs that contain paths of varying criticality; it is desirable to perform critical path nodes on fast modules and non-critical path nodes on slow modules in this case. Selection of a single module type makes this tradeoff impossible.

A module selection algorithm has been proposed by Ramachandran and Gajski [30] which performs component selection in conjunction with scheduling and operator binding using a distribution graph model [28] to estimate the effect of each compound decision on the area and performance of the final design. Ramachandran and Gajski's work expands this model by computing the distribution of each module type in the module library. Ishikawa and De Micheli [13] propose a module selection algorithm which uses heuristics to select modules types while meeting a latency constraint. A module selection algorithm for pipelined datapaths is proposed in [23] which uses a detailed module delay model, requiring increased CPU time.

The algorithm proposed in [8] performs allocation with module selection before scheduling, by using a hill climbing technique to explore the search space of different allocations. Since allocation is performed before scheduling, allocation decisions cannot make use of scheduling information to achieve improved results.

Scheduling of pipelined datapaths has been studied in several research projects such as [27, 18, 11, 7, 12]. Even though both area and performance are frequently critical due to the stringent throughput requirements of DSP applications, module selection has not been commonly incorporated.

4. SYSTEM OVERVIEW

The basic components of the algorithm are heuristic synthesis, time-and-area constrained synthesis, and area estimation. The heuristic synthesis component uses heuristic measures to choose scheduling, allocation, and module selection decisions to be included in the design. The time-and-area constrained synthesis component examines the design state after each heuristic decision and prunes away options that can be seen to lead to area or delay constraint violations. The area estimation component is used by the time-and-area component to determine which design decisions will lead to infeasible designs.

Heuristic decisions are made which limit the degrees of scheduling freedom of flowgraph nodes. Their effects are propagated throughout the design by the time-and-area constrained synthesis component. Such propagation may further force additional decisions automatically. Once such propagation is completely finished, a new allocation is predicted based on the resulting design state and new modules are added to the allocation if necessary. Decisions are made heuristically until all nodes will have been committed to clock cycles and bound to modules. Figure 1 is a flowchart showing the interaction between the time-and-area constrained synthesis and the heuristic synthesis.

One aspect of the heuristic component consists of allocation decisions. An allocation decision limits the freedom of allocated modules to be mapped to different module types. A novel aspect of the algorithm is that the allocation is composed of a number of *flexible modules* which are identified by the area estimation algorithm to perform the operations in the dataflow graph. A flexible module may be mapped to a *set* of feasible module types, rather than immediately being fixed to a single module type. This flexible representation allows the system to describe more accurately the information in the partial design state. When an allocation decision is made, the feasible module set of a flexible module is pruned.

While scheduling decisions determine the clock cycle at which a node will be executed, module selection decisions determine the flexible module which will perform the operation. Each node has a set of feasible clock cycles to which it may be scheduled. When a node is scheduled, this set is reduced to a single clock cycle. Each flowgraph node also has a feasible module set and can only be bound to a flexible module whose feasible module set shares some module types in common with the feasible module set of the node. This algorithm performs these two types of decisions in an intertwined fashion to allow all three tasks to benefit from partial design informa-

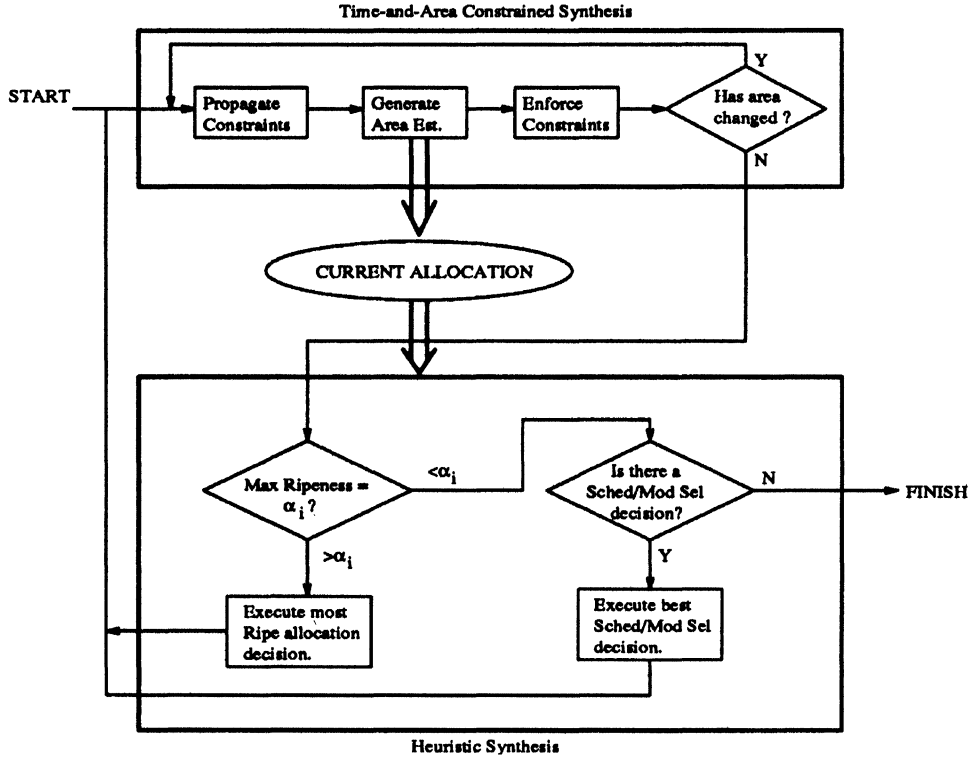

FIGURE 1 System flowchart.

tion during synthesis. The degree of intertwining is controlled by a user-defined parameter α_i.

The time-and-area constrained synthesis component is used to prune design options which can be shown to result in constraint violations. This is achieved by considering area consumption in performance determination while control step assignment possibilities are considered in area determination.

The area estimation component generates an estimate of the area by predicting an allocation which is minimally sufficient to perform the flowgraph nodes given the current state of scheduling. The area estimate is used by time-and-area constrained synthesis to determine which design options lead only to infeasible designs, and the predicted allocation produced is compared to the current allocation to determine if new modules need to be allocated.

Hardware utilization can be improved by *chaining* operations, that is, allowing two or more operations to be performed serially within one clock cycle. This alleviates underutilization by allowing hardware to be used in time in a clock cycle that would otherwise be wasted. The earliest and latest times at which a node can be scheduled, C_e and C_l respectively, are kept as a clock cycle and a time displacement within the clock cycle to enable appropriate handling of chaining.

This paper will first describe how time-and-area constrained synthesis assures that both constraints are met. Subsequently the area estimation algorithm will be presented followed by a description of the heuristic scheduling, module selection and allocation algorithm. Results demonstrating the effectiveness of the basic algorithm will be presented. Then synthesis of pipelined systems will be discussed, and results of pipelined synthesis with module selection will be presented.

5. TIME-AND-AREA CONSTRAINED SYNTHESIS

In order to satisfy both time and area constraints, it is necessary to determine which design decisions would lead only to infeasible designs. Once infeasible design options have been identified, they can be avoided by the heuristic synthesis component of the algorithm. A non-exhaustive search, using only information contained in the current state of the design, is sufficient to identify such infeasibilities. In order to determine that a design decision will violate the area constraint, it is necessary to generate an area estimate for the final design.

The scheduling and allocation possibilities are described by the feasible clock cycles of each node, the feasible module types of each node, and the feasible module types of each flexible module in the allocation. *Propagation* of the effects of each design decision may require that these sets be pruned for affected nodes and modules. *Constraint Enforcement* causes new design decisions to be made when they are necessary to guarantee that the area and delay constraints are not violated.

Propagation restricts the scheduling freedom of adjacent nodes. The scheduling freedom of a node is limited by restrictions imposed on the scheduling of adjacent nodes in the graph, and by the availability of hardware at different clock cycles. When a node's scheduling freedom is restricted, the C_e and C_l values of all adjacent nodes are recomputed. Any clock cycles which are no longer within the feasible scheduling range of a node are pruned. Computation of the C_e and C_l values considers the area as well as the performance constraint. In figure 2, nodes +X2, +X3, +X4, and +X5 all have C_l equal to 4, the last clock cycle. Consideration of the delay constraints only would allow +X1 to be scheduled as late as clock cycle 4 also (assuming chaining). Yet consideration of the area constraint shows that at most two addition modules can be allocated in a clock cycle.

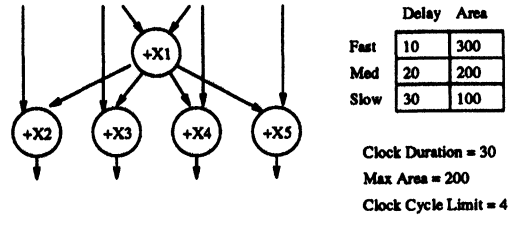

FIGURE 2 Earliest and latest feasible clock cycle computation.

Since only two addition operations can be performed in a clock cycle, it can be determined that operation +X1 can be performed no later than clock cycle 2.

Constraint enforcement limits scheduling and module selection options when area or delay limits are approached. When insufficient area is available to allocate a new module of a certain functionality, the already allocated modules must suffice for the scheduling of all remaining nodes. If the currently allocated hardware of a given functionality is fully utilized at a certain clock cycle, then no other flowgraph operations of that functionality may be committed to that clock cycle. This observation is propagated throughout the partial design state by pruning the feasible clock cycle sets of all appropriate nodes. Feasible module types of a flowgraph node will be restricted if they would result in infeasible schedules or a violation of the area constraint. If the feasible clock cycles of a node are restricted to 2 clock cycles, then a module type will be pruned from the node if its delay would result in a violation of the clock duration constraint at both adjacent clock cycles. This is determined by examining the time displacements within the clock cycle. When a module type has been pruned from all elements of the allocation, and remaining area is not sufficient to allocate a new module of that type, then that module type is pruned from all flowgraph nodes.

Additionally, the feasible module types of each flexible module are examined during constraint enforcement to determine if pruning is necessary. Area estimation relies on an optimistic estimate of the area based on the slowest feasible module type of each allocated module. Consequently, if there is not enough area remaining to upgrade the module from its slowest to its fastest module type, then the fastest module type is infeasible.

6. AREA ESTIMATION ALGORITHM

The area estimation component of the algorithm estimates the area remaining by predicting an allocation that is minimally sufficient to perform the flowgraph nodes in the partial design state. The area estimate is used by time-and-area constrained synthesis to determine which design options lead to constraint violation, and to determine if new modules need to be added to the allocation. The estimate generated by the area estimation component is never an overestimate; this consistent characteristic of our design estimate ensures that feasible design options are not pruned.

The method of estimating the area is an application of the pigeonhole principle[2] to nodes confined to ranges of clock cycles. We will use the dataflow graph in figure 3 to demonstrate the use of the pigeonhole principle in predicting a minimum allocation. In figure 3, four nodes must be scheduled within three clock cycles, therefore, by the pigeonhole principle, at least two addition modules must be allocated. The pigeonhole principle can be analogously generalized to consider nodes which are confined to ranges of clock cycles, as well as module types.

Our approach is illustrated in the dataflow graph of figure 4 wherein the three shaded nodes are scheduled to clock cycles (+4, +5, +6) while the other three are free to be scheduled over all three clock cycles (assuming chaining). The unscheduled nodes are annotated with their feasible module types. Clearly nodes +1, +2, and +3 must be scheduled within clock cycles 1, 2, and 3, and the total module availability over that range of clock cycles and over the feasible module range, ({Fast, Med} modules) is three. The availability in the range is figured by counting the number of clock cycles at which each

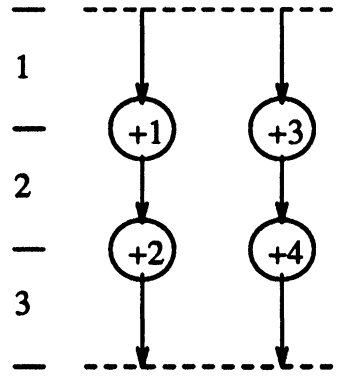

FIGURE 3 Four nodes confined to three clock cycles.

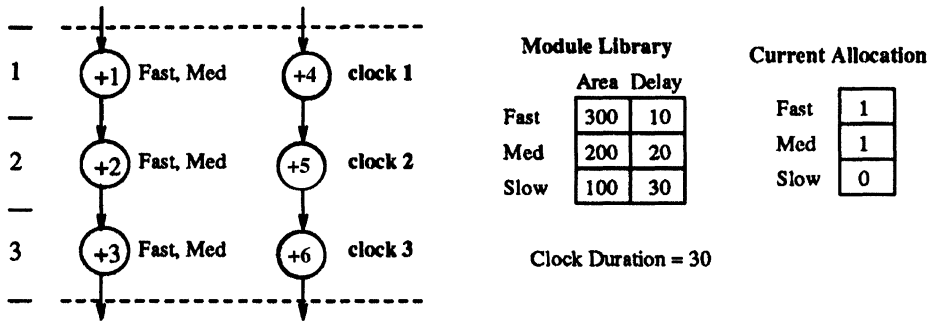

FIGURE 4 Six nodes confined to three clock cycles.

module whose type is a subset of the feasible module range is not utilized. Since there are two flexible modules in this range with three nodes already committed, the total remaining availability in this range is 3. Since there are also three unscheduled nodes in this range, the current allocation is sufficient to perform the schedule. An additional unscheduled node in the range would have resulted in a new module of type {Fast, Med} being introduced.

The algorithm performs optimistic estimates of area so that the partial design state is not unnecessarily constrained. To ensure that the area is an underestimate, the examination of ranges of module types is ordered by the smallest module type contained in the range. This causes smaller modules to be allocated first, and large modules to be allocated only if small modules are insufficient. In general, the availability is reduced to zero inside a range of clock cycles C and module types M if:

$$| MR_M \cap CR_C | = \sum_{c \in C} AV_{c,M} \qquad (1)$$

where equation MR_M is the set of nodes whose feasible module types are subsets of M, CR_c, is the set of nodes which must be scheduled within the set of clock cycles C, and $AV_{c,M}$ is the number of modules whose feasible module types are a subset of M and are not executing an operation in clock cycle c.

A new module is added to the current allocation when inequality 2 is true for some range of clock cycles C and module types M.

$$| MR_M \cap CR_C | = \sum_{c \in C} AV_{c,M} \qquad (2)$$

7. HEURISTICS

Once time-and-area constrained synthesis has pruned away all design decisions that are definitely infeasible, there remain a number of feasible design options. We propose an heuristic based approach to choose a design option in a computationally efficient manner.

Heuristic decisions are of two types, a combined scheduling/module selection decision or alternately an allocation decision. Each scheduling/module selection decision commits a node in the dataflow graph to be performed at a clock cycle, and to be performed by a particular flexible module. Each allocation decision refines the real allocation by pruning a feasible module type from a module.

The heuristic subsystem alternates between a scheduling/module selection phase and an allocation phase. We have observed that the order in which scheduling, module selection, and allocation are performed impacts the optimality of the design. Completion of one phase may limit the solution spaces of the subsequent phases in such a way that no feasible solutions which satisfy both area and delay constraints remain.

This algorithm intertwines the tasks of scheduling, allocation, and module selection, so that each task can be guided by partial information from the others.

Experimental results show that the degree to which scheduling and allocation decisions are intertwined can have a significant effect on the quality of the design.

7.1 Scheduling/Module Selection Decisions

The system schedules each node to a clock cycle, and binds each node to an allocated module. A node is committed to a clock cycle and bound to a flexible module simultaneously. First the node which is most ready to be committed is determined, and then the best clock cycle and module for node commitment is chosen. The node is selected based on the following criteria.

Module Type Similarity A node which has many feasible module types in common with a module should be bound early. Binding a node to a dissimilar module would result in constraining the feasible module types of the node and/or the module.

Clock Cycle Similarity A node which must be committed to a module which is unutilized at many of the node's feasible clock cycles should be committed prior to a node which has few clock cycles in common with any module. Binding a node to a module which is utilized during the feasible clock cycles of the node restricts the freedom of the node. Decisions which bind nodes to modules which are dissimilar in terms of available clock cycles are deferred, allowing the algorithm to take advantage of future module allocation that may be more compatible.

Scheduling Freedom We define the scheduling freedom of a node to be the number of (clock cycle, module type) pairs to which it may be committed. A node which has less freedom should be committed early in the scheduling because the node can lose its scheduling freedom as a result of propagation and constraint enforcement effects of intervening heuristic decisions.

Path Scheduling Freedom A node has less freedom if it is on path of the dataflow graph which has little scheduling freedom. The criticality of a path is a measure of the degree of scheduling freedom of a chain of nodes rather than a single node. If the path scheduling freedom for a node is low then it is on a path whose completion time is large compared to the maximum time in which the path must complete to meet the delay constraint. The nodes on such critical paths should be scheduled early because they have less freedom.

The flexible module to which a node will be bound and clock cycle at which a node will be performed are selected based on the criteria listed below.

Module Type Similarity This is the same Module Type Similarity criterion used to select the node to be scheduled. A node is bound to a flexible module which has many feasible module types in common with the node.

Uniform Hardware Usage By distributing nodes across the graph, this rule attempts to fully utilize hardware at all clock cycles. We define a heuristic measure of the collective scheduling freedoms of the predecessor and successor paths containing a node. By committing a node to the clock cycle which most evenly matches the scheduling freedoms of predecessor and successor paths of each node, node clustering into a few clock cycles, which would have resulted in underutilization, is avoided.

Hardware Availability This rule attempts to fully utilize hardware by committing nodes to states where fewer nodes are likely to use the hardware. A probabilistic estimate of the number of nodes that will require hardware at a clock cycle is generated by adding the path scheduling freedom values of all nodes which may be committed to that clock.

Mathematical formulations of these heuristic measures may be found in [10].

7.2 Allocation Decisions

For each flexible module in the allocation, the system can heuristically decide to prune either the fastest or the slowest module type from its feasibility set. When the feasible module type of such a module is pruned, that type is also pruned from the feasible module type

sets of all nodes bound to that module. For this reason, each node which is bound to the module must be examined to see if such pruning is acceptable.

The fastest module type can be eliminated from the feasible module type set if it is estimated that no node bound to the module will utilize this module type. A probabilistic determination is made as to whether the bound nodes must use this module type. The determination is based on the *Path Scheduling Freedom* of a node as previously described. If a node's paths have a large degree of scheduling freedom, then that node is less likely to require a fast module type. The slowest module type is pruned using a similar metric which measures the scheduling freedom of paths containing the node under the assumption that each node on the paths is performed by the slowest feasible module type. A fast module type is pruned only if none of the bound nodes require it, while a slow module type is pruned if there is a single bound node which cannot use it.

7.3 Intertwining Threshold

The user provides an input parameter, α_i, which determines how much scheduling information is needed before an allocation decision can be made. This parameter is used to control the degree of intertwining of the scheduling/module selection decisions and the allocation decisions. When deciding whether or not to prune the feasible module types of a flexible module, a weighted average of the scheduling freedoms of the paths containing the bound nodes is compared to α_i and pruning is performed if the weighted average is greater than α_i. Low values of α_i cause allocation decisions to be eager, which provides early direction to the scheduling, while high values cause scheduling decisions to be eager, giving early direction to allocation.

8. EXPERIMENTAL NON-PIPELINED SYNTHESIS RESULTS

We have conducted a set of experiments to test the ability of the heuristics to navigate, and of the time-and-area constrained synthesis to prune the search space. The first example in figure 5 demonstrates that the effects of the time and area constraints are successfully enforced. In this example, time-and-area constrained synthesis pruned all decisions that could be deduced infeasible from the initial constraints. Since almost all decisions were automatically pruned as a result of constraint enforcement, all scheduling, module selection, and allocation decisions were completed except for the limited freedom of nodes +X2 and +X14. In this example, the constraint enforcement part of the system automatically assigned all addition operations to medium speed adders, and all multiplication operations to slow multipliers.

In another set of experiments, we studied the ability of the heuristics to guide the search through the design space under tight constraints. We scheduled the differential equation example[28], the AR-filter[27] and the FIR-filter[27] flowgraphs with constraints and results shown in figures 6, 7, and 8, respectively. Under the given constraints, the solutions identified by the algorithm are the only feasible solutions. The solutions use a rich set of modules and would not have been feasible under the single module type assumption.

To demonstrate that module selection produces designs which utilize area and delay resources more efficiently than designs generated which use a single module type for each operator, we compared the results of our system to results generated by the HAL [29] algorithm on the AR-filter dataflow graph. The resulting area-time curves are shown in figure 9. A clock cycle duration of 250 ns was used for this experiment.

The HAL algorithm considers only one module of each functionality, so we supplied it with each pair of adder and multiplier modules in the library shown in figure 10. Our system, which uses the full library, is a better area/delay curve than HAL in almost every case.

In order to investigate the effect of changing the degree of intertwining of scheduling and allocation decisions, we performed scheduling on the FIR-filter example with different degrees of intertwining by changing the value of α_i. The results are shown in figure 11. The area of each result is marked on the

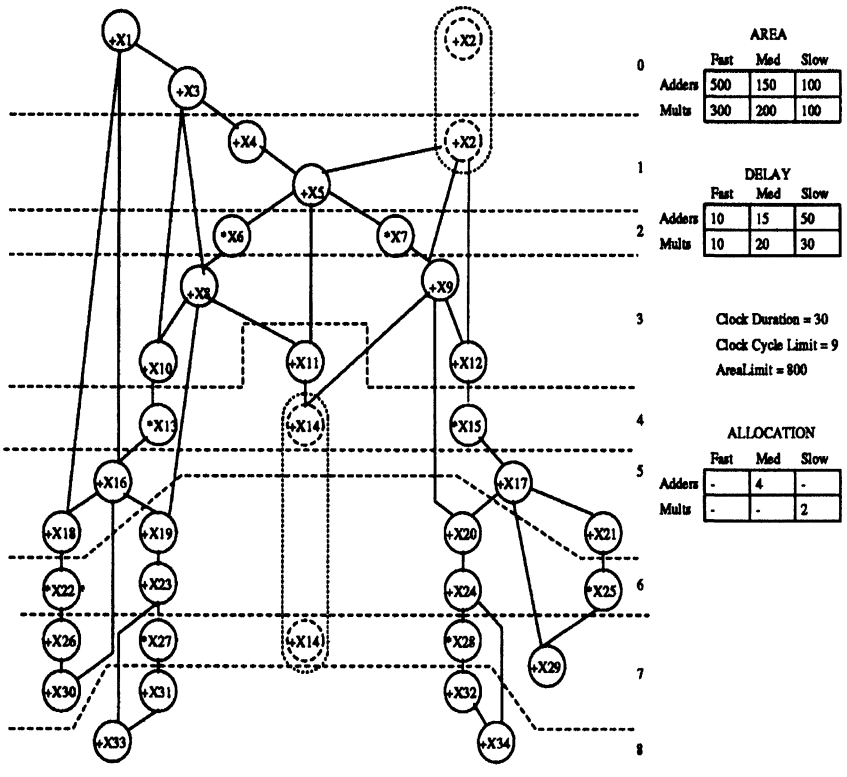

FIGURE 5 Fifth order elliptic filter benchmark.

graph, and is annotated with the allocation of modules corresponding to that result. In these results, the α_i parameter ranges from 0 to 1. Low values of α_i cause allocation decisions to be eager, which provides early direction to the scheduling. Eager allocation may cause allocation decisions to be made prematurely as occurred with the $\alpha_i = 0.33$ result shown. High values of α_i cause allocation decisions to be delayed until more information is available about the schedule. The scheduling decisions are made with less allocation information and therefore have a better chance of resulting in a suboptimal design. This occurred with the $\alpha_i = 0.5$ and $\alpha_i = 0.55$ results.

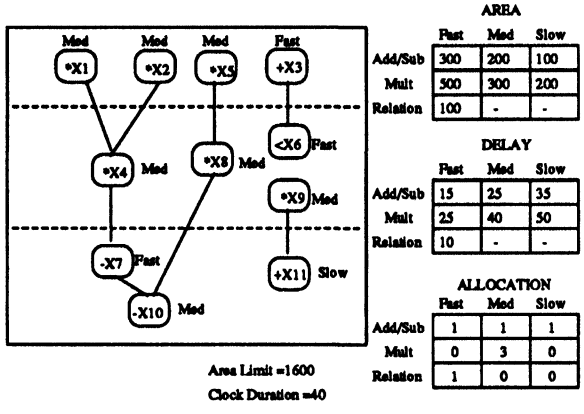

FIGURE 6 Differential equation results.

To observe how efficiently the algorithm uses area under different clock cycle constraints, we performed scheduling on the AR filter example with different clock cycle limits. The results are shown in figure 12. Using 8 clock cycles, it is possible to schedule the graph without any chaining, so only slow modules were used. As the number of clock cycles decreases, the number of fast modules needed to schedule the graph increases.

In another experiment, we scheduled the FIR-filter example with different clock durations and clock cycle limits, holding the total completion time constant at 180 ns. We define the total completion time to be

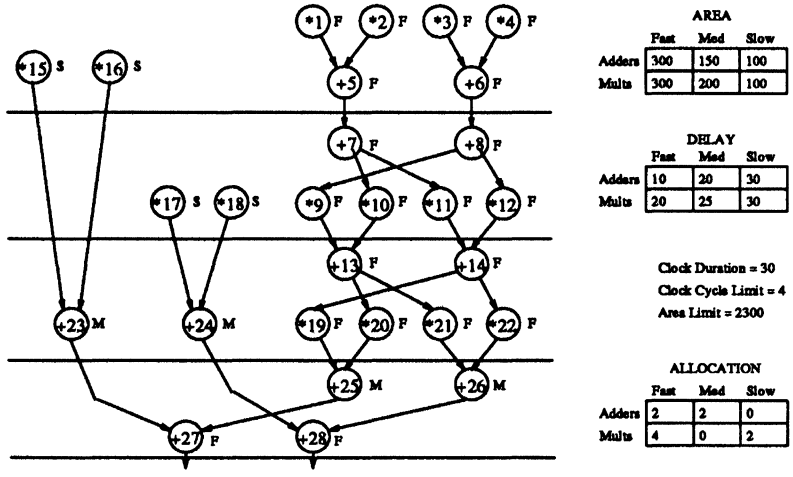

FIGURE 7 AR-filter scheduling.

the real time that is required to perform the entire algorithm, and we compute the real time as the product of the number of clock cycles and the duration of each clock cycle. The results are shown in figure 13.

To demonstrate the speed of the algorithm, we performed synthesis on the differential equation (DE), FIR-filter (FIR), and the AR-filter with different area and delay constraints. All experiments were performed on a Sun-4 SPARC-based CPU 20MHz machine with a Weitek 3170-based floating point unit. The average run times are shown in figure 14. Comparison to other algorithms is difficult since few other algorithms perform all of the tasks that this algorithm does, and published execution time results are limited. We have found a comparable execution time result by the HAL algorithm[28] which performs only a scheduling of the FIR-filter example with similar parameters in 30 seconds. These results show that despite simultaneous performance of scheduling, allocation, operator binding, and module selection, effective heuristics and tight constraint enforcement produce computationally effective solutions.

Usually, a larger clock duration with a smaller number of clock cycles will require area to increase because more operations must be performed in the same clock cycle, but this is not always the case, as is

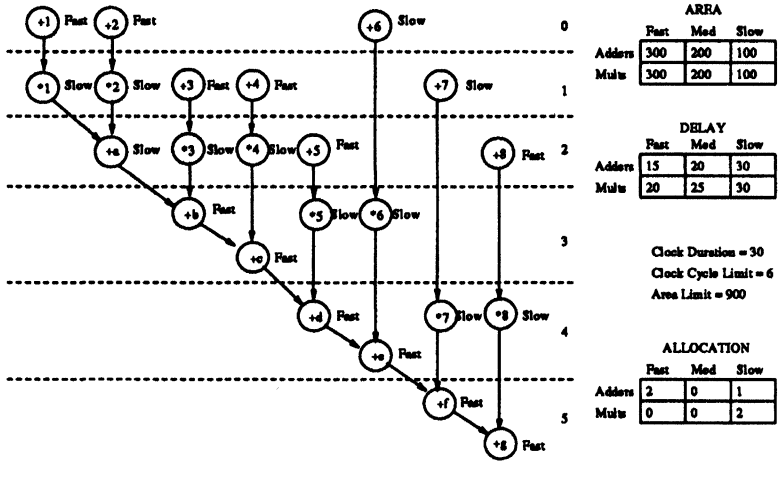

FIGURE 8 FIR-filter results.

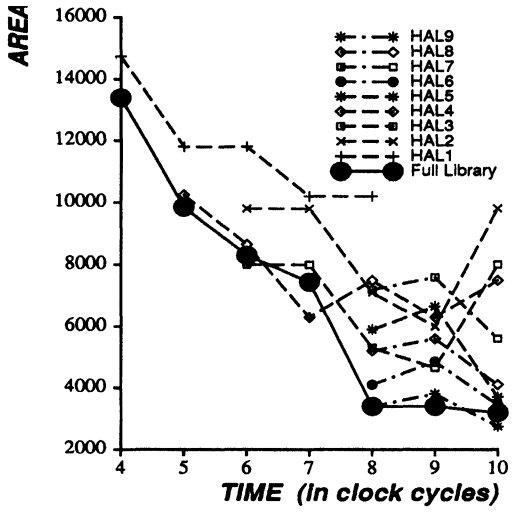

FIGURE 9 Comparison to HAL using limited libraries.

evidenced by the decrease in area when the clock duration was increased from 20 ns to 30 ns. The area may decrease as the clock duration increases because certain clock durations are more amenable to chaining.

9. CONSIDERATIONS FOR PIPELINED SYNTHESIS

Special considerations must be made when calculating hardware usage at a clock cycle in a pipelined system. Instances of two nodes which are scheduled to clock cycles c_i and c_j respectively, will be executed simultaneously if $c_i \ mod \ L = c_j \ mod \ L$ where L is latency. To consider this effect on hardware usage, we conclude that the hardware at a clock cycle c is fully utilized when the number of nodes committed to all equivalent clock cycles c_i such that $c_i \ mod \ L = c \ mod \ L$ equals the total allocated hardware. We have extended the basic area estimation algorithm to accommodate pipelined designs. When determining the number of nodes which must be committed to a range of clock cycles, all nodes are included which must be committed to the range of clock cycles being considered, or to any clock cycles which are equivalent modulo the latency.

Dependencies may exist between instances of operations in different iterations of the loop. When an inter-iteration dependency exists, the C_e and C_l values of the successor node must be computed differently. Clearly a node must be scheduled after all of its predecessor nodes in the dataflow graph, so $C_e(n) \geq C_e(m)$, where node m is the predecessor node of n. If this predecessor node is across an inter-iteration dependency which represents a dependence across i iterations, then the inequality must be changed to $C_e(n) \geq C_e(m) - (i * L)$. This change reflects the fact that the predecessor node is being performed in an instantiation of the loop which was initiated $(i * L)$ clock cycles before the current instantiation that the successor node is being performed in.

AREA

	Fast	Med	Slow
Adders	1350	750	400
Mults	1600	1100	650

DELAY

	Fast	Med	Slow
Adders	70	145	210
Mults	85	190	240

HAL1 =	FAST ADDER	FAST MULTIPLIER
HAL2 =	FAST ADDER	MEDIUM MULTIPLIER
HAL3 =	FAST ADDER	SLOW MULTIPLIER
HAL4 =	MEDIUM ADDER	FAST MULTIPLIER
HAL5 =	MEDIUM ADDER	MEDIUM MULTIPLIER
HAL6 =	MEDIUM ADDER	SLOW MULTIPLIER
HAL7 =	SLOW ADDER	FAST MULTIPLIER
HAL8 =	SLOW ADDER	MEDIUM MULTIPLIER
HAL9 =	SLOW ADDER	SLOW MULTIPLIER

FIGURE 10 Full module library.

10. PIPELINED SYNTHESIS RESULTS

We have performed the following experiments to test that the area and delay resources are used effectively in the scheduling of pipelined systems.

In order to test the ability of the system to perform under tight constraints, we scheduled the differential equation example presented in [29], containing its original inter-iteration dependencies. The resulting scheduling and module selection are shown in figure 15. The edges representing inter-iteration dependencies are shown in bold. All of the inter-iteration dependencies represent dependencies between successive iterations. Under these tight constraints, the system discovered a minimum area solution which fully

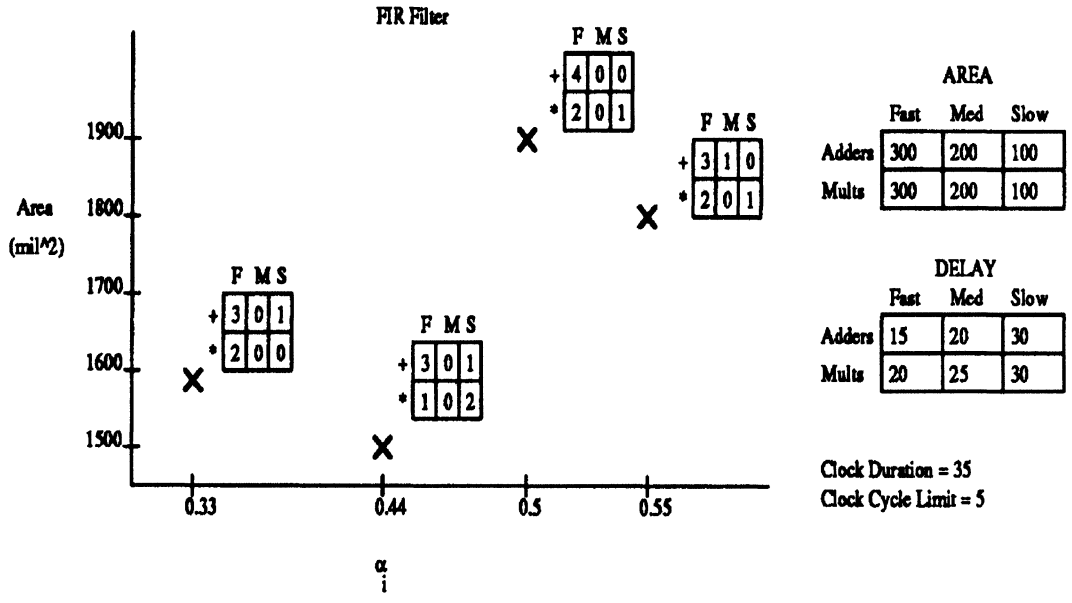

FIGURE 11 Scheduling with variable intertwining of the FIR-filter.

utilizes all adders and multipliers. In the given example, feasible scheduling and module selection possibilities are few. Time-and-area constrained synthesis pruned away all infeasible decisions early in the design process, guiding the heuristics by leaving few scheduling options. The solution uses different module types and would not have been feasible under the single module type assumption.

We have also scheduled the fifth-order elliptic filter dataflow graph with the original inter-iteration depen-

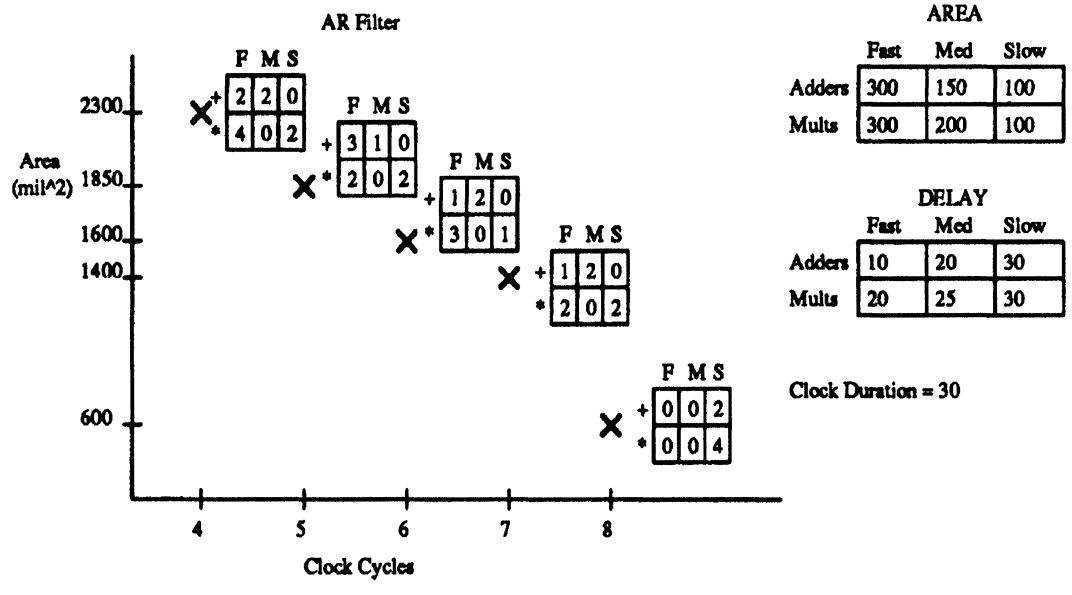

FIGURE 12 Scheduling AR-filter with variable clock cycles.

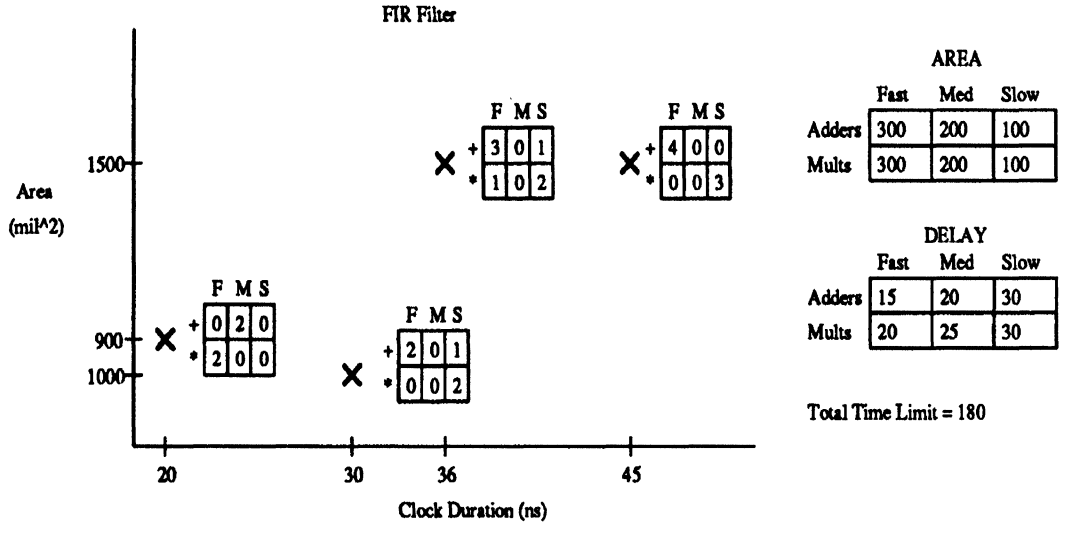

FIGURE 13 Scheduling FIR-filter with variable clock duration.

dencies first presented in [20]. The results and constraints of scheduling are shown in figure 16. The prescribed clock cycle limit was 14 but the resulting schedule utilized 10 clock cycles in order to meet the given latency constraint of 10 clock cycles. The area used by the design is the minimum area possible with the given constraints.

11. DISCUSSION SECTION

We have proposed an algorithm which generates a scheduling, allocation, and operator binding of a dataflow graph G using modules from a library of modules which is provided by the user. Synthesis is performed within a chip area constraint and timing constraints which include clock duration and the maximum number of clock cycles. The modules can have multiple functionality (i.e. an ALU) as long as there is only one module type which can perform each dataflow graph operation.

The area estimate is generated after each design decision of this system, so it is important that estimates be achieved in a computationally efficient manner to allow the system to explore tradeoffs quickly. For this reason, the area estimate includes only the functional unit area. As promising new work in layout estimation progresses ([33],[21]), new algorithms will allow a more detailed area estimate while still meeting satisfactory execution time constraints.

Dataflow Graph	Operations	Avg CPU
Diff. Equation	11	3.47 s
FIR Filter	23	67.73 s
AR Filter	28	68.9 s

FIGURE 14 Average run times for different examples (time in seconds).

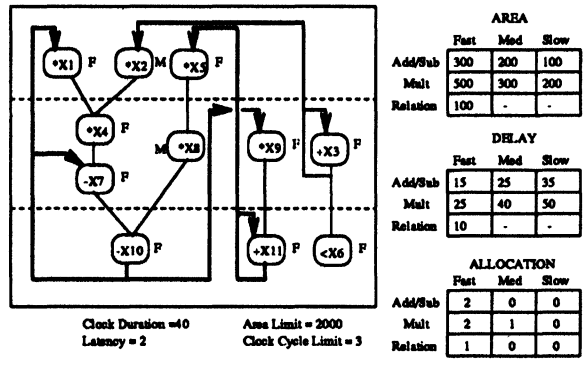

FIGURE 15 Differential equation results.

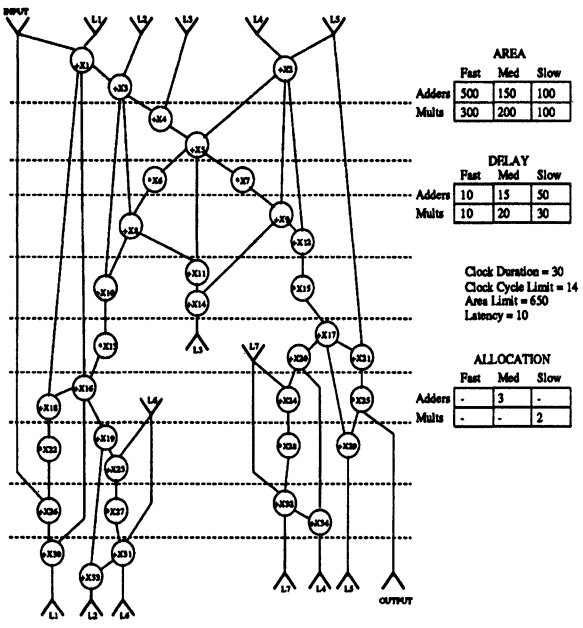

FIGURE 16 Fifth-order elliptic filter results.

The approach used by this system to satisfy area and delay constraints is flexible and may be extended to perform tradeoffs between other conflicting constraints. For instance, a module library could additionally capture power information. Approximation of power constraint satisfaction can be easily achieved if appropriate modeling of cumulative aspects of power is incorporated.

12. CONCLUSIONS

In this paper we have presented an algorithm which integrates module selection into high-level synthesis of pipelined and non-pipelined designs. Furthermore, we have illustrated an heuristic algorithm which intertwines module selection and scheduling decisions. The experimental results show that modules are selected appropriately and minimum area designs are accomplished. This system successfully performs time-and-area constrained scheduling, even under tight constraints when very few solutions are possible. This success is due both to the heuristics which guide the search through the design space toward feasible designs, and to the time-and-area constrained synthesis approach which guides the search by pruning infeasible design options. Performing module selection allows this system to find solutions under tight constraints when no solution would exist under the single module assumption.

In conclusion, the module selection system:

a. Imposes high-level parametric constraints.
b. Simultaneously satisfies multiple constraints such as performance and area.
c. Helps bridge the gap between high-level synthesis and physical design through more accurate representation of library components.

Acknowledgments

This work has been supported by a grant from Brooktree Inc. and National Science Foundation grant number CDA-9314748. A preliminary version of this paper has been presented in ISCAS93[9].

References

[1] P. H. Bardell, W. H. McAnney, and J. Savir, *Built-In Test for VLSI*. John Wiley and Sons, 1987.
[2] K. P. Bogart. *Introductory Combinatorics*. Harcourt Brace Jovanovich, 1990.
[3] L.-G. Chen and L.-G. Jeng. Optimal Module Set and Clock Cycle Selection for DPS Synthesis. *International Symposium on Circuits and Systems*, pages 2200–2203, June 1991.
[4] E. B. Eichelberger and T. W. Williams. A Logic Design Structure for LSI Testability. *Proceedings of the 14th Design Automation Conference, ACM-IEEE*, pages 462–468, 1977.
[5] D. D. Gajski, N. D. Dutt, A. C. Wu, and S. Y. Lin. *High-Level Synthesis: Introduction to Chip and System Design*. Kluwer Academic Publishers, 1992.
[6] Catherine H. Gebotys. Optimal Scheduling and Allocation of Embedded VLSI Chips. *Proceedings of the 29th Design Automation Conference, ACM-IEEE*, pages 116–119, June 1992.
[7] E. F. Girzyc. Loop Winding—A Data Flow Approach to Fuctional Pipelining. *Proceedings of the IEEE International Symposium of Circuits and Systems*, pages 382–385, May 1987.
[8] P. Gutberlet, J. Muller, H. Kramer, and W. Rosenstielo. Automatic Module Allocation in High Level Synthesis. *Proceedings of the European Design Automation Conference*, pages 328–333, August 1992.
[9] I. G. Harris and A. Orailoğlu. Intertwined Scheduling, Module Selection and Allocation in Time-and-Area Constrained

Synthesis. *International Symposium on Circuits and Systems*, pages 1682–1685, May 1993.

[10] Ian G. Harris and Alex Orailoğlu. SMA: An Algorithm for Scheduling, Module Selection and Allocation. Technical Report CS92-248, University of California, San Diego, June 1992.

[11] C. T. Hwang, Y. C. Hsu, and Y. L. Lin. Scheduling for Functional Pipelining and Loop Winding. In *Proceedings of the 28th Design Automation Conference*, pages 764–769,. 1990.

[12] K. S. Hwang, A. E. Casavant, C-T Chang, and M. A. d'Abreu. Scheduling and Hardware Sharing in Pipelined Data Paths. *Proceedings of the IEEE International Conference on Computer Aided Design*, pages 24–27, November 1989.

[13] M. Ishikawa and G. De Micheli. A Module Selection Algorithm for High-Level Synthesis. *International Symposium on Circuits and Systems*, pages 1777–1780, June 1991.

[14] R. Jain. MOSP: Module Selection for Pipelined Designs with Multi-Cycle Operations. *Proceedings of the IEEE Conference on Computer Aided Design*, pages 212–215, November 1990.

[15] R. Jain, M. J. Mlinar, and A. Parker. Area-Time Model for Synthesis of Non-Pipelined Designs. *Proceedings of the IEEE Conference on Computer Aided Design*, pages 48–51, November 1988.

[16] R. Jain, A. Mujumdar, A. Sharma, and H. Wang. Empirical Evaluation of Some High-Level Synthesis Scheduling Heuristics. *Proceedings of the 28th Design Automation Conference, ACM-IEEE*, pages 210–215, June 1991.

[17] R. Jain, A. Parker, and N. Park. Module Selection for Pipelined Synthesis. *Proceedings of the 25th Design Automation Conference, ACM-IEEE*, pages 542–547, June 1988.

[18] R. Karri and A. Orailoglu. ALPS: Algorithm for Pipeline Data Path Synthesis. In *Proceedings of MICRO-24*, pages 124–131, 1991.

[19] R. Karri and A. Orailoğlu. Scheduling with Rollback Constraints in High-Level Synthesis of Self-Recovering ASICs. In *Proceedings of the 22nd International Symposium on Fault-Tolerant Computing*, pages 519–526, July 1992.

[20] S. Y. Kung, H. J. Whitehouse, and T. Kailath. *VLSI and Modern Signal Processing*. Prentice-Hall, 1985.

[21] F. J. Kurdahi and A. C. Parker. Techniques for Area Estimation of VLSI Layouts. *IEEE Transactions on Computer Aided Design*, pages 81–92, January 1989.

[22] J. Lee, Y. Hsu, and Y. Lin. A New Integer Linear Programming Formulation for the Scheduling Problem in Data-Path Synthesis. *Proceedings of the IEEE Conference on Computer Aided Design*, pages 20–23, November 1989.

[23] S. Note, F. Cathoor, G. Goosens, and H. J. De Man. Combined Hardware Selection and Pipelining in High-Performance Data-Path Design. *IEEE Transactions on Computer Aided Design*, pages 413–423, April 1992.

[24] A. Orailoğlu and D. D. Gajski. Flow Graph Representation. In *Proceedings of the 23rd Design Automation Conference*, pages 503–506, 1986.

[25] A. Orailoğlu and R. Karri. The 'RELIABLE' Microarchitecture Synthesis System. *Journal of Computer and Software Engineering*. In Press.

[26] B. M. Pangrle, F. D. Brewer, D. A. Lobo, and A. Seawright. Relevant Issues in High-Level Connectivity Synthesis. *Proceedings of the 28th Design Automation Conference, ACM-IEEE*, pages 607–610, June 1991.

[27] N. Park and A. C. Parker. Sehwa: A Software Package for Synthesis of Pipelines from Behavioral Specifications. *IEEE Transactions on CAD*, 7(3):356–370, 1988.

[28] P. G. Paulin and J. P. Knight. Force-Directed Scheduling for the Behavioral Synthesis AIC's. *IEEE Transactions on Computer Aided Design*, 8(6):661–679, June 1989.

[29] P. G. Paulin, J. P. Knight, and E. F. Girczyc. HAL: A Multi-Paradigm Approach to Automatic Data Path Synthesis. *Proceedings of the 23rd Design Automation Conference, ACM-IEEE*, July 1986.

[30] L. Ramachandran and D. D. Gajski. Module Selection for Pipelined Synthesis. *Proceedings of the IEEE Conference on Computer Aided Design*, pages 92–95, November 1991.

[31] J-P. Weng and A. Parker. 3D Scheduling: High-Level Synthesis with Floorplanning. *Proceedings of the 28th Design Automation Conference, ACM-IEEE*, pages 668–673, June 1991.

[32] T. W. Williams and K. P. Parker. Design for Testability—A Survey. *IEEE Transactions on Computers*, pages 2–15, 1982.

[33] A. C-H Wu, V. Chaiyakul, and D. D. Gajski. Layout-Area Models for High-Level Synthesis. *Proceedings of the IEEE Conference on Computer Aided Design*, pages 34–37, November 1991.

Author Biographies

Ian G. Harris is a Ph.D. student in the Computer Science and Engineering Department at the University of California, San Diego. His current research interests are Built-In Self-Test and High-Level Synthesis. Mr. Harris received as B.S. in computer science at the Massachusetts Institute of Technology and an M.S. degree in computer science at the University of California, San Diego.

Alex Orailoğlu received the S.B. degree from Harvard College, *cum laude* in Applied Mathematics, in 1977. He has received the M.S. degree in Computer Science from the U. of Illinois, Urbana, in 1979 and the Ph.D. degree in Computer Science, from the U. of Illinois, Urbana, in 1983. Prof. Orailoğlu has been a member of the faculty of the Computer Science and Engineering Department at the University of California, San Diego since 1987. Prof. Orailoğlu leads research efforts at UC San Diego in the area of High-Level Synthesis of VLSI Designs. His research interests include the High-Level Synthesis of Fault-Tolerant Microarchitectures, the Synthesis of Testable VLSI Designs, and Knowledge-Based approaches to VLSI CAD.