Survey Paper

Recent Developments in High-Level Synthesis

YOUN-LONG LIN
Tsing Hua University

We survey recent developments in high level synthesis technology for VLSI design. The need for higher-level design automation tools are discussed first. We then describe some basic techniques for various subtasks of high-level synthesis. Techniques that have been proposed in the past few years (since 1994) for various subtasks of high-level synthesis are surveyed. We also survey some new synthesis objectives including testability, power efficiency, and reliability.

Categories and Subject Descriptors: B.5.1 [**Register-Transfer-Level Implementation**]: Design—*data-path design*; B.5.2 [**Register-Transfer-Level Implementation**]: Reliability and Testing—*automatic synthesis, hardware description languages, optimization*

General Terms: Design, Experimentation, Languages, Reliability

Additional Key Words and Phrases: Design automation, design methodology, high level synthesis, VLSI Design

1. INTRODUCTION

Very Large Scale Integrated Circuits (VLSI) technology provides densities of multiple million gates of random logic per chip. Chips of such complexity are very difficult, if not impossible, to design using the traditional *capture-and-simulate* design methodology. Furthermore, VLSI technology has also reached such a maturity level that it is well understood and no longer provides a competitive edge by itself. Instead, time to market is usually equally, if not more, important than area or speed. The industry has started looking at the product development cycle comprehensively to reduce the design time and to gain a competitive edge in the time-to-market race. Automation of the entire design process from conceptualization to silicon

Part of this article was previously published in Gajski et al. [1992]. The author is supported in part by the National Science Council of R.O.C.
Author's address: Department of Computer Science, Tsing Hua University, Hsin-Chu, Taiwan 30043, R.O.C.

or a *describe-and-synthesize* design methodology has become necessary [Gajski 1992].

As the complexities of chips increase, so will the need for design automation on higher levels of abstraction where functionality is easier to understand and tradeoff is more influential. There are several advantages to automating part or all of the design process and moving automation to higher levels. First, automation assures a much shorter design cycle. Second, it allows for more exploration of different design styles since different designs can be generated and evaluated quickly. Finally, if synthesis algorithms are well understood, design automation tools may out-perform average human designers in meeting most design constraints and requirements.

Synthesis is a translation process from a behavioral description into a structural description, similar to the compilation of a high-level language program in C or Pascal into an assembly program. Each component in the structural description is in turn defined by its own (lower-level) behavioral description. Synthesis, sometimes called design refinement, adds an additional level of detail that provides information needed for the next level of synthesis or for manufacturing of the design. This more detailed design must satisfy design constraints supplied along with the original behavioral description or generated by a previous synthesis step.

We define high-level synthesis (HLS) as a translation process from a behavioral description into a register-transfer-level (RTL) structural description. High-level synthesis has been a very hot research topic over the past 15 years. Comprehensive discussions of specific research approaches to HLS can be found in the literature.[1] We concentrate on its development over the past three years.

The rest of this paper is organized as follows. Section 2 describes the design flow of VLSI when HLS is used. Section 3 outlines the tasks and basic techniques. In Section 4, several new target architectures for HLS are surveyed. Advances in algorithmic techniques are described in Section 5. In Section 6, we survey some new objective functions that recent HLS systems are trying to optimize. Section 7 describes applications of HLS. Finally, Section 8 speculates on future directions of HLS.

2. DESIGN FLOW USING HLS

A behavioral description is used as the starting point for HLS. It specifies behavior in terms of operations, assignment statements, and control constructs in a hardware description language (HDL) (e.g., VHDL [Standard VHDL Language Reference Manual 1988] or Verilog [Thomas and Moorby 1991]). An HDL differs from a software programming language (e.g., C or Pascal) in its capability of expressing timing and concurrency of hardware.

[1]See Camposano and Wolf [1991], Gajski et al. [1992], Michel et al. [1992], and Walker and Camposano [1991].

The output from a high-level synthesizer consists of two parts: a datapath structure at the register-transfer level (RTL), and a specification of the finite state machine to control the datapath. At the RTL level, a datapath is composed of three types of components: functional units (e.g., ALUs, multipliers, and shifters), storage units (e.g., registers and memory), and interconnection units (e.g., buses and multiplexors). The finite state machine specifies every set of microoperations for the datapath to perform during every control step.

In the first step of HLS, the behavioral description is compiled into an internal representation. This process usually includes a series of compiler-like optimizations such as code motion, dead code elimination, constant propagation, common subexpression elimination, and loop unrolling. In addition, it may also apply some hardware-specific transformations such as syntactic variances minimization, retiming, and those taking advantage of the associativity and commutativity properties of certain operations. A control/data flow graph (CDFG) is a commonly used internal representation to capture the behavior. The control-flow graph (CFG) portion of the CDFG captures sequencing, conditional branching, and looping constructs in the behavioral description, and the data-flow graph (DFG) portion captures data-manipulation activity described by a set of assignment statements (operations).

The following three steps form the core of transforming a behavior into a structure: *scheduling, allocation*, and *binding*. Scheduling assigns operations of the behavioral description into control steps. A control step usually corresponds to a cycle of the system clock, the basic time unit of a synchronous digital system. Allocation chooses functional units and storage elements from the *component library*. There may be several alternatives among which the synthesizer must select the one that best matches the design constraints and maximizes the optimization objective. Binding assigns operations to functional units, variables to storage elements, and data transfers to wires or buses such that data can be correctly moved around according to the scheduling.

These three tasks are closely interdependent. For example, an optimal scheduling of operations to control steps without explicit performance and cost information of component allocation is impossible. Similarly, an optimal allocation of components cannot be performed without exact information on operation concurrency. Furthermore, there exists a performance/cost tradeoff by applying these tasks. For example, the most area-efficient design consists of the minimum number of slow components such that a larger number of control steps is required to execute the desired function. On the other hand, allocating more components allows exploiting parallel executions of operations so that a higher performance can be achieved at the expense of higher area cost. By adjusting the constraint parameters, the design space can be explored. One primary application of HLS is in helping the designers with exploring the design space and hence evaluating multiple implementation alternatives quickly.

3. BASIC TECHNIQUES

Given a CDFG, we have to perform scheduling, allocation, and binding to get a datapath. A byproduct of these tasks is a specification of the behavior of the controlling finite state machine. In this section we survey some basic techniques for scheduling, allocation, and binding.

3.1 Scheduling

Scheduling assigns operations in the behavioral description into control steps. Within a control step, a separate functional unit is required to execute each operation assigned to that step. Thus the total number of functional units required in a control step directly corresponds to the number of operations scheduled into it. If more operations are scheduled into each control step, more functional units are necessary, which results in fewer control steps for the design implementation. On the other hand, if fewer operations are scheduled into each control step, fewer functional units are sufficient, but more control steps are needed. Thus scheduling is an important task in HLS because it largely determines the tradeoff between design cost and performance.

There are two classes of scheduling problems: *time-constrained scheduling* and *resource-constrained scheduling*. Time-constrained scheduling minimizes the hardware cost when all operations are to be scheduled into a fixed number of control steps. However, resource-constrained scheduling minimizes the number of control steps needed for executing all operations given a fixed amount of hardware.

Integer Linear Programming (ILP) formulations for both resource-constrained scheduling and time-constrained scheduling [Hwang et al. 1991] have been proposed. However, the execution time of the algorithm grows exponentially with the number of variables and the number of inequalities. In practice the ILP approach is applicable only to very small problems. Nevertheless, the ILP approach has made the problems better understood.

Since the ILP method is impractical for large designs, heuristic methods that run efficiently at the expense of the design optimality have been developed. Heuristic scheduling algorithms generally use two techniques: *constructive approach* and *iterative refinement*. There are many approaches for constructive scheduling. They differ in the selection criteria used to choose the next operation to be scheduled.

The simplest constructive approach is the as soon as possible (ASAP) scheduling. First, operations are sorted into a list according to their topological order. Then, operations are taken from the list one at a time and placed into the earliest possible control step. The other simple approach is the as late as possible (ALAP) scheduling. The ALAP value for an operation defines the latest control step into which an operation can possibly be scheduled. In this approach, given a time constraint in terms of the number of control steps, the algorithm determines the latest possible control step in which an operation must begin its execution. The critical paths within the flow graph can be found by taking the intersection of the

ASAP and ALAP schedules such that the operations that appear in the same control steps in both schedules are on the critical paths.

In both ASAP and ALAP scheduling, no priority is given to operations on the critical path so that those operations may be mistakenly delayed when resource limits are imposed on scheduling. List scheduling overcomes this problem by using a global criterion for selecting the next operation to be scheduled. One example of a global priority function is mobility [Pangrle and Gajski 1987] which is defined as the difference between the ASAP and ALAP values of an operation. The other example is urgency [Girczyc et al. 1985], which is defined as the minimum number of control steps from the bottom at which an operation can be scheduled before a timing constraint is violated. In list scheduling, a list of ready operations is ordered according to the priority function and processed for each state.

The force-directed scheduling (FDS) [Paulin and Knight 1989] is another example that uses a global selection criterion to choose the next operation for scheduling. The FDS algorithm relies on the ASAP and ALAP scheduling algorithms to determine the range of control steps for every operation. Distribution graphs are then created to represent the probabilistic number of operations that will be performed in the control step for each type of operation in each control step. The main goal of the FDS algorithm is to reduce the total number of functional units used in the implementation of the design. The algorithm achieves the objective by uniformly distributing operations of the same type into all the available states.

We call algorithms similar to FDS "constructive" because they construct a solution without performing any backtracking. The decision to schedule an operation into a control step is made on the basis of a partially scheduled DFG; it does not take into account future scheduling of operations into the same control step. Due to the lack of a lookahead scheme and the lack of compromises between early and late decisions, the resulting solution may not be optimal. We can cope with this weakness by iteratively rescheduling some of the operations in the given schedule. One example of this approach was proposed by Park and Kyung [1991] which is based on the paradigm originally proposed for the graph-bisection problem by Kernighan and Lin (KL) [1970]. In this approach, an initial schedule is obtained using any scheduling algorithm. New schedules are obtained by rescheduling a sequence of operations that maximally reduces the scheduling cost. If no improvement is attainable, the process halts.

In the preceding discussions only blocks of straight-line code have been considered. However, in addition to blocks of straight-line code, a realistic design description usually contains both conditional and loop constructs. Many approaches [Kim et al. 1994; Wakabayashi and Yoshimura 1989] have been proposed to schedule conditional constructs. For example, in Wakabayashi and Yoshimura [1989] a conditional vector is used to identify mutually exclusive operations so that an operation can be scheduled in different control steps for different execution instances. Another approach is the path-based As Fast As Possible (AFAP) scheduling [Camposano 1991] which first extracts all possible execution paths from a given behavior and

schedules them independently. The schedules for the different paths are then combined by resolving conflicts among the execution paths. For loop constructs, different approaches, such as pipelining [Park 1988] and loop folding [Girczyc 1987] have been proposed.

3.2 Allocation and Binding

After scheduling, a datapath can be constructed in two steps: *unit allocation* and *unit binding*. Some researchers call unit allocation and unit binding, collectively, *datapath allocation*. Unit allocation determines the number and types of RT components to be used in the design. Since a real RT component library may contain multiple types of functional units, each with different characteristics (e.g., functionality, size, delay, and power dissipation), unit allocation needs to determine the number and types of different functional and storage units from the component library. Unit binding maps the operations, variables, and data transfers in the scheduled CDFG into the functional, storage, and interconnection units, respectively, while ensuring that the design behavior operates correctly on the selected set of components.

Unit binding consists of three interdependent tasks: functional-unit binding, storage binding, and interconnection binding. Functional-unit binding involves the mapping of operations in the behavioral description into the set of selected functional units. Storage binding maps data carriers (e.g., constants, variables, and data structures such as arrays) in the behavioral description onto storage elements (e.g., ROMs, registers, and memory units) in the datapath. Interconnection binding maps every data transfer in the behavior into a set of interconnection units for data routing.

There are three approaches to solve the allocation problem: constructive approaches, which progressively construct a design while traversing the CDFG; decomposition approaches, which decompose the allocation problem into its constituent parts and solve each of them separately; and iterative methods, which try to combine and interleave the solution of the allocation subproblems.

A constructive approach starts with an empty datapath and builds the datapath gradually by adding functional, storage, and interconnection units as necessary. For example, EMUCS [Hitchcock and Thomas 1983] and MABAL [Kucukcakar and Parker 1990] use a global criterion based on functional, storage, and interconnection costs to determine the next element to assign and where to assign it.

Although constructive algorithms are simple, the solutions they find can be far from optimal. In order to improve the quality of the results, some researchers have proposed a decomposition approach, where the allocation process is divided into a sequence of independent tasks; each task is transformed into a well-defined graph-theoretical problem and then solved with a proven technique. In the following, we describe allocation techniques based on three graph-theoretical methods: clique partitioning, left-edge algorithm, and the weighted bipartite matching algorithm.

Tseng and Siewiorek [1986] divided the allocation problem into three tasks of storage, functional-unit, and interconnection allocation which are solved independently by mapping each task to the well-known problem of graph clique-partitioning. In the graph formulation, operations, values, or interconnections are represented by nodes. An edge between two nodes indicates those two nodes can share the same hardware. Thus, the allocation problem, such as storage allocation, can be solved as finding the minimal number of cliques in the graph. Because finding the minimal number of cliques in the graph is an NP-hard problem, in Tseng and Siewiorek [1986] a heuristic approach is taken.

Although the clique-partitioning method when applied to storage allocation can minimize the storage requirements, it totally ignores the interdependence between storage and interconnection allocation. Paulin and Knight [1989] extend the previous method by augmenting the graph edges with weights that reflect the impact on interconnection complexity due to register sharing among variables.

Kurdahi and Parker [1987] apply the left-edge algorithm to solve the register-allocation problem. Unlike the clique-partitioning problem, which is NP-complete, the left-edge algorithm has a polynomial time complexity. Moreover, this algorithm allocates the minimum number of registers. However, it cannot take into account the impact of register allocation on the interconnection cost, as can the weighted version of the clique-partitioning algorithm.

Both the register and functional-unit allocation problems also can be transformed into a weighted bipartite-matching algorithm [Huang et al. 1990]. In this approach, a bipartite graph is first created that contains two disjoint subsets (e.g., one subset of registers and one of variables, or one subset of operations and one of functional units), and an edge connecting two nodes in different subsets represents the node in one subset that can be assigned to the node of the other subset (e.g., the variable in the variable subset can be assigned to the register in the register subset). Thus, the problem of matching each variable to a register is equivalent to the classic job-assignment problem. In Huang et al. [1990], a polynomial time maximum weight matching algorithm is employed to solve the problem. The matching algorithm, like the left-edge algorithm, allocates a minimum number of registers. It also takes partially into consideration the impact of register allocation on interconnection allocation since it can associate weights with the edges.

Given a datapath synthesized by constructive or decomposition methods, a further improvement may be achieved by reallocation, an iterative refinement approach. The most straightforward approach could be a simple assignment exchange using the pairwise exchange or the simulated annealing method. In addition, a more sophisticated branch-and-bound method can be applied by reallocating a group of different types of entities for datapath refinement [Tsay and Hsu 1990].

4. NEW ARCHITECTURES

4.1 Multiport Memory

A multiport memory can support multiple read and/or write accesses simultaneously. Using multiport memory gives the HLS more flexibility in binding variables to storages. Traditionally, variables are grouped into registers, and registers into register files (memory modules) before synthesizing interconnection between memory modules and functional units. Kim and Liu [1995] proposed placing emphasis on the interconnection. They tried to minimize the interconnection first and then to group the variables to form the memory modules later. Lee and Hwang [1995] proposed taking multiport memory into account as early as during scheduling. They defined a *multiport access variable* (MAV) for a control step, and let the scheduling algorithm equalize the MAVs across all control steps in order to achieve a better utilization of the memory.

4.2 Distributed Storage

Most traditional HLS systems are intended for centralized storage units. Every operation must access its input operands from and write its output operands to the storage units. For some regular iterative algorithms, this architecture may not be optimal. Aloqeely and Chen [1994] proposed a sequencer-based datapath architecture. A sequencer is a stack or queue connecting one functional unit to another. By letting variables intelligently "stay" in or "flow" through sequencers in the datapath for future use, high quality datapaths can be synthesized for many signal processing and matrix computation algorithms. Furthermore, less traffic is needed between the functional units and the central storage units, resulting in a simple interconnection complexity. A similar concept called *data routing* has been proposed for both HLS and code generation [Lanneer 1994].

4.3 Partitioned Bus

Ewering [1990] proposed a parameterized architecture called *partitioned buses*. Buses are partitioned into segments. Each functional unit and each storage unit is connected to one part of the segment. Most data transfers occur within a segment. Intersegment transfers are made possible through switches between segments. Since the loading of the bus is light, the synthesized circuit is fast. Scheduling and allocation is done such that the amount of intersegment data transfer is minimized.

5. NEW TECHNIQUES

There have been quite significant advances in HLS algorithms and heuristics. Various methods have been proposed for transforming the CDFG so that a high-quality circuit is easier to synthesize. Several groups have extended the ILP formulation for more general scheduling problems, more capability in handling large designs, or more problems in addition to scheduling. Algorithms originally developed for other problems have been

employed for HLS too. Several groups proposed various neural network models for various HLS tasks. As the solution techniques get more powerful, it is no longer necessary to divide the synthesis problem into multiple subtasks. A notable development is in performing all tasks together as a single task in order to achieve better quality of design.

5.1 Behavioral Transformation

Transformations can be applied to the design representation at various stages of HLS. During compilation of the input behavioral description into a control/data flow graph, several compiler-like optimizations can be performed to remove the extraneous syntactic constructs and redundancies. Flow-graph transformations are used to convert parts of the representation from one style (e.g., control flow) to another (e.g., data flow) and to change the degree of parallelism. Hardware-specific transformations use properties of register-transfer (RT) and logic components to perform optimizations (e.g., replacing a data-flow graph segment that increments a variable with an increment operation).

Since the flow-graph representation typically comes from an imperative language description, several standard compiler optimization techniques, such as constant folding and redundant operator elimination, can be performed on the representation [Aho et al. 1986]. Arrayed variables are another rich source of compiler-level optimizations for HLS [Grant and Denyer 1991; Orailogulu and Gajski 1986]. Since arrays in the behavioral representation get mapped to memories, a reduction in the number of array accesses decreases the overhead resulting from accessing memory structures [Kolson et al. 1994].

Certain compiler transformations are specific to the HDL used for describing the design. For example, when VHDL is used for design description, several approaches proposed by Bhasker and Lee [1990] can identify specific syntactic constructs and replace them with attributes on signals and nets to indicate their functions. Furthermore, in order to reduce the syntactic variation of descriptions with the same semantics, Chaiyakul et al. [1993] proposed a transformation technique using Assignment Decision Diagrams (ADD) to minimize syntactic variance in the description.

The graph capturing the behavior can be restructured. Tree height reduction is one of the commonly used flow-graph transformations to improve the parallelism of the design. Tree height reduction uses the commutativity and distributivity properties of language operators to decrease the height of a long expression chain, and exposes the potential parallelism within a complicated data-flow graph [Hartley and Casavant 1989; Nicolau and Potasman 1991].

Pipelining is another frequently applied transformation in HLS [Park 1988]. Other commonly used transformations include loop folding [Girczyc 1987; Lee et al. 1994], software pipelining [Goossens et al. 1990; Potkonjak and Srivastava 1995], and retiming [Potkonjak and Rabaey 1994].

Hardware-specific transformations at the logic, RT, and system levels can be applied to the intermediate representation. In general, these are

local transformations that use properties of hardware at different design levels to optimize the intermediate representation. For example, at the logic level, we can apply local Boolean optimization techniques [Darringer and Joyner 1980] to the intermediate representation. At the RT level, we can use pattern matching to detect and replace portions of the flow graph with simpler flow-graph segments. The pattern-matching transformations are based on RT semantics of the hardware components corresponding to the flow-graph operators [Rosenstiel 1986]. System level transformations can be used to divide parts of the flow graph into separate processes that run concurrently or in a pipelined fashion [Walker and Thomas 1989].

5.2 Advance in ILP Formulation

Achatz [1993] proposed an extension to the ILP formulation to enable it to handle multifunctional units as well as units with different execution times for different instances of the same operation type.

Wang and Grainger [1994] showed that the number of constraints in the original ILP formulation can be reduced without reducing the explored design space. Therefore, the computation can be more efficient or the formulation can be applicable to larger-sized problems.

Chaudhuri et al. [1994] performed an indepth formal analysis of the structure of the assignment, timing, and resource constraints, evaluated the structure of the scheduling polytope described by these constraints, and showed how to exploit that structure in a well-designed ILP formulation. They also proposed improvements to a well-structured formulation by adding new valid inequalities.

In the OSCAR system [Landwehr et al. 1994], a 0/1 integer programming model is proposed for solving scheduling, allocation, and binding. Gebotys [1994] proposed an integer programming model for the synthesis of multichip architecture. Her model simultaneously deals with partitioning, scheduling, and allocation. The search space is reduced by using a polyhedral theory.

Wilson et al. [1995] generalized the ILP approach in an integrated solution to the scheduling, allocation, and binding in datapath synthesis. A module may execute an arbitrary combination of operations, possibly using a different number of control steps for different types of operations. Operations may be bounded to a variety of modules, possibly requiring a different number of control steps depending on the module chosen.

5.3 New Approaches

Ly et al. [1995] proposed an idea of "behavioral templates" for scheduling. A template locks a number of operations into a relative schedule with respect to one another. It eases the handling of timing constraints, sequential operation modeling, prechaining of certain operations, and hierarchical scheduling.

Many neural net based scheduling algorithms have been proposed. ANSA [Unaltuna and Pitchumani 1995] is a three-phase neural network sched-

uler. Kawaguchi and Tokada [1995] combined simulated annealing and neural networks for the scheduling problem.

Genetic algorithms also find their application in high-level synthesis. Dhodhi et al. [1995] proposed a problem-space genetic algorithm (PSGA) for datapath synthesis. It performs concurrent scheduling and allocation with the objective of minimizing a cost function of the hardware resource and the total execution time. Heijligers et al.'s [1995] genetic algorithm uses an encoding technique that is capable of allocating supplementary resources during scheduling. They also paid attention to runtime efficiency by means of carefully designed analyzing methods.

Ly and Mowchenko [1993] proposed adapting simulated evolution (SE) to high-level synthesis. Simulated evolution has been successfully applied in other CAD areas including routing, partitioning, and placement. SE-based synthesis explores the design space by repeatedly ripping up parts of a design in a probabilistic manner and reconstructing them using application-specific heuristics. It combines rapid design iterations and probabilistic hill climbing to achieve effective design space exploration. The objects subject to ripping up and reconstruction are operation-to-step assignments and various binding.

InSyn [Sharma and Jain 1995] combines allocation and scheduling of functional, storage, and interconnect units into a single phase. It uses the concept of register state (free, busy, and undecided) for optimizing registers in a partial schedule where lifetimes of data values are not yet available. It alleviates bus contention by using reusable data values and broadcast, or selectively slowing down noncritical operations. InSyn can trade off distinct resource sets concurrently; that is, it can trade a functional unit for some registers, or vice versa. Estimation tools are utilized for resource allocation, design space pruning, and evaluation of the synthesized results.

6. ADVANCE IN OBJECTIVE FUNCTION

6.1 More Accurate Estimation

Traditional HLS systems characterize their synthesis results based on very crude estimation. Area is estimated with the sum of the area of functional units, storage units, and interconnects. Timing is estimated assuming that wiring delay is insignificant. As we move into the deep submicron era, both wiring area and timing can no longer be ignored. Many estimation methods have been proposed. Moreover, interaction between HLS and layout has been investigated.

Rim and Jain [1994] proposed a performance estimation tool. Given a data-flow graph, a set of resources, resource delay, and a clock cycle, their tool computes a lower-bound completion time for nonpipelined resource-constrained scheduling problems.

Chaudhuri and Walker [1996] proposed an algorithm for computing lower bounds on the number of functional units of each type required to schedule a data-flow graph in a specified number of control steps. The bounds are

found by relaxing either the precedence constraints or the integrity constraints on the scheduling problem. This bounding method is used to estimate FU area, to generate resource constraints for reducing the search space, or in conjunction with exact formulation for design space exploration.

Mecha et al. [1996] proposed a high-level area estimation method designed for standard cell implementation. They emphasized predicting the interconnection area.

Fang and Wong [1994] proposed simultaneously performing functional unit binding and floorplanning. Munch et al. [1995] proposed an analytical approach to capture the placement and binding problems in a single mixed ILP model. Proposed for a linear bit-slice architecture, the model is capable of minimizing the overall interconnect structure of the datapath.

6.2 HLS for Testability

Testability at the high level can be enhanced by minimizing the number of self-loops (self-adjacent registers). The main concern is in the tradeoff between testability improvement and area overhead.

Since multiplexors and buses can behave as switches, they can help to reduce the test costs. Gupta and Breuer [1995] proposed taking advantage of multiplexors and buses to reduce both the area overhead and test generation costs. They analyzed the locations of switches during the selection of partial scan registers. They also utilized the switches to setup paths for transporting test data.

Flottes et al. [1995] proposed a method to improve the testability by incorporating test constraints during register allocation and interconnect network generation. They analyzed the testability of a design at the behavioral level in the presence of loops and control constructs. Dhodhi et al. [1995] proposed a problem-space genetic algorithm (PSGA) for performing simultaneously scheduling and allocation of testable functional units and registers under area, performance, and testability constraints. Mujumdar et al. [1994] proposed a two-stage approach for binding for testability. First, they used a binder with test cost to generate a nearly loop-free design. Then, they used a loop-breaking method to identify self-loops in the design, and eliminated these loops by alternating the register and module binding.

Potkonjak et al. [1995] proposed methods for transforming a behavioral description so that synthesis of the new description requires less area overhead and partial scan cost. They proposed a two-stage objective function for estimating the area and testability as well as for evaluating the effects of a transformation. Then they used a randomized branch-and-bound steepest decent algorithm to search for the best sequence of transformations.

Testability can be improved via better controller design. Hsu et al. [1996] proposed a controllability measure for high-level circuit description and a high-level synthesis-for-testability technique. They improved the circuit testability by enhancing the controllability of the control flow.

6.3 HLS for Low Power

Due to low power requirements in many portable applications such as notebooks and mobile phones as well as packaging cost consideration, low power design technology is becoming very important in every aspect of VLSI design. A great deal of research effort has been spent on circuit and logic design for low power [Pedram 1996], and in the past few years there has been active research for low power for HLS.

For low power HLS, high-level power estimation techniques are needed. Raghunathan et al. [1996] proposed some techniques for estimating switching activity and power consumption at register-transfer level. They take into account the presence of glitching activity at various data path and control signals.

Rabaey et al. [1995] proposed an approach for high-level design guidance for low power using properties of given algorithms and architectures. Several relevant properties (operation count, the ratio of critical path to available time, spatial locality, and regularity) are identified. Significant emphasis is placed on exploiting the regularity and spatial locality for the optimization of interconnect power. Their scheduling, assignment, and allocation techniques [Mehra and Rabaey 1996] exploit the regularity and common computation patterns in the algorithm to reduce the fanins and fanouts of the interconnection wires, resulting in reduced bus capacitance and simplified bus structure.

Raghunathan and Jha [1994] proposed a datapath allocation method for low power. They also took into account the controller's effect on datapath power dissipation. Goodby et al. [1994] proposed achieving low power via pipelining and module selection. Musoll and Cortadella [1995] proposed a scheduling and resource-binding algorithm for reducing the activity of the function units by minimizing the transitions of their input operands. Kumar et al. [1995] measured activities of operations and carriers in a behavioral specification by simulating the DFG with user-supplied profiling stimuli. They selected a module set and schedule that minimized the switching activity.

Martin and Knight [1995] applied several low power techniques in behavioral synthesis including lowering supply voltage, disabling the clock of idle components, and architectural tradeoff.

6.4 HLS for Reliability

In many critical applications, fault-tolerance is very important. It is desirable that a system is capable of self-recovery in the presence of transient faults. In a self-recovering microarchitecture, intermediate results are compared at regular intervals, and if correct saved in registers (checkpoints). On the other hand, on detecting a fault, it rolls back to a checkpoint and retries. Orailoglu and Karri [1994] proposed a self-recovering microarchitecture synthesis system. They proposed an algorithm for the selection of good checkpoints that has low overhead while meeting the constraint on the number of clock cycles of a retry period.

Self-testing can be carried out concurrently with normal operations using otherwise idle functional units. Singh and Knight [1994] proposed a method to test hardware elements when they are not used. It generated a circuit to cycle test vectors through the idle hardware and produce a signature. Built-in self testing is achieved with reduced test-time overhead.

The hot-carrier reliability issue has also been dealt with in high-level synthesis. Karnik et al. [1995] proposed an iterative redesign method to improve the long-term reliability of a given high-level circuit. They used macromodels of standard circuit elements developed using a reliability simulation tool. Reliability improvement is due to capacitance reduction.

6.5 Controller Issues

Traditionally, the control unit specification is generated after the datapath is synthesis completed. Recently, some authors observed that there exists a tradeoff between the controller and the datapath. Rao and Kurdahi [1994] proposed a hierarchical approach in which the control logic overhead could be taken into account at each level of the hierarchy before the datapath was fully synthesized. Their approach is most suitable for behavior consisting of regular algorithms. Huang and Wolf [1994] studied how datapath allocation affects controller delay. They proposed an allocation approach that considers the controller's effect on the critical path delay. Therefore they were able to minimize the system clock length during allocation.

7. APPLICATIONS OF HLS

Over the years, HLS has been successfully applied for the designs of several narrow, application-specific domains. For example, many HLS systems, such as Cathedral [Lanneer et al. 1990], Hyper [Chu et al. 1989], and Phideo [Lippens et al. 1991], provide a design environment for digital signal processing (DSP) applications with various levels of throughput requirements. In contrast to computationally intensive DSP designs, many other HLS systems, such as HIS [Camposano et al. 1991], Callas [Ledeux et al. 1993], and Olympus [DeMicheli et al. 1990], have been designed for control-dominated circuit designs. In addition, an HLS system Mimola/Honeywell [Zimmermann 1980] has been created for instruction set processor design. The System Architect's Workbench from CMU [Thomas et al. 1990] has been used in designs for automotive applications [Fuhrman 1991].

A high-level synthesis tool even finds its use in education [Reese 1994]. With a datapath synthesizer capable of exploiting the design space, students are able to try different architectural decisions and evaluate their effectiveness.

Recently, HLS has been used in an embedded system design environment. A typical embedded system consists of off-the-shelf instruction set processors, memory, and ASICs. Each application is partitioned into two interacting parts: software on the instruction set processor and hardware on the ASICs. Several approaches [Ernst and Henkel 1992; Gupta and

DeMicheli 1992] have been developed to tackle the hardware–software codesign problems. An HLS tool is definitely needed to quickly give the cost and performance figures of the synthesized ASIC.

With the advent of field-programmable hardware such as field programmable gate arrays (FPGAs) and field programmable interconnect components (FPICs), many designs are implemented in FPGAs. Fast turnaround is more important to FPGA- than ASIC-based implementation. Using HLS provides an unmatchable fast turnaround time.

Taking advantage of the reprogrammability of FPGAs and FPICs, hardware emulation provides a very fast means for design verification. Currently, most hardware emulators take as input a logic-level structural description. With HLS, extensive verification can be performed earlier in the design process. For example, in Wehn et al. [1992] an HLS scheme is used for automatic mapping of behavioral description onto an ASIC emulation board.

In a custom computer, hardware resources are configured according to the characteristics of the application programs. HLS is essential here in converting an algorithm or a code segment into an FPGA/FPIC programming bitstream.

8. FUTURE DIRECTIONS

Although we have seen its many applications, HLS today is still not as indispensable as layout or logic synthesis. For high-level synthesis to move into mainstream design practice, its area efficiency and performance level must be competitive with those of traditional approaches. A possible approach is to embed more domain knowledge into the synthesizer. For instance, many DSP-oriented HLS systems have been developed. Unfortunately, this approach has one serious drawback. The more domain-specific a tool is, the smaller its customer base will be. Since HLS development is a difficult and complicated task, a small customer base may not be able to support its healthy growth.

In the deep submicron era, wiring delay will dominate gate delay. Therefore wiring delay must be taken into account as early in the design process as possible. Work has been done in dealing with the interaction between logic synthesis and layout, and we have surveyed work on simultaneous allocation and floorplanning. In the future, we shall see more interaction between high-level synthesis and layout. Every step of HLS should take layout into account.

With the rapid increase in the integration level, we must prepare to deal with (complicate) system-on-a-chip issues. It is impractical to design multimillion gate chips from scratch. Therefore it is essential to make good use of existing designs. A successful HLS tool should provide a smooth mechanism for the user to reuse a wide variety of components.

We also have to pay attention to design verification. Although HLS is indeed effective in reducing the time for synthesis, a quite significant portion of the design time is spent on simulation and validation, for which

HLS does not provide any help (to date). Since verification at the higher level is more efficient than that at the lower level, a correctness-preserving HLS design flow will significantly save validation time. Therefore it will give the designer more incentive to use HLS tools.

REFERENCES

ACHATZ, H. 1993. Extended 0/1 LP formulation for the scheduling problem in high level synthesis. In *Proceedings, EURO-DAC'93 with EURO-VHDL'93,* 226–231.

AHO, A. V., SETHI, R., AND ULLMAN, J. D. 1986. *Compilers: Principles, Techniques and Tools.* Addison-Wesley, Reading, MA.

ALOQEELY, M. AND CHEN, C. Y. R. 1994. Sequencer-based datapath synthesis of regular iterative algorithms. In *Proceedings of the Design Automation Conference* (San Diego, CA), 155–160.

BHASKER, J. AND LEE, H.-C. 1990. An optimizer for hardware synthesis. *IEEE Des. Test Comput. 7*, 5 (Oct.), 20–36.

CAMPOSANO, R. 1991. Path-based scheduling for synthesis. *IEEE Trans. Comput. Aided Des. Integ. Circuits Syst. 10,* 1 (Jan.), 85–93.

CAMPOSANO, R. AND WOLF, W., EDS. 1991. *High Level VLSI Synthesis,* Kluwer Academic, Norwell, MA.

CAMPOSANO, R., BERGAMASCHI, R. A., HAYNES, C., PAYER, M., AND WU, S. M. 1991. The IBM high level synthesis system. In *High Level VLSI Synthesis,* R. Camposano and W. Wolf, Eds. Kluwer Academic, Norwell, MA.

CHAIYAKUL, V., GAJSKI, D. D., AND RAMACHANDRAN, L. 1993. High level transformation for minimizing syntactic variances. In *Proceedings of the Design Automation Conference (DAC)* (Dallas, TX, June), 413–418.

CHAUDHURI, S. AND WALKER, R. A. 1996. Computing lower bounds on functional units before scheduling. *IEEE Trans. Very Large Scale Integ. (VLSI) Syst. 4,* (June), 273–279.

CHAUDHURI, S., WALKER, R. A., AND MITCHELL, J. E. 1994. Analyzing and exploiting the structure of the constraints in the ILP approach to the scheduling problem. *IEEE Trans. Very Large Scale Integ. (VLSI) Syst. 2,* 4, 456–471.

CHU, C. M., POTKONJAK, M., THALER, M., AND RABAEY, J. 1989. HYPER: An interactive synthesis environment for high performance real time applications. In *Proceedings of the International Conference on Computer Design (ICCD)* (Oct.), 432–435.

DARRINGER, J. A. AND JOYNER, W. H. 1980. A new look at logic synthesis. In *Proceedings of the Design Automation Conference (DAC).*

DHODHI, M. K., AHMAD, I., AND ISMAEEL, A. A. 1995. High level synthesis of data paths for easy testability. *IEE Proc. Circuits, Devices Syst. 142,* 4 (Aug.), 209–216.

DHODHI, M. K., HIELSCHER, F. H., STORER, R. H., AND BHASKER, J. 1995. Datapath synthesis using a problem-space genetic algorithm. *IEEE Trans. Comput.-Aided Des. Integ. Circuits Syst. 14,* 8 (Aug.), 934–944.

ERNST, R. AND HENKEL, J. 1992. Hardware-software codesign of embedded controllers based on hardware extraction. In handout from *The First International Workshop on Hardware-Software Codesign.*

EWERING, C. 1990. Automatic high level synthesis of partitioned busses. In *Digest of Technical Papers, International Conference on Computer-Aided Design (ICCAD)* (Santa Clara, CA, Nov.), 304–307.

FANG, Y.-M. AND WONG, D. F. 1994. Simultaneous functional-unit binding and floorplanning. In *Digest of Technical Papers, International Conference on Computer-Aided Design (ICCAD)* (Santa Clara, CA), 317–321.

FLOTTES, M. L., HAMMAD, D., AND ROUZEYRE, B. 1995. High level synthesis for easy testability. In *Proceedings, The European Design and Test Conference* (Paris, France), 198–206.

FUHRMAN, T. E. 1991. Industrial extensions to university high level synthesis tools: Making it work in the real world. In *Proceedings of the Design Automation Conference (DAC)* (San Francisco, CA, June), 520–525.

GAJSKI, D. D., DUTT, N., WU, A., AND LIN, S. 1992. *High Level Synthesis—Introduction to Chip and System Design.* Kluwer Academic, Norwell, MA.

GEBOTYS, C. H. 1994. An optimization approach to the synthesis of multichip architectures. *IEEE Trans. Very Large Scale Integ. (VLSI) Syst. 2,* 1, 11–20.

GIRCZYC, E. F. 1987. Loop winding—a data flow approach to functional pipelining. In *Proceedings of the International Symposium on Circuits and Systems,* 382–385.

GIRCZYC, E. F., BUHR, R. J. A., AND KNIGHT, J. P. 1985. Applicability of a subset of Ada as an algorithmic hardware description language for graph-based hardware compilation. *IEEE Trans. Comput.-Aided Des. Integ. Circuits Syst. 4,* 2 (April).

GOODBY, L., ORAILOGLU, A., AND CHAU, P. M. 1994. Microarchitecture synthesis of performance-constrained low-power VLSI designs. In *Proceedings of the International Conference on Computer Design (ICCD),* 323–326.

GOOSSENS, G., RABAEY, J., VANDEWALLE, J., AND DE MAN, H. 1990. An efficient microcode compiler for application specific DSP processors. *IEEE Trans. Comput.-Aided Des. Integ. Circuits Syst. 9,* 9 (Sept.), 925–937.

GRANT, D. M. AND DENYER, P. B. 1991. Address generation for array access based on modulus *m* counter. In *Proceedings of the European Conference on Design Automation (EDAC)* (Paris, France), 118–123.

GUPTA, R. AND BREUER, M. A. 1995. Partial scan design of register-transfer level circuits. *J. Electr. Test. Theory Appl. 7,* 1–2 (Aug.), 25–46.

GUPTA, R. AND DE MICHELI, G. 1990. Partitioning of functional models of synchronous digital systems. In *Digest of Technical Papers, International Conference on Computer-Aided Design (ICCAD)* (Santa Clara, CA, Nov.), 216–219.

GUPTA, R. K. AND DE MICHELI, G. 1992. System level synthesis using re-programmable components. In *Proceedings of the European Conference on Design Automatio (EDAC)* (Amsterdam, the Netherlands), 2–7.

HARTLEY, R. AND CASAVANT, A. 1989. Tree-height minimization in pipelined architectures. In *Digest of Technical Papers, International Conference on Computer-Aided Design (ICCAD)* (Santa Clara, CA, Nov.), 112–115.

HEIJLIGERS, M. J. M., CLUITMANS, L. J. M., AND JESS, J. A. G. 1995. High level synthesis scheduling and allocation using genetic algorithms. In *Proceedings of the Asia and South-Pacific Design Automation Conference (ASP-DAC)* (Chiba, Japan), 61–66.

HITCHCOCK, C. Y. AND THOMAS, D. E. 1983. A method of automatic data path synthesis. In *Proceedings of the Design Automation Conference (DAC)* (June), 484–489.

HSU, F. F., RUDNICK, E. M., AND PATEL, J. H. 1996. Enhancing high-level control-flow for improved testability. In *Digest of Technical Papers, International Conference on Computer-Aided Design (ICCAD)* (San Jose, CA, Nov.), 322–328.

HUANG, C.-Y., CHEN, Y.-S., LIN, Y.-L., AND HSU, Y.-C. 1990. Data path allocation based on bipartite weighted matching. In *Proceedings of the Design Automation Conference (DAC)* (Orlando, FL, June), 499–504.

HUANG, S. C.-Y. AND WOLF, W. H. 1994. How datapath allocation affects controller delay. In *Proceedings of the International Symposium on High Level Synthesis* (Niagra Falls, Canada), 158–163.

HWANG, C.-T., LEE, J.-H., AND HSU, Y.-C. 1991. A formal approach to the scheduling problem in high level synthesis. *IEEE Trans. Comput.-Aided Des. Integ. Circuits Syst. 10,* 4 (April), 464–475.

IEEE 1988. Sehwa: A software package for synthesis of pipelined from behavioral specifications. *IEEE Trans. Comput.-Aided Des. Integ. Circuits Syst. 7,* 3 (March), 356–370.

KARNIK, T., TENG, C.-C., AND KANG, S.-M. 1995. High-level hot carrier reliability-driven synthesis using macro-models. In *Proceedings, the IEEE 1995 Custom Integrated Circuits Conference* (Santa Clara, CA) 65–68.

KAWAGUCHI, T. AND TODAKA, T. 1995. Operation scheduling by annealed neural networks. *IEICE Trans. Fund. Electr. Commun. Comput. Sci. E78-A,* 6 (June), 656–663.

KERNIGHAN, K. H. AND LIN, S. 1970. An efficient heuristic procedure for partitioning graph. *Bell Syst. Tech. J. 49,* 2 (Feb.), 291–307.

KUMAR, N., KATKOORI, S., RADER, L., AND VEMURI, R. 1995. Profile-driven behavioral synthesis for low-power VLSI systems. *IEEE Des. Test Comput. 12,* 3 (Fall), 70–84.

KIM, T. AND LIU, C. L. 1995. A new approach to the multiport memory allocation problem in datapath synthesis. *INTEGRATION, VLSI J. 19,* 3, 133–160.

KIM, T., YONEZAWA, N., LIU, J. W. S., AND LIU, C. L. 1994. A scheduling algorithm for conditional resource sharing—a hierarchical reduction approach. *IEEE Trans. Comput.-Aided Des. Integ. Circuits Syst. 13,* 4 (April), 425–437.

KOLSON, D. J., NICOLAU, A., AND DUTT, N. 1994. Integrating program transformations in the memory-based synthesis of image and video algorithms. In *Digest of Technical Papers, International Conference on Computer-Aided Design (ICCAD)* (San Jose, CA), 27–30.

KUCUKCAKAR, K. AND PARKER, A. C. 1990. Data path tradeoff using MABAL. In *Proceedings of the Design Automation Conference (DAC)* (Orlando, FL, June), 511–516.

KUCUKCAKAR, K. AND PARKER, A. C. 1991. CHOP: A constraint-driven system level partitioner. In *Proceedings of the Design Automation Conference (DAC)* (San Francisco, CA, June), 514–519.

KURDAHI, F. J. AND PARKER, A. C. 1987. REAL: A program for register allocation. In *Proceedings of the Design Automation Conference (DAC)* (Miami Beach, FL, June), 210–215.

LAGNESE, E. AND THOMAS, D. 1991. Architectural partitioning for system level partitioner. *IEEE Trans. Comput.-Aided Des. Integ. Circuits Syst. 10,* 7 (July), 847–860.

LANNEER, D., CORNERO, M., GOOSSENS, G., AND DE MAN, H. 1994. Data routing: A paradigm for efficient datapath synthesis and code generation. In *Proceedings of the International Symposium on High Level Synthesis* (Niagra Falls, Canada), 17–22.

LANNEER, D., NOTE, S., DEPUYDT, F., PAUWELS, M., CATTHOOR, F., GOOSSENS, G., AND DE MAN, H. 1990. Architectural synthesis for medium and high throughput signal processing with the new CATHEDRAL environment. In *High Level VLSI Synthesis,* R. Camposano and W. Wolf, Eds., Kluwer Academic, Norwell, MA.

LANDWEHR, B., MARWEDEL, P., AND DOMER, R. 1994. OSCAR: Optimum simultaneous scheduling, allocation and resource binding based on integer programming. In *Proceedings, EURO-DAC '94 with EURO-VHDL '94,* 90–95.

LEDEUX, S., ET AL. 1993. The Siemens high level synthesis system CALLAS. *IEEE Trans. Very Large Scale Integ. (VLSI) Syst. 1,* 3 (Sept.), 144–153.

LEE, H.-D. AND HWANG, S.-Y. 1995. A scheduling algorithm for multiport memory minimization in datapath synthesis. In *Proceedings of the Asia and South-Pacific Design Automation Conference (ASP-DAC),* 93–100.

LEE, T.-F., WU, A. C.-H., LIN, Y.-L., AND GAJSKI, D. 1994. A transformation-based method for loop folding. *IEEE Trans. Comput.-Aided Des. Integ. Circuits Syst. 13,* 4 (April), 439–450.

LIPPENS, P., ET AL. 1991. PHIDEO: A silicon compiler for high speed algorithms. In *Proceedings of the European Conference on Design Automatio (EDAC),* 436–441.

LY, T., KNAPP, D., MILLER, R., AND MACMILLEN, D. 1995. Scheduling using behavioral templates. In *Proceedings of the Design Automation Conference (DAC)* (San Francisco, CA), 101–106.

LY, T. A. AND MOWCHENKO, J. T. 1993. Applying simulated evolution to high level synthesis. *IEEE Trans. Comput.-Aided Des. Integ. Circuits Syst. 12,* 3 (March), 389–409.

MARTIN, R. S. AND KNIGHT, J. P. 1995. Power-profiler: Optimizing ASICs power consumption at the behavioral level. In *Proceedings of the Design Automation Conference (DAC)* (San Francisco, CA), 42–47.

MCFARLAND, M. C. 1986. Using bottom-up design techniques in the synthesis of digital hardware from abstract behavioral descriptions. In *Proceedings of the Design Automation Conference (DAC),* (Las Vegas, NV, June), 474–480.

MCFARLAND, M. C. AND KOWALSKI, T. J. 1990. Incorporating bottom-up design into hardware synthesis. *IEEE Trans. Comput. Aided Des. Integ. Circuits Syst. 9,* 9 (Sept.), 938–950.

MECHA, H., FERNANDEZ, M., TIRADE, F., SEPTIEN, J., MOTES, D., AND OLCOZ, K. 1996. A method for area estimation of data path in high level synthesis. *IEEE Trans. Comput. Aided Des. Integ. Circuits Syst. 15,* 2 (Feb.), 258–265.

MEHRA, R. AND RABAEY, J. 1996. Exploiting regularity for low-power design. In *Digest of Technical Papers, International Conference on Computer-Aided Design (ICCAD),* (San Jose, CA, Nov.), 166–172.

MICHEL, P., LAUTHER, U., AND DUZY, P., EDS. 1992. *The Synthesis Approach to Digital System Design.* Kluwer Academic, Norwell, MA.

DE MICHELI, G., KU, D. C., MAILHOT, F., AND TRUONG, T. 1990. The Olympus synthesis system for digital design. *IEEE Des. Test Mag. 7*, 1, 37–53.

MUJUMDAR, A., JAIN, R., AND SALUJA, K. 1994. Incorporating testability considerations in high level synthesis. *J. Electr. Test. Theory Appl. 5,* 1 (Feb.), 43–55.

MUNCH, M., WEHN, N., AND GLESNER, M. 1995. Optimum simultaneous placement and binding for bit-slice architectures. In *Proceedings of the Asia and South-Pacific Design Automation Conference (ASP-DAC)* (Chiba, Japan), 735–740.

MUSOLL, E. AND CORTADELLA, J. 1995. Scheduling and resource binding for low power. In *Proceedings, the Eighth Symposium on System Synthesis,* 104–109.

NICOLAU, A. AND POTASMAN, R. 1991. Incremental tree height reduction for high level synthesis. In *Proceedings of the Design Automation Conference (DAC)* (San Francisco, CA, June), 770–774.

ORAILOGULU, A. AND GAJSKI, D. D. 1986. Flow graph representation. In *Proceedings of the Design Automation Conference (DAC)* (Las Vegas, NV, June), 503–509.

ORAILOGLU, A. AND KARRI, R. 1994. Coactive scheduling and checkpoint determination during high level synthesis of self-recovering microarchitectures. *IEEE Trans. Very Large Scale Integ. (VLSI) Syst. 2,* 3 (Sept.), 304–311.

PANGRLE, B. M. AND GAJSKI, D. D. 1987. Design tools for intelligent silicon compilation. *IEEE Trans. Comput.-Aided Des. Integ. Circuits Syst. 6,* 6 (Nov.), 1098–1112.

PARK, I.-C. AND KYUNG, C.-M. 1991. Fast and near optimal scheduling in automatic data path synthesis. In *Proceedings of the Design Automation Conference (DAC)* (San Francisco, CA, June), 680–685.

PARK, N. AND PARKER, A. 1988. Sehwa: A software package for synthesis of pipelined data path from behavioral specification. *IEEE Trans. Comput. Aided Des. Integrated Circuits Syst.* 7, 3 (March), 356–370.

PAULIN, P. G. AND KNIGHT, J. P. 1989. Force-directed scheduling for the behavioral synthesis of ASICs. *IEEE Trans. Comput.-Aided Des. Integ. Circuits Syst. 8,* 6 (June), 661–679.

PEDRAM, M. 1996. Power minimization in IC design: Principles and applications. *ACM Trans. Des. Autom. Electr. Syst. 1,* 1 (Jan.), 3–56.

POTKONJAK, M. AND RABAEY, J. 1994. Optimizing resource utilization using transformation. *IEEE Trans. Comput.-Aided Des. Integ. Circuits Syst. 13,* 3 (March), 277–292.

POTKONJAK, M. AND SRIVASTAVA, M. 1995. Rephasing: A transformation technique for the manipulation of timing constraints. In *Proceedings of the Design Automation Conference (DAC)* (San Francisco, CA), 107–112.

POTKONJAK, M., DEY, S., AND ROY, R. K. 1995. Synthesis-for-testability using transformations. In *Proceedings of the Asia and South-Pacific Design Automation Conference (ASP-DAC),* 485–490.

POWELL, S. R. AND CHAU, P. M. 1990. Estimating power dissipation of VLSI signal processing chip: The PFA technique. In *Proceedings of VLSI Signal Processing IV,* 250–259.

RABAEY, J., GUERRA, L., AND MEHRA, R. 1995. Design guidance in the power dimension. *1995 International Conference on Acoustics, Speech, and Signal Processing,* 2837–2840.

RAGHUNATHAN, A. AND JHA, N. K. 1994. Behavioral synthesis for low power. In *Proceedings of the International Conference on Computer Design (ICCD),* 318–322.

RAGHUNATHAN, A., DEY, S., AND JHA, N. K. 1996. Register-transfer level estimation techniques for switching activity and power consumption. In *Digest of Technical Papers, International Conference on Computer-Aided Design (ICCAD)* (San Jose, CA, Nov.), 158–165.

RAO, D. S. AND KURDAHI, F. J. 1994. Controller and datapath trade-offs in hierarchical RT-level synthesis. In *Proceedings of the International Symposium on High Level Synthesis,* 152–157.

REESE, B. 1994. Using HYPER to teach datapath design techniques in an ASIC design course. In *Proceedings, IEEE International ASIC Conference and Exhibit,* 200–203.

RIM, M. AND JAIN, R. 1994. Lower-bound performance estimation for the high level synthesis scheduling problem. *IEEE Trans. Comput.-Aided Des. Integ. Circuits Syst. 13,* 4 (April), 451–458.

ROSENSTIEL, W. 1986. Optimizations in high level synthesis. *Microprocess. Microprogram. 18,* 543–549.

SHARMA, A. AND JAIN, R. 1995. InSyn: Integrated scheduling for DSP applications. *IEEE Trans. Signal Process. 43,* 8 (Aug.), 1966–1977.

SINGH, R. AND KNIGHT, J. 1994. Concurrent testing in high level synthesis. In *Proceedings of the International Symposium on High Level Synthesis* (Niagra Falls, Canada), 96–103.

STANDARD VHDL LANGUAGE REFERENCE MANUAL. 1988. The Institute of Electrical and Electronics Engineers, Inc., New York.

THOMAS, D. E. AND MOORBY, P. 1991. *The Verilog Hardware Description Language.* Kluwer Academic, Norwell, MA.

THOMAS, D., LAGNESE, E., WALKER, R., NESTOR, J., RAJEN, J., AND BLACKBURN, R. 1990. *Algorithmic and Register-Transfer Level Synthesis: The System Architect's Workbench.* Kluwer Academic, Norwell, MA.

TSAY, F.-S. AND HSU, Y.-C. 1990. Data path construction and refinement. In *Digest of Technical Papers, International Conference on Computer-Aided Design (ICCAD)* (Santa Clara, CA, Nov.), 308–311.

TSENG, C. J. AND SIEWIOREK, D. P. 1986. Automatic synthesis of data path on digital systems. *IEEE Trans. Comput. Aided Des. Integ. Circuits Syst. 5,* 3 (July), 379–395.

UNALTUNA, M. K. AND PITCHUMANI, V. 1995. ANSA: A new neural net based scheduling algorithm for high level synthesis. In *IEEE Symposium on Circuits and Systems,* Vol 1 (Seattle, WA), 385–388.

VAHID, F. AND GAJSKI, D. D. 1992. Specification partitioning for system design. In *Proceedings of the Design Automation Conference (DAC)* (Anaheim, CA, June), 219–224.

WAKABAYASHI, K. AND YOSHIMURA, T. 1989. A resource sharing control synthesis method for conditional branches. In *Digest of Technical Papers, International Conference on Computer-Aided Design (ICCAD)* (Santa Clara, CA, Nov.), 62–65.

WALKER, R. A. AND CAMPOSANO, R., EDS. 1991. *A Survey of High Level Synthesis.* Kluwer Academic, Norwell, MA.

WALKER, R. A. AND THOMAS, D. E. 1989. Behavioral transformations for algorithmic level IC design. *IEEE Trans. Comput. Aided Des. Integ. Circuits Syst. 8,* 10 (Oct.), 1115–1128.

WANG, X. J. AND GRAINGER, S. R. 1994. The reduction of the number of equations in the ILP formulation for the scheduling problem in high level synthesis. In *Proceedings, the Second International Conference on Concurrent Engineering and Electronic Design Automation* (Bournemouth, UK), 483–487.

WEHN, N., HERPEL, H.-J., HOLLSTEIN, T., POECHMUELLER, P., AND GLESNER, M. 1992. High level synthesis in a rapid-prototype environment for mechantronic systems. In *Proceedings, EURO-DAC'92 with EURO-VHDL'92* (Hamburg, Germany), 188–193.

WILSON, T. C., MUKHERJEE, N., GARG, M. K., AND BANERJI, D. K. 1995. An ILP solution for optimum scheduling, module and register allocation, and operation binding in datapath synthesis. *VLSI Design 3,* 1, 21–36.

ZIMMERMANN, G. 1980. MDS: The Mimola design method. *J. Digital Syst. 4,* 3, 337–369.

Received October 1996; revised December 1996; accepted December 1996.