Synthesis Systems for Digital Design (invited paper)

Giovanni De Micheli
Computer Systems Laboratory
Stanford University

Abstract

This paper describes computer-aided synthesis systems for digital design. It addresses the design of VLSI single-chip digital systems from behavioral specifications to semi-custom implementation, such as macro-cells, standard-cells, sea-of-gates or programmable gate-arrays. An overview of state of the art synthesis system is presented first, with an emphasis on those systems that are either commercially available or that are developed at major universities. Then the Olympus synthesis system, developed at Stanford University, is described in detail, as well as the results of its use for three chip designs.

Giovanni De Micheli is Associate Professor of Electrical Engineering and, by courtesy, of Computer Science at Stanford University. From 1984 to 1986 he worked at the IBM T.J. Watson Research Center, Yorktown Heights, New York, where he was project leader of the Design Automation Workstation group. He was co-director of the Advanced Study Institute on Logic Synthesis and Silicon Compilation, held in L'Aquila, Italy, under the sponsorship of NATO in 1986 and in 1987. He received a Dr. Eng. degree, *Summa cum Laude*, in Nuclear Engineering from the Politecnico di Milano, Italy, in 1979, a M.S. and a Ph.D. degree in Electrical Engineering and Computer Science from the University of California, Berkeley in 1980 and 1983 respectively. His research interests include several aspects of the computer-aided design of integrated circuits with particular emphasis on automated synthesis, optimization and verification of VLSI circuits.

1 Introduction

Very Large Scale Integration (VLSI) processors are the heart of computers, signal and image processors and other systems that elaborate digital information. Their design is critical to the progress of the electronic industry: more powerful processors are needed every day and their design time has to shrink because of the competitiveness in the marketplace. Application-Specific Integrated Circuits (ASIC) have today a large share of the market, due to the expansion of the semiconductor sector into the communication, automotive and consumer field. In these areas, it has shown to be effective to migrate applications from software implementations to specific hardware chips.

Computer-Aided Design (CAD) techniques are necessary ingredients for the fast design of processors which yield maximal performance with the state-of-the-art technology. In particular, circuit synthesis and optimization techniques have been effectively used to synthesize chips (or components) from high-level specifications without (or with limited) human intervention. The confidence in computer-aided circuit synthesis methods has grown tremendously in the recent years.

VLSI circuits are designed with different styles and technologies, according to the criticality of the performances and the projected production volume [1]. Today semi-custom implementations (e.g. standard-cells, gate-arrays or sea-of-gates) and structured custom design methodology (e.g. macro-cell based design) have shown to be viable solutions to achieve high-performance circuits, with reduced engineering costs due to the automation of all (or part of) the design process.

The wide-spread use of the semi-custom methodologies was boot-strapped by the progress in CAD synthesis systems. At first, **physical design synthesis** systems were developed to fully support the placement and wiring of standard cells, gate arrays and sea of gates. These tasks are fully understood today, even though they remain difficult problems from a computational standpoint. Several design systems are commercially available from vendors, or distributed from universities, to support physical design. Research in this area is now directed to some critical problems (e.g. layout of very high preformance systems) or to those that involve emerging design methodologies (e.g. electrically programmable gate-arrays [28]).

In the recent years, **logic synthesis systems** have been the object of extensive investigation and commercial implementations have shown to be practical for product-level design of digital circuits. Semi-custom circuit implementations have again motivated the development of logic synthesis techniques, particularly in the direction of exploiting multiple-level (or multiple-stage) logic structures. Multiple-level logic circuit implementations are more flexible and faster than two-level implementations, such as Programmable Logic Arrays. As a result, several techniques for multiple-level logic synthesis techniques have been investigated and clever algorithms for combinational logic synthesis have been reported in the literature [1] [2] [3] [4]. Most logic synthesis systems provide two capabilities: technology independent circuit optimization and technology mapping into library parts. Most systems support at present only synchronous logic circuit design, that guarantees race-free design.

In general semi-custom design entry is done by means of logic schematic capture programs. The major task of a designer is to transform the hardware specification into a logic schematic, which is generally validated by simulation. However, conceiving and entering logic schematics of large scale systems is a time-consuming, tedious and error-prone task. Logic-level languages, with structured programming constructs have become common design representations. Nevertheless the complexity of describing large designs can be reduced only by raising the level of abstraction from the logic to the functional domain.

Today **high-level synthesis** systems are the object of intensive investigation and start being available in the commercial arena. Such system support the automatic, or computer-assisted, transformation of behavioral specifications of a digital system in a Hardware Description Language (HDL) to a logic specification. The HDL descriptions, that have been most commonly used, are based on procedural semantics, that can be very powerful in terms of expressing hardware and communication. Transformation into logic level description is a complex task, because of the multiple choices that can be taken in partitioning the system into serial or parallel computation blocks, under constraints on the figures of merit of the hardware being designed, such as silicon area and timing. The complexity of the task, the various flavors of HDLs available and the remoteness of the behavioral description primitives (that are better understood by software than by hardware designers) justify the delay of wide-spread use of high level synthesis systems. I think that such system will mature and be used widely in the early nineties.

The combination of high, logic and physical level synthesis systems into a **hardware compiler**, called also silicon compiler or full automated synthesis system, is one of the major challenges in the design automation field [1]. Silicon compilation was postulated in the seventies by Johannsen [7]. However more than one decade of advances in CAD algorithms and tools was required to make it acceptable to the designer community. Today fully automatic hardware compilation has shown to be feasible and it has also proven to be a time-effective way of designing ASIC and general purpose processors [6]. Among the main advantages of hardware compilation, I would like to stress the possibility of experimenting with architectural trade-offs to match the implementation technology. Indeed, architectural changes can be performed as quickly as the modification of a program and hardware compilation time is in the order of minutes of computation.

In this paper I review the state of the art of those synthesis systems that are either commercially available or that are developed at major universities. It is important to remember that other synthesis systems have been developed by industries, research centers and consortia, but they are not available to the general user. Moreover, due to the evolution of the focus of research and interest in the field, I concentrate on the high-level and logic-level facets of hardware compilers. Then, I present the Olympus synthesis system, that is being designed at Stanford University. Olympus supports the design of synchronous digital systems from behavioral specifications into semi-custom implementation. Eventually, I present some results related to the use of the Olympus system for three chip designs.

2 Overview of Synthesis Systems

A general taxonomy of the existing synthesis systems is hard to determine, because of the evolving nature of the field. However, some distinctions may be useful.

Commercial systems and research systems differ in many respects. The former systems are designed to satisfy the needs of a set of customers and their development is driven by this necessity. They are generally more robust than research systems, because they are designed on the basis of methodologies and algorithms developed and tested for some years. Research systems are at the forefront of the most recent advances in the field. However, in some cases, the full design path from description to silicon is not supported or these systems have been used only for few real hardware designs.

Most hardware synthesis systems claim to support full top-down synthesis. However, some are limited in the set of tasks and in the representation levels they support. Few systems support high-level synthesis

from truly behavioral specifications. Some systems do not support physical design. However, this is a minor drawback, because they provide translations into the format supported by common physical design systems.

Some hardware synthesis systems are dedicated to some sectors of the marked, for example Digital Signal Processors (DSP). Some others are geared toward processor design. It is difficult to provide full support to designing a wide spectrum of applications, due to the assumptions on the related hardware primitives and to the different priorities in optimizing the figures of merit of the design.

An important feature of some synthesis systems is the openness to augmenting them by incorporating user designed routines for the synthesis of special purpose components, or for specific synthesis of some components. Consider for example the synthesis of logic circuits that involve arithmetic multiplication: the logic primitives for synthesizing such a component is different from the basic logic gates and must be exploited in order to achieve an efficient design.

The links of the hardware synthesis systems to the simulation and verification systems is vital. Indeed, high-level descriptions must be routinely simulated and validated. In the near future, formal verification techniques will play a larger role in the validation phase.

2.1 Early Synthesis Systems and their Evolution

It is interesting to trace the origin of the present most successful synthesis systems from their ancestors. Research at some major CAD centers in the seventies has matured into a few successful projects, that have been then the seed for following developments. As an example, work on high-level synthesis systems originated at Carnegie Mellon University [6] and then was perfected and engineered at ATT Bell Laboratories [6] and at the University of Southern Califorinia [11].

Silicon compilation started at Caltech, with the pioneering work of Johannsen on the **Bristle-Blocks** program [7]. This experience has been seminal and encouraged other projects, such as the **Genesil** synthesis system, which is a commercial tool marketed by Silicon Compiler Systems Inc.

The **MacPitts** hardware compiler, developed at MIT by Southard, was among the first to use algorithmic descriptions as a starting point for top-down synthesis. These ideas were then expanded in the **MetaSyn** project [6]. A similar approach to hardware compilation, based on algorithmic descriptions, was introduced by Jeffrey Fox in the SILC compiler at GTE. Later, some of the ideas implemented in SILC evolved into the **SilcSyn** synthesis and design system, marketed by Silc Technologies Inc.

Research on synthesis at the IBM T.J.Watson Research center culminated in the development of the **Yorktown Silicon Compiler**, a top down synthesis system from behavioral specifications to macro-cell based layout [6]. The heart of the Yorktown Silicon Compiler is a powerful logic synthesis package, called Yorktown Logic Editor (YLE), designed by R.Brayton. While the YLE design benefited from the large experience at IBM on logic synthesis, as in the case of the parallel development of the **Logic Synthesis System** (LSS) [3] and the interaction with the research groups at U.C.Berkeley [2] and U.C.Boulder[4] , the YLE still remains the archetype of modern multiple-level logic synthesis systems. The YLE model has been influential on the development of **Design Compiler**, a logic synthesis system marketed by Synopsys Inc., as well as on the logic synthesis tools developed later at U.C.Berkeley [2].

2.2 Present Commercial Synthesis Systems

2.2.1 Genesil and AutoLogic from Silicon Compiler Systems

Silicon Compiler Systems Inc. has been marketing synthesis systems for a few years. Among their products, Genesil and AutoLogic are described here because their scope fits into the frame of this review. While Genesil has been available since 1985, AutoLogic is a fairly new product. Genesil has been described extensively, for example in [6].

Genesil is a top-down silicon compiler and integrates system specification, simulation and synthesis of the masks. The system consists of a set of block compilers for RAMs, ROMs, PLAs, data-path and random logic. Functional blocks are specified hierarchically, by means of tabular inputs called forms. A functional simulator and a timing verifier help the designer in refining the system specification. Then the system generates automatically the layout.

AutoLogic is a tool for logic synthesis and optimization of ASIC designs. It manipulates netlists and therefore it can be inserted in any design flow where netlist are available. When used as stand alone tool, input netlist can be generated by a Register Transfer Level or by a Finite State Machine description. Netlists are first optimized by algorithms, that do peep-hole optimization and support hill-climbing procedures. Then library-based mapping is supported.

2.2.2 SilcSyn from SILC Technologies

SilcSyn is a set of synthesis and designs tools marketed by Silc Technologies Inc. The SylcSyn synthesis system performs high-level and logic level synthesis, starting from behavioral descriptions in the DDL proprietary language and soon from VHDL. It outputs netlists of components from a given library. The logic netlist format is compatible with the tools for physical design of several vendors and with the EDIF standard. The SilcSym simulator allows the user to verify the input and output descriptions. The synthesis system supports also design for testability, by generating a register transfer scan structure during synthesis. A test pattern generator provides also test vectors.

Hardware described in the DDL language may be in terms of several concurrent and interacting processes. Communication protocols may be specified and the physical implementation of the processes themselves can run at different clock frequencies. DDL descriptions are Lisp-based and consists of Data-Control-Machines, i.e. the combination of data-flow and control statements, and Combinational-Logic-Machines. Architectural features such as series/parallel execution of blocks of hardware is explicitly defined by the user in the description.

The high-level synthesis portion of SilcSyn supports resource sharing. The resulting structure can be examined by the user who may desire to modify it by changing the original description and running high-level synthesis again before the logic synthesis tools. It also supports performance driven synthesis of arithmetic units such as adders and subtractors and automatic state encoding. The logic synthesis tool, called Builder, performs technology independent multiple-level logic synthesis and optimization. Technology mapping into a cell library is driven by a rule-based system. The output of the Builder can be interfaced to CAE workstations for physical design.

2.2.3 The Design Compiler from Synopsys

Synopsys offers a robust set of tools for logic synthesis and technology mapping, called Design Compiler. A high-level compiler can be used to map logic descriptions in the Verilog simulator language into netlists that can be processed by by the logic synthesis tools. The Design Compiler evolved from Socrates, a former program for logic synthesis developed at Calma-G.E., that supported transformations among PLA-based representations, netlists and Boolean expression.

The overall structure of the Design Compiler is centered upon the use of hierarchical combinational and/or sequential circuits. The Design Compiler supports the optimal design of logic circuits under constraints, such as: clock waveforms; drive on inputs and load on outputs; maximum area and maximum/minimum propagation delay along any path; preserve locally the circuit structure, etc ... Technology mapping is done by using a rule-based approach. Rules are fired in a predefined order and controlled by an algorithm. Each rule transforms a subcircuit into an equivalent one, if a marginal gain is achieved in the objective function and if the corresponding constraints are satisfied.

2.2.4 The VLSI-Logic Tools from VTI

VLSI Technology Inc. (VTI) offers a set of tools for logic synthesis, which are used also inside the company for ASIC design. The tools comprise a capture system, a finite state machine synthesizer, a two-level logic optimizer, a multi-level logic extractor and a technology mapper. The entire design methodology is centered around the finite state machine model. Design is entered as state transition diagrams or using a textual description.

Logic synthesis is performed as follows. First, the logic is separated from the state information, which is always implemented by a minimal number of registers. Then state assignment is done by using a constrained embedding algorithm. At this point, two-level logic is optimized by a minimizer which is an internal version of the Espresso minimizer. Needless to say, flattening to two-level representation can be very limiting. Arithmetic functions and datapath cannot be efficiently represented. For example, n-bit parity functions require 2^{n-1} product-terms.

Multiple-level logic circuits are extracted from two-level representations by generating kernels and extracting sub-expressions using a priority scheme. The heuristics for chosing the kernels privilege three estimates: i) literal gain, i.e. reduction in estimated area complexity; ii) performance loss, i.e avoiding those kernels that would lead to timing constraint violations; iii) performance gain, i.e. privileging those kernels that would reduce the data ready at some outputs.

Technology mapping is achieved by transforming first the logic representation into NANDs. Then the network is traversed starting from the inputs, and groups or NANDs are collapsed to complex gates, if such a mapping is possible. This is done by checking on a set of rules in a predefined order, that estimate circuit improvement following application of each rule. High fanout gates are processed separately and buffering techniques are used.

2.3 Present Research Synthesis Systems

2.3.1 The System Architect's Workbench

Research at Carnegie Mellon University on high level system specifications opened the way to a set of tools for high-level synthesis, developed over more than one decade. These tools, are now collected under the name of System Architect's Workbench [10]. Their purpose is to explore architectural choices. Hardware systems are described in ISPS, that can be simulated and compiled into an intermediate data-flow format called Value Trace (VT). The Value Trace can be edited graphically, to perform operations such as partitioning and expansion of selected blocks. It can be annotated, to provide a link between the behavioral specification and the corresponding structural domain by program Coral.

Synthesis in the System Architect Workbench is in terms of hardware resources, i.e. predefined library macrocells, such as ALUs, adders and multipliers. No explicit link to logic synthesis tools has been reported. Therefore, the output of the workbench in a netlist of macrocells, whose internal details is not dealt with by the system. It is important to remark that the choices in interconnecting the resources (data-path allocation) and in defining an appropriate sequencing (control scheduling) are difficult problems, whose outcome affect substantially the performance and the area cost of the corresponding hardware.

The workbench consists of a set of tools. Aparty is an automatic partitioner, based on a cluster search. Cstep is responsible for deriving the hardware control portion: it is based on a list scheduling algorithm, under resource constraints. Emucs is a global data allocator, that binds resources based on the interconnection cost. Busser synthesizes the bus interconnection, by optimizing the hardware using a clique covering algorithm. Sugar is a dedicated tool for microprocessor synthesis. It recognizes some specific components of a processor (e.g. an instruction decode unit) and takes advantage of these structures in synthesis. All the tools are interfaced to each other, and they have been used successfully for a few years.

2.3.2 The Berkeley Synthesis Systems

Research on synthesis at U.C.Berkeley addressed two major areas: physical and logic synthesis. Several tools for physical synthesis have been developed and distributed by U.C.Berkeley. Most notably the Mosaico macro-cell design system [13], the Bear layout system [14], the Orca system [15] for sea-of-gates and the Timberwolf program for standard cell placement and routing. Most tools have access to a unified design database, called Oct [12]. A graphic browser, Vem, supports geometric and symbolic graphic editing.

The logic synthesis tools evolved from an extensive study of two-level logic circuits, culminating in the development of the Espresso logic minimizer [16] (in cooperation with IBM), and a set of tools for designing combinational and sequential circuits based on Programmable Logic Arrays [9]. Recently a powerful multiple-level minimizer, MisII, was developed [2]. It supports technology independent optimization of multiple-level logic representations in terms of area and delay and a powerful mapper that transforms a multiple-level network into an interconnection of cells from a library. The mapping can optimize the total cost in terms of area, as well as timing.

The logic synthesis tools are fully interfaced to the Oct database, and therefore physical designs can be derived from logic descriptions. In addition, functional descriptions in the BDS language, that can be simulated

by the Decsim simulator of Digital Equipment Corporation, can be mapped into logic circuits amenable for logic synthesis. While BDS provides a convenient way of specifying Boolean circuits, its level of abstraction is closer to the logic model than to the behavioral one. Therefore, no structural/architectural synthesis and trade-offs are possible, such as those supported by the System Architect's workbench.

2.3.3 The Cathedral Synthesis Systems

The Cathedral project was developed at IMEC, in connection with the Catholic University of Leuven in Belgium and other partners under the auspices of project Esprit of the European Community. Cathedral rejects the idea of the existence of a general purpose silicon compiler, in analogy with the present lack of software compilers for multiple source languages and back-ends. Therefore, Cathedral is designed to map behavioral descriptions of a particular class of designs, namely Digital Signal Processors (DSP), into a particular hardware model. Cathedral-I is a hardware compiler for bit-serial digital filters. Cathedral-II is a synthesis system for single-chip concurrent bit-parallel processors. Typical applications are speech synthesis and analysis, digital audio, modems, etc ... Cathedral-III targets hard-wired bit-sliced architectures, intended for the implementation of algorithms in real-time video, image and communication domain. Cathedral-IV is used for implementing very repetitive algorithms for video processing. Cathedral-I and II have been extensively described in the literature [6].

The general design methodology in Cathedral-II is called "meet in the middle" strategy. There are two sets of tasks in the system. The former is compiling behavioral descriptions into an interconnection of instances of primitive modules, such as arithmetic components. The latter is a set of parametrizable module generators for these modules, that construct the physical layout and that can be viewed as a set of procedures called by the high-level compiler. The basic components of the architecture are six execution units, which are prototypes of data-path, memories, I/O units and controllers.

Hardware description is done in the Silage language. Hardware compilation includes the following tasks: system partitioning into processes and protocols; data-path synthesis, i.e. mapping partitioned behavior into execution-units while minimizing the interconnection busses; control synthesis based on a microcode model. The data-path synthesis step is done with the aid of an architecture knowledge data base. Control synthesis is based on a heuristic scheduling algorithm. The physical layout is achieved by invoking the module generators. These modules can be seen as a library of high-level cells. They are designed to be portable across different technologies.

3 The Olympus Synthesis System

The Olympus Synthesis System is a research synthesis system for digital circuits, with emphasis on high-level and logic synthesis. Its intended use is for application-specific circuit designs to be implemented in a semi-custom methodology. The Olympus system is under development at Stanford University. However, some tools have reached enough maturity to support full chip design. The system has been applied to the standard benchmark circuits proposed for the High-level and Logic synthesis workshops, as well as to the design of three chips, that are described later.

The Olympus system can be used as a vertical system for chip design, from behavioral specifications to layout. However, intermediate formats are available for introducing and extracting circuit descriptions. For

example, logic descriptions can be entered into the systems so that is can be used for technology remapping purposes. Similarly, logic descriptions can be diverted to other synthesis systems for alternative implementation styles. As a result, the U.C.Berkeley physical design system [13] [12] can be used as back-end of the Olympus system.

Digital systems that are synthesized by the Olympus system are described at the behavioral level in a hardware description language called **HardwareC** [26], because of its similarity to the C programming language. Systems are modeled in HardwareC in terms of **concurrent and interacting processes**. A process describes a piece of hardware that implements cyclically a sequence of tasks. Processes define blocks of hardware that can execute concurrently. They represent a means of describing coarse-grain parallelism. There is much latitude in interpreting the hardware implementation of a process, which depends on the ultimate complexity of the hardware being built. For example, a multi-chip system can be described by associating a process to each chip. Alternatively, a single chip can implement multiple-processes.

HardwareC allows for two mechanisms for process communication: parameter passing and message passing. The former model assumes the existence of a shared medium (e.g. shared bus or shared memory) that interconnects the hardware implementation of the processes. The designer choses such a medium and provides the appropriate protocol for communication by describing it in the HardwareC program. The latter model uses a simple point-wise send/receive mechanism, that can be used for synchronization or for data transfer. The corresponding hardware for communication is automatically synthesized as well as the protocol.

Hardware descriptions can be made hierarchical, by using procedure calls. The hardware implementation of procedures that are called more than once, may be shared to reduce the area-cost of an implementation, when it is possible according to the system timing. Alternatively, the HardwareC description may be explicit in binding a procedure to a dedicated piece of hardware. This mechanism is useful to describe two pieces of hardware that may have identical representations but require separate implementations. Consider, for example, the case of describing counters that increment two different variables and whose execution overlap in time.

A hardware design may be fully specified by HardwareC primitives. In addition, HardwareC descriptions may include reference to predesigned library parts, if desired. For example, calls to specific adder circuits, Schmidt triggers, etc ... Then, synthesis techniques do not affect these parts and preserve them in the final implementation.

3.1 Hercules: High-level Synthesis

Hercules is a program for high-level synthesis of HardwareC descriptions. It generates (hierarchical) logic level circuit descriptions in terms of Boolean primitives. At present, only synchronous designs are synthesized. Hercules assumes that the implementation of each process is controlled by one clock. Therefore, the output of Hercules can be modeled in terms of concurrent synchronous finite-state machines.

Hercules supports three major tasks: behavioral synthesis, structural synthesis and mapping. Behavioral synthesis involves the parsing of HardwareC, syntax checking and semantic analysis and the resolution of multiple assignments to a variable by means of a mechanism called reference stack [17]. HardwareC descriptions are transformed into parse trees, that are the object of typical transformations of optimizing compilers, such as variable and constant propagation and dead code elimination. Eventually, the hardware description is transformed into an intermediate form, called SIF or **sequencing intermediate form**. Here, processes and

procedures are represented by hierarchical directed graphs, where vertices represent tasks and edges temporal dependencies due to dataflow dependencies, or sequencing dependencies provided in the hardware description or caused by hardware sharing. Such graphs are acyclic, because looping constructs are broken by the use of hierarchy. They are also polar, because each block of code has a single-entry and single-exit. Note that all operations that do not have explicit dependencies can execute in parallel.

The sequencing intermediate form is the basis for structural synthesis, that determines the optimal hardware sharing. Trade-offs between parallel and serial implementations are encompassed by the hardware sharing mechanism. Since the motivation of hardware sharing is related to reducing the area penalty, then area estimates of the resources are of utmost importance. In addition, hardware sharing may affect the performance of the system. Therefore the resource execution times must be known. Hercules does not assume the use of specific resources. A resource is a procedure, and therefore it is an application-specific piece of hardware. Such resources are circuits constructed by logic synthesis techniques. In order to evaluate the area/timing cost of each resource, Hercules generates first a structure that corresponds to a maximally parallel implementation. Circuits are then constructed and evaluated by logic synthesis techniques. The information about area/timing of each resource is then used to explore series/parallel trade-offs by using hardware sharing techniques. The result is a stepwise refinement of an initial structure.

Once an optimal structure with respect to hardware sharing is determined, structural mapping provides the appropriate assignment of hardware circuits to operations, their interconnection and the generation of the appropriate control circuits. Control synthesis in Hercules is complicated by the presence of data-dependent delays in the hierarchical sequencing model. A scheme for synchronization that uses an interconnection of atomic control finite-state machines, allows Hercules to handle hierarchical control structures (one per level of the SIF hierarchy) with no penalty in execution time. The structure that Hercules generates is a (hierarchical) netlist of a logic view of the circuit, in a format called **SLIF** or **structure/logic intermediate form.**

3.2 Minerva: Logic-level Synthesis

Minerva is the logic synthesizer in the Olympus system. Minerva can process the information generated by Hercules in the SLIF format. In general, since such format can support arbitrary calls to predefined parts (e.g. synchronizers, Schmidt triggers, etc.) Minerva extracts from SLIF the information that is specific to logic synthesis and that is called **synchronous logic network.** A synchronous logic network is a model for synchronous sequential multiple-level logic. It is described in terms of logic variables and logic expressions. A notation for synchronous delays allows us to include synchronous registers in the model.

Most logic synthesis systems deal with synchronous circuits by partitioning them into an interconnection of a combinational logic component and registers. The combinational portion of the circuit is optimized by combinational logic algorithms. Then registers are added back to the circuit. Needless to say, such optimization techniques are limited in their scope by this partitioning strategy. In contrast to other logic synthesis tools, Minerva does not partition a circuit into registers and combinational logic. It operates on the synchronous logic network model and it performs logic transformations while determining the registers position. Therefore, it may achieve results that are potentially better than those obtained by other logic synthesis tools that operate on combinational logic only and that are therefore limited by the partitioning step.

In particular, Minerva can optimize circuits with the following goals: i) minimize the circuit area (under cycle time constraints) or ii) minimize the cycle time (under area constraints). Minerva uses an extension of

the following transformations: elimination, resubstitution, extraction and decomposition that were defined for combinational logic synthesis [1] [2]. These transformations operate within and across the register boundaries. They incorporate **retiming** techniques [23] and extend the power of this technique to networks with variable topology: i.e. the optimal register position is determined under the assumption that the underlying network is being transformed, while preserving I/O equivalence.

Representations of synchronous logic networks may be manipulated by program **Janus**, that works as an interface among different tools. Janus can generated logic views in the appropriate format for the Thor simulator and for the Lsim simulator [21] in the M language. Janus can also provide an interface to the MisII combinational logic synthesis program [2] from U.C.Berkeley, and therefore an entry point to the corresponding physical design system [12]. In addition, Janus provides an interface to the NDL format used by the LSI Logic design system for sea-of-gates implementation as well as to the ADL format used by the Actel logic design system for electrically programmable gate array implementations [28].

3.3 Ceres: Semi-custom Technology Mapper

Ceres is a program for technology mapping. It addresses the problem of conforming an arbitrary synchronous logic network to a network that uses parts from a given library. Ceres tries to achieve an optimal technology mapping by choosing a coverage of the network in terms of library components that minimize the area or the cycle time. Since both these problems are computationally intractable, Ceres uses a heuristic strategy. Ceres differs from other technology mapping tools [5] [27] in two main respects. First, it uses algorithms, as in [27], as opposed to applying rules as in [5] and in most other commercial tools. Second, Ceres exploits Boolean operations to match subsets of a Boolean networks to library parts, as opposed to using tree matching techniques as in [27] [29].

Ceres maps first the registers to the corresponding library elements. Then, it processes the remaining combinational network from the registers outputs to their inputs and it isolates subnetworks whose vertices have unit fanout. These sub-networks are then processed one at a time and mapped. First they are cast into a canonical form, e.g. a decomposition into two-inputs NAND gates is performed. Such a decomposition yields a directed acyclic graph, with a sink node corresponding to a vertex in the original network with multiple fanout. Then, the sub-network graph is visited starting from the vertices at maximal distance from the source. At each vertex, for each element of the library, a Boolean tautology algorithm checks whether the library element can cover a portion of the sub-network having that vertex as a sink. If a match is found, the cost of that match in terms of area or timing is computed. The cost depends on the library element that matches as well as on the cost of the matching of the portions of the network that feed into the chosen element. Since the matching proceeds from the furthest vertices to the sink, a local optimal choice at each vertex yields an optimum implementation of the sub-network. However, the overall coverage of the network is not guaranteed to be optimal, because of the partitioning strategy into sub-networks with single-fanout vertices.

The algorithm for tautology-based matching exploits symmetries in the description of library cells to speed up the matching process. Note that while the Ceres matching algorithm can be potentially more expensive in terms of computation time, it matches efficiently library components with multiple literals, such as exclusive ORs and Majority functions. In contrast, tree-matching based algorithm do not exploit the use of these parts efficiently. Experimental results on standard benchmark circuits have shown that Ceres can achieve better than other algorithmic-based mappers [2].

3.4 Castor/Pollux: Macro-cell Generator

Castor and **Pollux** are twin programs that generate the layout of a macro-cell implementing a synchronous logic network [20]. They can be used as a back-end to Olympus, when a macro-cell design style is desired. Castor and Pollux generate the macro-cell layouts that resemble a standard-cell implementation. However, they do not require a cell library and therefore their use is alternative to that of Ceres. The twin programs generate a layout of a cell for each expression in the synchronous logic network. Cells are then stacked in rows and routed by channel routing techniques.

Program Castor addresses the problem of finding the optimal implementation of a CMOS gate (in terms of area) implementing a logic expression and represented by a Boolean factored form. Castor assumes a structured design methodology, with parallel polysilicon stripes forming the gates of the transistors, that should connect by abutment as much as possible, to reduce contact area and parasitic capacitance. Given a logic expression, Castor determines a symbolic matrix with the relative positions of the polysilicon and metal lines as well as the contacts [20]. It uses a two-stage algorithm. In the former stage, an optimal alignment of the polysilicon gates is sought. Then, a packing of the metal stripes is achieved. Program Pollux generates the layout of the macro-cell. It operates within the frame of the GDT environment [21]. It generates first a symbolic description of each master cell in the L language and an interconnection netlist. Then the cells are placed and routed.

3.5 Thor: Functional and Logic Simulation

Thor is an event-driven functional and logic simulator [24]. It was developed at Stanford on the basis of the work on the CSIM simulator done at U.Colorado at Boulder. Thor can simulate an interconnection of functional models in the C programming language. Such descriptions can be generated by program Janus from synchronous logic networks in the SLIF format. Unfortunately Thor cannot simulate directly HardwareC descriptions. However, a new HardwareC simulator is currently under development.

3.6 Circuit Design Using the Olympus system

The Olympus Synthesis System has been tested on the usual benchmark examples of the High-level and Logic synthesis workshops. It has also been used to design three application-specific digital circuits at Stanford University, namely a Bi-dimensional Discrete Cosine Transform (BDCT) chip [19], a Digital Audio Input Output chip [18] and a decoder chip for the Multi Anode Microchannel Array (MAMA) detector [25], to be used on the space telescope.

The BDCT chip may be used to reduce redundancy of video information in low bit-rate transmission channels and to perform video compression for image storage and retrieval. For example, video images (or portions) may be compared with previously stored ones. The difference between the two images may be transformed by the BDCT operator, coded and broadcast to the receiving station where the image is reconstructed. The BDCT chip architecture is defined by a set of equations [19] that can be solved row-by-row and column-by-column by two simpler Mono-dimensional transforms. Therefore the chip was modeled by two concurrent processes, that communicate by means of a shared memory array. A BDCT chip of 8-th order was described in HardwareC and synthesized in a macro-cell methodology. The chip image was about $9 \times 9mm^2$ in a 2μ CMOS technology.

The DAIO chip is an interface between a standard 16/32 microprocessor bus and audio devices, such as compact disk or digital audio tape player, that follow the AES protocol. The chip performs a serial to parallel data conversion. It has to demodulate the input serial data stream that is encoded by bi-phase marks. The input data stream comes in frames, with stereo-channel and other information. Preambles that denote beginning of the frames have to be detected and the proper information has to be formatted in 32-bit words. The DAIO chip has been described in HardwareC, compiled and mapped into a representation suitable for implementation in a LSI Logic 9K sea of gates. Such representation has been transferred then to the LSI Logic modular design environment. The chip counted about 6000 equivalent gates. An implementation as an Electrically Programmable Gate Array [28] is under progress at present.

The MAMA chip is designed to decode optical information received by a Multi Anode Microchannel Array), which is a multi-anode photo-multiplier. The photocathode does photon/electron conversion and grid anode array determines the position of an event. The position of a given event is decoded through coincidence discrimination, and the decoding algorithm exploits the geometry of the anode array. The MAMA detector chip implements the decoding algorithm. It was described in HardwareC and synthesized in a LSI Logic 9K sea of gates in a standard case for prototyping and a radiation-hardened one for space applications.

4 Conclusions

Hardware synthesis from behavioral description to layout is becoming a reality. Several research and commercial tools are nowadays available, even though some complicated problems still need to be solved. While physical and logic level synthesis are today standard practice for product-level design, high-level synthesis techniques will probably be used early in the nineties for this purposes. Nevertheless, some research and commercial tools are already available and being used on an experimental basis.

The Olympus Synthesis System, being developed at Stanford University, is a prototype of a vertically integrated synthesis system. It has been used to design three research chips, starting from behavioral specifications in the HardwareC language. It provides a workbench to experiment on algorithms for computer aided synthesis as well as a vehicle for fast turnaround design.

5 Acknowledgements

This research has been sponsored by NSF, Digital Equipment Corporation and ATT under a Presidential Young Investigator Award and by the Stanford Center of Integrated Systems under a seed grant.

References

[1] R.Brayton, G.De Micheli, A.Sangiovanni-Vincentelli and P.Antognetti, Editors, *Design Systems for VLSI Circuits: Logic Synthesis and Silicon Compilation*, Martinus Nijhoff, 1987.

[2] R. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, A. Wang "MIS: A Multiple-Level Logic Optimization System", *IEEE Transactions on CAD/ICAS*, Vol. CAD-6, No. 6, November 1987, pp. 1062-1081.

[3] J.Darringer, D.Brand, J.Gerbi, W.Joyner and L.Trevillyan, "LSS: A System for Production Logic Synthesis", *IBM Journal of Res. and Dev.*, Vol 28, No 5, pp. 537-545, Sep 1984.

[4] K.Bartlett, W.Cohen, A.De Geus and G.Hachtel, "Synthesis and Optimization of Multilevel Logic under Timing Constraints" *IEEE Transactions on CAD/ICAS*, Vol CAD-5 No. 4, pp.582-596, Oct. 1986.

[5] D. Gregory, K. Bartlett, A. deGeus, and G. Hachtel, "A System for Automatically Synthesizing and Optimizing Combinational Logic", *Proceedings of the 23rd ACM/IEEE Design Automation Conference* , pp. 79-85, June 1986.

[6] D.Gajski, *Silicon Compilation*, Addison Wesley, 1988

[7] D.Johannsen, "Bristle-Blocks, a Silicon Compiler" *16th Design Automation Conference*, June 1982, pp. 594-604.

[8] G. De Micheli, 'Performance-oriented synthesis in the Yorktown Silicon Compiler', *IEEE Trans on CAD/ICAS*, Vol CAD-6, NO 5, Sept 1987, pp.751-765.

[9] G.De Micheli, M.Hofmann, A.Newton and A. Sangiovanni-Vincentelli, "A Design System for PLA-based Digital Circuits" in A. Sangiovanni-Vincentelli editor, *Advances in Computer-Aided Engineering Design*, Jay Press, 1985.

[10] D.Thomas, E.Dirkes, R.Walker, J.Rajan, J.Nestor, R. Blackburn, "The System Architect's Workbench", *Proceedings of the 25th Design Automation Conference*, Las Vegas, NV, pp. 337-343, June 1988.

[11] J.Granacki, D.Knapp and A.Parker, "The ADAM Advanced Design Automation System", *Proceedings of the 24th Design Automation Conference*, Miami, Fl, pp. 35-41, June 1987.

[12] D.Harrison, S.David, P. Moore, R.Spickelmier and A.R.Newton, "Data Management and Graphics Editing in the Berkeley Design Environment", *Proceedings ICCAD*, Santa Clara, CA, Nov. 1986, pp. 20-24.

[13] J. Burns, A. Casotto, M. Igusa, F. Marron, F. Romeo, A. Sangiovanni-Vincentelli, C. Sechen, H. Shin, G. Srinath and H. Yaghutiel, "Mosaico: An integrated Macro-cell Layout System", *Proceedings VLSI 87*, Vancouver, Canada, Aug 1987.

[14] W-M Dai, H.Chen, R.Dutta, M.Jackson, E.Kuh, M.Marek-Sadowska, M.Sato, D.Wang and X-M Xiong, "BEAR: A New Building-Block Layout System" *Proceedings ICCAD* Santa Clara, CA, Nov 87, pp. 34-37.

[15] Mitsuru Igusa, Mark Beardslee and Alberto Sangiovanni-Vincentelli, "ORCA A Sea-of-gates Place and Route System", *Proceedings 26th DAC*, Las Vegas, June 1989.

[16] R. K. Brayton, G. D. Hachtel, C. T. McMullen, A. L. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*, Kluwer Academic Publishers, 193 p. 1984.

[17] G. De Micheli, D. Ku "HERCULES - A system for High- Level Synthesis", *Proceedings of 25th Design Automation Conference*, Anaheim, pp. 483-488, 1988.

[18] M.Ligthart, A.Bechtolsheim, G.De Micheli and A.El Gamal "Design of a Digital Audio Input Output chip", *Proceedings of the Custom Integrated Circuit Conference*, San Diego, 1989.

[19] V.Rampa and G.De Micheli, "Computer Aided Synthesis of a Discrete Cosine Transform Chip", *Proceedings of the International Symposium on Circuits and Systems*, Portland, 1989, pp. 220-225.

[20] F. Mailhot, G. De Micheli "Automatic Layout and optimization of static CMOS cells", *Proceedings of ICCD*, Rye, N.Y., pp. 180-185, 1988.

[21] Silicon Compiler System *GDT Database and Language Tools Reference*, version 3.2, May 1988.

[22] G. De Micheli, T.Klein "Algorithms for Sequential Logic Synthesis" *Proceedings of ISCAS*, Portland, May 1989, pp. 756-761.

[23] C.Leiserson, F.Rose and J.Saxe "Optimizing Synchronous Circuitry by Retiming", in R.Bryant, Editor *Third Caltech Conference on VLSI*, Computer Science Press, 1983.

[24] R. Alverson, T. Blank, K. Choi, S. Y. Hwang, A. Saltz, L. Soule, T. Rokicki "THOR User's Manual: tutorial and commands", *Technical Report*: CSL-TR-88-348, Computer System Laboratory, Stanford University, January 1988.

[25] David Kasle, "Decoding Techniques for fine-fine geometry Multi-Anode Microchannel Arrays" *Proceedings of SPIE - The International Society for Optical Engineering*, Vol. 932, pp.280-284, 1988.

[26] D. Ku and G. De Micheli, "HardwareC: A language for Hardware Design" *Technical Report CSL-TR-88-362*, Stanford university, August 1988.

[27] E. Detjens, G. Gannot, R. Rudell, A. Sangiovanni-Vincentelli and A. Wang, "Technology Mapping in MIS" , *Proceedings of the ICCAD*, pp.116-119, November 1987.

[28] A. El Gamal, J. Green, J. Reyneri, E. Rogoyski, K. A. El-Ayat and A. Mohsen, "An architecture for Electrically Configurable Gate Arrays", *IEEE Journal of Solid-State Circuits*, Vol. 24, No. 2, pp.394-398, April 1989.

[29] K. Kreutzer, "Technology Binding and Local Optimization by DAG Matching", *Proceedings of the 24th ACM/IEEE Design Automation Conference*, 1987.