

A Multicomponent Synthesis Environment for VHDL Specifications.¹

Ranga Vemuri, Nand Kumar, Ning Ren, and Ram Vemuri
Laboratory for Digital Design Environments
Department of ECE, Mail Location 30
University of Cincinnati, Cincinnati, OH 45221
e-mail: ranga.vemuri@uc.edu

Abstract: The Multicomponent Synthesis System (MSS) is a vertically integrated collection of design synthesis tools for top-down design of multichip modules (MCMs) from behavioral specifications. MSS consists of a high-level synthesis subsystem, a partitioning engine, a collection of structural silicon compilers, a test structure compiler, a test-bench compiler, a packaging compiler, and various component libraries and technology files.

A typical design flow through MSS begins with a behavioral specification in VHDL and a performance specification in terms of area and clock-speed of the digital system to be implemented. The specification is then processed by the various tools in MSS.

MSS is a VHDL-centered design environment. MSS blends synthesis and simulation tools operating at various levels of abstraction to quickly design correct application-specific MCMs. Functional test bench compilation is used as the primary means of achieving this goal.

This paper discusses the various synthesis and test algorithms used in MSS and their interaction. Through a small example we show how the designer can steer the design process to generate an application-specific MCM design.

1 Introduction

Application-specific multichip modules (MCMs) contain many ASICs. The design and development of application-specific MCMs is growing in importance, especially in defense and aerospace related applications. Low production quantities and the need to quickly produce working systems in these application areas do not justify the use of long and expensive development methodologies traditionally used for MCM development. An integrated CAD environment for application-specific MCMs should, therefore, offer the following facilities: (1) Automated synthesis; (2) Design Verification tools; (3) Testability support; and (4) Performance-driven design flow.

We have been developing an integrated design environment for multichip modules. The environment, called MSS (Multicomponent Synthesis System), contains several research tools we have developed over the the past three years along with several industrial strength tools. The MSS environment, shown in Figure 1, is centered around VHDL, and WAVES. It provides four levels of automated synthesis support all the way from the behavior level to MCM placement and routing, three levels of simulation support including behavioral, register and switch levels, and tools for automated test bench compilation and

¹An expanded version of this paper has been accepted to appear in IEEE Computer

design validation for all synthesized designs. This paper briefly discusses the various novel algorithms used in MSS for Multicomponent synthesis. We also illustrate the interaction between our research tools and some commercial tools through a small example.

2 Find Example

We will use a tutorial example *Find*, in this paper to illustrate the algorithms and results. *Find* performs bubble sort followed by binary search. Due to its small size *Find* is used in this paper as the running example. (Such small specifications, however, do not require MCMs. *Find* merely illustrates the features of MSS. We use two relatively large examples *Move Machine* and the *Viper Microprocessor*, to illustrate the results).

- *Find*: The *Find* has an array of eight 4-bit numbers. On power-up, *Find* sorts these numbers in ascending order. Then it accepts a 4-bit number, searches for the number in this array and, if found, returns the array index. Behavioral specification of the *Find* is shown in Figure 2. The interface to the *Find* entity is kept quite simple for the sake of clarity. Writing a behavioral specification is the first and most important step in generating an MCM design using MSS. Various tools in the MSS system can only interpret the VHDL constructs used in the behavioral specification, but have no way of determining whether the specification itself is ‘correct’ (represents the user’s intentions). Hence, the user must ensure that the specification does in fact reflect what she wants. In case of the *Find* example our user has decided to write the description this way, although others may very well have chosen to write it differently.

3 High Level Synthesis

High level synthesis is the process of generating a register level design from a behavioral specification. The register level design contains a data path and a finite state controller. Our high level synthesis system, called DSS (Distributed Synthesis System) is a collection of parallel algorithms [11]. The data paths generated by DSS contain register level modules selected from a parameterized module library. This module library contains VHDL descriptions of the register level modules and information about the performance of each module such as its area and delay time. Since the modules are parameterized in terms of their bit-widths, the area and delay profiles are stored as equations or tables.

DSS accepts a behavioral specification in a synthesizable subset of VHDL [10] and performance constraints in terms of the desired clock period and the upper limit on the area. DSS views the clock period as the maximum delay of the combinational blocks in any register transfer and the area constraint as the sum of the areas of the register level components used in the data path.

The behavior specification is first translated into a data flow graph representation called VIF (VHDL Intermediate Format). The VIF representation is organized as a collection of segments, one segment for each process, subprogram, wait statement and while-loop in the VHDL specification. The VIF representation is then loaded into DSS during the initialization phase. DSS consists of the following major tasks:

- *Scheduling and Performance Estimation*: The operations in the data flow graph are assigned relative time steps during scheduling. Also, the arithmetic operations are bound to physical ALUs available in the module library. In DSS, the scheduler simultaneously explores many alternative designs. To do so, it first generates all valid module sets from the module library. A valid module set is a collection of register level modules selected (with duplicates allowed) from

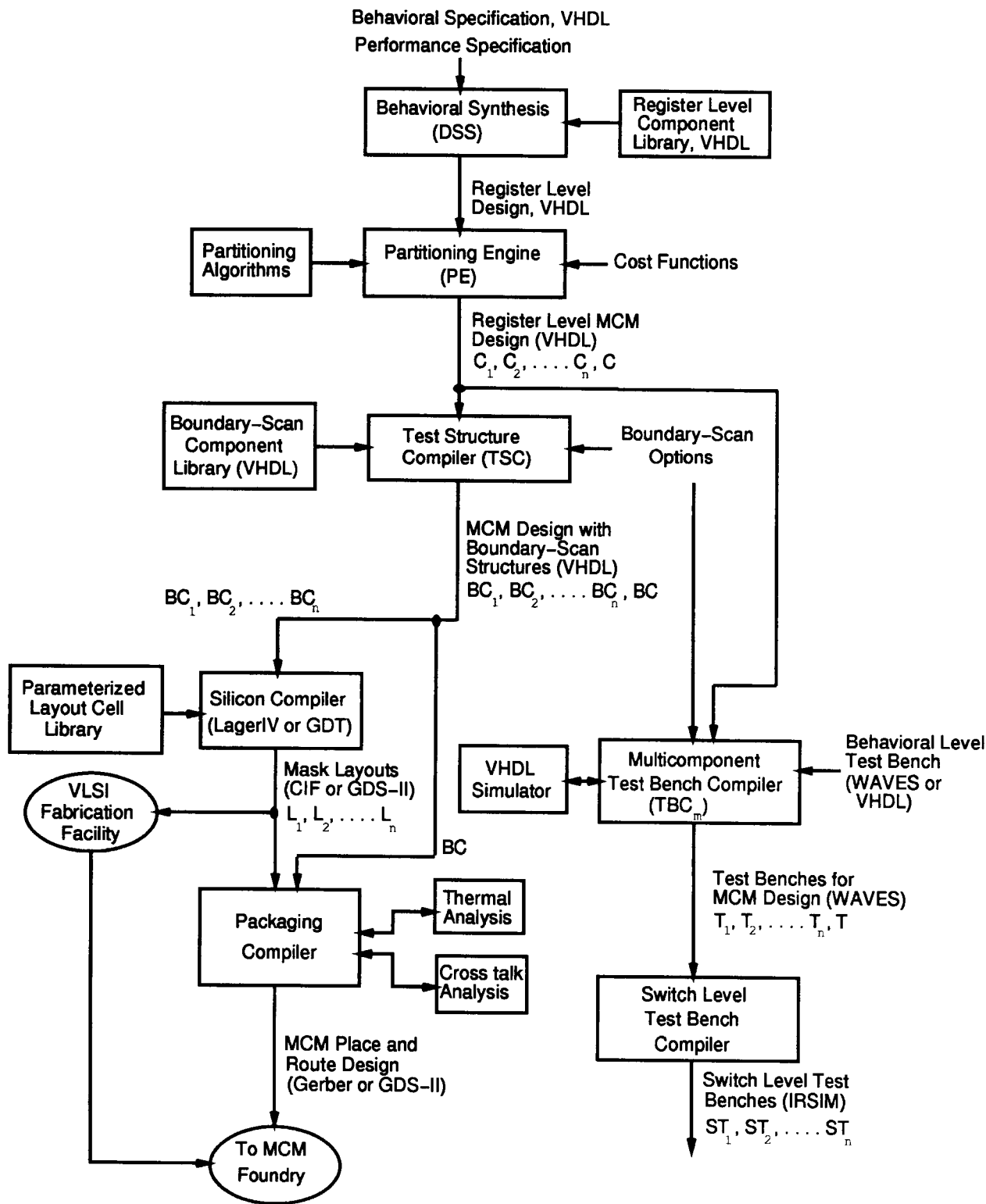


Figure 1: Multicomponent Synthesis System, MSS

```

entity find is
  port (x : in bit_vector(3 downto 0);
        index : out bit_vector(3 downto 0));
end find;

architecture find of find is
type int_array is array (0 to 7) of bit_vector(3 downto 0);
signal list : int_array := ("1000", "0111", "0110", "0101",
                            "0100", "0001", "0010", "0011");

begin
  sort : process
    variable i,j: integer := 0; -- %width 4
    variable low,high,mid,found : integer := 0; -- %width 4
    variable temp : bit_vector(3 downto 0);
    variable sorted : bit := '0';
    begin

      if (sorted = '0') then -- bubble sort
        i := 0;
        while (i < 8) loop
          j := i + 1;
          while (j < 8) loop
            if bits_to_int(list(j)) < bits_to_int(list(i)) then
              temp := list(j); list(j) <= list(i); list(i) <= temp;
              wait for 0 ns; -- for synchronization
            end if;
            j := j+1;
          end loop ;
          i := i + 1;
        end loop ;
        sorted := '1';
      end if;

      index <= "1111"; -- binary search
      low := 0;
      high := 8;

      found := 0;
      while (( low < high) and (found = 0)) loop
        mid := (low + high) / 2;
        if (x = list(mid)) then found := 1; end if;
        if (x > list(mid)) then low := mid + 1; end if;
        if (x < list(mid)) then high := mid; end if;
      end loop;

      if (found = 1) then index <= int_to_bits4(mid); end if;
      wait on x;

    end process;
end find;

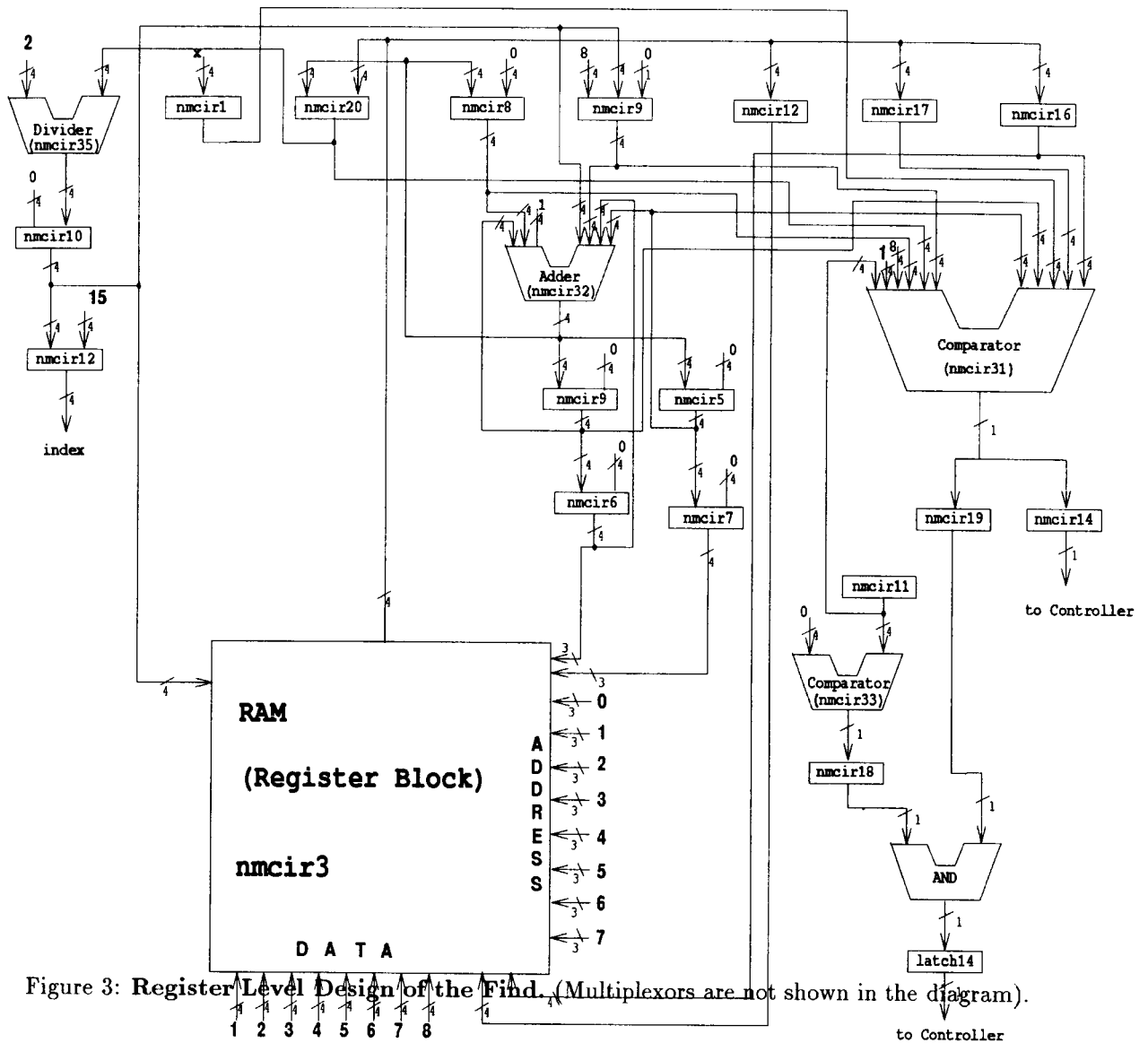
```

Figure 2: Behavioral Specification of the Find in VHDL

the module library such that (a) the selected modules are sufficient to implement all operations in the data flow graph, (b) no module in the module set has a delay greater than the specified clock period constraint, (c) together, the modules in the module set do not violate the area constraint specified by the user, and (d) the number of copies of any module does not exceed the maximum number needed to implement full parallelism available in the data flow graph. Then, for each valid module set, the data flow graph is scheduled using a variation of Paulin's force-directed scheduling algorithm [9]. The force-directed algorithm uses an efficient heuristic to produce the fastest possible schedule subject to the resource constraints imposed by the modules available in the valid module set. Following the scheduling step, a performance estimator is invoked to estimate the area and clock-speed of the design that would be generated from the scheduled data flow graph. Estimation of area considers the effects of introducing registers, multiplexers, routing, and the controller. Execution time estimate is based on the number of control steps used to schedule the data flow graph. The estimator generates a plot showing area vs. execution time tradeoff for various module sets. The user can select a tradeoff point or the DSS will select the fastest of the least area designs. Scheduling and tradeoff analysis in DSS is explained in detail elsewhere [3].

- *Register Optimization:* Each carrier (roughly speaking, a carrier is an edge connecting two operation nodes in the data flow graph) in the data flow graph represents the need for a register. Two carriers are said to be compatible if their life-spans do not overlap. The problem of register optimization is to find the best grouping of compatible carriers such that each group can be bound to a physical register. Within any group, life-spans of any two carriers should not overlap. Fewer groups lead to fewer registers in the data path. Like scheduling, register optimization is an NP-hard problem. Our register optimization algorithm is based on partitioning the data flow graph at the VIF segment boundaries. In the first step, intra-segment optimization is carried out based on extensive intra-segment life-span analysis. The best grouping of compatible carriers within each segment is determined. In the second step, inter-segment register optimization is done. During this step various carrier groups determined in the intra-segment optimization are further combined into larger groups. Intra-segment optimization does not require any further life-span analysis, but simply exploits the data-dependence relationships (calling sequences) among the segments as represented in the data flow graph. At the end of register optimization, a few groups of compatible carriers are left. Each group is then bound to a physical register selected from the module library. Details of register optimization in DSS are discussed in [11]. At the end of register optimization, the interconnect structure of the design is formed by inserting multiplexers at appropriate places.
- *Controller Generation:* Behavioral specification in VHDL is viewed as a collection of communicating processes. While facilitating abstract specification, this view complicates controller organization and generation. The controller is conceptually organized as a collection of communicating synchronous state machines, one for each VIF segment. A privileged finite-state machine called *root FSM* controls all segment state machines, called *leaf FSMs*. *Leaf FSMs* control the actual register transfers in the data path. The interaction among the *root FSM* and the *leaf FSMs* resembles what happens in the VHDL simulation cycle. During each cycle of operation, the *root FSM* updates all the signal registers and invokes all the *leaf FSMs*. When all the *leaf FSMs* reach the privileged *wait* states, a special signal, called *finish*, is generated. In simulation terms, the *finish* signal indicates the suspension of all processes and the absence of pending transactions. Then the *root FSM* begins a new cycle. We should point out that in spite of the conceptual organization as a collection of state machines, the design is actually a synchronous one and in fact assumes the standard 2-phase clocking scheme.

The entire DSS executes on several uniprocessor and multiprocessor platforms. For this paper, we executed DSS on a Sun SPARCstation 1. The register level design generated by DSS for the *Find* specification is shown in Figure 3.



4 Partitioning for Multichip Designs

The register level design generated by DSS may be too large to fit on one chip. In such cases, it should be partitioned into several chips subject to area and pin constraints. The *partitioning engine*, PE is used for this purpose. PE has a flexible framework within which the user has access to various partitioning algorithms, performance evaluators as well as an interactive interface to facilitate manual partitioning if desired.

The partitioning problem involves grouping the register level elements (register, ALUs, multiplexors etc.) into several groups. Each group of elements is bound to a VLSI chip and all the chips together are bound to a multichip module. Let the register level design R consist of register level elements $\{r_1, r_2, \dots, r_n\}$. These components should be grouped into disjoint groups $\{g_1, g_2, \dots, g_m\}$, $1 \leq m \leq n$, subject to the following constraints:

1. *Area Constraint:*

$$\forall g_i, \sum_{r \in g_i} \text{area}(r) \leq A$$

where $\text{area}(r)$ denotes the area of the register level module r , and A denotes the area any one chip.

2. *Pin Constraint:*

$$\forall g_i, \text{pin_count}(g_i) \leq P$$

where, P is the maximum number of I/O pins allowed for any chip.

We use cost functions to drive the partitioning algorithms. The *area* and *pin_count* of a chip and the total number of interconnection wires among the chips are used by the partitioning engine.

Partitioning algorithms usually attempt to minimize the total number of chips and/or the total number of interconnections among the chips. Partitioning is a combinatorial problem; even restricted versions are known to be NP-hard. Various heuristics representing tradeoffs between the time required to generate a partition and the quality of the partition generated have been proposed. No one algorithm can produce efficient designs in all situations. The partitioning engine, shown in Figure 4 has access to several different partitioning algorithms which the user can choose from. PE also has an interactive front-end to facilitate manual partitioning. In this paper, we briefly describe two such partitioning algorithms.

Hierarchical Clustering: In the hierarchical clustering technique we assume that none of the register level elements violates the area or pin constraints. A cluster represents a VLSI chip. A register level element which is included in a cluster is said to be *bound*; otherwise it is said to be *free*. Initially, a free element is selected and included in a new cluster. A free element is added to this cluster if such an addition does not violate the area and pin constraints on the chips. In case of multiple choices, the element which causes the maximum increment in the cluster area is selected. This process is repeated until no more elements can be added to the cluster. Further clusters are created until all elements are bound. This completes the first iteration of the clustering algorithm. In the next iteration, the clusters formed in the previous iteration are viewed as elements and further clustering is attempted. This iterative process continues until no further clustering occurs during an iteration. Hierarchical clustering is a deterministic greedy algorithm and usually gives sub-optimal partitions. But it is quite fast and memory efficient.

Genetic Partitioning: Randomized search algorithms which intermittently allow inferior designs in order to reach a final optimal design are known to be effective in solving combinatorial problems.

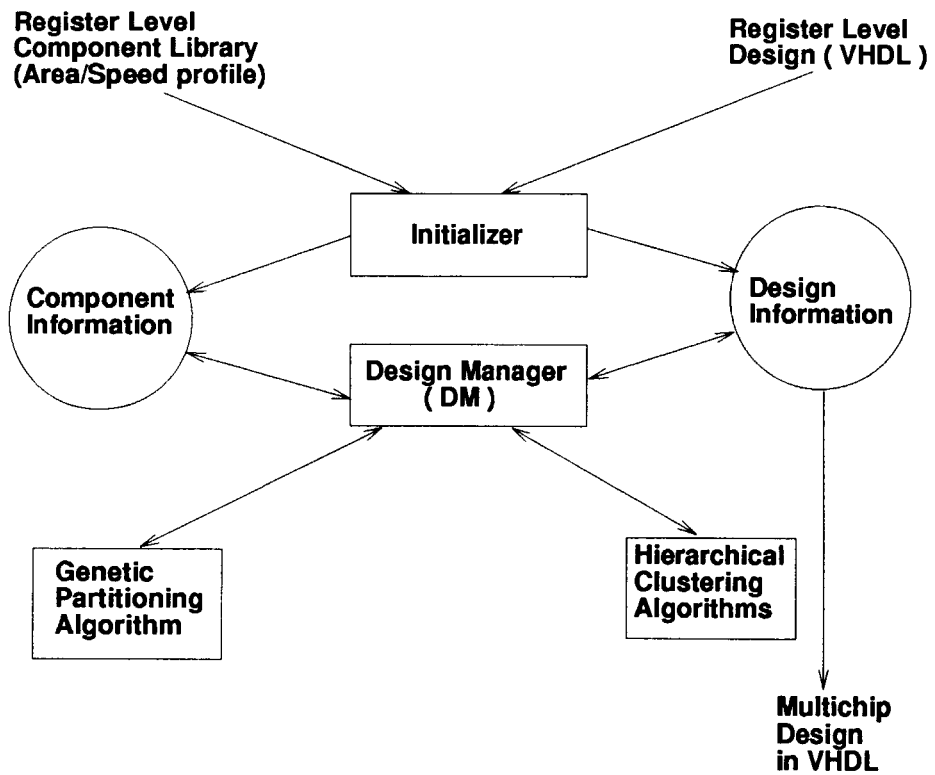


Figure 4: Partitioning Engine, PE

```

genetic partitioning algorithm
1  N: population size (number of partitions in a generation) := 100
2  F: percent of new generation produced by selection := 20
3  C: percent of new generation produced by crossover := 80
4  M: percentage of partitions mutated := 20
5  begin
6      create a random set of N partitions.
7      evaluate the fitness of each partition.
8      while (stopping criteria not satisfied) do
9          begin
10             create F percent of new population of partitions by selection.
11             create C percent of new population of partitions by crossover.
12             replace the current generation by new generation of partitions.
13             mutate M percent of the current partitions.
14             evaluate the fitness of each partition.
15             save the partition with the best fitness.
16         end
17     end

```

Figure 5: **Genetic Partitioning Algorithm**

Genetic algorithms [4] are one such class of algorithms. We have developed a genetic algorithm to solve our partitioning problem [12]. A genetic algorithm searches through generations of populations (solutions to the given problem) to arrive at a population that contains an individual solution which meets the given criteria. As the population evolves during the search process, the average fitness of the population improves.

Our genetic partitioning algorithm, shown in Figure 5, initially creates a number of random partitions. The *fitness* of a partition is defined as,

$$\frac{1}{1+PD+AD}$$

where PD is the net deviation from the pin constraint on each chip and AD is the net deviation from the area constraint on each chip. Fitness is value between 0.0 and 1.0 The algorithm uses the following operators in order to generate the next generation of partitions:

- *Selection*: A randomly selected partition which is highly fit (fitness value > 0.8) can be moved into the next generation. Twenty percent of the partitions in the new generation are created using selection.
- *Crossover*: Two highly fit partitions P_1, P_2 in the current population are randomly selected. The largest chip C (in terms of the total area of the register level components in it) in P_1 is copied into P_2 . Register level elements in C which are currently assigned to other chips in P_2 are deleted from those other chips. Both P_1 and P_2 are moved into the next generation of partitions. Eighty percent of the partitions in the new generation are created using crossover.
- *Mutation*: Mutation involves randomly selecting a partition and moving a register level element from some randomly selected chip in the partition to another randomly selected chip. Mutation is applied to twenty percent of the partitions in the new generation.

Genetic partitioning algorithm terminates when a termination criterion is satisfied. The criterion usually is that a partition with fitness 1.0 be found. In addition, to find optimal partitions (not just constraint satisfying partitions), other criteria such as the computation time, number of generation to be searched or a lower limit on a measure of global optimality such as the total number of interconnection wires or the total number of chips are also specified. The genetic partitioning algorithm is quite slow but can find better quality partitions, in terms of the number of chips and the number of interconnections among the chips, than the hierarchical clustering algorithm. Complete details of the genetic partitioning algorithm can be found in [12].

For our *Find* example, the design is partitioned, using hierarchical clustering, into 5 chips. Figure 6 shows the resulting design.

5 Test Structure Compiler

Complexity and limited probing accessibility in multichip modules necessitates incorporation of test structures in the design to improve their observability and controllability. IEEE standard boundary-scan architecture provides a means of gaining access to the chips buried inside the multichip module [5].

The *test structure compiler*, TSC, automatically incorporates the IEEE standard test access port (TAP), scan cells and other devices necessary to support boundary scanning. To each chip in the multichip design generated at the end of the partitioning phase, TSC adds a 16-state TAP controller, a 3-bit instruction register, an instruction decoder, a bypass register, scan cells for all the pins, and, optionally, a device identification register. The following instructions are supported by default: EXTEST, BYPASS, SAMPLE/PRELOAD, INTEST and IDCODE. Four pins, TCK, TMS, TDI, and TDO, are added to each chip. A fifth pin, TRST*, is optional.

TSC has access to a library of boundary-scan structures also written in VHDL. TSC has three major phases. First, relevant design and connectivity information is extracted from the VHDL files generated at the end of the partitioning phase. Then, during the boundary-scan synthesis phase, all necessary test structures are added in each chip and appropriate connections are added among the new test structures and the existing I/O pins of the chip. Finally, during the serial-scan path formation phase, the chips are interconnected such that the TDO of one chip goes to the TDI of another during the test mode. The test access port is added to the overall MCM design. Thus a serial scan path is established through the entire MCM and can be exercised in its test mode.

The resulting MCM design is output by TSC in the form of several VHDL files, one for each chip and one for the connectivity information among all the chips. Figure 7 shows a sketch of the *Find* design with the boundary-scan architecture included.

6 Silicon Compilation

Mask layouts for each chip in the MCM are generated using the LagerIV silicon compiler tools [6]. LagerIV is a collection of logic synthesis and physical design automation tools for VLSI chips. Specifications to LagerIV are written in two formats: SDL (Structure Description Language) is used to specify structural composition of existing library modules and standard cells and BDS is used to specify the behavior of finite state controllers. We wrote two translators, *v2sdl* and *v2bds* to generate appropriate SDL and BDS files from the register level VHDL description of the chip being synthesized. These files are then processed by the LagerIV tools to generate fabricatable mask layouts. We currently use 2.0 micron scalable CMOS technology; the design process however is completely technology-independent. The layouts can be generated in both the CIF and the GDS-II formats.

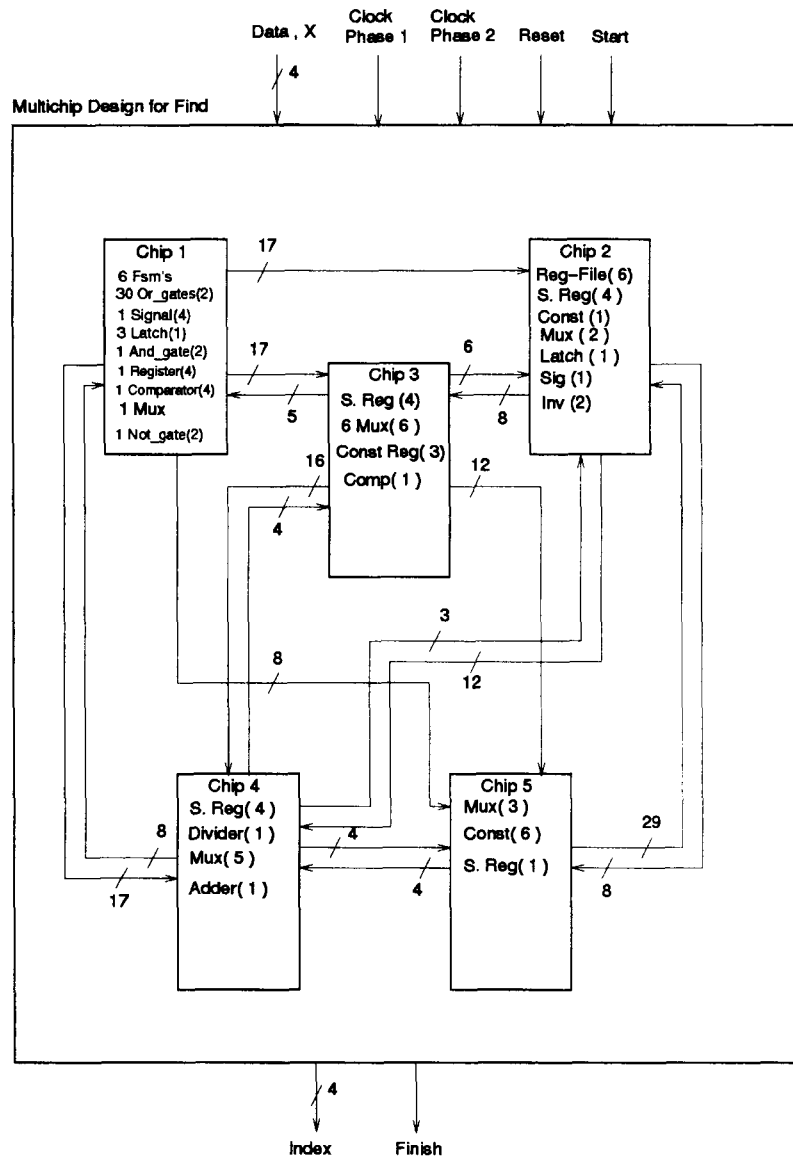


Figure 6: Multichip Design of the Find

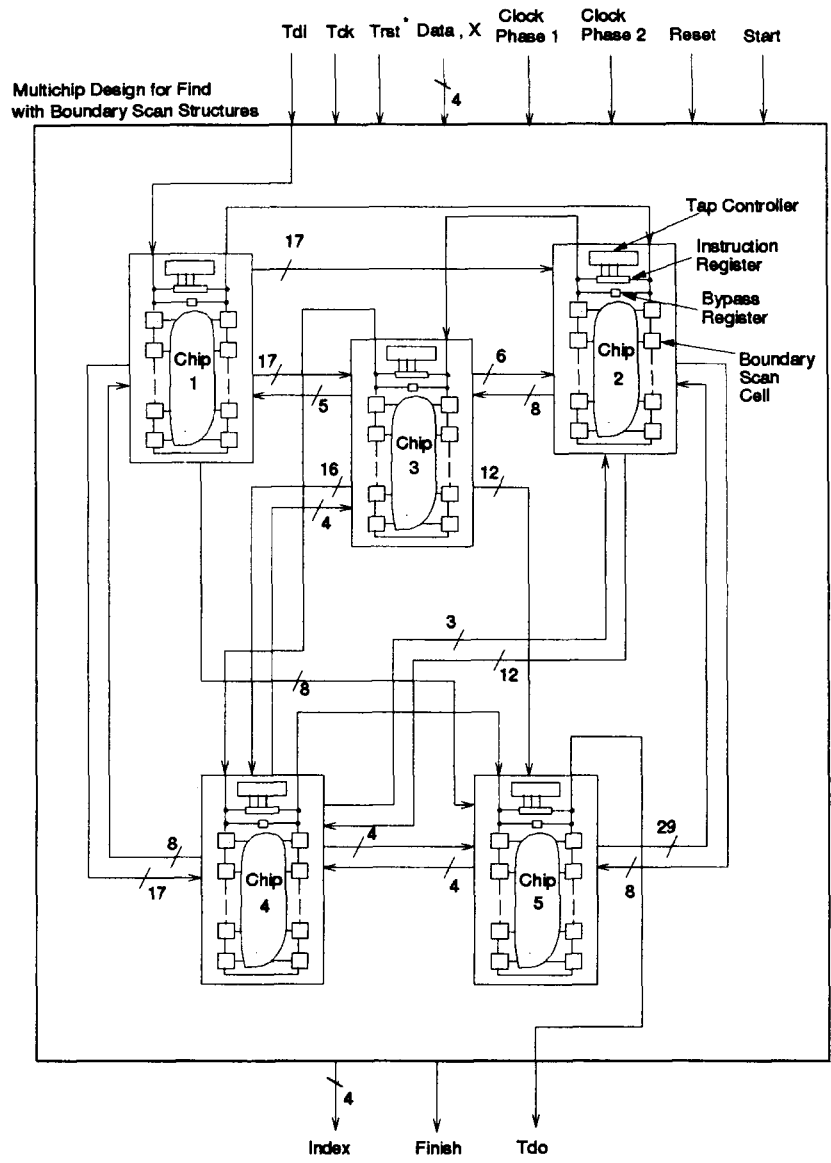


Figure 7: Multichip Design of the Find with Boundary-Scan Structures

7 Design Verification Methodology

Although the designs generated by various synthesis tools are expected to be correct ‘by construction’, there might remain software errors in the synthesis tools resulting in incorrect designs. To make sure that the designs generated meet the functional specifications and to avoid costly post-fabrication test and repair, extensive simulations are conducted at various stages in the design process.

Design verification within the MSS is done through simulations at various stages in the design process. Our approach is primarily based on functional testing [8] at each level of design abstraction, namely, the behavioral level, the register level, and the switch level. All the component generators in the various component libraries in MSS were extensively tested before adding them to the libraries.

To facilitate functional testing, we provide two test bench compilers. A *test bench* is an executable textual description of the test vectors and expected results. WAVES is an IEEE standard for writing test benches [15]. When simulating the behavioral specifications we expect the user to have used good functional testing practices so that fault coverage is possible. We use these test vectors to automatically generate functional tests at lower levels of abstraction. All that the user has to do is to provide behavior level test vectors in the form of a VHDL file or preferably a WAVES file. The multicomponent test bench compiler, TBC_m can be used to generate test benches for each component as well as for the entire MCM design [13]. These test benches are in the form of WAVES files. These files can be used in conjunction with any VHDL simulator for register level simulations of the MCM design.

WAVES test benches can also be used for post-fabrication testing of the MCM and its constituent chips (both before and after being embedded in the MCM package) provided the ATE (Automated Test Equipment) used accepts WAVES files. Many ATE manufacturers are developing software packages to facilitate inputs in WAVES format.

For post-layout simulations of the chips we use a switch level simulator called IRSIM. Circuit descriptions for IRSIM can be extracted from the CIF files generated by the silicon compiler. IRSIM uses a fairly accurate timing model based on the extracted capacitances to provide both functional and timing simulations. More detailed simulations (using a circuit simulator such as Spice) are prohibitively expensive for such large designs. Note however that all the component generators in the library used in conjunction with the silicon compiler have been simulated using Spice. The switch level test bench compiler, TBC_s , can be used to automatically generate IRSIM stimulus files from the WAVES external files produced by TBC_m .

Simulation results, both at the register level and at the switch level, can be automatically compared with the expected responses. Two programs, $COMPARE_m$ and $COMPARE_s$, are provided for this purpose.

8 MCM Physical Design

The final stage in the design process is the design of the MCM package itself. We currently use the Mentor Graphics’ MCM, hybrid, and PCB design tools for placement, routing and analysis of the package. We use the term MCM *packaging compiler* to refer to the MCM physical design tools collectively.

MCM packaging compiler takes as its input the top level VHDL description of the multichip design and the GDS-II files for the mask layouts of each chip. A program, *v2net*, is used to generate a net-list file from the VHDL file which has the connectivity information among the chips. Various geometry files which indicate the bounding-box information are created from GDS-II files.

The packaging compiler, shown in Figure 8 operates in two steps. During the placement step, all chips

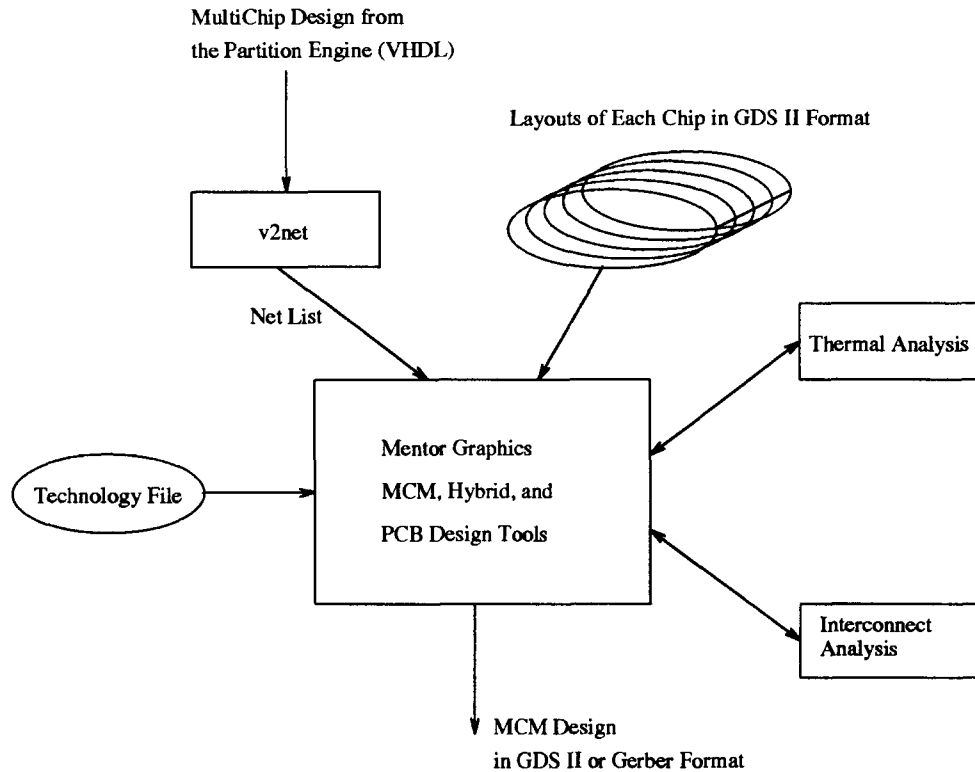


Figure 8: Packaging Compiler

are placed on the chip layer. During the routing step, the interconnect is generated to connect the chips as per the net list specification. A number of wiring layers are used for this purpose. Placement and routing steps attempt to minimize the total area, signal delays, and the interconnect wire lengths. Interactive placement may be attempted to generate tighter designs. Figure 9 shows a placed and routed MCM design for the *Find* example. The manufacturing artwork can be generated in either GDS-II or the Gerber formats.

Following successful placement and routing, thermal analysis and cross-talk analysis are performed. Dense MCM designs can generate considerable amount of heat. Placement and routing of MCMs should determine hot-spots and avoid the possibility of thermal failure. The thermal analysis tool simulates the three types of heat transfer, namely, conduction, convection and radiation. The results of thermal analysis can be examined and if the results indicate potential hot-spots, the thermal analysis information can be fed back into the place and route steps to arrive at a more thermally-sound MCM design.

9 Results

Our tools in the MSS environment have been under development for one to three years. We recently completed streamlining the entire design process from specification to fabricatable MCM designs. We have successfully fabricated and tested several VLSI chips, including the Move Machine, synthesized using the MSS environment. We are currently in the process of fabricating some MCMs generated using MSS.

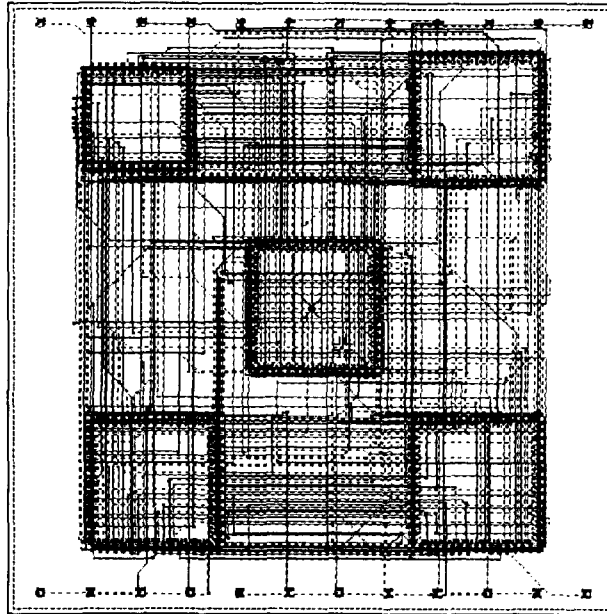


Figure 9: Layout of the Find MCM.

Table 1 shows some data pertaining to the three example MCM designs. The target technology for the chips is 2.0 micron scalable CMOS in each case. We had chosen a smaller feature size, say 1.0 micron CMOS, then none of the examples would have required partitioning. Note that none of the examples are pin-limited. In fact, the primary interface of each design has relatively few pins, 74 in case of Viper. But once we start partitioning the design, due to area limitation, the number of pins needed for the individual chips is quite high due to the large number of wires connecting register level elements in the design. The large number of pins in each individual chip contributes to the relatively large number of boundary scan devices introduced. Each pin needs a boundary-scan cell and each chip needs a TAP controller. The clock is relatively slow in each case, only about 10 MHz. This is again due to the 2.0 micron technology we are using and the way the library cells are designed; they are not designed for speed. All the arithmetic units are combinational blocks with ripple carry features. Our synthesis software itself is independent of the technology used; if we replace the library by a better one, we get better designs. The MCM technology was a hybrid technology with 4 signal layers and assumes surface mounting.

To give a realistic picture of the design times involved, we show, in Table 2, the approximate person-hours (wall-clock time) taken to complete various design tasks for the three examples, starting with behavioral specification development through MCM place and route. The user has been using the various tools in the MSS environment for a while but is not the developer of any of the tools. The times shown are for one design iteration. Subsequent iterations should take shorter times. As the various algorithms and tools in MSS are improved, the synthesis times are expected to be significantly shorter. Amount of simulation time will probably decrease at a much slower rate (assuming the same hardware platform); the simulators are already quite fast and efficient. Layout synthesis, both at the chip level and the package level, has been a key contributor to the design time. However, the layouts generated are reasonably compact. On the other hand, behavior synthesis and partitioning (at least the clustering algorithm) are fast, but the designs generated by these tools have room for improvements. We are currently investigating various algorithms for high-level synthesis as well as

		find	MM	viper
1	Lines of Behavior Specifications	62	80	450
2	Number of chips	5	3	5
3	Total number of pins of all chips	439	346	1530
4	No. of I/O pins (before boundary scan)	15	49	69
5	No. of I/O pins (after boundary scan)	20	54	74
6	No. of register level components			
	a. excluding boundary scan devices	90	49	158
	b. including boundary scan devices	559	422	1733
7	Area constraint	25 sq. mm	50 sq. mm	50 sq. mm
8	Pin constraint	120	150	450
9	Clock period constraint	100 ns.	100 ns.	100 ns.
10	Number of interconnections among the chips	235	215	640

Table 1: Design Data for the Three Examples

design partitioning to generate better quality multichip designs.

10 Acknowledgments

This research is sponsored in part by the Wright Laboratories of the USAF under contract number F33615-91-C-1811, the Defense Advanced Research Projects Agency under order no. 7056 monitored by the Federal Bureau of Investigation under contract no. J-FBI-89-094 and by an ACM/SIGDA Graduate Scholarship. The authors would also like to thank Darrell Barker, John Hines, Bob Reese, and Neal Stollon for their help and useful comments.

References

- [1] W. J. Cullyer, "Implementing Safety Critical Systems: The VIPER Microprocessor", pp. 1-26, in G. Birtwistle and P. A. Subrahmanyam (eds.), *VLSI Specification, Verification, and Synthesis*, Kluwer Academic Publishers, 1988.
- [2] W. Dai (ed.), *Proceedings of the Second Multichip Module Conference*, March 1992.
- [3] Rajiv Dutta, Jay Roy, and Ranga Vemuri, "Distributed Design Space Exploration for High Level Synthesis Systems", 29th Design Automation Conf, June 1992.
- [4] D. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA: Addison-Wesley, 1989.
- [5] *IEEE Standard Test Access Port and Boundary-Scan Architecture*, IEEE Std. 1149.1-1990.
- [6] Rajeev Jain et al., "An Integrated CAD System for Algorithm-Specific IC Design", *IEEE Transactions on Computer Aided design*, Vol 10, No. 4, April 1991.
- [7] R. Johnson et. al. (ed.), *Multichip Modules*, IEEE Press, 1991.
- [8] K.W. Lai and D.P. Siewiorek, "Functional Testing of Digital Systems," *Proc. 20th Design Automation Conference*, June 1983.
- [9] P. G. Paulin and J. P. Knight, "Force Directed Scheduling in automatic Data Path Synthesis," *Proc. 24th DAC*, July 1987, pp. 195-202.
- [10] Jay Roy and Ranga Vemuri, "On the Appropriate Usage of VHDL: The Synthesis Point of View," Tech. Memo TM-DDE-89-08, Dept. of ECE, Univ. of Cincinnati, Ohio 53 pp., Dec. 1989, (revised Apr. 1990, May 1990).

		find	MM	viper
1	Behavior specifications preparation & simulation	2 hrs.	8 hrs	56 hrs
2	No. of behavior level test vectors	8	20	116
3	Behavior synthesis	2 mins.	3 mins.	5 mins
4	Partitioning engine			
	a. Hierarchical clustering	2 mins.	2 mins.	5 mins.
	b. Genetic partitioning	6 hrs.	5 hrs.	10 hrs.
5	Test structure compilation	1 min.	1 min.	1 min.
6	Silicon compilation (all the chips)			
	a. without BSD	30 mins.	30 mins.	1 hr.
	b. with BSD	6 hrs.	5 hrs.	15 hrs.
7	Packaging compilation (place & route)	3 hrs.	2 hrs.	9 hrs.
8	Thermal analysis	2 mins.	5 mins.	10 mins.
9	Multicomponent test bench compilation (including simulation of the register level design)	30 mins.	30 mins.	1 hr.
10	Simulation of multichip design with boundary scan options at register level (each chip + overall) & comparison of results	45 mins.	45 mins.	2 hrs.
11	Switch level test bench compilation (all chips)	3 mins.	3 mins.	10 mins.
12	switch level simulation & comparison of results (all chips)	3 hrs.	4 hrs.	6 hrs.
	Total hours	22	26	101

Table 2: Design & Simulation Times for the Examples

- [11] Jay Roy, Nand Kumar, Rajiv Dutta, and Ranga Vemuri, "DSS: A Distributed High-Level Synthesis System", *IEEE Design & Test of Computers*, June 92, pp. 18-32.
- [12] Ram Vemuri and Ranga Vemuri, "A Genetic Algorithm for Multichip Partitioning", TM-ECE-DDE-92-26, June 1992.
- [13] Raghu Vutukuru, P. Subba Rao and Ranga Vemuri, "Boundary Scan Test Structures and Test-Bench Compilation in a Multichip Module Synthesis System", IEEE Multichip Modules Conference, March 1992.
- [14] Robert A. Walker and Raul Camposano, *A Survey of High-Level Synthesis Systems*, Kluwer Academic Publishers, Norwell, Mass., 1991.
- [15] *User's Guide to WAVES (Waveform and Vector Exchange Specification)*, Draft Document Version 4.4, December 10, 1990.