

# Synthesis of Testable RTL Designs using Adaptive Simulated Annealing Algorithm

C.P. Ravikumar	Sumit Gupta *	Akshay Jajoo
Department of Electrical Engg.	S3 India	Synopsys India
Indian Institute of Technology	Prestige Meridian	Sheriff Towers
Hauz Khas	M.G. Road	Richmond Road
New Delhi 110016	Bangalore 560001	Bangalore 560001

## Abstract

With several commercial tools becoming available, the high-level synthesis of application-specific integrated circuits is finding wide spread acceptance in VLSI industry today. Existing tools for synthesis focus on optimizing cost while meeting performance constraints or vice versa. Yet, verification and testing have emerged as major concerns of IC vendors since the repercussions of chips being recalled are far-reaching. In this paper, we concentrate on the synthesis of testable RTL designs using techniques from Artificial Intelligence. We present an adaptive version of the well known Simulated Annealing algorithm and describe its application to a combinatorial optimization problem arising in the high-level synthesis of digital systems. The conventional annealing algorithm was conceived with a single *perturb* operator which applies a small modification to the existing solution to derive a new solution. The Metropolis criterion is then used to accept or reject the new solution. In some of the complex optimization problems arising in VLSI design, a set of *perturb* functions become necessary, leading to the question of how to select a particular function for modifying the current system configuration. The adaptive algorithm described here uses the concept of reward and penalty from the theory of learning automata to “learn” to apply the appropriate perturb function. We have applied both the conventional simulated annealing algorithm and the adaptive simulated annealing algorithm to the problem of *testability-oriented datapath synthesis* for signal processing applications. Our experimental results indicate that the adaptive algorithm can yield better solutions in shorter time.

## Key Words

Test Synthesis, Adaptive Algorithms, Simulated Annealing, RTL Synthesis

---

\* This research was conducted when Sumit Gupta and Akshay Jajoo were students of IIT Delhi.

# 1 Introduction

Application specific integrated circuits are being increasingly used in real-time embedded applications related to signal processing and control. Automatic synthesis of application-specific systems has been adopted by the industry to reduce the time to market the product. Synthesis involves the solution of a number of difficult combinatorial optimization problems such as hardware-software partitioning, task scheduling, and resource allocation. These problems are often discrete constrained optimization problems. For example, there may be constraints on the performance of the system or the cost of the system. The objective function in the synthesis effort can be the time to market, the cost of the system, or the performance of the system. In this paper, we shall focus on the synthesis of testable ASICs assuming Built-in Self Test (BIST) as the testing strategy. The objective of the synthesis problem is to minimize the area and time overheads associated with testing. This problem has been addressed in the literature using several different approaches. Papachristou, Chiu and Harmanani [10] introduced the problem of synthesizing testable data paths and posed the problem as a combinatorial optimization. In particular, they introduced the idea of testable logic blocks (TLB) in a data path. A logic block such as an adder which computes a binary operation is testable if there exist registers  $R_A$  and  $R_B$  in the data path connected to the inputs of the block which can be configured as pseudorandom test pattern generators (PRPG). There must exist a register  $R_C$  connected to the output of the block which must be concurrently configurable as a multiple-input signature analyzer register (MISR).

A register  $R$  is said to be self-adjacent if it is possible to trace a purely combinational path from the output of  $R$  to the input of  $R$ ; self-adjacent registers pose difficulties during scheduling of test sequences. Thus, if the input to a logic block comes from a self-adjacent register  $R$ , then it becomes necessary to configure  $R$  as a PRPG *as well as* an MISR during test mode. While this is impossible to do using conventional BILBOs [5], the effect can be achieved at considerable overhead in terms of area using concurrent BILBOs [1]. Avra [2] has described a graph-coloring approach to the problem of designing testable data paths using conventional BILBO registers. Njinda, Lin, and Breuer presented a branch-and-bound heuristic for the synthesis of testable data paths with the objective of minimizing the total time to test the data path [7]. Alternately, their approach can minimize the total test area overhead which is the overhead resulting from transforming some of the registers in the data path into BILBO registers [5].

Simulated Annealing has been used as an optimization technique in a number of applications related to electronic design automation. For example, the popular TimberWolf placement and routing package is based on Simulated Annealing [14]. Annealing has also been applied to problems such as travelling salesperson [4], circuit partitioning [12], global routing [3], channel routing [3], scan path design [13]. There are several reasons for the popularity of Simulated Annealing, the principal reason being that it is easy to code the algorithm. In relation to greedy heuristics such as  $k$ -opt for the travelling salesperson problem or the Kernighan-Lin heuristic for the circuit partitioning problem [6], Simulated Annealing holds the promise of reaching the *global optimum* solution. Indeed, a number of authors have reported significantly better solutions for optimization problems using Simulated Annealing [4, 12, 3, 13]. In this paper, we intend to present an enhancement to the Simulated Annealing algorithm. The enhanced algorithm, which

we call *Adaptive Simulated Annealing*, is capable of “learning” [9] the best *move* which must be made at any stage in order to modify the current system configuration. We illustrate the power of adaptive simulated annealing for the testability oriented data path synthesis problem.

This paper has been organized as follows. In the next section, we describe adaptive simulated annealing. Section 3 provides a description of the testability properties associated with data paths and a formulation of the testability-oriented synthesis problem. Experimental results are reported in Section 4 and conclusions are presented in Section 5.

## 2 Adaptive Simulated Annealing

The conventional Annealing algorithm operates by starting with an initial solution to the combinatorial optimization problem and *improving* the solution through a series of *moves*. Unlike a greedy algorithm which only accepts system configurations resulting from better moves, annealing probabilistically accepts inferior moves.

Let  $S$  and  $S'$  denote the system configuration before and after the move. Let  $cost(S)$  indicate the value of the objective function applied to system  $S$ . Then  $\delta = cost(S') - cost(S)$  indicates the change in the level of the objective function. In a minimization problem,  $\delta < 0$  indicates a better move. While annealing accepts better moves unconditionally, it accepts inferior moves with probability  $e^{-\frac{\delta}{T}}$ , where  $T$  is the *temperature* parameter [4]. Annealing begins with a high value of temperature and decreases the temperature by a factor  $\alpha < 1$  at each stage until the temperature reaches a predefined lower limit. At each temperature, the algorithm makes a large number of moves until a convergence criterion is satisfied.

In multiobjective optimization problems, it is common to define an entire repertoire of moves, each of which is expected to improve one aspect of the system. As an example, consider the example of the testability-oriented (pipelined) data path synthesis problem. A data path can be viewed in an abstract sense as the tuple  $\langle L, S, A, B \rangle$ , where  $L$  is the latency cycle associated with the pipeline,  $S$  is the operation schedule,  $A$  is the resource allocation, and  $B$  is the binding information. The resource allocation  $A$  can be further decomposed into  $\langle A_f, A_m \rangle$ , where  $A_f$  represents the allocation of functional units and  $A_m$  is the allocation of memory elements. Similarly, binding  $B$  can be viewed as a tuple  $\langle B_f, B_m \rangle$ , where  $B_f$  is the binding of operations to functional units and  $B_m$  is the binding of variables to memory elements. Given a data path, a variety of moves can be conceived to modify the configuration of the data path.

Figure 1 illustrates the search space in the data path synthesis problem. At the top of the hierarchy is the set of possible latency cycles for pipelined implementation. For any one of these latency cycles, there are several possible schedules. For every schedule, several resource allocations are possible, and for each resource allocation, there exist several bindings. A data path may be viewed as a path from the root of the tree in Figure 1 to a leaf node, where the nodes on the path correspond to the selection of values for the tuple  $\langle L, S, A, B \rangle$ . The search space of data paths is very large even for reasonably large data flow graphs. A transformation to the data path shifts the location of the corresponding path in the search tree. We illustrate a number of such transformations taking the example data flow graph in Figure 2(a).

*The Delay transformation* changes the latency cycle corresponding to the pipelined data path.

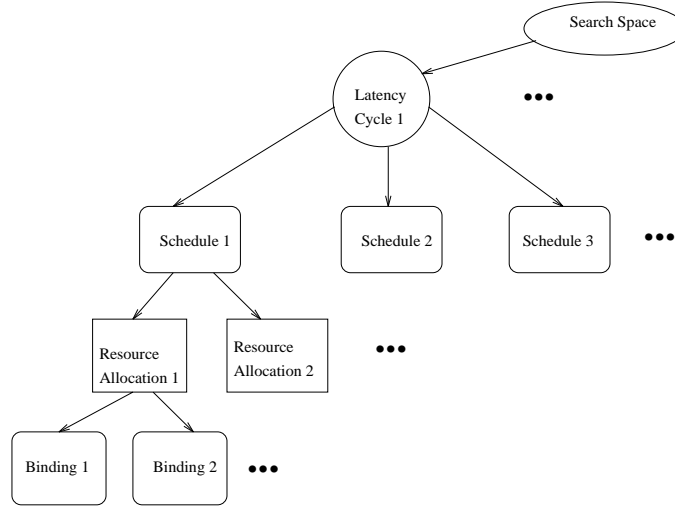


Figure 1: A Hierarchical View of the Search Space in the Data Path Synthesis Problem

For example, if the constant latency cycle (3) has been used, it can be changed to the constant latency cycle (4) through the insertion of a pure delay.

*The Reschedule Transformation* changes the schedule information by moving an operation  $O$  up or down by one control step. The reschedule transformation is applicable to an operation  $O$  if the control step at which  $O$  is scheduled in an ASAP schedule is different from the control step in an ALAP schedule [11]. For example, the operation  $C$  in Figure 2(a) can be rescheduled for the second  $c$ -step.

*The Type-1 Functional Binding Transformation* is applicable to an operation  $O$  if more than one operator of type  $O$  have been allocated. The transformation consists of mapping the operation  $O$  to a different operator of the same type. For instance, operation  $D$  in Figure 2(b) can be moved to the other adder in the data path.

*The Type-2 Functional Binding Transformation* is applicable to operations  $O_1$  and  $O_2$  of the same type which have been mapped to two different functional operators  $F_1$  and  $F_2$ . The transformation binds  $O_1$  to  $F_2$  and  $O_2$  to  $F_1$ . As an example, operations  $A$  and  $C$  in the data path of Figure 2 can be swapped so as to bind them to the other adder.

*The Type-1 Memory Binding Transformation* is applicable to a variable  $V$  which can be mapped to more than one register (memory element). Note that two variables can share the same memory element if and only if their life times do not overlap. The transformation consists of remapping variable  $V$  to a different memory element. As an example, the variable  $Y_2$  can be bound to register  $R_4$ .

*The Type-2 Memory Binding Transformation* is applicable to variables  $V_1$  and  $V_2$  which have been mapped to memory elements  $M_1$  and  $M_2$ . The transformation remaps  $V_1$  to  $M_2$  and  $V_2$  to  $M_1$ . A precondition for such a transformation to be valid is that  $V_1$  does not overlap with any other variable mapped to  $M_2$  and vice versa. As an example, the mapping of

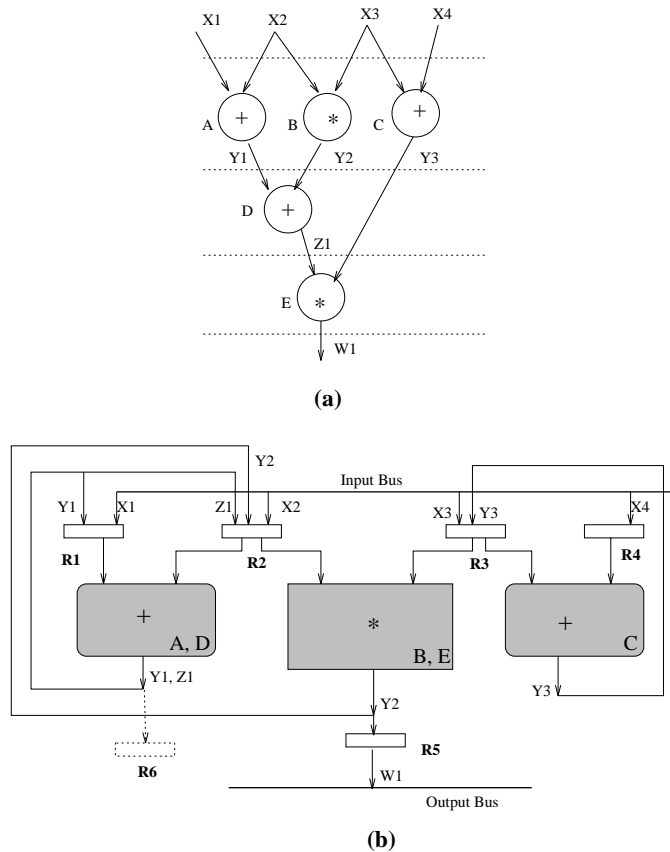


Figure 2: (a) A sample data flow graph (b) A data path corresponding to the sample data flow graph

variables  $Y_1$  and  $Y_3$  can be changed so that they are remapped to registers  $R_3$  and  $R_1$  respectively.

Each of the transformations listed above affects the properties of the data path such as area, performance, and testability. The transformations to the latency cycle and the schedule information represent macroscopic changes in the properties, whereas the rebinding transformations represent small changes. The testability properties such as presence or absence of self-adjacent registers are influenced significantly by the rebinding and the rescheduling transformations. We have observed through experimentation that the order in which the transformations are applied influences the quality of the resulting data path. Thus, in a Simulated Annealing algorithm, the selection of a transformation is a crucial decision which affects the properties of the resulting solution. We therefore developed an adaptive mechanism for the selection of the perturbation operator based on the theory of learning automata [9] to “learn” to apply the appropriate perturb function. Initially, each of the  $K$  operators has equal probability ( $1/K$ ) of being selected. Depending on the outcome of the perturbation, we update the probability of selection of the various operators according to the following rules. If the new solution resulting from the transformation  $i$  is an improved solution, we reward the operator  $i$  by increasing the probability of selection for operator  $i$  by  $\epsilon$ . At the same time, the selection probability for the remaining operators is

decreased by an amount of  $\frac{\epsilon}{K-1}$ . In a similar way, if a transformation  $i$  results in an inferior configuration, we penalize the operator  $i$  by reducing its selection probability by an amount  $\epsilon$  and increasing the selection probability of the remaining operators by an amount of  $\frac{\epsilon}{K-1}$ . The amount  $\epsilon$  is selected as follows.

$$\epsilon = \frac{\epsilon_0}{1 + e^{-\delta/T}} \quad (1)$$

where  $\epsilon_0$  is a constant smaller than 1 and close to 0. In our implementation, we selected  $\epsilon_0 = 0.02$ . Equation 1 ensures that the reward (or penalty) is proportional to the amount of improvement  $\delta$  in the quality of the solution.

### 3 Testability Oriented Synthesis

The synthesis tool reported in this paper takes as input the behavioural description of the algorithm and a module library, and generates the description of a self-testing data path. The BILBO-BIST test methodology is assumed. An initial latency cycle is selected from a set of predefined latency cycles. The force-directed scheduling algorithm [11] which aims at minimizing the hardware resources of the data path is used to generate an initial schedule. Multi-cycle operations are supported. Following the scheduling step, a graph coloring algorithm is used to allocate resources (both functional modules and registers). Multiplexers are used to share expensive resources such as multipliers and adders among several operations. The graph coloring algorithm also generates an initial binding of operations to modules and a binding of variables to registers.

Since additions and multiplications are the most frequently used operations in digital signal processing, we shall assume that the data flow graph mainly consists of these two binary operations. The concept of a 2-1 embedding has been used to achieve the testability of the data path. A 2-1 embedding of a functional unit  $M$  consists of a minimum of two registers  $R_A$  and  $R_B$  which are connected to the two inputs of the module  $M$  and at least one register  $R_C$  which is connected to an output of  $M$ . In the data path of Figure 2, the multiplier module is part of a 2-1 embedding consisting of registers  $R2$ ,  $R3$ , and  $R5$ . A module which is part of a 2-1 embedding is testable by configuring two of the registers connected to its inputs in the PRPG mode and configuring a output register in the MISR mode. In the example, we can test the multiplier by realizing registers  $R2$ ,  $R3$ , and  $R5$  as BILBO registers. In the test mode,  $R2$  and  $R3$  are configured as pseudorandom test pattern generators, whereas  $R5$  is configured as a signature register. We say a data path is testable if each functional module is part of at least one 2-1 embedding. Using this definition, the data path of Figure 2(b) is not testable since none of the two adders can be part of a 2-1 embedding. By adding a register  $R6$  (shown dotted in Figure 2(b)) at the output of one of the adders, we can make the adder testable. The other adder can be made testable by changing the binding of variables to registers such that  $Y1$  is mapped to  $R3$  and  $Y3$  is mapped to  $R1$ . If the modified embedding is used, the second adder will be the part of a 2-1 embedding which consists of input registers  $R3$  and  $R4$  and output register  $R2$ .

A test plan is generated for testing the modules in minimum number of test sessions using the branch and bound algorithm described by Njinda, Lin, and Breuer [7]. The testability of a data path is measured using the following metrics:

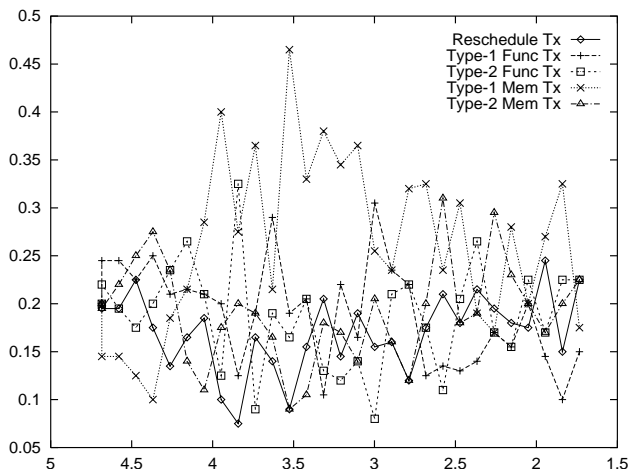


Figure 3: Variation of Selection Probabilities of Transformation operators for the Elliptic Wave Filter example

1. the number of registers to be configured as BILBOs in order to ensure that each module is part of a 2-1 embedding
2. total test application time.

## 4 Results

We have implemented the Adaptive Simulated Annealing algorithm for testability-oriented synthesis on a Sun SPARCstation 20. The implementation requires about 10,000 lines of C code including an X-windows based graphical user interface.

Figure 3 shows the variation of the selection probabilities for five transformations as a function of the temperature. The data flow graph used in this experiment corresponds to the Elliptic Wave Filter (EWF) benchmark [8]. Another similar plot is shown for the benchmark Auto Regressive (AR) Filter in Figure 4. We see that the variation of the selection probabilities depends on the specific problem being solved. This was confirmed by our experiments on several other benchmark problems such as BiQuad, DiffEq, and so on. In other words, there does not exist a procedure to predict the selection probabilities ahead of time. This is the primary motivation for the *learning automaton* [9] approach towards the computation of selection probabilities used in this paper. Some general trends can, however, be noticed in the variation of the selection probabilities. For instance, in all examples, we observed that the Type-1 memory binding transformation is a useful transformation throughout the course of the annealing algorithm.

We compared the relative performance of Simulated Annealing and Adaptive Simulated Annealing for several benchmarks including the BiQuad filter, AR filter, and Elliptic Wave filter. In our implementation of the conventional simulated annealing algorithm, one of the 6 possible transformations is selected at random; the probability of selecting any transformation  $i$  is  $\frac{1}{6}$ . We studied the effect of the initial solution during these experiments. In the first set of experiments, the initial schedule was generated using the force-directed scheduling algorithm [11]. In the sec-

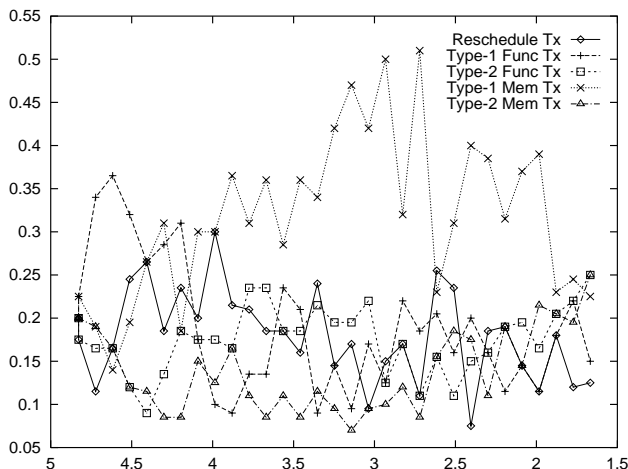


Figure 4: Variation of Selection Probabilities of Transformation operators for the Auto Regressive Filter example

ond set of experiments, we used the As Soon As Possible (ASAP) scheduling algorithm. The results for the two sets of experiments are shown in Tables 1 and 2. For each of the benchmarks, we show the result using the conventional simulated annealing algorithm followed by the result obtained using the adaptive algorithm. The *cost* in the table is a compound measure of the three objective functions, namely, the silicon area, performance, and testability. The area measure  $a$  refers to the area of the functional blocks such as adders and multipliers as well as the area of interconnect modules and registers. The pipeline latency  $\ell$  is used as the performance parameter. The testability factor  $\tau$  consists of the number of BILBOs in the data path and the test application time. The cost function is given by

$$Cost = \beta_1 \cdot (a \times \ell) + \beta_2 \cdot \tau \quad (2)$$

where  $\beta_1$  and  $\beta_2$  are constants. We indicate in the tables the number of functional modules, the number of registers, the number of multiplexors, the number of BILBO registers, and the test application time in the final solution. The execution time of the algorithm (in seconds on a Sun SPARCStation 20) is shown in the last column. As the tables indicate, the adaptive simulated annealing algorithm performs better than the conventional annealing algorithm both in terms of the quality of the resulting solution and the execution time. The advantage from the adaptive annealing algorithm is underlined by the fact that the modifications to the original annealing algorithm are simple.

## 5 Conclusions

We have presented an enhancement to the Simulated Annealing algorithm which improves the execution time of the algorithm as well as the quality of the resulting solution. The adaptive simulated annealing algorithm learns to select the perturbation operator which is used to modify the current solution. This algorithmic enhancement is useful when there are a large number of

Bench	Cost	Lat	# Mod	# Reg	# Mux	# Bilbo	Test Time	Exec Time
biquad	772	8	6	20	34	10	7	164
biquad	745	8	6	20	32	6	12	123
arfilt	984	8	8	28	44	12	12	1761
arfilt	945	8	7	28	42	10	12	1167
ewf	1638	14	5	29	52	7	9	579
ewf	1569	14	5	29	45	7	9	530

Table 1: Relative Performance of Simulated Annealing and Adaptive Simulated Annealing. Force Directed scheduling is used to generate the initial schedules

Bench	Cost	Lat	# Mod	# Reg	# Mux	# Bilbo	Test Time	Exec Time
biquad	761	8	6	20	35	6	7	160
biquad	737	8	6	20	29	8	14	133
arfilt	1115	8	8	28	61	11	12	2583
arfilt	1060	9	6	28	52	8	12	1475
ewf	1690	15	4	29	52	4	9	875
ewf	1608	14	5	29	49	4	9	698

Table 2: Relative Performance of Simulated Annealing and Adaptive Simulated Annealing with ASAP initial schedules

transformational operators which can be defined. A good application domain for the adaptive algorithm is the automatic synthesis of digital systems, where a variety of perturbation operators can be used to modify a design. These perturbation operators can be behavioral or structural. For instance, in the data path synthesis problem, behavioral transformations include changes to the data flow graph such as tree height reduction. Structural transformations include changing the pipeline latency, schedule, allocation, or binding. We have applied the adaptive simulated annealing algorithm to the problem of testability-oriented data path synthesis problem using BILBO-BIST as the test technology. Our results show that adaptive simulated annealing gives both speed and quality advantages in relation to the conventional annealing algorithm.

## References

- [1] M. Abramovici, M.A. Breuer, and A.D. Friedman. *Digital Systems Testing and Testable Design*. W.H. Freeman and Company, NY, 1990.
- [2] L. Avra. Allocation and Assignment in High-Level Synthesis for Self-Testable Data Paths. In *Proceedings of International Test Conference*, pages 463–472, 1991.
- [3] P. Banerjee, M. Jones, and J. Sarjent. Parallel Simulated Annealing algorithms for Cell Placement. *IEEE Transactions on Parallel and Distributed Systems*, 1(1):91–105, January 1990.
- [4] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671–680, May 1983.
- [5] B. Konemann, J. Moucha, and G. Zwierhoff. Built-in logic block observation technique. In *Proceedings of IEEE Test Conference*, pages 37–41, 1979.
- [6] S. Lin and B. Kernighan. Computer solutions of the travelling salesman problem. *The Bell System Technical Journal*, December 1965.
- [7] S.-P. Lin, C. Njinda, and M. Breuer. Generating a Family of Testable Designs using the BILBO Methodology. *Journal of Electronic Testing: Theory and Applications*, pages 71–89, 1993.
- [8] MCNC. Benchmarks for the fifth international workshop on high-level synthesis. Available via anonymous FTP at [mcnc.mcnc.org](http://mcnc.mcnc.org), 1991.
- [9] K.S. Narendra and M.A.L. Thathachar. *Principles of Learning Automata*. Printice Hall, 1989.
- [10] C. Papachristou, S. Chiu, and H. Harmanani. A Framework for High-Level Synthesis with Self- Testability. In *Proceedings of the International Conference on Computer-Aided Design*, 1991.
- [11] P. G. Paulin and J. P. Knight. Force-Directed Scheduling for the Behavioral Synthesis of ASIC's. *IEEE Transactions on CAD*, 8(6):661–678, June 1989.

- [12] C.P. Ravikumar. *Parallel Algorithms for VLSI Physical Design*. Ablex Publishing Corporation, 1996.
- [13] C.P. Ravikumar and H. Rasheed. Simulated annealing for target-oriented partial scan. In *IEEE International Conference on VLSI Design*, 1994. Calcutta, India.
- [14] C. Sechen and A. Sangiovanni-Vincentelli. Timberwolf 3.2 : A new standard cell placement and global routing package. In *Proceedings of IEEE/ACM Design Automation Conference*, pages 432–439, 1986.