

Synthesis of optimal 1-hot coded on-chip controllers for BIST hardware *

D. Mukherjee, C. Njinda and M. A. Breuer

Department of Electrical Engineering-Systems
University of Southern California
Los Angeles, CA 90089-0781

Abstract

We present a procedure for merging on-chip controllers for BIST circuitry to reduce hardware overhead. Instead of starting with one minimal state assignment and then performing state, input and output encoding, we pick the 1-hot code state assignment and implicitly search the space of minimum prime compatible state covers to obtain an optimal merged controller. This procedure uses knowledge of the greatest lower bounds on states, arcs, next-state and the output logic of the merged controller to prune the search space.

1 Introduction

Designers use DFT and BIST testable design methodologies (TDMs), such as LSSD and BILBO, to obtain testable chips. A *test schema* describes the execution characteristics or control strategy of a TDM. When a TDM is *embedded* in an actual circuit, the *test schema* is customized to the specific circuit and is then called a *test plan*[1]. A circuit can have more than one embedding, each giving rise to a *test plan*. A test plan is in essence a *control strategy* for an embedding. Embeddings can share data drivers, receivers and transfer paths. Thus the control of test plans leads to a complex design problem. To date there is no methodology that, given a set of embeddings, synthesizes a control entity with the objective of minimizing area overhead, functional delay, and at the same time meet constraints on test time.

1.1 Focus of this paper

Consider the figure shown in Fig. 1 that has three combinational blocks (kernels) *L1*, *L2* and *L3* that are to be tested using the BILBO TDM test methodology. *PG1*, *PG2* and *PG3* are pattern generators and *SA1* is a signature analyzer. When the test mode signal *TM* is 0, the circuit operates in *normal mode*, and is in the *test mode* when *TM* is 1.

In normal mode registers are controlled by the load/hold signal *ld/hld*. *PG*, *SA* and *shift* are signals that set the registers in pattern-generate, signature-collect or shift mode, respectively. In test mode the *shift* signal has precedence over the *PG* and *SA* signals. Test plans 1,2 and 3, shown in Fig. 2, correspond to three embeddings of the BILBO TDM. The test plans are represented as directed graphs, where the vertices represent *phases* and the edges represent *transitions* of the test plans. *Head* and *Tail* are pseudo-states that

*This work was supported by the Defense Advanced Research Projects Agency and monitored by the Federal Bureau of Investigation under Contract No. JFBI90092.

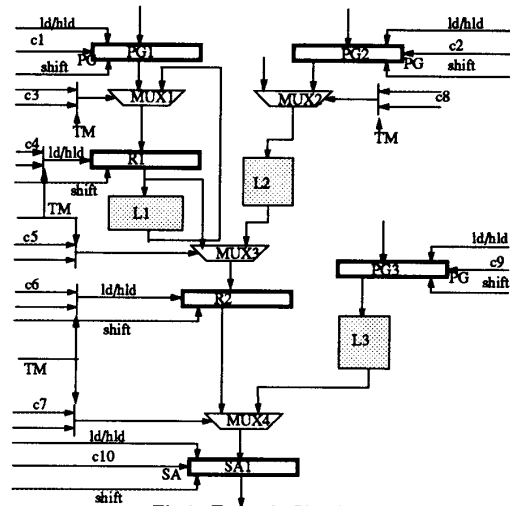


Fig 1 : Example Circuit 1

represent the processes of shift-in and shift-out of test data and results, respectively. Logic values on the control lines *c_i* are given alongside each vertex in the test plans. Unspecified control lines are don't cares. For example, *phase 1* of test plan 1 activates the pattern-generate mode of *PG1* while *SA1* holds its state. In *phase 2*, *MUX1* is configured to accept data from *PG1*, *R1* loads data and *PG1* and *SA1* hold their states.

For each kernel in a design there will be one or more test controllers that execute the test plans associated with the kernel. However, the test plans do not perform the set-up task for the test environment, e.g., initialize the BILBO registers or scan out the final signature. The test controller(s) for a kernel will thus not only have to execute the actual test plan(s) but also have to set up the test environment. Since set-up is a common feature of all test plans, a master controller can perform all housekeeping functions and hand over control to simpler controllers to execute different test plans.

In this paper we attempt to solve the problem of merging these simpler controllers or submachines to minimize the complexity of the control logic. We focus on the BILBO TDM and assume that test plans are executed serially, i.e. one after the other. Each test plan is assumed to be non-pipelined. The methodology is applicable to other TDMs (such as partial scan) and to test plans that operate in a pipeline manner.

2 The test environment and test controller model

We next define the environment in which the test controller operates.

There are n test plans to be executed. The *length* of a test plan i , denoted by $l(i)$, represents the number of phases or states in the test plans. Three counters are used in the test environment. A *test counter* (TCNTR), a *shift counter* (SCNTR) and a *test plan counter* (TPCNTR). Only TCNTR is on a scan path in the chip. Signal DEC-TC decrements TCNTR, while signals INC-SC and INC-TPC increment the other counters. RST-SC and RST-TPC reset SCNTR and TPCNTR. Signals TC, TPC and SC are set high when TCNTR, TPCNTR and SCNTR reach terminal counts of $0, n$, and *scan-chain-length* respectively, where *scan-chain-length* is the number of flip-flops on the scan chain. Test data is assumed to be stored off-chip.

A Moore machine model $M = (I, O, S, \delta, \lambda)$ of the overall test controller that sets up the test environment and executes each of the n test plans is shown in Fig. 3. I is the set of input lines, O the set of output lines, S the set of states, δ the state transition function and λ is the output function.

- $I = \{TC, SC, TPC, TM, tp_1, \dots, tp_n\}$ where tp_i is activated when test plan i is to be executed.
- $O = \{c_1, \dots, c_k, INC-SC, DEC-TC, INC-TPC, RST-SC, RST-TPC, SHIFT\}$ where c_1, \dots, c_k are the set of control lines controlling the circuit under test.
- $S = \{Idle_1, Idle_2, Head, Tail, S_a, S_b, S_{1(1)}, \dots, S_{1(l_1)}, S_{2(1)}, \dots, S_{2(l_2)}, \dots, S_{n(1)}, \dots, S_{n(l_n)}\}$
- $\delta: I_{val} \times S \rightarrow S \mid I_{val}$ set of all input values in the space spanned by I .
- $\lambda: S \rightarrow O_{val} \mid O_{val}$ is the set of all output values in the space spanned by O .

This machine can be decomposed into two parts. One

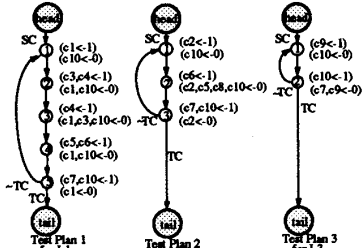


Fig 2 : Test plans 1, 2 and 3

part consisting of states $Idle_1, Idle_2, Head, Tail, S_a$ and S_b (Region B in Fig. 3), deals with the set-up process. The second part (Region A) deals with the actual execution of the test plans. We can therefore model the overall controller as interacting submachines M_{common} and $M_{tp_i}, i=1, \dots, n$, where M_{tp_i} executes test plan i . Our primary focus will be on merging the submachines M_{tp_i} into one FSM.

2.1 Implementation details of the submachines

M_{common} can be implemented using any classical technique. In addition to the *Head* and *Tail* states, M_{tp_i} has q_i states. These states will be implemented using the 1-hot encoded state assignment, (one of the many choices in state assignment) and thus q_i flip-flops

are required. Though the number of flip-flops in a 1-hot coded assignment is larger than that in an assignment using the minimum number of flip-flops, the 1-hot code offers at least four tangible benefits. (1) The flip-flops in the controller can be distributed throughout the circuitry to reduce control wire routing overhead. (2) The controller can be made acyclic so there is no sequential test generation problem. (3) Since the controller generates a walking 1 pattern in test mode, output logic, if any, is fully tested. (4) If the test controller is made part of the scan path, then the 1-hot coded controller provides an efficient means of executing scan tests to supplement BILBO test. From the examples considered to date, we have observed that the output logic for a 1-hot coded controller is trivial. Each control line is either driven directly by a flip-flop or at most by an OR(NOR) gate. We have thus restricted the output logic such that each output is driven at most by one OR/NOR gate. This restriction helps prune the search space and appears to be quite realistic. However, our procedure is flexible enough to accommodate either a 1-level synthesizer, a 2-level synthesizer (Espresso)[2], or a multilevel synthesizer (MIS)[3].

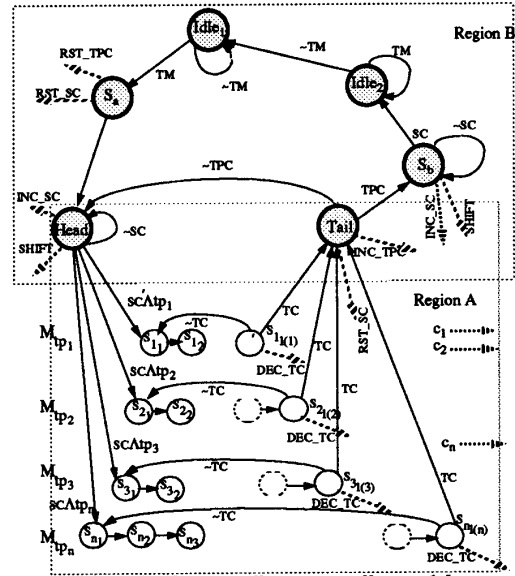


Fig 3 : The overall test controller model

For the 1-hot code the transition functions of the submachines M_{tp_i} are refined as follows:

- $\delta_{entry}: (SC, Head) \rightarrow S_{i1} \mid S_{i1}$ is the *start* state of M_{tp_i} ,
- $\delta_{exit}: (TC, S_{i(l(i))}) \rightarrow Tail \mid S_{i(l(i))}$ is the *end* state of M_{tp_i} ,
- $\delta_i: S_{ij} \rightarrow S_{i(j+1)}$ for $j=1, \dots, l(i) - 1$
- $\delta_{feedback}: (\sim TC, S_{i(l(i))}) \rightarrow S_{i1}$

3 Merging 1-hot encoded machines

We now formulate the *merging problem*. Given n submachines each controlling a test plan, obtain one machine M_{mrg} that has (1) *Head* and *Tail* as its entry and exit states, (2) controls each of the test plans in turn under control of inputs from the test plan counter,

and (3) has a 1-hot coded implementation with minimum next state and output logic.

Satisfying the first two conditions in the merging problem is trivial. We will thus focus on satisfying the third condition. Fig. 4 is a state transition table of a M_{mrg} for the example of Fig. 1, where the states of the three test plans (Fig. 2) have been concatenated together. The unspecified outputs appear as an x in the table. M_{mrg} is an incompletely specified machine and the number of states can probably be reduced. Using the classical pair chart technique the *maximum compatibles* (MCs) and the *prime compatibles* (PCs) can be found[4]. The PCs for this problem are the same as the MCs. In a classical state minimization technique

	tp ₁	tp ₂	tp ₃	c ₁	c ₂	c ₃	c ₄	c ₅	c ₆	c ₇	c ₈	c ₉	c ₁₀
1	2			1	x	x	x	x	x	x	x	x	0
2	3			0	x	1	1	x	x	x	x	x	0
3	4			0	x	0	1	x	x	x	x	x	0
4	5			0	x	x	x	1	1	x	x	x	0
5	1			0	x	x	x	x	x	1	x	x	1
6	7			x	1	x	x	x	x	x	x	x	0
7	8			x	0	x	x	0	1	x	0	x	0
8	6			x	0	x	x	x	x	1	x	x	1
9		10		x	x	x	x	x	x	x	x	1	0
10		9		x	x	x	x	x	0	x	0	0	1

Fig 4: The state transition table of M_{mrg}

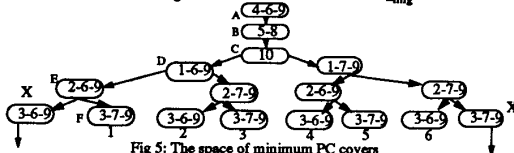


Fig 5: The space of minimum PC covers

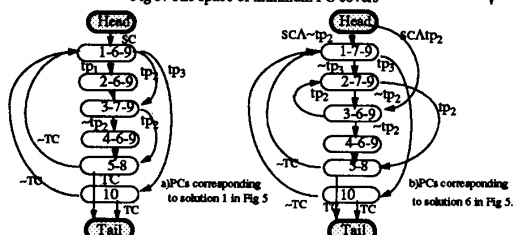


Fig 6: Transition graphs of M_{mrg} for two different choices of PCs

the next step would be to find *one* minimal PC cover for the set of states in the state transition table. Then state, input and output encodings are determined so as to minimize the resulting logic implementation[2, 5]. Herein lies the major difference between classical techniques and ours. *We have already decided to use the 1-hot code.* To find the simplest hardware realization, it is necessary to consider *all* the minimal PC covers for this machine. In fact, the actual search space in most cases is larger than the space of all minimal PC covers. This is because a state may appear in more than one PC in a minimum PC cover.

To illustrate how the choice of a minimal cover affects the implementation of our merged 1-hot coded controller, consider the six possible combinations of PCs shown in Fig. 5, all of which give a minimal cover. For example, PCs A,B,C,D,E and F correspond to the first minimal cover. Figs. 6(a) and (b) are the state transition graphs of the merged controller corresponding to two different minimal covers. Hardware implementation of the two graphs shows that the im-

plementation of the transition graph of Fig. 6(a) has less logic than that for Fig. 6(b). We will show in the next section that there is a direct relationship between the number of arcs in M_{mrg} and the amount of next state logic.

3.1 Greatest lower bound of the number of states, arcs, next state logic and output logic of M_{mrg}

M_{tp_i} has q_i states for $i=1, \dots, n$. Two cases need to be considered.

- Each of the q_i are distinct and can be ordered in a strictly decreasing manner, i.e. $q_1 > q_2 > q_3 > \dots > q_n$
- The q_i are not distinct and the states can be ordered in a non-increasing manner, e.g. $q_1 = q_2 = \dots = q_{t-1} > q_t > q_{t+1} = q_{t+2} = \dots = q_{s-1} > q_s > \dots > q_{s-1+1} = q_{s-1+2} = \dots = q_t > \dots > q_t$

The M_{tp_i} , with equal number of states can be grouped together. In case 2, there are s such groups. For case 1, $s=n$. Refer to [6] for proofs.

Proposition 1: For either of the above two cases, the greatest lower bound or *inf* of the number of states in M_{mrg} is $\max(q_i)$ which, in terms of the ordered sequences above, is q_1 .

Theorem 1: The *inf* of the number of arcs in the state transition graph of M_{mrg} is q_1+s+1 , where s is the number of state groups.

Corollary 1: Merging starting states (S_{i_1}) to form S_{mrg_1} and merging ending states $S_{i_{q_1}}$ to form $S_{mrg_{q_1}}$ for $i=1, \dots, n$ is a *necessary* condition for achieving the *inf* of arcs in M_{mrg} .

We can think of a transition arc between two states as being partitioned into two parts, the outgoing arc (from a state) and the incoming arc (to a state). The existence of a single outgoing transition arc from a state implies that no logic is needed for the implementation of that outgoing arc. However when a state has more than one outgoing transition arc, then each transition is a function of some subset of $\{tp_1, \dots, tp_n\}$, and the *greatest lower bound* of logic needed to "implement" each outgoing arc is a product term of 2 literals. We will assign a cost of 1 to a product of 2 literals. A product of k literals has cost $k-1$. To implement two incoming transition arcs to a state we need the sum of 2 literals. We assign a cost of $k-1$ to a sum of k literals.

Proposition 2: Let M_{mrg} and M'_{mrg} be two merged machines corresponding to two different choices of minimum PC covers. If the number of arcs of M'_{mrg} is greater than the number of arcs of M_{mrg} , then the cost of implementing the next state logic of M_{mrg} is less than the cost of implementing the next state logic of M'_{mrg} .

Theorem 2: The following two conditions are *sufficient* for achieving the minimum cost next state logic in M_{mrg} : (1) the number of arcs in M_{mrg} equals the *inf* of arcs, and (2) all the outgoing transition arcs that distinguish various test groups, s , emerge from the same state S_{mrg_j} for some $i=1, \dots, q_1-2$.

Corollary 2: Let k_1, k_2, \dots, k_s be the number of test plans in the s groups, when these groups are ordered such that $k_1 \geq k_2 \geq \dots \geq k_s$. Let r be the number of groups which have only one test plan each. Then the minimum cost of the next state logic, denoted by C_{nsl} is given by $C_{nsl} = \min\{n, \sum_{i=2}^s k_i + 2(s-1) - r\} + s + 3$.

Proposition 3: Let L_{ol_i} be the minimum output logic for 1-hot coded submachines M_{tp_i} $i=1, \dots, n$. Then the greatest lower bound on the output logic for M_{mrg} is $L_{ol_{mrg}} = \max \{L_{ol_i} \mid i=1, \dots, n\}$.

4 Procedure SOHOT

We have developed an implicit enumeration procedure SOHOT (Synthesis of Optimal 1-HOT controllers) that incrementally builds up the transition graph of the merged machine, M_{mrg} , which has a minimum number of states and a minimum amount of input and output logic. The basic data structure of SOHOT is a n -ary tree called *Search.tree*. Each node N_i in *Search.tree* has some properties attached to it. One of these properties is a directed graph DG . $DG=(V, E)$ represents the state transition graph (partial or complete) of a M_{mrg} . *Freq.list* is a list of states in M_{tp_i} , $i=1, \dots, n$, sorted in nondecreasing order of their occurrences in the PCs. The basic idea in SOHOT is to pick the states in *Freq.list* which occur more than once in the PCs and assign them to different PCs. The y states that occur only once in the PCs are all picked at the same time and appear as the vertices of the DG associated with the root node N_1 of *Search.tree*. All edges that will connect these y states in M_{mrg} constitute the edge set of this DG. The assignment of a state to a PC creates a new node in *Search.tree*. Each of these new nodes inherits the DG of its parent. A new vertex is added to each of these DGs to reflect the addition of the new state. Information from each of these DGs is used to compute various metrics that establish a lower bound on the number of states (*lbs*), next state logic (*lba*) and the output logic (*lbo*) of a M_{mrg} whose partial transition graph is the corresponding DG. The nodes in *Search.tree* are kept in a *Node.list*. The search strategy is *best first* with a node having the least value of *lbs/lba/lbo* expanded first. The precedence relationship is $lbs > lba > lbo$. Global variables are assigned values for the states, arcs, total logic, output logic and shared logic respectively for the best complete DG found at a leaf node *Search.tree*. Metrics for a node to be expanded are compared against these global variables and depending on the outcome of the comparisons, the node is either pruned or expanded. The procedure stops either when there are no more nodes to be expanded, or at any leaf node there is a DG whose next state and output logic equals the *inf* of the next state and output logic respectively. The latter is a very powerful feature of SOHOT because it efficiently prunes the search space.

5 Results

Some results using SOHOT are presented in Table 1. Ckt 1 is the example shown in Fig 1. The entry *Total # states* in the table corresponds to $\sum_{i=1}^n q_i$, where n is the number of test plans for a particular circuit. The *inf* of arcs, next state logic (*nsf*), output logic (*of*) and states are also tabulated. It is important to note that the search space is not simply the space of all minimal PC covers. A state may occur in more than one PC in the minimal cover and an optimal assignment of this state to a PC increases the search space. For example, Ckt 1 has 6 PC covers, but the entire search space has 18 leaf nodes. As seen from the table, M_{mrg} for Ckt 1 has 11 arcs, 6 states, and 10 2-input gates, and the procedure only examined 4 leaf nodes out of a possible 18. Ckt 2 is interesting because the next

state logic and output logic of M_{mrg} at one of the leaf nodes is equal to the *inf* of the next state logic and output logic, respectively, and the procedure terminated without searching any further. In this case, only 2 out of a possible 18 nodes were examined. For Ckt 3, the procedure examined only 2 leaf nodes out of 10. Ckt 4 has only 1 answer. Except for an OR gate driving a control line in Ckt 1 and Ckt 3, all other control lines of the circuits are driven directly from the flip-flops of their merged machines.

Ckt Name	Total # states	Inf Arcs	Inf nsf	Inf of	Inf States	Leaf nodes in search space	# of Arcs	# of States	next state logic	out-put logic	shared logic	Total logic	Leaf nodes generated
Ckt 1	10	9	9	0	5	18	11	6	10	1	1	10	4
Ckt 2	15	10	9	0	6	18	10	6	9	0	0	9	2
Ckt 3	12	9	9	0	5	10	12	6	13	1	1	13	2
Ckt 4	13	8	7	0	5	1	9	5	10	0	0	10	1

Table 1

6 Conclusion

Contemporary state machine synthesis techniques perform state minimization first and then do state, input and output encoding. In this paper we have shown that for some specific problems, such as the test controller synthesis problem, it is possible to use certain properties specific to the nature of the controllers and come up with an implicit enumeration technique to achieve an optimal design. We have focused on the BILBO TDM and have assumed that test plans are being executed serially. However the test controller model for partial scan is almost identical to that for BILBO, and SOHOT can be applied with minor modifications to this TDM. In this paper we have used the Moore model. Using a Mealy model may produce different results. We have also assumed that the inputs are available fully decoded and restricted ourselves to a single level of output gates. In the future we will remove these restrictions.

References

- [1] M. S. Abadir and M. A. Breuer. A knowledge based system for designing testable VLSI chips. *IEEE Design and Test of Computers*, pages 56-68, August 1985.
- [2] R. K. Brayton, G. D. Hachtel, C. McMullen, and A. Sangiovanni-Vincentelli. *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic, Boston, MA, 1984.
- [3] R. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang. MIS: A multiple-level logic optimization system. *IEEE Trans. Computer-Aided Design*, pages 1062-1081, November 1987.
- [4] A. D. Friedman and P. R. Menon. *Theory and Design of Switching Circuits*. Computer Science Press, 1975.
- [5] S. Devadas, H. K. T. Ma, A. R. Newton, and A. Sangiovanni-Vincentelli. Mustang: State assignment of finite state machines targeting multi-level logic implementations. *IEEE Trans. Computer-Aided Design*, pages 1290-1300, December 1988.
- [6] D. Mukherjee, C. Njinda, and M. A. Breuer. Synthesis of optimal 1-hot coded on-chip controllers for BIST hardware. Technical Report CENG 91-20, University of Southern California, 1991.