On the use of Reset to Increase the Testability of Interconnected Finite-State Machines

Irith Pomeranz and Sudhakar M. Reddy +
Electrical and Computer Engineering Department
University of Iowa
Iowa City, IA 52242
U.S.A.

Abstract

We propose a *DFT* solution for synchronous sequential circuits described as interconnections of finite-state machines, that takes into account specific requirements for justification of test sequences and propagation of fault effects occurring during test generation. We present this solution in the context of the output sequence justification problem. The proposed *DFT* solution is based on the use of reset. Three types of reset mechanisms are considered, having increasing overhead and increasing flexibility. The third type allows every output sequence over the output alphabet of a machine to be justified.

1. Introduction

A synchronous sequential circuit may be represented as an interconnection of finite-state machines due to the design process or through decomposition. The view of a synchronous sequential circuit as an interconnection of finite-state machines was found useful for logic optimization [1]-[7], verification [4], and testing [8]-[12]. The partitioning of a large circuit into smaller finite-state machines has the advantage that the problem of interest can be solved for each component separately at a significantly lower computation time than for the complete circuit. An analysis of the constraints imposed by adjacent machines allows each machine to be analyzed in the context of the complete circuit. However, a decomposition with an arbitrary interconnection structure may not allow the complexity of the problem to be reduced compared to direct consideration of the original circuit, and methods based on decomposition are most effective when the interconnection structure is simple and the interactions between the various machines are limited. When this is not the case, design and synthesis for testability can be used to ensure that test generation can be carried out on a simplified circuit.

One of the important subproblems of the test generation problem on a decomposed circuit is the *output sequence justification problem* (to be distinguished from the state justification problem), defined as follows [12].

The output sequence justification problem: Given a finite-state machine M and an output sequence O, find an initial state s_0 and an input sequence I such that when M is started from state s_0 and the input sequence I is applied, M produces the output sequence O.

This problem occurs at several points during the test generation process, as illustrated by Figure 1. To apply a test sequence T to detect faults in the embedded machine M in Figure 1, M_1 and M_2 must produce specific output sequences corresponding to T. In addition, to propagate fault effects appearing on the outputs of M through M_4, M_3 must produce an appropriate output sequence. The output sequence justification problem must be solved in this case for M_1, M_2 and M_3.

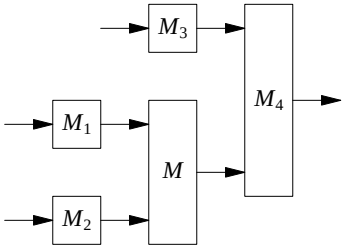


Figure 1: An interconnection of finite-state machines

In [12], a procedure for solving the output sequence justification problem was proposed. It was observed in [12] that in most cases, an arbitrary output sequence O of a submachine M_i cannot be be justified by an input sequence of M_i for any initial state. The solution in [12] consists of applying the output sequence justification procedure repeatedly while a test sequence T is generated for a submachine M, to ensure that every output sequence required to justify T remains justifiable as T is extended to detect additional faults. However, detectable faults remain, for which a test sequence cannot be justified in this way. Moreover, cycles in the interconnection structure may make this approach too computation intensive. The approach of [11] alleviates this problem and other controllability and observability problems by modifying each submachine such that it has a transparent test mode where it transmits its input sequence to its output. During testing, a machine can be placed in this mode and its input (output) sequences can be easily translated into output (input) sequences. Several supporting *DFT* procedures are used in [11] to make this approach feasible. First, it is necessary to ensure that the input and output spaces of adjacent submachines match. For example, in Figure 1, a value v on the output of submachine M can be translated into a value v on the output of the succeeding submachine M_4 only if M_4 has at least $\lceil \log_2 v \rceil$ outputs. To ensure the matching of input and output spaces, internal lines must sometimes be made controllable and/or observable. This is achieved by placing flip-flops on these lines and inserting the flip-flops into a scan chain. In addition, cycles in the interconnection graph are broken by placing additional flip-flops in the scan chain. The approaches of [9] and [10] consist of embedding a predefined test function in each submachine.

+ Research supported in part by NSF Grant No. MIP-9220549, and in part by NSF Grant No. MIP-9357581

ED&TC '97 on CD-ROM

In this work, we consider a *DFT* solution to the output sequence justification problem that differs from other *DFT* solutions [9]-[11] in that it addresses directly the need to justify specific output sequences during test generation. Instead of allowing any output sequence to be justified, the proposed *DFT* procedure considers a given output sequence (or set of output sequences) O of each submachine M and inserts hardware only in order to justify O. The proposed solution is based on the use of reset as a *DFT* approach. In the extreme, each submachine may be reset independently; however, we attempt to minimize the number of reset lines and the hardware overhead to implement reset by minimizing the overall number of reset states for the complete circuit. We distinguish between three approaches to using reset for a given submachine. The approaches differ in the hardware overhead and in the flexibility they provide. In the first approach, called *single-reset single-state* ($1r1s$), reset is applied to a submachine only once, before the input sequence to justify O is applied. In the second approach, called *multiple-reset single-state* ($mr1s$), reset is applied to the machine multiple times. Each time, reset brings the machine to the same state. In the third approach, called *multiple-reset multiple-state* (*mrms*), the machine may be brought to different states at different times by multiple applications of reset. Denoting the class of output sequences that can be justified by a submachine M operating under the approaches above by J_{1r1s}, J_{mr1s} and J_{mrms}, respectively, we have $J_{1r1s} \subseteq J_{1rms} \subseteq J_{mrms}$. This is because every approach contains the reset configurations of the previous ones. As we show in Section 2, the class J_{mrms} contains every sequence comprised of output values over the output alphabet of the machine. The use of reset and the tailoring of the solution to a given output sequence have several consequences.

(1) The hardware overhead of *DFT* solutions that are tailored to a given problem need never be higher than the hardware overhead of generic *DFT* solutions, or solutions that do not take specific test sequences into account. Typically, tailored solutions are significantly cheaper than generic ones.

(2) The use of reset instead of scan avoids the test application time overhead involved in scanning test patterns in and out.

(3) Reset to 0 (1) can be implemented by adding a single AND (OR) gate to the next-state function of a flip-flop. Reset to both values requires two gates. To implement N_{rs} different reset states for an interconnection of m submachines with a total of n flip-flops, $\lceil \log_2(N_{rs}+1) \rceil$ reset lines are required with a decoding logic containing at most $N_{rs}+2n+1$ gates. By keeping the number of reset states low, this hardware overhead can be kept low. Alternatively, reset to multiple states can be implemented using a (partial) scan chain that will be activated only when it is necessary to change the circuit state.

(4) The addition of reset does not require resynthesis.

DFT to facilitate the output sequence justification problem is studied as a stand-alone problem in order to bring out the advantages in terms of flexibility and completeness, and the disadvantages in terms of hardware overhead. In a complete test generation procedure, *DFT* should be used only if the test generation procedure cannot otherwise generate a test to detect a fault. In addition, even for a fault that cannot be detected, the test generation procedure can attempt to generate a test sequence that requires a minimal hardware overhead for justification, by exploring various test sequences. Thus, the hardware overhead in the context of a complete test generation procedure is expected to be significantly lower than demonstrated here.

Faults in the reset hardware are not studied in this work. A discussion of such faults can be found in [13].

The paper is organized as follows. In Section 2 we define three versions of the output sequence justification problem for a given submachine M. In Section 3 we describe solutions to the problems defined in Section 2 and present experimental results. In Section 4 we extend the output sequence justification problem to the case of multiple submachines. This formulation is aimed at minimizing the overall hardware overhead for a circuit represented as an interconnection of submachines. Section 5 concludes the paper.

2. A single submachine

We use the following notation. For a sequence A of length L, $A[u]$ is the value of A at time unit u, $0 \le u < L$. Input sequences are denoted by I and output sequences by O. To describe the application of reset while an input sequence I is applied, we define a reset sequence R as follows. $R[u] = s$ if the machine is reset to state s at time unit u; otherwise, $R[u] = -$. For a machine with a single reset state s_0, $R[u] \in \{s_0, -\}$ for $0 \le u < L$. The reset sequence where $R[0] = s_0$ and $R[u] = -$ for $0 < u < L$ is denoted by R_0.

If $R[u] = s$, we assume that the machine is reset to state s at the beginning of time unit u, and that the next-state and output at time unit u are computed by applying input pattern $I[u]$ at state s. The sequence of states the machine goes through under input sequence I and reset sequence R is denoted by ST.

Next, we define three problems depending on the type of reset sequences allowed. The first one is similar to the output sequence justification problem of [12].

The single-reset problem: Given a finite-state machine M, a reset state s_0 and an output sequence O, find an input sequence I such that when the input sequence I is applied to M together with R_0, M produces the output sequence O.

The multiple-reset single-state problem: Given a finite-state machine M, a reset state s_0 and an output sequence O, find an input sequence I and a reset sequence R such that the following conditions are satisfied. (1) $R[u] \in \{s_0, -\}$ for $0 \le u < L$. (2) If I is applied to M together with R, M produces the output sequence O.

The multiple-reset multiple-state problem: Given a finite-state machine M, a reset state s_0 and an output sequence O, find an input sequence I, a set of reset states S and a reset sequence R such that the following conditions are satisfied. (1) $s_0 \in S$. (2) $|S|$ is minimum. (3) $R[u] \in S$ for $0 \le u < L$. (4) If I is applied to M together with R, M produces the output sequence O (for an implementation using scan, the number of time units where $R[u] \ne -$ should be minimized instead of $|S|$).

The following examples show instances of the three problems defined above. The examples are given for MCNC finite-state machine benchmark $dk15$ whose state table is shown in Table 1. The input and output patterns are given using their decimal values. The reset state is assumed to be state 0.

Table 1: $dk15$

	NS,z							
PS	x=0	x=1	x=2	x=3	x=4	x=5	x=6	x=7
0	0,18	0,20	3,18	0,17	3, 9	0,10	3,10	3,21
1	0,18	0,20	3,18	2, 4	2, 9	0,10	3,16	2, 4
2	2, 5	0, 2	3, 2	0,17	2, 9	0,10	3,10	3,21
3	2, 5	0, 2	3, 2	2, 4	2,20	0, 8	1,10	2, 4

Example 1: The output sequence (10,10,20,2,9,10) can be justi-

fied by applying the input sequence (5,6,4,1,4,6) and the reset sequence R_0 as shown in Table 2(a).

Table 2: Output sequences of *dk*15

(a) Example 1

time	0	1	2	3	4	5
input	5	6	4	1	4	6
reset	0	-	-	-	-	-
state	0	0	3	2	0	3
output	10	10	20	2	9	10

(b) Example 2

time	0	1	2	3	4	5
input	0	5	4	4	2	0
reset	0	-	-	-	0	-
state	0	0	0	3	2→0	3
output	18	10	9	20	18	5

(c) Example 3

time	0	1	2	3	4	5
input	3	1	6	4	3	0
reset	0	-	1	-	1	-
state	0	0	0→1	3	2→1	2
output	17	20	16	20	4	5

Example 2: The output sequence (18,10,9,20,18,5) cannot be justified by any input sequence using the reset sequence R_0 (this will be shown in Section 3.1). However, it can be justified by using the input sequence (0,5,4,4,2,0) and the reset sequence (0,-,-,-,0,-) as shown in Table 2(b). Note that under input pattern 4 applied at time unit 3, the machine goes from state 3 to state 2. However, due to the application of reset at time 4, the present-state at time 4 is 0. This is indicated by the symbol 2→0.

Example 3: The output sequence (17,20,16,20,4,5) cannot be justified by using a single reset state 0. The output sequence O can be justified using the sequences shown in Table 2(c). At time unit 4, we have a choice between resetting the machine to state 1 or state 3, both of them capable of producing an output value 4. However, since state 1 is necessary at time 2, we prefer to repeat the reset state 1 and keep the set of reset states minimum, {0,1}.

Next, we consider several necessary and sufficient conditions for the existence of a solution to each one of the output sequence justification problems defined above. Let the output patterns that can be produced by a machine M in state s be A(s). For example, for *dk*15, A(0) = {9, 10, 17, 18, 20, 21}, A(1) = {4, 9, 10, 16, 18, 20}, and so on. Let $A = \bigcup_s A(s)$. Then M can only produce output sequences in 2^A, the power set of A. Using reset sequences with a single reset state s_0, the first pattern of any justifiable output sequence O must belong to A(s_0). Using reset sequences with multiple reset states, any output sequence in 2^A can be justified as follows. Consider an arbitrary output sequence O of length L. To produce $O[u]$, $0 \le u < L$, we select a state s such that $O[u]$ can be produced from s by an input pattern i. We set $R[u] = s$ and $I[u] = i$. Since $O[u] \in$ A, this can be done for every $0 \le u < L$. During test generation, it is easy to ensure that all the output sequences of a machine M are defined over A. Therefore, the examples and experimental results in this work use output sequences over A.

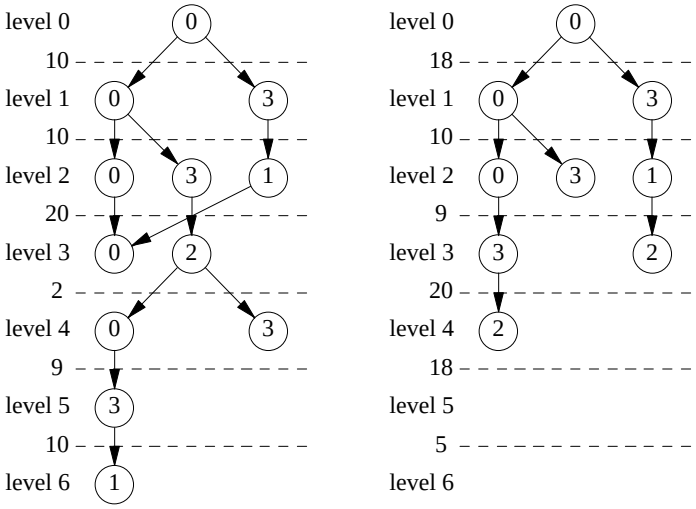
3. Using reset for a single submachine

In this section, we present procedures to solve the output sequence justification problems defined in Section 2. The procedures are described for an n state machine with states $\{s_0, s_1, \cdots, s_{n-1}\}$. The reset state for normal operation is assumed to be s_0. The output sequence to be justified is denoted by O.

3.1 The single-reset problem

The single-reset problem is similar to the one solved in [12]. We present here a different solution which is easier to extend to the problems with multiple applications of reset.

The proposed procedure constructs a levelized graph, called the *state-transition graph of O*. The graph is similar to the ones used in [14] for designing input/output experiments for finite-state machines. Each level contains at most n states. In level 0 of the graph we have the reset state s_0. Every state s_i in level u of the graph has at most n edges to states in level $u+1$. All the edges correspond to state-transitions that produce the output value $O[u]$. The end-state of an edge corresponds to the next-state under the corresponding state-transition. For illustration, the graph for the output sequence (10,10,20,2,9,10) of *dk*15 is shown in Figure 2(a). The dashed lines in Figure 2(a) separate two levels. The numbers next to the dashed lines are the corresponding output patterns. A state becomes a terminal state in one of two cases. (1) It cannot produce the required output pattern. (2) It is in level L of the graph. If all the states at a level $u < L$ become terminal, no input sequence exists to justify the given output sequence starting from the reset state. The output sequence (18,10,9,20,18,5) considered in Example 2 above is shown in Figure 2(b) and demonstrates this case.



(a) (10,10,20,2,9,10) **(b) (18,10,9,20,18,5)**

Figure 2: state-transition graphs for *dk*15

To find an input sequence to justify a given output sequence from its state-transition graph (assuming the graph has at least one state at level L), we start from any state at level L and trace the state-transitions backwards to the reset state along any path. For each state-transition, we find an appropriate input pattern and include it in the appropriate place in the input sequence. For example, in Figure 2(a), the last state-transition from state 3 to state 1 producing an output 10 can be accomplished under an input pattern 6. This pattern is entered as $I[5]$.

The procedure illustrated above is extended to allow a maximal prefix of O to be justified. This procedure will be useful in the following subsections where reset is used for justification. The extended procedure is shown in Figure 3. The procedure

maintains the state-transition graph implicitly by storing the set of states at level u in a set S_u.

Procedure 1: Justifying a maximum prefix of an output sequence O with reset state s_0

(1) Set $S_0 = \{s_0\}$ and $u = 0$.
(2) If $u = L$ go to Step 7.
(3) Set $S_{u+1} = \phi$.
(4) For every state $s \in S_u$ and for every input pattern i:
  (a) Let the next-state for present-state s and input pattern i be s' and let the output pattern be o.
  (b) If $o = O[u]$, add s' to S_{u+1}.
(5) If $S_{u+1} = \phi$, set $u = u - 1$ and go to Step 7.
(6) Set $u = u + 1$ and go to Step 2.
(7) Trace an input sequence by starting from any state in S_u and selecting state-transitions from a state in $S_{u'}$ to the selected state in $S_{u'+1}$, for $u' = u-1, u-2, \cdots, 0$.

Figure 3: Procedure 1

3.2 The multiple-reset single-state problem

To solve the multiple-reset single-state problem, we call Procedure 1 iteratively. Initially, Procedure 1 is used to justify the longest possible prefix of O. At an arbitrary iteration, we have input, state and reset sequences to justify a prefix O' of O. Let the last justified output value be at time unit u. We replace the state at time unit u by s_0 and check if it is possible to find an input sequence I'' to justify a subsequence O'' of O that starts at time unit u, assuming the state at time unit u is s_0. We keep reducing u as long as no subsequence I'' is found. If a subsequence I'' is found starting at time unit u, the reset, input and output sequences are updated and O' is redefined. This process is illustrated by the following example. The general procedure is shown in Figure 4.

Example 4: Consider the output sequence (18,10,4,9,4,9,8) of *dk*15. Applying Procedure 1 to this sequence, we find that it is possible to justify the output values up to time unit 3, as shown in Table 3(a).

Next, we consider the following options. Applying reset at time unit 4 would change the state at time unit 4 to 0. We call Procedure 1 with the remainder of the output sequence, (4,9,8), however, state 0 cannot produce an output pattern 4, and therefore, no additional input values can be justified. Applying reset at time unit 3 would change the state at time unit 3 to 0. We call Procedure 1 with the remainder of the output sequence, (9,4,9,8), and find the input sequence (4,3,4) to justify two additional output patterns beyond those already justified in Table 3(a). The new sequence is shown in Table 3(b).

Next, we consider the following options. Applying reset at time unit 6 would change the state at time unit 6 to 0. We call Procedure 1 with the remainder of the output sequence, (8), however, state 0 cannot produce an output pattern 8, and therefore, no additional input values can be set. Applying reset at time unit 5 would change the state at time unit 5 to 0. We call Procedure 1 with the remainder of the output sequence, (9,8) and find the input sequence (4,5). The output sequence is now justified.

Procedure 2 covers the given output sequence O by using input subsequences that start from the reset state s_0 and produce subsequences of O. The ability to justify a subsequence of O by setting $R[u] = s_0$ for some u is not affected by setting $R[u'] = s_0$ at another time unit u'. In other words, the decision to apply reset at any time unit is independent of the application of reset at any other time unit. Thus, if there exists input and reset sequences to justify O, Procedure 2 will find them.

Table 3: Output sequence (18,10,4,9,4,9,8) of *dk*15

(a) Iteration 1

time	0	1	2	3	4	5	6
input	0	6	3	4			
reset	0	-	-	-			
state	0	0	3	2	2		
output	18	10	4	9	4	9	8

(b) Iteration 2

time	0	1	2	3	4	5	6
input	0	6	3	4	3	4	
reset	0	-	-	0	-	-	
state	0	0	3	2→0	3	2	2
output	18	10	4	9	4	9	8

Procedure 2: Justifying an output sequence O with multiple-reset single-state s_0

(1) Define $O' = (\phi)$ and $L' = 0$ (O' is the justified part of O and L' is its length). Let $R = (-, -, \cdots, -)$ and $I = (\phi)$. Let $ST = (-, -, -, \cdots, -)$.
(2) Set $u = L'$.
(3) If $ST[u] = s_0$ go to Step 7 (the state at time u is already s_0; no additional justification is possible by reset to s_0).
(4) Define $O'' = (O[u], O[u+1], \cdots, O[L-1])$.
(5) Call Procedure 1 with O'' and s_0. Let the input sequence found by Procedure 1 be I'' of length L''.
(6) If $u + L'' > L'$ (I'' allows to increase the length of the justified part of O):
  (a) Add O'' to O' starting at time u (i.e., set $O'[u] = O''[0]$, $O'[u+1] = O''[1]$, $\cdots$, $O'[u+L''-1] = O''[L''-1]$).
  (b) Add I'' to I starting at time u.
  (c) Set $R[u] = s_0$ and $R[u'] = -$ for $u' > u$.
  (d) Update the state sequence ST according to I'' and R.
  (e) Set $L' = u + L''$.
  (f) If $O' = O$, stop: I and R are the required sequences.
  (g) Go to Step 2.
(7) Set $u = u - 1$. If $u < 0$, stop: O cannot be justified.
(8) Go to Step 3.

Figure 4: Procedure 2

It is possible to solve the multiple-reset single-state problem by extending Procedure 1 as follows. Whenever a state-transition exists from state s' to state s'', the next-state can be changed to s_0 by applying reset. Consequently, for every edge from s' to s'', we also include an edge from s' to s_0. In the resulting state-transition graph, each path defines input and reset sequences for the output sequence. However, Procedure 2 is easier to extend to the case of multiple reset states considered next.

3.3 The multiple-reset multiple-state problem

To justify a given output sequence O using multiple reset states, Procedure 2 is extended as follows. At every time unit u along O' ($u = L', L'-1, \cdots, 0$ where L' is the length of O'), the sequence O'' is defined as the remaining, unjustified subsequence of O, as in Procedure 2. However, instead of trying to justify O'' starting from s_0, we use every possible state as the starting state at time u. The first state that produces an input sequence I''

allowing the length of O' to be extended is accepted. To minimize the total number of reset states required to justify O, we start with a set of reset states S including only s_0. Whenever possible, we use a reset state that is already in S. Only if no sequence I'' exists using a state from S, a new reset state is selected and entered into S. The extended procedure is referred to as Procedure 3.

We point out that for an output sequence defined over A, a sequence I'' can always be found for $u = L'$, and no additional time units need to be considered. This is because there is always a state from which any given output value can be justified. It is possible to extend Procedure 3 to consider all the states at time unit $u = L'$ and select the one that results in a maximal increase in the length of O'' (instead of selecting the first one). This extension did not result in a significant improvement in the results obtained. It is also possible to consider each state in S under multiple time units u.

3.4 Experimental results

The goal of the experiments reported here is to demonstrate the hardware overhead involved in justifying a given output sequence by using reset. In the context of a complete test generation procedure, it is expected that reduced hardware overhead would be achieved by allowing backtracking to modify the output sequences that need to be justified. We applied Procedures 1-3 to MCNC finite-state machine benchmarks. As output sequences we use 100 random sequences for each machine and we report averages over these 100 experiments. Each sequence is of length 20. The sequences were defined over the set of output patterns A that can be produced by the machine. Although actual test sequences to detect all the faults in a given machine may be longer, a test generation procedure can generate test sequences that are easier to justify than the random sequences considered here.

The results of Procedures 1, 2 and 3 are shown in Table 4 as follows. Under the $1r1s$ column we show the information using the single-reset procedure (Procedure 1). The information for the multiple-reset single-state procedure (Procedure 2) is shown under column $mr1s$. For each procedure, we show the number of unjustifiable sequences followed by the number of justifiable sequences. For Procedure 2 we also show the average number of times reset is applied during the justifiable sequences (column *reset jst*). There are only seven circuits (marked with asterisks) where the use of multiple applications of reset helped justify sequences that could not be justified using a single application of reset. The results obtained using the multiple-reset multiple-state procedure (Procedure 3) are shown under column *mrms*. In this case, all the output sequences can be justified. Under subcolumn *reset n* we show the average number of applications of reset, followed by the average number of reset states used in every justified sequence. In most cases, the average number of reset states is low, between 1 and 3 states. Even when the sequence length was increased to 100, the number of reset states required increased only slightly in most cases. The run time in seconds on an HP 730 workstation is less than one second for Procedures 1 and 2. The highest run time of Procedure 3 is 13.05 seconds for circuit *fetch*. The run time is over 2 seconds only for four circuits.

We also restricted the output sequences such that their first pattern was randomly selected out of A(0), and the remaining patterns were randomly selected out of A. Thus, the sequences satisfy the necessary condition for being justifiable using a single reset state. As can be expected, more output sequences are justifiable using a single application of reset. Using multiple applications of reset with a single reset state allowed more sequences to be justified in 8 of the circuits. The complete set of results is omitted for space considerations.

Table 4: Results of Procedures 1, 2 and 3

	1r1s			mr1s			mrms	
	nseq			nseq		reset	reset	
circuit	unjst	jst		unjst	jst	jst	n	st
bbara	100	0		100	0	0.00	8.97	3.00
bbsse	100	0		100	0	0.00	17.30	5.81
bbtas	100	0	*	99	1	2.00	4.41	2.00
cse	100	0		100	0	0.00	15.61	5.96
dk14	100	0		100	0	0.00	8.03	3.39
dk17	100	0	*	99	1	4.00	7.52	2.55
dk27	100	0	*	99	1	5.00	9.81	2.31
dvram	100	0		100	0	0.00	18.59	13.05
ex2	49	51		49	51	1.00	1.07	1.49
ex4	100	0		100	0	0.00	17.88	7.63
ex5	78	22	*	51	49	1.92	2.00	1.51
ex7	49	51		49	51	1.00	1.00	1.49
fetch	100	0		100	0	0.00	18.94	12.46
firstex	100	0	*	98	2	4.00	8.09	1.98
keyb	100	0		100	0	0.00	10.63	2.00
lion	0	100		0	100	1.00	1.00	1.00
lion9	51	49		51	49	1.00	1.37	1.51
log	100	0		100	0	0.00	18.45	12.05
mark1	100	0		100	0	0.00	18.37	10.21
rie	100	0		100	0	0.00	18.34	12.83
shiftreg	93	7		93	7	1.00	4.37	1.93
tav	100	0	*	99	1	5.00	11.64	2.00
train11	0	100		0	100	1.00	1.00	1.00
train4	90	10	*	0	100	3.28	3.28	1.00

4. Multiple submachines

In the previous sections we considered the selection of a reset sequence for a single machine. In practice, it is necessary to consider multiple machines to minimize the hardware overhead of output sequence justification using reset. The general problem is formulated as follows.

The multiple-reset multiple-state problem for multiple machines: Given finite-state machines $M_1, M_2, \cdots, M_m$, a reset state s_{0_i} and an output sequence O_i for M_i, $1 \leq i \leq m$, find an input sequence I_i and a reset sequence R_i for every $1 \leq i \leq m$, such that the following conditions are satisfied. (1) If I_i is applied to M_i together with R_i, M_i produces the output sequence O_i, $1 \leq i \leq m$. (2) The number of different reset configurations (defined below) is minimum.

A *reset configuration* is a pattern $(R_1[u]R_2[u]\cdots R_m[u])$. For each such pattern, the reset hardware has to contain appropriate gates to implement the reset of M_i to state $R_i[u]$, $1 \leq i \leq m$. By minimizing the number of reset configurations, we attempt to minimize the hardware overhead of reset.

The procedure for solving the multiple-reset multiple-state problem for two machines M_1, M_2 is referred to as Procedure 4. Procedure 4 is based on Procedure 3 of Section 3.3, with the following changes. The set of reset states S used in Procedure 3 is replaced by a set of reset state pairs, also denoted S. A pair (s_1, s_2) implies that M_i is reset to state s_i, $i = 1, 2$. The size of S is minimized by always trying to use a pair of reset states that has already been used, before new pairs are tried. The two output sequences O_1 and O_2 are justified simultaneously by repeatedly extending the length of the shorter of the two prefixes

that have already been justified. A change is accepted if it lengthens the shorter prefix without reducing the length of the longer prefix. This is always possible when O_i is defined over A_i, $i = 1, 2$, for the following reasons. For the shorter justified prefix, it is always possible to choose a reset state that allows the next output value to be justified. For the longer justified prefix, the actual state reached can be selected as the reset state to ensure that the same prefix length is obtained.

Experimental results of Procedure 4 for pairs of circuits from Table 4 are shown in Table 5. The results include the run time, the number of times reset is applied and the number of reset states when only the output sequence of Circuit 1 is justified using Procedure 3 (column *only 1*), when only the output sequence of Circuit 2 is justified using Procedure 3 (column *only 2*), and when both sequences are justified simultaneously using Procedure 4 (column *both*). In the latter case, the subcolumn *reset st* gives the number of reset state pairs. To interpret the results, we denote by N_{rs_i} the number of reset states required to justify output sequence i, $i = 1, 2$, and by N_{rsp} the number of reset state pairs required to justify both output sequences simultaneously. The number of lines needed to encode N reset states is $\lceil \log_2 N \rceil$. Thus, one needs to compare $\lceil \log_2 N_{rs_1} \rceil + \lceil \log_2 N_{rs_2} \rceil$ with $\lceil \log_2 N_{rsp} \rceil$. Viewed in a different way, the number of reset state pairs when each output sequence is justified separately is $N_{rs_1} \cdot N_{rs_2}$, and this number should be compared to N_{rsp}. We show the value of $N_{rs_1} \cdot N_{rs_2}$ in the last column of Table 5. On the average, N_{rsp} is half of $N_{rs_1} \cdot N_{rs_2}$. In a complete test generation procedure, it is expected that N_{rsp} would be significantly smaller than in the stand-alone experiments reported here, as *DFT* would be used only if the test generation procedure cannot otherwise generate a test to detect a fault. In addition, test sequences that require a minimal hardware overhead for justification can be generated. We point out that the run time for the pair of circuits *dvram* and *ex*2 is the largest since *dvram* and *ex*2 have two of the largest numbers of states among the circuits considered.

Table 5: Results of Procedure 4 (pairs of machines)

		only 1			only 2			both			
			reset			reset			reset		
circ1	circ2	time	n	st	time	n	st	time	n	st	
bbara	bbsse	0.24	8.99	3.00	1.20	17.18	5.72	6.2	18.33	9.77	17.2
bbsse	bbtas	1.25	17.12	5.75	0.06	4.43	2.00	6.4	17.62	7.00	11.5
bbtas	cse	0.04	4.43	2.00	1.14	15.53	5.99	2.9	16.19	7.95	12.0
cse	dk14	1.07	15.42	5.97	0.09	7.86	3.36	7.9	17.16	11.98	20.1
dk14	dk17	0.08	8.13	3.34	0.07	7.67	2.56	0.7	12.29	7.09	8.5
dk17	dk27	0.03	7.57	2.63	0.04	9.59	2.35	0.5	13.68	6.92	6.2
dk27	dvram	0.05	9.66	2.46	8.74	18.58	12.98	24.1	19.27	17.68	31.9
dvram	ex2	8.78	18.80	13.38	0.09	1.05	1.47	201.8	18.80	14.96	19.7
ex2	ex4	0.06	1.06	1.43	0.22	17.95	7.58	3.8	17.95	9.98	10.8
ex4	ex5	0.26	17.72	7.62	0.03	2.13	1.53	1.3	17.83	12.34	11.7
ex5	ex7	0.02	1.91	1.57	0.06	1.00	1.47	0.3	1.93	1.92	2.3
ex7	fetch	0.06	1.00	1.43	5.76	18.90	12.54	12.9	18.90	15.07	17.9
fetch	first	5.73	18.92	12.87	0.05	8.48	1.99	75.9	19.47	16.72	25.6
first	keyb	0.07	7.58	1.99	0.50	10.79	2.00	2.4	14.53	4.44	4.0
keyb	lion	0.47	10.26	2.00	0.01	1.00	1.00	1.3	10.26	2.00	2.0
lion	lion9	0.01	1.00	1.00	0.05	1.42	1.53	0.1	1.42	1.53	1.5
lion9	log	0.05	1.41	1.57	5.25	18.45	12.08	21.5	18.50	15.78	19.0
log	mark1	5.22	18.60	12.34	0.15	18.48	10.09	81.9	19.89	19.84	124.5
mark1	rie	0.22	18.49	10.36	6.50	18.58	12.91	12.3	19.93	20.27	133.7
rie	shift	6.46	18.31	13.10	0.06	4.07	1.92	48.7	18.60	17.80	25.1
shift	tav	0.05	4.13	1.93	0.04	11.85	2.00	0.9	13.65	6.93	3.9
tav	trn11	0.06	11.51	2.00	0.06	1.00	1.00	0.4	11.51	2.00	2.0
trn11	trn4	0.03	1.00	1.00	0.02	3.53	1.00	0.2	3.53	1.00	1.0
avrge										10.04	22.3

5. Concluding remarks

We proposed a *DFT* solution to the output sequence justification problem. The proposed solution considers specific output sequences required during test generation. Thus, it allows the *DFT* solution to be tailored to the circuit. The proposed solution was based on the use of reset. Three types of reset hardware were considered, having increasing overhead and increasing flexibility, including a single application of reset at the beginning of the sequence, multiple applications of reset that bring the machine to the same reset state, and the use of multiple reset states. The third type allows every output sequence over the output alphabet of the machine to be justified.

References

[1] J. Kim and M. M. Newborn, "The Simplification of Sequential Machines with Input Restrictions", IEEE Trans. on Computers, Dec. 1972, pp. 1440-1443.

[2] S. Devadas, "Optimizing Interacting Finite State Machines Using Sequential Don't Cares", IEEE Trans. on Computer-Aided Design, Dec. 1991, pp. 1473-1484.

[3] J. Kukula and S. Devadas, "Finite State Machine Decomposition by Transition Pairing", in Proc. 1991 Intl. Conf. on Computer-Aided Design, Nov. 1991, pp. 414-417.

[4] H. Cho, G. D. Hachtel, E. Macii, B. Plessier and F. Somenzi, "Algorithms for Approximate FSM Traversal", in Proc. 30th Design Autom. Conf., June 1993, pp. 25-30.

[5] Y. Watanabe and R. K. Brayton, "The Maximum Set of Permissible Behaviors for FSM Networks", in Proc. 1993 Intl. Conf. on Computer-Aided Design, Nov. 1993, pp. 316-320.

[6] H.-Y. Wang and R. K. Brayton, "Input Don't Care Sequences in FSM Networks", in Proc. 1993 Intl. Conf. on Computer-Aided Design, Nov. 1993, pp. 321-328.

[7] J.-K. Rho and F. Somenzi, "Don't Care Sequences and the Optimization of Interacting Finite State Machines", IEEE Trans. on Computer-Aided Design, July 1994, pp. 865-874.

[8] I. Pomeranz and S.M. Reddy, "A Divide-and-Conquer Approach to Test Generation for Large Synchronous Sequential Circuits", in Proc. 22nd Fault-Tolerant Computing Symposium, July 1992, pp. 230-237.

[9] S. Kanjilal, S. T. Chakradhar and V. D. Agrawal, "Test Function Embedding Algorithms with Application to Interconnected Finite State Machines", IEEE Trans. on Computer-Aided Design, Sept. 1995, pp. 1115-1127.

[10] S. Kanjilal, S. T. Chakradhar and V. D. Agrawal, "A Partition and Resynthesis Approach to Testable Design of Large Circuits", IEEE Trans. on Computer-Aided Design, Oct. 1995, pp. 1268-1276

[11] F. Fummi, U. Rovati and D. Sciuto, "Testable Synthesis of High Complex Control Devices", in Proc. EURO-DAC '95, Sept. 1995, pp. 117-122.

[12] I. Pomeranz and S. M. Reddy, "On Test Generation for Interconnected Finite-State Machines - The Output Sequence Justification Problem", in Proc. 1996 Europ. Design & Test Conf., March 1996, pp. 380-385.

[13] I. Pomeranz and S. M. Reddy, "On the Detection of Reset Faults in Synchronous Sequential Circuits", in Proc. 1997 VLSI Design Conf., Jan. 1997.

[14] Z. Kohavi, *Switching and Finite Automata Theory*, McGraw-Hill, 1978.