Symbolic Optimization of FSM Networks Based on Sequential ATPG Techniques

Fabrizio Ferrandi [#] Franco Fummi [#] Enrico Macii [‡] Massimo Poncino [‡] Donatella Sciuto [#]

[#] Politecnico di Milano
Dip. di Elettronica e Informazione
Milano, ITALY 20133

[‡] Politecnico di Torino
Dip. di Automatica e Informatica
Torino, ITALY 10129

Abstract

This paper presents a novel optimization algorithm for FSM networks that relies on sequential test generation and redundancy removal. The implementation of the proposed approach, which is based on the exploitation of input don't care sequences through regular language intersection, is fully symbolic. Experimental results, obtained on a large set of standard benchmarks, improve over the ones of state-of-the-art methods.

1 Introduction

The most natural way of representing complex designs is through networks of interacting FSMs, where each network component represents a module that can be synthesized and optimized separately. Clearly, treating each component in isolation may not lead to a globally optimal design. In fact, this approach prevents the exploitation of the freedom introduced by the mutual interaction of each module with its neighbors in the realization of each component.
To overcome this problem, and thus obtain a better quality of the global design, synthesis and optimization techniques that take into account the interaction between the network components have been developed in the last few years. One of the most popular approaches is based on the concept of input don't care sequences (IDCSs). Let us consider the FSM network of Figure 1.

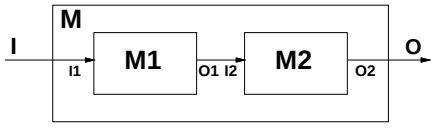

Figure 1: FSM Network with Cascade Decomposition.

In such network there exists a cascade decomposition; in fact, the driving machine M_1 feeds the driven machine M_2, while no information flows from M_2 to M_1. As observed by Unger in [1], machine M_2 may have more unspecified input behaviors when it is connected to machine M_1 than when it operates in isolation. These unspecified behaviors are the IDCSs of M_2 and can be used for optimization purposes. In [2], Kim and Newborn have proposed an exact algorithm for the optimization of the driven machine based on IDCSs. On the other hand, Devadas [3] has presented an efficient, though approximate, modification to Unger's method. In [4], Rho and Somenzi have introduced extensions to Kim and Newborn's procedure. Finally, Wang and Brayton [5] have generalized the approach of [2] to FSM networks of arbitrary topology.

In this paper, we present an algorithm for the optimization of M_2 which is based on sequential ATPG and redundancy removal. Random pattern simulation is initially executed at the inputs of M_1 to remove easy-to-detect faults from M_2. Then, test sequences for hard-to-detect faults in M_2 are generated at the inputs of M_2. Finally, a regular language intersection is performed to verify if at least one of such sequences can be generated at the outputs of M_1. If this is not the case, the fault is redundant and it can be removed from M_2. The standard way for checking regular language intersection in a cascade network requires the traversal of the composed machine $M_1 \rightarrow M_2 \times M_2^F$, where M_2^F indicates the faulty M_2. Unfortunately, reachability analysis on such a machine is usually too memory and time consuming. We present methods to simplify both M_1 and $M_2 \times M_2^F$ by means of finite automata, so as to make the task of the traversal program easier.
Test sequences for faults contained in M_2 are generated using a modified version of the ATPG procedure of [6], which is able to operate on both state-graphs and gate-level descriptions. On the other hand, the algorithms for simplifying $M_2 \times M_2^F$ and for optimizing M_2 are original contributions of this work. All the procedures rely on BDD-based symbolic techniques, and have been implemented in SIS [7]. Results obtained on standard benchmarks are satisfactory.

2 Optimization of FSM Networks

2.1 Overview

Optimization algorithms based on IDCSs and redundancy removal are funded on the following principle. Let f be a fault in machine M_2, and let T_f be the set of all input sequences (to machine M_2) which detect f. Then, the fault can be removed if either T_f is the empty set, or T_f is non-empty but none of the input sequences in T_f can be generated at the outputs of machine M_1.
A simple way to check whether fault f in M_2 is redundant or not with respect to the global circuit consists of injecting f into M_2 to obtain the faulty machine M_2^F, taking the product $M_2 \times M_2^F$, feeding it with M_1, and generating the test sequences for f on the so obtained FSM. If no sequence exists, f is redundant and can be removed. Unfortunately, this approach is not always applicable to real cases, due to the limitation imposed by the size of the FSM to be handled. In fact, even if BDD-based techniques are employed, the representation of either the cascade machine $M_1 \rightarrow M_2 \times M_2^F$, or its set of reachable states, or both may become too large to be manipulated.

33rd Design Automation Conference ®

DAC 96 - 06/96 Las Vegas, NV, USA

The algorithm we propose simplifies the method above and is based on regular language intersection. Let A_f be the finite automaton accepting the language $\mathcal{L}(A_f)$ of all the test sequences in T_f (an implicit algorithm to compute A_f is summarized in Section 2.2), and let $\mathcal{L}(M_1^O)$ be the regular language of all the output sequences generated by M_1. Because M_1 and M_2 are connected, languages $\mathcal{L}(A_f)$ and $\mathcal{L}(M_1^O)$ are defined over the same alphabet. Then, fault f is redundant if and only if there exists no sequence generated by M_1 which is accepted by A_f. Formally:

$$\mathcal{L}(M_1^O) \cap \mathcal{L}(A_f) = \emptyset. \quad (1)$$

We can verify condition (1) through reachability analysis of the automaton $P = A_1 \times A_f$ [8], where A_1 is a deterministic finite automaton accepting language $\mathcal{L}(M_1^O)$. Each state p of P is actually a pair (a_1, a_f), where a_1 is a state of A_1 and a_f is a state of A_f. Therefore, if the set of reachable states of P contains at least one state $p = (a_1, a_f)$ such that a_f is an accepting state of A_f, then $\mathcal{L}(A_1) \cap \mathcal{L}(A_f) \neq \emptyset$, indicating that fault f is irredundant. In essence, A_f can be viewed as the "interesting" portion of $M_2 \times M_2^F$ that has to be fed by M_1. We can only guarantee that A_f is not larger than $M_2 \times M_2^F$; however, what happens in practice is that $A_f \ll M_2 \times M_2^F$, so chances of a successful traversal of P increase when A_f is used instead of $M_2 \times M_2^F$.

Automaton A_1 can be computed by first getting a non-deterministic version, A_1', as in Kim and Newborn's procedure, and then determinizing it using Rabin and Scott's algorithm [9]. Unfortunately, this algorithm is exponential in the number of states of A_1'. Recently, Wang and Brayton have proposed a new method for FSM network optimization [10] that first calculates a possibly non-deterministic automaton A_1' accepting language $\mathcal{L}(M_1^O)$ through a procedure similar to Kim and Newborn's. Then, a deterministic FSM, D_1, simpler than M_1 but generating the same output language as M_1, is derived from A_1' and synthesized. Finally, the cascade of D_1 and M_2 is optimized using traditional techniques. The advantage of this method over Kim and Newborn's is that it does not require the determinization of A_1'. In fact, simple construction techniques are given to obtain an implementation of D_1 from A_1', even when A_1' is non-deterministic. Clearly, we can use a similar approach to build A_1 from A_1'. However, determinizing A_1' is not required by our algorithm. In fact, for our purposes, it is sufficient to traverse the product automaton $P' = A_1' \times A_f$ to check if condition (1) holds.

2.2 Building Automaton A_f

In [6] we have proposed a symbolic algorithm to compute the set T_f for a fault f contained into a FSM. We have modified the procedure to generate the automaton A_f of all the test sequences for a fault f located into M_2. For efficiency reasons, A_f is constructed directly during test generation. Each state of A_f is labeled as s/s_F to indicate that, after the application of an input sequence, the machines M_2 and M_2^F will be in states s and s_F, respectively. Such a sequence is obtained by concatenation of the edge labels on the path connecting the root state (i.e., the initial state) of A_f to the state labeled s/s_F.

The procedure that computes A_f does not require the calculation of the test sequences for fault f in an explicit way (i.e., one sequence at a time) but, instead, it determines all of them simultaneously. Two steps are needed: First, all the minimum length test sequences for fault f are stored as a directed, levelized graph, called in [6] the *expansion graph*. Then, the automaton A_f is constructed by choosing the accepting state among all the "distinguishing" states reached at the last deepest level of the expansion graph, and by adding the back-edges required to make the automaton accepting test sequences of any length.

We have observed that the use of A_f instead of $M_2 \times M_2^F$ is always beneficial for the optimization procedure. The following result supports the experimental evidence.

Theorem 2.1 The state graph of A_f is always smaller than or equal to the state graph of $M_2 \times M_2^F$.

Sketch of Proof: Consider the worst case (the set of test sequences for f in M_2 forms the input language of $M_2 \times M_2^F$), and suppose that we construct A_f starting from $M_2 \times M_2^F$ (in reality, A_f is always incrementally built during test generation). All the accepting states of A_f are equivalent. Pick one of them and collapse all the others into it. Then, when $M_2 \times M_2^F$ has only one distinguishing state, A_f and $M_2 \times M_2^F$ are of the same size. Otherwise, A_f is smaller. See [11] for more details. ∎

Corollary 2.2 A_f is always a sub-graph of the minimum automaton accepting the input language of $M_2 \times M_2^F$.

2.3 Verifying Condition (1)

In this section, we prove that determinizing the (possibly) non-deterministic automaton A_1' is not strictly required to verify condition (1) through the traversal of $P' = A_1' \times A_f$. Consider machine M_1, and let $\mathcal{L}(M_1^O)$ be the regular language it produces. Let A_1' be the finite automaton accepting the regular language $\mathcal{L}(A_1') = \mathcal{L}(M_1^O)$ and computed using [2]. By construction, A_1' is completely specified, but it may be non-deterministic. We have the following result.

Theorem 2.3 Condition $\mathcal{L}(M_1^O) \cap \mathcal{L}(A_f) = \emptyset$ can be verified via reachability analysis of $P' = A_1' \times A_f$.

Sketch of Proof: A_f is deterministic, while A_1', in general, is not. If there is a state in P' which is reachable from the reset state, and such that it corresponds to the accepting state of A_f, then there exists at least one sequence which belongs to both $\mathcal{L}(A_1')$ and $\mathcal{L}(M_1^O)$ by construction. Therefore, there exists at least one sequence which is generated by M_1 and accepted by A_f. See [11] for more details. ∎

2.4 Optimization Algorithm

Figure 2 shows the pseudo-code of our algorithm. The procedure starts by constructing, once and for all, automaton A_1', by creating the collapsed fault list, F_{list}, for faults in M_2, and by initializing to $\emptyset$ the set VT_m of all test sequences for all such faults. Then, a set of random sequences, T, is generated and simulated over the cascade network, and F_{list} is properly updated. Then, the optimization loop begins. A fault, f, is extracted from F_{list} and inserted into M_2 to get M_2^F, the automaton A_f is constructed, and the product $P' = A_1' \times A_f$ is calculated. The set of reachable states of P' is now computed

through symbolic traversal, and the set of states belonging to $Reached(P')$ corresponding to the accepting states of A_1' and A_f is extracted. If such set is empty, then fault f is redundant, so it is removed from M_2, and the new F_{list} for the optimized M_2 is created; the set of randomly generated test sequences T is then simulated on $M_1 \rightarrow M_2$ and F_{list} is updated accordingly. Otherwise, the valid set of test sequences, VT_f, is determined, accumulated into VT_m, and finally simulated over machine M_2. Upon convergence, i.e., no faults are left in F_{list}, the procedure returns the optimized M_2; since A_1' accepts all the output sequences of M_1, the optimized M_2 is guaranteed to be 100% irredundant when driven by M_1.

```
procedure Optimize_Network (M1,M2) {
  A1' = Kim_Newborn_Step_1(M1);
  VTm = ∅.
  Flist = Create_Fault_List(M2);
  T = Generate_Random_Sequences(M1 → M2);
  Flist = Upd_Fault_List(Simulate(M1 → M2,T), Flist);
  while (Flist ≠ ∅) {
    f = Get_Fault(Flist);
    M2^F = Inject_Fault(M2);
    Af = Generate_Test_Sequences(M2,M2^F,f);
    P' = A1' × Af;
    Reached(P') = Symbolic_Traversal(P');
    if (Accepting_States (Reached(P')) == ∅) {
      M2 = Redundancy_Removal(M2,f);
      Flist = Create_Fault_List(M2);
      Flist = Upd_Fault_List(Simulate(M1 → M2,T), Flist);
    }
    else {
      VTf = Extract_Sequences(Af);
      VTm = VTm ∪ VTf;
      Flist = Upd_Fault_List (Simulate(M2, VTm), Flist);
    }
  }
  return(M2);
}
```

Figure 2: The Optimize_Network Algorithm.

It is important to observe that machine M_2 is assumed to be irredundant when considered in isolation. Then, automaton A_f never represents the empty language.

3 Experimental Results

We present results for networks built as the cascade of Mcnc'91 symbolic FSMs [12] and Iscas'89 sequential circuits [13]. The symbolic FSMs have been initially state minimized in isolation using Stamina, and then encoded using Jedi. The encoded state transition graphs have been synthesized with SIS. Finally, the resulting networks have been optimized by first running script.rugged and then by removing all the redundant faults. For the sequential circuits a gate-level implementation was already available. The optimization has then been performed through script.rugged followed by redundancy removal.

Table 1 shows our findings. The first 12 examples have been constructed as in [10]; the following 5 have been taken from [4]. Finally, the last 10 consist of the cascade of larger Mcnc'91 and Iscas'89 benchmarks; as far as we know, no results are available in the literature for the latter.

All the circuits have been optimized by iteratively applying procedure Optimize_Network followed by script.rugged. Up to 5 iterations were required to reach convergence.

Redundancy removal techniques which are more efficient than the one of SIS are available, e.g., [14]. Their use inside our optimization procedure may lead to better results.

For some examples run times are large. This is an intrinsic limitation of the method, since the optimization is fault-based. Notice, however, that the execution times in Table 1 refer to the overall optimization process; therefore, they include the time spent to run script.rugged, which may account for a large fraction of the total time (e.g., 410 seconds out of 1948 for circuit bbsse-planet).

In the rightmost columns of the table we compare our data to the ones reported in Table 1 of [10] (circuits from ex1-s510 to sand-styr) and in Table I of [4] (circuits from c1 to sd1). The comparison is made against the published results; some discrepancies may then exist between the reference circuits, due to different initial optimizations. We have improved the area results of Wang and Brayton for 8 circuits out of 12. Execution times in [10] were obtained on a machine, the DEC 3000/500 AXP, which is approximately 4 times faster than the Sparc-10. Then, we can conclude that our method is, on average, slower by a factor of two, but there are cases (e.g., planet-s510) where we are sensibly faster. Our area results are also better than the ones by Rho and Somenzi [4] (three wins and two ties). However, larger execution times – about one order of magnitude – were required.

Some better "absolute" results were reported in Table 2 of [10] for the benchmarks we have considered. However, it is important to notice that such results were obtained by applying re-encoding and re-synthesis to the circuits initially optimized through the algorithm summarized in Section 2.1. Since the objective of this section is evaluating the effectiveness of procedure Optimize_Network, rather than targeting "best ever" results on the benchmarks in Table 1, for the sake of fairness, we have chosen to compare our data to the ones reported in Table 1 of [10]. However, as future work, we are definitely interested in testing out the performance of re-encoding and re-synthesis when applied to the optimized FSM networks produced by our tool.

Theorem 2.1 says that the size (i.e., the number of states) of A_f is always smaller than or equal to the size of $M_2 \times M_2^F$. In general, a smaller set of states does not necessarily imply a smaller BDD for such a set. Also, a finite state structure with fewer states does not necessarily imply a smaller BDD for its transition relation [15]. However, in our case, we have experimentally observed that the use of A_f instead of $M_2 \times M_2^F$ is beneficial concerning both the aspects we have just mentioned. Table 2 presents the results. Since $M_2 \times M_2^F$ and A_f are fault-dependent, data have been averaged over all the considered faults. Savings, in terms of BDD nodes of the transition relation, are over one order of magnitude on all the examples.

4 Conclusions

We have presented a new algorithm for the optimization of the driven machine of a cascade FSM network which is based on sequential ATPG and redundancy removal. Experimental results have shown the effectiveness of the proposed technique on a large set of benchmarks.

Example	I	O	C	*Our Method*									*From the Literature*			
				S_1	IS_2	IL_2	F_2	RR_2	TR_2	FS_2	FL_2	*Time*	IS_2	IL_2	FS_2	FL_2
ex1-s510	9	7	19	18	47	290	522	219	303	14	43	258.7	47	248	12	37
ex7-dk16	2	3	2	4	27	213	314	174	140	17	68	65.1	27	348	16	75
s820-s510	18	7	19	24	47	290	489	66	423	10	43	423.5	47	248	16	34
s832-s510	18	7	19	24	47	290	522	143	379	4	12	133.2	47	248	5	15
bbsse-keyb	7	2	7	23	19	180	291	146	145	15	97	108.7	19	314	18	170
keyb-dk16	7	3	2	29	27	213	314	88	226	25	119	151.2	27	348	20	94
s510-keyb	19	2	7	47	19	180	399	123	276	19	82	212.9	19	314	16	93
sand-ex1	11	19	9	32	18	209	432	199	233	10	65	255.4	20	280	9	66
bbsse-planet	7	19	7	13	48	616	1027	702	325	44	440	1948.2	48	617	44	454
planet-s510	7	7	19	48	47	290	522	362	160	38	191	544.3	47	248	35	165
s510-planet	19	19	7	47	48	616	1027	266	761	43	428	4720.9	48	617	45	439
sand-styr	11	10	9	32	30	602	959	269	690	27	336	1759.4	30	596	27	375
c1	9	7	2	7	13	109	224	33	191	12	91	41.4	13	116	12	93
c2	9	7	19	20	48	616	1027	634	393	45	520	2079.5	48	606	44	651
c3	2	5	5	16	16	73	138	25	113	2	6	4.3	16	23	2	6
c1b	7	7	7	13	10	89	174	45	129	10	73	35.4	13	116	10	87
sd1	1	1	2	3	4	15	28	20	8	4	13	2.0	6	17	4	13
s1494-s510	8	7	19	48	47	290	522	92	430	4	11	152.0	–	–	–	–
s1488-s420	8	2	19	48	18	77	156	7	149	2	1	14.4	–	–	–	–
s298-ex4	3	9	6	218	14	71	156	81	75	8	37	41.3	–	–	–	–
s298-tbk	3	3	6	218	16	195	291	84	207	4	18	175.6	–	–	–	–
s208-dk16	11	3	2	18	27	213	314	177	137	17	72	132.5	–	–	–	–
tma-tbk	7	3	6	18	16	195	291	40	251	6	31	130.3	–	–	–	–
pma-s1494	8	19	8	24	48	655	1167	386	781	39	528	1891.1	–	–	–	–
pma-s1488	8	19	8	24	48	659	1161	390	771	45	512	1546.8	–	–	–	–
dk16-s298	2	6	3	27	218	2717	2278	808	1470	135	1336	30754.2	–	–	–	–
tbk-s298	6	6	3	16	218	2717	2278	769	1409	144	1463	57401.7	–	–	–	–

Table 1: Experimental Results.

I = # of primary inputs of $M_1 \rightarrow M_2$.
O = # of primary outputs of $M_1 \rightarrow M_2$.
C = # of communication signals of $M_1 \rightarrow M_2$.
S_1 = # of states of M_1.
IS_2 = # of states of the initial M_2.
IL_2 = # of literals (fact. form) of the initial M_2.
F_2 = # of faults in M_2.
RR_2 = # of faults removed by random patterns.
TR_2 = # of faults removed by targeted test sequences.
FS_2 = # of states of the optimized M_2.
FL_2 = # of literals (fact. form) of the optimized M_2.
Time = CPU time (in seconds on a SUN Sparc-10 with 64MB of RAM).

Example	$M_2 \times M_2^F$			A_f		
	NT	*S*	*NS*	*T*	*S*	*NS*
ex1-s510	8731	33	171	307	16	80
ex7-dk16	3487	33	93	418	30	84
s820-s510	14697	20	92	323	17	83
s832-s510	12934	44	193	303	17	72
bbsse-keyb	6322	24	87	341	19	65
keyb-dk16	3540	41	102	502	38	92
s510-keyb	6565	30	77	507	30	77
sand-ex1	3659	19	73	321	18	69
bbsse-planet	14137	45	160	688	39	140
planet-s510	11783	81	329	586	27	132
s510-planet	15476	52	185	821	47	166
sand-styr	4584	27	84	506	27	84
c1	1122	15	40	210	14	38
c2	15550	47	166	823	47	166
c3	209	10	51	104	5	26
c1b	1517	24	38	343	24	38
sd1	74	9	5	56	9	5
s1494-s510	12934	44	193	303	17	72
s1488-s420	1504	9	38	144	8	36
s298-ex4	493	14	54	127	8	33
s298-tbk	2910	22	44	467	20	35
s208-dk16	3048	41	93	485	40	89
tma-tbk	3139	26	44	550	25	38
pma-s1494	13675	55	195	866	47	165
pma-s1488	14364	44	158	817	44	158
dk16-s298	214818	1414	1094	11612	1088	842
tbk-s298	214695	1430	1060	13687	1374	816

Table 2: Comparison of $M_2 \times M_2^F$ to A_f.

NT = # of BDD nodes of the transition relation.
S = # of reachable states.
NS = # of BDD nodes of the set of reachable states.

Acknowledgments

Interesting discussions we had with Fabio Somenzi and Abelardo "Baron" Pardo are acknowledged.

References

[1] S. Unger, *Asynchronous Sequential Switching Circuits*, John Wiley, 1969.

[2] J. Kim, M. Newborn, "The Simplification of Sequential Machines with Input Restrictions," *IEEE TC*, 1972.

[3] S. Devadas, "Optimizing Interacting Finite State Machines Using Sequential Don't Cares," *IEEE TCAD*, 1991.

[4] J. Rho, F. Somenzi. "Don't Care Sequences and the Optimization of Interacting Finite State Machines," *IEEE TCAD*, 1994.

[5] H. Wang, R. Brayton, "Input Don't Care Sequences in FSM Networks," *ICCAD-93*.

[6] F. Ferrandi, F. Fummi, E. Macii, M. Poncino, D. Sciuto, "Test Generation for Networks of Interacting Finite State Machines Using Implicit Techniques," *GLSVLSI-96*.

[7] E. Sentovich, K. Singh, C. Moon, H. Savoij, R. Brayton, A. Sangiovanni-Vincentelli, "Sequential Circuits Design Using Synthesis and Optimization," *ICCD-92*.

[8] J. Hopcroft, J. Ullman, *Formal Languages and Their Relation to Automata*, Addison-Wesley, 1969.

[9] M. Rabin, D. Scott, "Finite Automata and Their Decision Problems," *IBM Journal of Research and Development*, 1959.

[10] H. Wang, R. Brayton, "Exploitation of Input Don't Care Sequences in Logic Optimization of FSM Networks," *ICCAD-95*.

[11] F. Ferrandi, F. Fummi, E. Macii, M. Poncino, D. Sciuto, *ATPG-Based Symbolic Optimization of FSM Networks*, Internal Report, Politecnico di Torino – DAI, 1995.

[12] S. Yang, "Logic Synthesis and Optimization Benchmarks User Guide Version 3.0," *MCNC Technical Report*, 1991.

[13] F. Brglez, D. Bryan, K. Koźmiński, "Combinational Profiles of Sequential Benchmark Circuits," *ISCAS-89*.

[14] L. Entrena, K. T. Cheng, "Sequential Logic Optimization by Redundancy Addition and Removal," *ICCAD-93*.

[15] K. Ravi, F. Somenzi, "High-Density Reachability Analysis," *ICCAD-95*.