Integrating Wrapper Design, TAM Assignment, and Test Scheduling for SOC Test Optimization

Haidar M. Harmanani and Rana Farah
Department of Computer Science and Mathematics
Lebanese American University
Byblos, 1401 2010, Lebanon

***Abstract*— Test time minimization for core-based designs is tightly integrated with wrapper design and TAM capacity. This paper presents a method to determine minimum SOC test schedules with wrapper design and TAM optimization based on simulated annealing. The method can handle SOC test scheduling with and without power constraints in addition to precedence constraints that preserve desirable orderings among tests. We present experimental results using the ITC 2002 benchmarks.**

I. Introduction

Advances in VLSI technology have made the reuse of embedded cores imperative in order to increase design productivity. Embedded cores are typically tested after integration with the integrator ensuring test access to the embedded cores using Test Access Mechanism (TAM) and wrappers [11]. A TAM performs the on-chip test pattern transport between the *source* and the *core* as well as between the *core* and the *sink* and is characterized by its transport capacity. The core is surrounded with a *test wrapper* that provides switching functionality between *normal* access and *test*. The test wrapper forms the interface between the embedded cores to the rest of the IC as well as to the TAM [1]. In order to test the cores, the internal scan-chains, the wrapper input cells and the wrapper output cells are connected in order to form *wrapper chains*. There is a clear trade-off between test time and TAM capacity since the length of the wrapper chains directly affects the core's test time while the number of wrapper chains affects the number of TAM bits. For example, if a single chain is used, the test time becomes long while if the scanned elements are connected into n chains, it is possible to load all n chains concurrently and thus reduce the test time. However, each wrapper chain's input and output must be connected to a TAM wire. Iyengar et al. [6] noted that increasing the number of TAM bits may not always guarantee decreasing test time as the test time may hit a Pareto optimal point. For all points with the same test time, a Pareto-optimal point is the point where the least number of TAM wires is used [5]. Another related problem is test time reduction by maximizing the simultaneous test of all cores. The problem, known as test scheduling, determines the order in which various cores are tested and has been shown to be $\mathcal{NP}$-complete. An effective test scheduling approach must minimize test time while addressing resource conflicts among cores arising from the use of shared TAMs, on-chip BIST engines and power dissipation constraints.

A. Related Work

The test scheduling problem has been addressed based on *fixed* and *flexible* width test bus architectures [10]. Iyengar et al. [5] first formulated the wrapper/TAM optimization problem using ILP, and proposed later several heuristic techniques based on rectangle packaging [6]. Huang et al. [3] modeled the problem as a restricted 3-D bin packing problem and proposed a heuristic to solve it while Goel et al. [2] proposed a heuristic for fixed-width architecture. Su et al. [9] formulated the problem using graph-based approach and solved it using tabu search. Zou et al. [12] proposed a test scheduling method based on simulated annealing. Recently, Pouget et al. [8] proposed a constraint-driven wrapper design and test scheduling approach.

B. Problem Description

This paper presents a method for integrated test time minimization, TAM assignment and wrapper design for SOCs using simulated annealing. Formally, given a SOC with N_C cores, a total TAM width $\mathcal{W}$, and a set of design and test constraints, the problem we address is to minimize the overall test time such that, (i) the test schedule for the entire SOC is efficient, (ii) the TAM wires are optimally partitioned and assigned to cores (iii) the wrapper configuration for each core is determined, and (iv) $\mathcal{W}$ is not exceeded. The remainder of the paper is organized as follows. Section II formulates the *SOC test scheduling problem* and describes the annealing configuration, the initial solution, and the neighborhood functions. Section III presents the SOC test scheduling algorithm. We conclude with *experimental results* in section IV.

II. Test Scheduling Algorithm

The test time of an SOC is based on the cores test times as well as on the determination of test start times. The cores test time is based on the TAM assignments as well as on the wrapper design. In this section, we describe the integration of the TAM assignment problem, the wrapper design problem, and the test scheduling problem using simulated annealing.

A. Configuration Representation

we map the problem to an initial configuration using a vector representation where a cell corresponds to a 3-D core as shown in Figure 1. The first dimension corresponds to *test start time*, S_i; the second dimension corresponds to TAM width, and the third dimension corresponds to the maximum power that a

978-1-4244-2332-3/08/$25.00 © 2008 IEEE

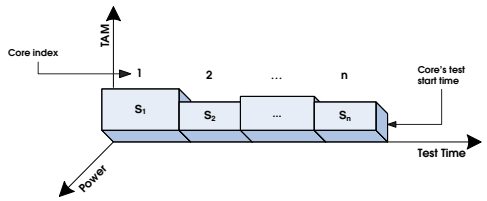

Fig. 1. Configuration representation

core may dissipate. The core *test finish time*, F_i, is equal to the test start time plus the core test time, T_i; that is, $F_i = T_i + S_i$. The test start time, S_i, is not constant and it changes to the end times of other cores.

B. Initial Configuration

The initial configuration is chosen by assigning each core one TAM bit and computing the individual cores test times as well as the theoretical test schedule lower bound, LB_T, which is equal to $\max\{\max(\tau_i), \sum_{i=1}^{n} \frac{\tau_i}{N_{tam}}\}$ for N_{tam} TAM width. The algorithm next iterates over all cores in order to ensure that all test times are lower or equal to LB_T by allocating additional TAM bits. The wrapper input cells, the core scan-chains, and the wrapper output cells are serially connected in order to form the minimum number of balanced wrapper chains and are adapted to the number of assigned TAM bits using the BFD algorithm. The test time for core i is determined based on $t_i = (1 + \max(s_i, s_o)) \times p + \min(s_i, s_o)$, where s_i and s_o denote respectively the scan-in and scan-out time for the core, and p_i denotes the number of test patterns applied to core i [7]. The initial configuration is next generated based on a serial schedule. However, when considering precedence constraints, the initial configuration is chosen based on the topological sorting of all tests represented in the precedence constraint graph. The result of the topological sorting is a valid serial solution that incorporates precedence constraints.

C. Neighborhood Functions

The algorithm explores test schedules, TAM assignment, and wrapper design using the following transformations:

1) *Test Schedule Exploration:* the algorithm randomly selects a random core i from the current configuration and changes its starting time S_i to the *end time* F_j of a randomly chosen core j ($i \neq j$), $0 \leq i, j \leq N_c$. If core j is selected such that $j = N_c$, then the start time of core i is set to 0.
2) *Test Schedule Swap:* the algorithm randomly selects two cores and swaps their test start times while considering precedence and concurrency constraints.
3) *TAM Exploration:* the algorithm finds the longest testing path in the schedule. For each core in the path, n TAM bits are added such that the core test time is lowered to the next Pareto-optimal point. The test time improvement, $\delta_t = t_{i+1} - t_i$, is computed and the algorithm accepts the change for the core that has the highest rate of change, $\frac{\delta_t}{n}$. If no change was affected, then algorithm repeats for other paths in the schedule.

```
Integrated_TestScheduling()
{
  MaxTime = Total allowed time for the annealing process
  LB_T ← lower bound test time.
  Assign one TAM bit to each core. Compute the Core test time, C_i.
  Design the Core Wrapper using the BFD algorithm.
  do {
      M = M_0;
      do{
        Violation ← True;
        Tries ← 0;
          while (Violation == True and tries < limit) {
              if (iteration % 10 == 0)
                  NewS = TestScheduleSwap(CurrentS);
              else if (iteration % 11 == 0)
                  NewS = TAMExplore(CurrentS);
              else if (iteration % 100 == 0)
              NewCost=TAMReduction(CurrentS);
              else
                  NewS = TestScheduleExplore(CurrentS);
                  check power, TAM, and precedence constraints
                  if there are violations then
                  Violation ← False;
                  tries++;
              }
        NewCost=Cost(NewS)
        Δ_Cost = NewCost − CurrentCost
        if (Δ_Cost < 0) {
          CurrentS=NewS
          CurrentCost=Cost(CurrentS);
          If (NewCost < BestCost) then
            BestS=NewS
            BestCost = Cost(BestS)
        }
        else if random < e^(−Δ_Cost/T) then      // T = temperature
          CurrentS=NewS
          CurrentCost = Cost(CurrentS);
        iteration++;
        M = M − 1
      } while (M ≥ 1)          // M is time until next parameter update
      Time = Time + M;
      T = α ∗T;                          // α = Cooling rate
      M = β ∗ M;                         // β = Iteration multiplier
  } while (Time < MaxTime);
  CompactSchedule(BestS);
  Return(BestS);
}
```

Fig. 3. Annealing SOC test scheduling algorithm

4) *TAM Reduction:* the algorithm randomly selects a core and reduces its TAM bits to the previous Pareto-optimal point; the core is placed at the end of the schedule.

The neighborhood transformations are followed by deterministic functions that compact the schedule horizontally in the test time direction as well as vertically in the TAM direction. The algorithms for both transformations are shown in Figures 4 and 5.

III. Test Scheduling Algorithm

Each annealing configuration represents a test schedule with the corresponding test adaptation but with a different cost. The test scheduling algorithm, shown in Figure 3, assigns initially one TAM bit to embedded cores and computes the individual test times. The algorithm ensures that the test time, t_i, for each core is lower than the lower bound, LB_T, by assigning the appropriate number of TAM bits. The test wrapper is next designed by configuring the scan elements into wrapper chains using the BFD algorithm. During every annealing iteration, the neighborhood of the configuration is explored using the

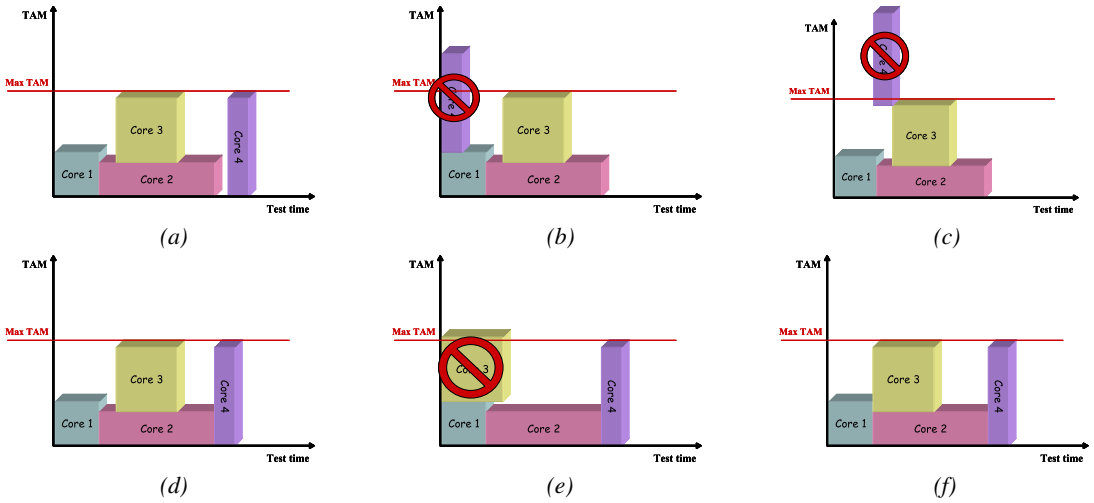

Fig. 2. Illustrating the test scheduling algorithm: (a) Current configuration; (b) Set $S_4 = 0$, causing a TAM violation; (c) Attempt to repair by setting $S_4 = F_1$, causing another TAM violation; (d) Successful repair by setting $S_4 = F_2$; (e) Compact vertically by setting $S_3 = 0$, causing another TAM violation; (f) Successful repair by setting $S_3 = F_1$

```
Compress_TimeDirection()
{
  SortList ← List of cores sorted by start time (in decreasing order).
  For each core_i ∈ CoreList,
    if S_i ≠ 0 and S_i ≠ F_j; j = 1, 2, ..., N_C
      if core_i does not violate any constraints
        S_i ← 0
      else ∀ k, k = 1, 2, ..., N_C
        Get the next core k
        if F_k < S_i
          S_i ← F_k
        else
          increment k
}
```

Fig. 4. Test time *compaction*

```
Compress_TAMDirection()
{
  SortList ← List of cores sorted by start time (in decreasing order).
  For each core_i ∈ SortList ,
    if there are still free TAM pins
      ∀  k, k = 1, 2, ..N_C
        if S_i < S_k and there is no TAM violation
          S_k = S_i
}
```

Fig. 5. TAM bits *compaction*

neighborhood functions in order to concurrently 1) minimize the cores test time; 2) minimize the number of TAM bits within the proper Pareto-optimal limits, and 3) avoid test conflicts. The neighborhood solutions use a constructive approach that leads to incremental feasible test schedules. The variation in the cost functions, Δ_C, is computed and if negative then the transition from C_i to C_{i+1} is accepted. If the cost function increases, the transition is accepted with a probability based on the Boltzmann distribution. The temperature is gradually decreased throughout the algorithm from a high starting value, $T_0 = 4000$, where almost every proposed transition, positive or negative, is accepted to a freezing temperature, $T_f = 0.001$, where no further changes occur.

IV. Experimental Results

We have implemented the proposed test scheduling algorithm and verified the approach using the ITC'02 benchmark suites. We compare our results to [2], [3], [4], [5], [6], [8], [9], [12]. The detailed results are shown in I, where it can be shown that our algorithm reported favorable test times in all attempted cases, and even found some new optimum test times in some examples. The simulated annealing cooling schedule was determined experimentally and is as follows: α = 0.99, β = 1.05, T_0 = 4000, M = 5.

Acknowledgment

This work was supported in part by a grant from the Lebanese National Council for Scientific Research (CNRS) and by the Lebanese American University.

References

[1] M. Bushnell and V. Agrawal. *Essentials of Electronic Testing for Digital, Memory & Mixed-Signal VLSI Circuits*. Kluwer, 2000.

[2] S.K. Goel and E.J. Marinissen. SOC Test Scheduling Design for Efficient Utilization of Bandwidth. *TODAES*, 8(4):399–429, 2003.

[3] Y. Huang, et al. Optimal Core Wrapper Width Selection and SOC Test Scheduling on 3-D Bin Packing Algorithm. *Proc. ITC*, pp. 74–82, 2002.

[4] V. Iyengar, K. Chakrabarty, and E. Marinissen. Efficient Wrapper/TAM Co-optimization for Large SoCs. In *Proc. DATE*, pp. 491–498, 2002.

[5] V. Iyengar et al. Test Wrapper and Test Access Mechanism Co-Optimization for System-On-a-Chip. *JETTA*, 18:213–230, 2002.

[6] V. Iyengar, K. Chakrabarty, and E. J. Marinissen. On Using Rectangle Packing for SOC Wrapper/TAM Co-Optimization. In *Proc. VTS*, pp. 253–258, 2002.

[7] E.J. Marinissen, S.K. Goel, and M. Lousberg. Wrapper Design for Embedded Core Test. In *Proc. ITC*, pp. 911–920, 2000.

[8] J. Pouget, E. Larsson, and Z. Peng. Multiple-Constraint Driven System-on-Chip Time Optimization. *JETTA*, pp. 599–611, 2005.

[9] C. Su and C. Wu. A Graph-Based Approach to Power-Constrained Test Scheduling. *JETTA*, 20:45–60, 2004.

[10] Q. Xu and N. Nicolici. Resource-Constrained System-On-a-Chip Test: A Survey. *IEE Proceedings*, 152(1):67–81, January 2005.

[11] Y. Zorian, E.J. Marinissen, and S. Dey. Testing Embedded Core-Based System Chips. In *Proc. ITC*, pp. 130–143, 1998.

[12] W. Zou, S. R. Reddy, I. Pomeranz, and Y. Huang. SOC Test Scheduling Using Simulated Annealing. In *Proc. VTS*, pp. 325–330, 2003.

TABLE I

ITC'02 BENCHMARK RESULTS

		Fixed-Width					Flexible-Width				
SOC	$\mathcal{W}_{max}$	ILP [5]	Par_eval [4]	TR-Architect [2]	Lagrange [12]	GRP [6]	3-D Packing [3]	Tabu [9]	SA_2 [12]	[8]	SA
a586710	16	–	–	41523868	–	–	42198943	42067708	32626782	–	32417445
	24	–	–	28716501	–	–	27785885	27907180	23413604	–	22973206
	32	–	–	22475033	–	–	21735586	22704821	18838663	–	17195389
	40	–	–	19048835	–	–	19041307	19041307	14260216	–	14249791
	48	–	–	15315476	–	–	15071730	15212440	12811087	–	12811087
	56	–	–	13415476	–	–	14945057	13401034	12573448	–	11486603
	64	–	–	12700205	–	–	12754584	11567464	10659014	–	9572169
u226	16	–	–	18663	–	–	13416	18663	13333	–	13333
	24	–	–	13331	–	–	10750	14745	8084	–	8084
	32	–	–	10665	–	–	6746	10665	6746	–	6746
	40	–	–	8084	–	–	5332	8084	5332	–	5332
	48	–	–	7999	–	–	5332	7999	5332	–	5332
	56	–	–	7999	–	–	4080	7999	4080	–	4080
	64	–	–	7999	–	–	4080	7999	4080	–	4080
f2126	16	–	–	372125	–	–	357109	357089	357088	–	357088
	24	–	–	335334	–	–	335334	335334	335334	–	335334
	32	–	–	335334	–	–	335334	335334	335334	–	335334
	40	–	–	335334	–	–	335334	335334	335334	–	335334
	48	–	–	335334	–	–	335334	335334	335334	–	335334
	56	–	–	335334	–	–	335334	335334	335334	–	335334
	64	–	–	335334	–	–	335334	335334	335334	–	335334
t512505	16	–	–	10530995	–	–	10531003	11210100	10530995	–	10530995
	24	–	–	10453470	–	–	10453470	10525823	10453470	–	10453470
	32	–	–	5268868	–	–	5268872	6370809	5268868	–	5268868
	40	–	–	5228420	–	–	5228420	5240493	5228420	–	5228420
	48	–	–	5228420	–	–	5228420	5239111	5228420	–	5228420
	56	–	–	5228420	–	–	5228420	5228474	5228420	–	5228420
	64	–	–	5228420	–	–	5228420	5228489	5228420	–	5228420
p34392	16	998733	1033210	1010821	1021510	1053491	1016640	995739	944768	-	940626
	24	720858	882182	680411	729864	759427	681745	690425	628602	-	637263
	32	591027	663193	551778	630934	544579	553713	544579	544579	-	544579
	40	544579	544579	544579	544579	544579	544579	544579	544579	-	544579
	48	544579	544579	544579	544579	544579	544579	544579	544579	-	544579
	56	544579	544579	544579	544579	544579	544579	544579	544579	-	544579
	64	544579	544579	544579	544579	544579	544579	544579	544579	-	544579
g1023	16	-	-	34459	-	-	31444	32602	31139	-	31777
	24	-	-	22821	-	-	21409	22005	21024	-	20261
	32	-	-	16855	-	-	16489	17422	15890	-	16332
	40	-	-	14794	-	-	14794	14794	14794	-	14794
	48	-	-	14794	-	-	14794	14794	14794	-	14794
	56	-	-	14794	-	-	14794	14794	14794	-	14794
	64	-	-	14794	-	-	14794	14794	14794	-	14794
p93791	16	1771720	1786200	1791638	1775586	1932331	1791860	1767248	1757452	1827819	1777840
	24	1187990	1209420	1185434	1198110	1310841	1200157	1178776	1169945	1220469	1204168
	32	887751	894342	912233	936081	988039	900798	906153	878493	945425	943087
	40	698583	741965	718005	734085	794027	719880	737624	718005	787588	760868
	48	599373	599373	601450	599373	669196	607955	608285	594575	639217	638993
	56	514688	514688	528925	514688	568436	521168	539800	509041	–	557890
	64	460328	473997	455738	472388	517958	549233	485031	447974	457862	489591
d281	16	–	–	8444	–	–	7948	8156	7946	–	7347
	24	–	–	6408	–	–	5486	5830	5485	–	4992
	32	–	–	5084	–	–	4070	4640	4070	–	3926
	40	–	–	3964	–	–	3926	3926	3926	–	3926
	48	–	–	3926	–	–	3926	3926	3926	–	3926
	56	–	–	3926	–	–	3926	3926	3926	–	3926
	64	–	–	3926	–	–	3926	3926	3926	–	3926
d695	16	42568	42644	44307	–	44545	42716	41905	41604	41847	42382
	24	28292	30032	28576	–	31569	28639	28231	28064	29106	29103
	32	21566	22268	21518	–	23306	21389	21467	21161	20512	21456
	40	17901	18448	17617	–	18837	17366	17308	16993	18691	17480
	48	16975	15300	14608	–	16984	15142	14643	14182	17257	14779
	56	13207	12491	12462	–	14974	13208	12493	12085	–	12708
	64	12941	12941	11033	–	11984	11279	11036	10723	13348	11069
p22810	16	462210	468011	458068	434922	489192	446684	465162	438619	473418	464809
	24	361571	313607	299718	313607	330016	300723	317761	289287	352834	310456
	32	312659	246332	222471	245622	245718	223462	236796	218855	236186	233101
	40	278359	232049	190995	194193	199558	184951	193696	175946	195733	197566
	48	278359	232049	160221	164755	173705	167858	174491	147944	159994	166169
	56	268472	153990	145417	145417	157159	145087	155730	126947	–	145417
	64	260638	153990	133405	133628	142342	128512	145417	109591	128332	123543