Efficient Wrapper/TAM Co-Optimization for Large SOCs

Vikram Iyengar[†], Krishnendu Chakrabarty[†] and Erik Jan Marinissen[‡]

[†] Department of Electrical & Computer Engineering
Duke University, Durham, NC 27708, USA
{vik,krish}@ee.duke.edu

[‡] Philips Research Laboratories
5656 AA Eindhoven, The Netherlands
erik.jan.marinissen@philips.com

Abstract

Core test wrappers and test access mechanisms (TAMs) are important components of a system-on-chip (SOC) test architecture. Wrapper/TAM co-optimization is necessary to minimize the SOC testing time. Most prior research in wrapper/TAM design has addressed wrapper design and TAM optimization as separate problems, thereby leading to results that are sub-optimal. We present a fast heuristic technique for wrapper/TAM co-optimization, and demonstrate its scalability for several industrial SOCs. This extends recent work on exact methods for wrapper/TAM co-optimization based on integer linear programming and exhaustive enumeration. We show that the SOC testing times obtained using the new heuristic algorithm are comparable to the testing times obtained using exact methods. Moreover, more than two orders of magnitude reduction can be obtained in the CPU time compared to exact methods. Furthermore, we are now able to design efficient test access architectures with a larger number of TAMs.

1 Introduction

The general problem of system-on-chip (SOC) test integration includes the design and optimization of wrapper/TAM architectures and test scheduling. Test wrappers form the interface between cores and TAMs, and TAMs transport test data between SOC pins and test wrappers [15]. Test scheduling determines the order in which tests are applied. We focus here on wrapper/TAM co-design to minimize testing time under TAM width constraints. Wrapper/TAM design is challenging because (i) wrapper and TAM optimization must be carried out in conjunction [8], (ii) TAMs must be designed to minimize testing time under the constraint of limited chip I/Os available for testing, and (iii) wrapper/TAM co-optimization techniques must be scalable for industrial SOCs containing not only a large number of cores with hundreds of I/O terminals and scan chains, but also a large number of TAMs.

Most prior research has either studied wrapper design and TAM optimization as independent problems [1, 4, 5, 12], or not addressed the issue of sizing the TAMs to minimize SOC testing time [14]. Alternative approaches that combine TAM design with test scheduling [9, 13] do not address the problem of wrapper design and its relationship to TAM optimization. New techniques for wrapper/TAM co-optimization are therefore needed to minimize testing time under TAM width constraints. Such techniques should be scalable for SOCs that employ a large number of TAMs.

The first integrated method for wrapper/TAM co-optimization was proposed in [8]. TAM optimization was carried out by enumerating over the different partitions of TAM width as well as over the number of TAMs on the SOC. Integer linear programming (ILP) was used to calculate the optimal core assignment and resulting testing time for each partition. A drawback of this approach is that the wrapper/TAM designs considered in [8] are limited to a small number of TAMs in order to maintain feasible compute times. However, if the total number of TAM wires on the SOC is large, the testing time can often be reduced by increasing the number of TAMs. This is because of two reasons. Firstly, when there are multiple TAMs of different widths, a larger number of cores can be assigned to TAMs whose widths match the cores' own test data requirements; thus the number of unnecessary (idle) TAM wires assigned to cores is reduced. Secondly, multiple TAMs provide greater test parallelism, thereby decreasing total testing time. The methods in [8] are therefore inadequate for large industrial SOCs.

In [8], four problems structured in order of increasing complexity were formulated, such that they serve as stepping stones to the problem of wrapper/TAM co-optimization for SOCs. We first review these four problems.

1. $\mathcal{P}_{\mathbf{W}}$: Design a *wrapper* for a given core, such that the core testing time is minimized, and the TAM width required for the core is minimized.
2. $\mathcal{P}_{\mathbf{AW}}$: Determine (i) an *assignment* of cores to TAMs of given widths, and (ii) a *wrapper* design for each core such that SOC testing time is minimized. (Item (ii) corresponds to $\mathcal{P}_{\mathbf{W}}$.)
3. $\mathcal{P}_{\mathbf{PAW}}$: Determine (i) a *partition* of the total TAM width among the given number of TAMs, (ii) an *assignment* of cores to the TAMs, and (iii) a *wrapper* design for each core such that SOC testing time is minimized. (Items (ii) and (iii) together correspond to $\mathcal{P}_{\mathbf{AW}}$.)

These three problems lead up to $\mathcal{P}_{\mathbf{NPAW}}$, the more general problem, described as follows.

4. $\mathcal{P}_{\mathbf{NPAW}}$: Determine (i) the *number* of TAMs for the SOC, (ii) a *partition* of the total TAM width among the TAMs, (iii) an *assignment* of cores to TAMs, and (iv) a *wrapper* design for each core, such that SOC testing time is minimized. (Items (ii), (iii) and (iv) together correspond to $\mathcal{P}_{\mathbf{PAW}}$.)

The above four problems are all $\mathcal{NP}$-hard [8]. Therefore, efficient heuristics are needed for large problem instances.

In this paper, we present heuristics to effectively solve the wrapper/TAM co-optimization problems reviewed above for large SOCs containing multiple TAMs. As in [8], we use the test bus model for TAMs. To solve $\mathcal{P}_{\mathbf{W}}$, we use the *Design_wrapper* algorithm presented in [8]. We then describe a new algorithm to solve problem $\mathcal{P}_{\mathbf{AW}}$ efficiently. The solution to $\mathcal{P}_{\mathbf{AW}}$ thus obtained is a good approximation of the optimal solution for a given TAM width partition. An efficient technique for TAM partition enumeration using solution-space pruning is used to obtain the TAM width partition and number of TAMs with the lowest testing time (problems $\mathcal{P}_{\mathbf{PAW}}$ and $\mathcal{P}_{\mathbf{NPAW}}$). This yields an intermediate solution to $\mathcal{P}_{\mathbf{NPAW}}$. In the final step, an exact mathematical programming model is used to optimize the final assignment of cores and the SOC testing time. This two-step approach allows us to apply our methods to design effective wrapper/TAM architectures for large industrial SOCs. We show

[1] This research was supported in part by the National Science Foundation under grant number CCR-9875324 and by an IBM Graduate Fellowship.

that while the SOC testing times obtained using the new heuristic algorithm are comparable to the testing times obtained using exact methods, over two orders of magnitude reduction is obtained in the CPU time needed. This is especially important since in some cases, the minimum SOC testing time is obtained for a larger number of TAMs, which we could not compute earlier in a reasonable amount of execution time.

2 New algorithm for core assignment

The first problem $\mathcal{P}_{\mathbf{W}}$ is that of designing an optimal wrapper for the I/O terminals and internal scan chains of a core, such that the core testing time is minimized. To solve $\mathcal{P}_{\mathbf{W}}$, we use an algorithm based on the Best Fit Decreasing (BFD) heuristic for the Bin Packing problem [6]. Our *Design_wrapper* algorithm (proposed earlier in [8]) has two priorities: (i) minimizing core testing time, and (ii) minimizing the TAM width required for the test wrapper. These priorities are achieved by balancing the lengths of the wrapper scan chains designed, and identifying the number of wrapper scan chains that actually need to be created to minimize testing time. Priority (ii) is addressed by the algorithm since it has a built-in reluctance to create a new wrapper scan chain, while assigning core-internal scan chains to the existing wrapper scan chains.

The second problem $\mathcal{P}_{\mathbf{AW}}$ is that of assigning cores to TAMs of given widths. An ILP model was developed to solve $\mathcal{P}_{\mathbf{AW}}$ exactly in [8]. The CPU time for this ILP model was reasonably short for a single execution and optimal solutions for the core assignment problem were easily obtained. However, $\mathcal{P}_{\mathbf{AW}}$ is an $\mathcal{NP}$-hard problem, and execution times can get high for problem instances larger than those encountered in [8]. Furthermore, solutions to the problems $\mathcal{P}_{\mathbf{PAW}}$ and $\mathcal{P}_{\mathbf{NPAW}}$ were obtained by enumerating optimal ILP solutions to $\mathcal{P}_{\mathbf{AW}}$ for each TAM width partition on the SOC. As a result, CPU times for $\mathcal{P}_{\mathbf{PAW}}$ and $\mathcal{P}_{\mathbf{NPAW}}$ were found to be prohibitively large, especially for an industrial SOC. Moreover, during execution of the ILP model of $\mathcal{P}_{\mathbf{AW}}$, values of SOC testing time $\mathcal{T}$ are calculated only at the end of each ILP iteration. Therefore, execution of the ILP model cannot be halted prematurely if it is discovered that the testing time of a TAM has already exceeded the best-known value $\mathcal{T}$ calculated earlier. Therefore, a faster algorithm for $\mathcal{P}_{\mathbf{AW}}$ that produces efficient results in polynomial time is clearly needed.

Here, we present a new heuristic algorithm for $\mathcal{P}_{\mathbf{AW}}$ based on the relationship between TAM width and core testing times calculated for $\mathcal{P}_{\mathbf{W}}$. The testing time of Core i when connected to a TAM j of width w_j is denoted by $T_i(w_j)$. We design the heuristic for $\mathcal{P}_{\mathbf{AW}}$, based on an approximation algorithm for the problem of scheduling n independent jobs (tests) on m parallel, equal processors (TAMs) [3]. The pseudocode for our heuristic *Core_assign* is presented in Figure 1. Intuitively, in each iteration the algorithm calculates the summed testing time on each TAM by adding up the testing times of all the cores assigned to that TAM. Then the core with the largest testing time (among all unassigned cores) is assigned to the TAM with the shortest current summed testing time. Furthermore, during core assignment, if the time T_r on any TAM r exceeds the best-known value $\mathcal{T}$ computed earlier, the algorithm returns $\mathcal{T}$ and halts. This plays a significant role in reducing computation when *Core_assign* is executed a large number of times, as will be shown in Section 3.

We illustrate the *Core_assign* algorithm using an example SOC containing five cores and three TAMs. The testing times for the five cores when assigned to the TAMs of widths 8, 16, and 32 are shown in Figure 2 (a). Initially, the testing time on all TAMs is 0 cycles;

Procedure *Core_assign*(**B**,**C**,$\mathcal{T}$)

```
1 Let C be the set of cores;
2 Let B be the set of TAMs;
3 Let T be the best-known testing time for (B,C);
4 For each core i ∈ C{
5   For each TAM j ∈ B{
6     Find T_i(w_j) using Design_wrapper;}}
7 For each TAM j ∈ B{
8   Set testing time T_j on TAM j to 0;}
9 While C ≠ ∅{
10  Select TAM j ∈ B, such that T_j is minimum;
11    If there are two or more such TAMs{
12      Select TAM j, such that w_j is maximum;}
13  Select Core i ∈ C, such that T_i(w_j) is maximum;
14    If there are two or more such cores{
15      Select TAM k ∈ B, such that (w_k < w_j AND w_k is maximum);
16      Select Core i, such that T_i(w_k) is maximum;}
17  Assign Core i to TAM j;
18  Determine TAM r ∈ B, such that T_r is maximum;
19  If T_r ≥ T{
20    Return SOC testing time T;}
21  C = C - {i};}
22 Return SOC testing time T_r;
```

Figure 1. New algorithm for core assignment.

Cores	Testing time (cycles) TAM 1 32 bits	TAM 2 16 bits	TAM 3 8 bits
1	50	100	200
2	75	95	200
3	90	100	150
4	60	75	80
5	120	120	125

(a)

Core	TAM	Testing time (cycles)
1	2	100
2	3	200
3	2	100
4	1	60
5	1	120

(b)

Figure 2. Core testing times for (a) the SOC used to illustrate *Core_assign*, and (b) the final assignment.

therefore TAM 1 of width 32, being the widest, is considered first. Core 5 has the highest testing time on TAM 1, therefore Core 5 is assigned to TAM 1. Next, there is a choice between Cores 1 and 3 to be assigned to TAM 2 of width 16. We choose to assign Core 1 to TAM 2 here because the testing time for Core 1 on TAM 3 is higher than the testing time for Core 3 on TAM 3 (Line 14 of *Core_assign*). Next Core 2 is assigned to TAM 3. TAM 2 is now the minimally loaded TAM; therefore, Core 3 is assigned to TAM 2. Finally, Core 4 is assigned to TAM 1. Figure 2 (b) presents the final assignment of cores to TAMs. The testing times on TAMs 1, 2, and 3 are 180, 200, and 200 clock cycles, respectively. The complexity of *Core_assign* is $\mathcal{O}(N^2)$, where N is the number of cores in the SOC. *Core_assign* executes two orders of magnitude faster than the ILP model in [8]; hence, a significantly larger number of *Core_assign* iterations can be executed in the time taken to execute the ILP model.

3 TAM width partitioning

In this section, we describe how the *Core_assign* heuristic is used to develop an algorithm to quickly reach an intermediate solution to $\mathcal{P}_{\mathbf{PAW}}$ and $\mathcal{P}_{\mathbf{NPAW}}$. We also demonstrate how, once an approximate solution for $\mathcal{P}_{\mathbf{NPAW}}$ has been reached, an exact mathematical programming model for $\mathcal{P}_{\mathbf{AW}}$ can be executed to perform a final optimization of the core assignment, thus achieving near-optimal results with little computation time.

3.1 Fast algorithm for $\mathcal{P}_{\mathbf{NPAW}}$

In [8], it was shown that the $\mathcal{P}_{\mathbf{AW}}$ ILP model takes a relatively small time to execute for problem instances of resonable size. This can be exploited to execute the model for each unique TAM width partition and record the partition and core assignment with the best testing time. Solutions to $\mathcal{P}_{\mathbf{PAW}}$ and $\mathcal{P}_{\mathbf{NPAW}}$ can thus be obtained. This method is applicable because the number of unique partitions of TAM width is relatively small for a small number of TAMs. The number of unique partitions $p_B(W)$ for a given total TAM width W, and a given number B of TAMs can be estimated using partition theory in combinatorial mathematics [10]. In [10], $p_B(W)$ is shown to be approximately $\frac{W^{B-1}}{B!(B-1)!}$ for $W \mapsto \infty$. For $B = 2$, $p_2(W) = \left\lfloor \frac{W}{2} \right\rfloor$. For $B = 3$, the number of partitions can be shown to be $\sum_{i=0}^{\left\lfloor \frac{W}{3} \right\rfloor} \left\lfloor \frac{W-(3i+1)}{2} \right\rfloor$ [8]. From this formula, $p_3(64) = 341$. Therefore, the execution time for SOCs having three TAMs is reasonable, even for large W. The challenge to effective partition enumeration lies in the fact that for $B \geq 3$, there is no simple and systematic method to enumerate only the unique partitions. In fact, no exact formula is available to calculate the total number of unique partitions for a given value of W and B. The value of $p_B(W)$ can thus only be approximated assuming $W \mapsto \infty$. One way to ensure that only unique partitions are evaluated is to discard, prior to evaluation, each new partition that appears to be a cyclical isomorphism of a previously handled partition. However, the memory requirements for this method and the number of partition comparisons required to be performed grow exponentially with B and severely limit the scalability for large B. Furthermore, as B increases, the time required to enumerate unique partitions and evaluate them using ILP increases significantly. This method is therefore inadequate for industrial SOCs having multiple TAMs.

In this subsection, we use *Core_assign* to develop a fast method to evaluate width partitions; this effectively addresses the problems inherent to the ILP model and "enumeration-comparison" method described above. The new heuristic employs extensive solution-space pruning, and is thus applicable to wrapper/TAM design for industrial SOCs having a large number of TAMs. In our experiments with industrial SOCs, we were able to evaluate width partitions and testing times for wrapper/TAM architectures having upto ten TAMs within a few minutes. Test access architectures having more than ten TAMs could also be evaluated, but were found to be less useful for testing time minimization because testing time increases significantly as the relative width of each TAM decreases beyond a threshold.

The new algorithm *Partition_evaluate* for problems $\mathcal{P}_{\mathbf{PAW}}$ and $\mathcal{P}_{\mathbf{NPAW}}$ is presented in Figure 3. This algorithm employs three levels of solution-space pruning. Firstly, the number of partitions enumerated is significantly limited by the restriction in Line 1 of the recursive function *Increment*. To enumerate partitions of W over B TAMs, *Increment* dynamically creates B nested loops with loop variables $w_1 \ldots w_B$. Without the restriction in Line 1, enumeration would be as follows: $\{w_1 + \ldots + w_B\} = \{1 + \ldots + 1 + (W - B + 1)\}$, $\{1 + \ldots + 2 + (W - B)\}$, ..., $\{(W - B + 1) + 1 + \ldots + 1\}$. However, a sizeable number of repeated partitions is prevented by establishing an upper bound $\left\lfloor \frac{W - \sum_{k=0}^{j-1} w_k}{B-(j-1)} \right\rfloor$ on each variable w_j during enumeration. For example, for $W = 8, B = 4$, the first three partitions enumerated are $\{1+1+1+5\}$, $\{1+1+2+4\}$, and $\{1+1+3+3\}$, respectively. If the restriction of Line 1 were not present, the repeated partition $1 + 3 + 1 + 3$ would also subsequently be enumerated.

Procedure *Partition_evaluate*(W)

```
1 Let W = total TAM width;
2 Let C = set of cores;
3 Let B_max = upper limit on number of TAMs;
4 For B = 1 to B_max {
5   Let set of TAMs B = {w_1, w_2, ..., w_B};
6   Set SOC testing time T = ∞;
7   For TAM j = 1 to (B − 2), set w_j = 1;
8   Set w_{B−1} = 0; Set flag = 0;
9   While flag ≠ 1{
10    Increment(B, B − 1, W);
11    New SOC testing time T_new = Core_assign(B,C,T);
12    If T_new < T{
13      Set T = T_new; Set B_best = B;}
14 Output B_best, T;
```

Procedure *Increment*(**B**, j, W)

```
1 If w_j < ⌊ (W − Σ_{k=0}^{j−1} w_k) / (B − (j − 1)) ⌋ {
2   Set w_j = w_j + 1;
3   Set w_B = W − Σ_{k=1}^{B−1} w_k;
4   Return;}
5 Else{
6   If j = 1{
7     Set flag = 1; Return;}
8   Else Increment(B, j − 1, W);}
```

Figure 3. Fast algorithm for partition evaluation.

However, Line 1 establishes an upper bound of 2 on w_2; thus, partition $1 + 3 + 1 + 3$ is not enumerated. Secondly, Lines 18 to 20 in *Core_assign* terminate the evaluation of any partition for which the testing time T_r of any TAM r has already exceeded the best-known value of testing time $\mathcal{T}$ calculated previously. The number of such partitions, whose evaluation can be terminated was found to be large in our experiments; therefore the execution time of *Partition_evaluate* is reduced significantly. Finally, *Partition_evaluate* uses heuristic algorithm *Core_assign* to evaluate partitions. This $\mathcal{O}(N^2)$ algorithm significantly reduces the computation performed.

Table 1 presents some experimental data on the efficiency of *Partition_evaluate* for partition-space pruning. The number of possible unique partitions $p_B(W)$ (estimated using $p_B(W) = \frac{W^{B-1}}{B!(B-1)!}$) is presented for several values of W and B. We present results for $W \geq 44$, because the formula $p_B(W) = \frac{W^{B-1}}{B!(B-1)!}$ is accurate only for larger values of W. The number of partitions P_{eval} that are actually evaluated to completion (pruned by Line 1 of the *Increment* function and by lines 18 to 20 of *Core_assign*) is presented for an example SOC p21241 from Philips. (The relevant details of this SOC are presented in Section 4.) Finally, the efficiency η of our heuristics is calculated using $\eta = \frac{P_{eval}}{p_B}$. Here, $\eta = 0.01$ implies that approximately 1% of the number of unique partitions are evaluated to completion by *Partition_evaluate*. We choose p21241 to illustrate the efficiency of our heuristics, because the exhaustive method [8] was found to be inadequate for wrapper/TAM co-design for p21241; the method did not complete even for $B = 3$. From Table 1, it can be seen that *Partition_evaluate* evaluates on average only 2% of the unique partitions. Thus there is a significant reduction in the execution time using this heuristic compared to the exhaustive method.

W	$B=6$			$B=8$		
	$p_B(W)$	P_{eval}	η	$p_B(W)$	P_{eval}	η
44	1909	46	0.02	1571	170	0.1
48	2949	46	0.02	2889	48	0.02
52	4401	65	0.01	5059	100	0.02
56	6374	111	0.02	8499	110	0.01
60	9000	278	0.03	13776	172	0.01
64	12428	708	0.06	21643	256	0.01

Table 1. Efficiency of the *Partition_evaluate* heuristic.

3.2 Final optimization step

Partition_evaluate provides a fast approximation of the optimal values of TAM width partition and testing time. We further improve on this result by performing a final optimization step using the ILP model for $\mathcal{P}_{\mathbf{AW}}$ [8]. Since this final step is performed only once, and since the execution time for a single iteration of the ILP model for $\mathcal{P}_{\mathbf{AW}}$ is relatively small, this results in a near-optimal solution to $\mathcal{P}_{\mathbf{NPAW}}$ in a short execution time. Here, we repeat the ILP model for $\mathcal{P}_{\mathbf{AW}}$ from [8] for reasons of completeness, and to comment on its complexity.

To model $\mathcal{P}_{\mathbf{AW}}$, consider an SOC consisting of N cores and B TAMs of widths $w_1, w_2, \ldots, w_B$. The time taken to test Core i assigned to TAM j, given by $T_i(w_j)$ clock cycles, is calculated using *Design_wrapper*. We introduce binary variables x_{ij} (where $1 \leq i \leq N$ and $1 \leq j \leq B$), which are used to determine the assignment of cores to TAMs in the SOC. Let x_{ij} be a 0-1 variable defined as follows:

$$x_{ij} = \begin{cases} 1, & \text{if Core } i \text{ is assigned to TAM } j \\ 0, & \text{otherwise} \end{cases}$$

The time needed to test all cores on TAM j is given by $\sum_{i=1}^{N} T_i(w_j) \cdot x_{ij}$. Since all the TAMs can be used simultaneously for testing, the system testing time equals $\max_{1 \leq j \leq B} \sum_{i=1}^{N} T_i(w_j) \cdot x_{ij}$. The ILP model for $\mathcal{P}_{\mathbf{AW}}$ can be formulated as follows.

Objective: Minimize testing time $\mathcal{T}$, subject to

1. $\mathcal{T} \geq \sum_{i=1}^{N} T_i(w_j) \cdot x_{ij}, 1 \leq j \leq B$, i.e., $\mathcal{T}$ is the maximum testing time on any TAM
2. $\sum_{j=1}^{B} x_{ij} = 1, 1 \leq i \leq N$, i.e., every core is assigned to exactly one TAM

The number of variables and constraints for this model (a measure of its complexity) is given by $N \cdot B$, which is $\mathcal{O}(N^2)$, and $N+B$, which is $\mathcal{O}(N)$, respectively. This ILP model uses the best width partition obtained from *Partition_evaluate* to optimize the core assignment and obtain a near-optimal wrapper/TAM architecture in the final step of our co-optimization methodology.

4 Experimental results

In this section, we present experimental results on our wrapper/TAM co-optimization methodology for four example SOCs. The first, d695, is an academic benchmark SOC from Duke University. The other three SOCs p93791, p21241, and p31108 are from Philips. The number (e.g., 93791) in each SOC name is a measure of its test complexity. We calculate the SOC test complexity number using the formula presented in [8].

The experimental results presented in this paper were obtained at Duke University using a Sun Ultra 10 with a 333 MHz processor and 256 MB memory. The results in [8] were obtained at Philips Research Laboratories using a Sun Ultra 80 with a 450 MHz processor and 4096 MB memory. For the problems in this paper, we found that the Sun Ultra 80 leads to five times faster execution compared to the Sun Ultra 10. Therefore, the CPU times reported in [8] have been multiplied by a factor of five to facilitate a comparison with the CPU times reported here. Note that we achieve an order of magnitude improvement in CPU time over [8] even without the 5× adjustment factor. Note also that all SOC testing times in this section are expressed in clock cycles.

4.1 Results for SOC d695

In this subsection, we present experimental results for SOC d695. SOC d695 consists of two ISCAS'85 and eight ISCAS'89 benchmark circuits [8].

W	Results in [8] for $B=2$: w_1+w_2	Core assignment	Testing time $\mathcal{T}_{old}$ (cycles)	Exec. time $\mathcal{E}_{old}$ (sec)
16	6+10	(1,2,1,1,2,2,1,1,2,1)	45055	5
24	6+18	(2,1,1,1,2,2,1,1,1,2)	29501	5
32	11+21	(1,2,1,1,2,2,2,1,1,1)	25442	5
40	8+32	(2,1,1,1,2,2,1,1,2,2)	21359	10
48	16+32	(2,1,1,1,2,1,2,2,2,2)	19938	10
56	19+37	(1,2,1,1,2,1,2,2,1,2)	18434	10
64	20+44	(1,2,1,1,2,1,2,2,1,2)	18205	15

(a)

W	New co-optimization method for $B=2$: w_1+w_2	Core assignment	$\mathcal{T}_{new}$ (cycles)	$\mathcal{E}_{new}$ (sec)	$\Delta\mathcal{T}$ (%)	Ratio $\frac{\mathcal{E}_{new}}{\mathcal{E}_{old}}$
16	8+8	(2,1,2,1,1,2,1,2,1,2)	45055	1	+0.00	0.2
24	12+12	(2,2,1,1,1,2,1,1,2,2)	34455	1	+16.79	0.2
32	16+16	(2,1,2,1,2,1,1,2,1,2)	25828	1	+1.52	0.2
40	20+20	(2,1,1,1,1,2,1,2,1,2)	22848	1	+6.97	0.1
48	20+28	(1,2,1,2,1,1,2,2,1,2)	22804	1	+14.37	0.1
56	23+33	(2,2,2,2,2,1,1,1,2,2)	18940	1	+2.74	0.1
64	32+32	(1,1,1,2,1,1,2,2,1,2)	18869	1	+3.65	0.08

(b)

W	Results in [8] for $B=3$: $w_1+w_2+w_3$	Core assignment	$\mathcal{T}_{old}$ (cycles)	$\mathcal{E}_{old}$ (sec)
16	3+5+8	(2,2,1,1,2,3,1,1,3,3)	42568	20
24	2+5+17	(2,2,2,1,3,3,3,1,3,2)	28292	50
32	4+10+18	(1,2,2,1,2,3,3,1,1,3)	21566	80
40	4+17+19	(2,1,2,1,2,3,2,1,2,3)	17901	120
48	4+19+25	(3,3,3,2,3,2,3,1,3,1)	16975	200
56	5+18+33	(3,2,1,1,3,2,3,1,2,3)	13207	265
64	5+17+42	(2,2,2,1,3,2,3,1,2,3)	12941	420

(c)

W	New co-optimization method for $B=3$: $w_1+w_2+w_3$	Core assignment	$\mathcal{T}_{new}$ (cycles)	$\mathcal{E}_{new}$ (sec)	$\Delta\mathcal{T}$ (%)	Ratio $\frac{\mathcal{E}_{new}}{\mathcal{E}_{old}}$
16	5+5+6	(1,1,1,2,1,3,3,2,2,2)	42952	1	+0.9	0.06
24	8+8+8	(3,1,3,2,3,1,2,3,1,2)	30032	1	+6.15	0.03
32	8+12+12	(2,2,2,2,2,1,3,3,3,3)	24851	1	+15.23	0.02
40	7+16+17	(2,2,2,1,2,3,2,1,1,3)	18448	1	+3.06	0.01
48	16+16+16	(3,2,3,2,3,1,3,1,2,2)	17581	1	+3.57	0.01
56	17+19+20	(2,2,2,1,3,2,2,3,1,1)	15510	1	+17.44	0.004
64	18+20+26	(2,2,1,1,2,3,2,3,1,1)	15442	1	+19.33	0.003

(d)

Table 2. Results for d695 (Problem $\mathcal{P}_{\mathbf{PAW}}$).

Table 2 compares the results obtained in [8] with the results of the new wrapper/TAM co-optimization method for d695 for $B=2$ and $B=3$ (Problem $\mathcal{P}_{\mathbf{PAW}}$). The testing times $\mathcal{T}_{new}$ of the new method are comparable to the testing times $\mathcal{T}_{old}$ in [8]. However, the CPU times $\mathcal{E}_{new}$ of the new method are at least an order of magnitude less than the CPU times $\mathcal{E}_{old}$ in [8] for larger values of W. The core assignment vector follows the notation introduced in [5] and further used in [8]. Each position in the vector refers to the core number and the entry in each position refers to the TAM to which the corresponding core is assigned. The percentage change in testing time using the new method is calculated using the formula $\Delta\mathcal{T}$ (%) $= \frac{\mathcal{T}_{new} - \mathcal{T}_{old}}{\mathcal{T}_{old}} \times 100$.

W	New co-optimization method: B	TAM partition	Core assignment	$\mathcal{T}_{new}$ (cycles)	$\mathcal{E}_{new}$ (sec)	$\Delta\mathcal{T}$ (%)	Ratio $\frac{\mathcal{E}_{new}}{\mathcal{E}_{old}}$
16	4	3+3+ 5+5	(3,3,4,1,3, 4,1,4,1,2)	42644	1	+0.18	0.06
24	3	8+ 8+8	(3,1,3,2,3, 1,2,3,1,2)	30032	1	+6.15	0.03
32	4	7+6+ 9+10	(3,4,3,2,4, 3,2,2,1,1)	22268	1	+3.26	0.02
40	3	7+16+ 17	(2,2,2,1,2, 3,2,1,1,3)	18448	1	+3.06	0.01
48	5	5+2+8+ 16+17	(3,4,3,2,4, 5,5,1,4,3)	15300	2	-9.86	0.01
56	5	5+8+11+ 16+16	(5,3,5,1,5, 4,2,1,2,3)	12941	2	-2.01	0.009
64	6	5+3+8+ 12+18+18	(4,3,4,1,6, 5,3,1,2,4)	12941	7	+0.00	0.02

Table 3. New results for d695 ($\mathcal{P}_{\mathbf{NPAW}}$).

While the methods of [8] are limited to $B \leq 3$ due to high computation cost, the heuristic proposed in this paper can evaluate more efficient TAM designs with higher values of B. Table 3 presents the results obtained with the new wrapper/TAM co-optimization methodology for d695 over a larger number of TAMs (Problem $\mathcal{P}_{\mathbf{NPAW}}$, $B \leq 10$). The testing times and CPU times in [8] have already been presented in Table 2. The best results in [8] were obtained for $B = 3$. The testing times obtained using the new co-optimization technique are better than or equal to the best testing times in [8] for larger values of W. For $W \leq 40$, the new testing times are on average only 3% larger than those reported in [8]. We improve upon the testing times compared to [8] for larger values of W because the exhaustive approach of [8] did not terminate within a reasonable CPU time for these values of W for larger B. Only the best results for $B \leq 3$ were reported in [8]. Since the exhaustive method did not terminate within a reasonable CPU time for $B \geq 3$, the ratio $\frac{\mathcal{E}_{new}}{\mathcal{E}_{old}}$ reported in Table 3 is obtained using the value of $\mathcal{E}_{old}$ for $B = 3$. There is an improvement of two orders of magnitude in the CPU times in all cases. The new technique is therefore scalable for industrial SOCs having multiple TAMs, as illustrated in the following subsections.

4.2 Results for SOC p21241

SOC p21241 contains 28 cores. Of these, 6 are memory cores and 22 are scan-testable logic cores. Table 4 presents a summary of the data for the 28 cores. Test data for this SOC has not been published before.

Circuit (core)	Number range: Test patterns	Functional I/Os	Scan chains	Scan chain lengths: Min	Max
Logic cores	1–785	37–1197	1–31	1	400
Memory cores	222–12324	52–148	0	–	–

Table 4. Ranges in test data for the 28 cores in p21241.

Tables 5 and 6 present experimental results for p21241 for $B = 2$. The exhaustive method [8] did not run to completion for $B = 3$, even after two days of execution. The results for $\mathcal{P}_{\mathbf{NPAW}}$ over a larger number of TAMs ($1 \leq B \leq 10$) using the new co-optimization method are presented in Table 7. For $W \geq 24$, the testing times obtained using the new co-optimization technique are on average 25% lower than those obtained using the Exhaustive method in [8]. This is because using *Partition_evaluate*, we were able to partition W among a larger number of TAMs than was possible using Exhaustive. The values of $\mathcal{E}_{old}$ in Table 7 are for $B = 2$. The new CPU times are

W	Exhaustive method: $w_1 + w_2$	Core assignment	Testing time $\mathcal{T}_{old}$ (cycles)	Exec. time $\mathcal{E}_{old}$ (sec)
16	6+10	(2,1,2,2,2,1,1,1,1,1,1,1,1, 1,2,1,1,1,1,1,1,1,1,2,2,2,1)	462210	11
24	8+16	(2,2,2,1,1,2,1,1,1,1,1,1,2,1, 1,1,1,1,1,1,1,1,1,1,2,2,2,1)	361571	24
32	10+22	(1,2,2,2,1,2,2,2,2,1,1,2,2,2, 1,1,2,1,2,1,1,2,2,1,2,2,1,2)	312659	49
40	10+30	(1,2,2,2,2,2,2,2,2,1,1,2,1,2, 1,1,1,1,2,1,1,1,1,1,2,2,2,1)	278359	60
48	10+38	(1,2,2,2,2,2,2,2,2,2,2,2,1,1 1,1,1,1,2,1,1,1,1,2,2,2,2,2)	268472	84
56	10+46	(1,2,2,2,2,2,2,2,2,2,2,2,2,1, 1,1,1,1,1,1,1,2,1,1,2,2,1,2)	266800	80
64	10+54	(1,2,2,2,2,2,2,2,2,1,1,2,1,2, 2,1,2,1,2,1,1,2,2,2,2,2,2,1)	260638	122

Table 5. Exhaustive results for p21241 for $B = 2$ ($\mathcal{P}_{\mathbf{PAW}}$).

W	New co-optimization method: $w_1 + w_2$	Core assignment	$\mathcal{T}_{new}$ (cycles)	$\mathcal{E}_{new}$ (sec)	$\Delta\mathcal{T}$ (%)	Ratio $\frac{\mathcal{E}_{new}}{\mathcal{E}_{old}}$
16	6+10	(2,1,2,2,2,1,1,1,1,1, 1,1,1,1,1,2,1,1,1, 1,1,1,1,1,2,2,2,1)	462210	1	+0.00	0.1
24	10+14	(1,2,1,1,2,2,2,2,2,1, 1,2,2,1,1,1,1,1,1, 1,1,1,1,1,1,2,2,1)	365947	2	+1.21	0.08
32	10+22	(1,2,2,2,1,2,2,2,2,1, 1,2,2,2,1,1,2,1,2 1,1,2,2,1,2,2,1,2)	312659	9	+0.00	0.18
40	12+28	(1,2,2,2,1,2,2,2,2,1, 1,1,2,1,1,1,2,2,1 1,1,1,1,2,2,2,1,1)	290644	2	+4.41	0.03
48	20+28	(1,2,1,2,1,2,2,1,2, 1,1,1,1,2,1,1,2,1, 1,1,2,2,2,2,1,2,2,2)	290644	1	+8.25	0.01
56	28+28	(1,2,1,1,1,1,1,2,1,1 1,1,1,1,2,1,2,1,2 1,2,1,1,1,2,2,1,1,1)	290644	1	+8.97	0.01
64	29+35	(1,2,1,1,1,1,1,2,1,1, 1,1,1,1,2,1,x,1,2, 1,2,1,1,1,2,2,1,1)	271330	6	+4.10	0.05

Table 6. New results for p21241 for $B = 2$ ($\mathcal{P}_{\mathbf{PAW}}$).

comparable to the CPU times of the Exhaustive method. The final optimization step took between 10 and 20 seconds to complete in all cases. This was because the ILP models for this SOC were particularly intractable. This also explains why the Exhaustive method from [8] did not terminate within a reasonable CPU time for $B > 2$ for p21241.

For $\mathcal{P}_{\mathbf{NPAW}}$ and $W = 16$, the *Partition_evaluate* heuristic returns a partition of 1+1+4+10. This yields a testing time of 468011 cycles after the final optimization step (Table 7). However, a lower testing time of 462210 cycles is actually achievable using only 2 TAMs (Table 6). This is because the testing time for four TAMs returned by *Partition_evaluate* is lower than that for two TAMs before the final optimization step. Similarly, for $W = 64$, *Partition_evaluate* obtains a partition of 13+10+10+10+21 (five TAMs), which gives a lower (heuristic) testing time than does the partition obtained for $W = 56$: 5+4+10+10+10+17 (six TAMs). However, after the final optimization step, the partition for $W = 56$ is able to achieve the same testing time as that for $W = 64$. This anomalous behavior of our algorithm is due to the fact that *Partition_evaluate* uses heuristics to quickly approximate the final result. Therefore, the partition that actually provides the lowest testing time after final (exact) optimization might not be returned by *Partition_evaluate*.

4.3 Results for SOC p31108

SOC p31108 contains 19 cores. Of these, 15 are memory cores and 4 are scan-testable logic cores. Table 8 presents the data for

W	New co-optimization method: B	TAM partition	Core assignment	$\mathcal{T}_{new}$ (cycles)	$\mathcal{E}_{new}$ (sec)	$\Delta\mathcal{T}$ (%)	Ratio $\frac{\mathcal{E}_{new}}{\mathcal{E}_{old}}$
16	4	1+1+ 4+10	(4,3,4,4,4,4,3,2,3,2, 2,1,4,2,4,2,2,1,1, 2,1,1,3,1,1,1,4,1)	468011	10	+1.25	0.9
24	3	4+10+ 10	(2,2,2,1,3,3,3,1,2,1, 1,2,1,1,1,1,1,1,1, 1,1,1,1,1,2,3,2,1)	313607	41	-13.27	1.71
32	4	1+10+ 10+11	(4,4,3,2,2,2,2,2,2,1, 1,3,4,2,1,1,2,2,1, 3,4,1,2,4,2,3,1,1)	246332	192	-21.21	3.92
40	5	5+5+ 10+10+ 10	(4,4,3,5,3,1,1,1,3,1, 1,5,3,2,1,3,5,3,1 4,3,1,1,4,4,5,3,5)	232049	60	-16.64	1.00
48	6	5+3+ 10+10+ 10+10	(5,5,2,2,3,2,3,2,3,2, 2,2,2,2,2,2,2,3,2, 2,4,2,2,2,3,6,2,1)	232049	15	-13.57	0.18
56	6	5+4+ 10+10+ 10+17	(5,5,2,2,3,2,3,2,3,2, 2,2,2,2,2,2,2,3,2, 2,4,2,2,2,3,6,2,1)	153990	69	-42.28	0.86
64	5	13+10+ 10+10+ 21	(6,4,4,3,2,4,4,5,4,2, 4,6,5,4,3,5,4,4,3, 5,6,4,5,1,6,5,5,2)	153990	138	-40.92	1.31

Table 7. New results for p21241 ($\mathcal{P}_{\mathbf{NPAW}}$).

the 19 cores. Test data for this SOC has not been published before.

Circuit (core)	Number range: Test patterns	Functional I/Os	Scan chains	Scan chain lengths: Min	Max
Logic cores	210–745	109–428	1–29	8	806
Memory cores	128–12236	11–87	0	–	–

Table 8. Ranges in test data for the 23 cores in p31108.

W	Exhaustive method: $w_1 + w_2$	Core assignment	Testing time $\mathcal{T}_{old}$ (cycles)	$\mathcal{E}_{old}$ (sec)
16	8+8	(1,2,2,2,2,1,1,1,1,2, 1,1,2,2,1,1,1,1,1)	1080940	5
24	9+15	(1,2,2,2,2,2,1,2,1,2, 2,1,1,2,1,1,2,1,2)	820870	7
32	11+21	(1,2,2,2,2,1,1,2,2,2, 2,2,2,2,1,1,2,1,2)	733394	9
40	15+25	(1,2,2,2,2,1,2,2,2,2, 2,2,2,1,2,1,1,1,2)	721564	10
48	16+32	(1,1,2,2,2,2,1,2,2,1, 2,1,1,2,2,1,1,2,2)	709262	14
56	16+40	(1,1,2,2,2,2,1,2,2,1, 2,2,2,1,2,1,1,2,2)	704659	18
64	16+48	(1,1,2,2,2,2,2,2,2,1 2,2,2,2,2,2,2,2,2)	700939	18

Table 9. Exhaustive results for p31108 for $B = 2$ ($\mathcal{P}_{\mathbf{PAW}}$).

Tables 9, 10, 11, and 12 compare the results obtained by the exhaustive method with the results obtained by the new co-optimization method for p31108 for $B = 2$ and $B = 3$ (Problem $\mathcal{P}_{\mathbf{PAW}}$). For $B = 4$, the exhaustive method of [8] did not provide a solution even after two days of CPU time. Table 13 presents the new experimental results for p31108 (Problem $\mathcal{P}_{\mathbf{NPAW}}$). For $W \leq 32$, the testing times obtained using the new co-optimization technique are on average 15% higher than those obtained using the Exhaustive method. For $W \geq 40$, we reach the optimum testing time of 544579 cycles. The testing time of this SOC does not decrease beyond 544579 cycles as W is increased beyond 40 and B is increased beyond 3. This is because the testing time for Core 18 in p31108 reaches a minimum value of 544579 cycles when the width of the TAM to which it is assigned reaches 10 bits. Note that in Tables 11, 12 and 13, for $W \geq 40$, Core 18 is always assigned to a TAM, whose width is always 10 bits or more and which does not have any other cores assigned to it; thus our method achieves the theoretical lower bound on testing time for this SOC. For $W = 56$ and $W = 64$, TAM 1 is not used since the algorithm is able to assign the cores to the remaining TAMs, while achieving the lower bound of 544579 cycles. Results are shown for six TAMs, however, since the *Partition_evaluate* heuristic obtains a lower testing time for six TAMs than for five TAMs before the final optimization step. The values of $\mathcal{E}_{old}$ in Table 13 are for $B = 3$. The new CPU times are on average between 1 and 2 orders of magnitude less than the CPU times of the exhaustive method. This is because the individual $\mathcal{P}_{\mathbf{AW}}$ Exhaustive models for p31108 took particularly long to solve. This significantly affected the CPU time of the exhaustive method for $\mathcal{P}_{\mathbf{PAW}}$ and $\mathcal{P}_{\mathbf{NPAW}}$.

W	New co-optimization method: w_1+ w_2	Core assignment	$\mathcal{T}_{new}$ (cycles)	$\mathcal{E}_{new}$ (sec)	$\Delta\mathcal{T}$ (%)	Ratio $\frac{\mathcal{E}_{new}}{\mathcal{E}_{old}}$
16	8+8	(1,2,2,2,2,1,1,1,1,2, 1,1,2,2,1,1,1,1,1)	1080940	1	+0.00	0.2
24	10+14	(2,1,2,2,2,1,1,2,1,1, 2,1,1,2,1,1,1,2,2)	928782	1	+13.15	0.14
32	16+16	(1,2,1,2,2,2,1,2,1,2, 2,1,1,1,1,2,1,1,2)	750490	1	+2.33	0.11
40	16+24	(1,2,2,2,2,2,2,2,2,2, 2,2,1,2,2,2,1,1,2)	721566	1	+0.0002	0.1
48	16+32	(1,1,2,2,2,2,1,2,2,1, 2,1,1,2,2,1,1,2,2)	709262	1	+0.00	0.07
56	16+40	(1,1,2,2,2,2,1,2,2,1, 2,2,2,1,2,1,1,2,2)	704659	1	+0.00	0.06
64	16+48	(1,1,2,2,2,2,2,2,2,1 2,2,2,2,2,2,2,2,2)	700939	1	+0.00	0.06

Table 10. New results for p31108 for $B = 2$ ($\mathcal{P}_{\mathbf{PAW}}$).

W	Exhaustive method: w_1+ $w_2 + w_3$	Core assignment	Testing time $\mathcal{T}_{old}$ (cycles)	Exec. time $\mathcal{E}_{old}$ (sec)
16	1+7+8	(1,3,2,3,2,1,1,2,3,3, 3,1,2,2,3,1,2,2,1)	998733	222
24	9+7+8	(2,3,2,2,1,2,2,3,2,2, 3,3,3,2,3,3,2,1,2)	720858	325
32	17+5+ 10	(2,1,3,3,2,2,2,1,2,1, 2,2,1,2,1,2,2,3,2)	591027	1576
40	9+10+ 21	(1,3,1,1,3,3,1,3,1,1, 3,1,1,1,3,1,3,2,3)	544579	1081
48	9+10 29	(1,3,1,1,3,3,1,3,1,1, 3,1,1,1,3,1,3,2,3)	544579	6198
56	9+10 37	(1,3,1,1,3,3,1,3,1,1, 3,1,1,1,3,1,3,2,3)	544579	11331
64	17+15+ 32	(3,3,1,1,3,1,3,3,3,1, 1,1,1,3,3,3,3,2,1)	544579	1125

Table 11. Exhaustive results for p31108: $B = 3$ ($\mathcal{P}_{\mathbf{PAW}}$).

W	New co-optimization method: w_1+ $w_2 + w_3$	Core assignment	$\mathcal{T}_{new}$ (cycles)	$\mathcal{E}_{new}$ (sec)	$\Delta\mathcal{T}$ (%)	Ratio $\frac{\mathcal{E}_{new}}{\mathcal{E}_{old}}$
16	4+6+ 6	(3,2,1,3,1,1,1,1,1,1, 3,3,1,3,3,3,1,3,1)	1174710	10	+17.62	0.04
24	6+9+ 9	(1,2,1,2,3,2,2,1,3,1, 1,2,3,1,1,3,2,3,2)	729872	10	+1.25	0.03
32	6+12+ 14	(1,3,1,1,3,3,3,2,3,1, 1,3,1,3,3,3,3,2,2)	680591	13	+15.15	0.008
40	9+15+ 16	(1,3,3,3,3,3,1,3,3,1, 1,1,1,3,1,1,3,2,3)	544579	12	+0.00	0.01
48	9+16+ 23	(1,3,3,3,3,3,1,3,3,1, 1,1,1,3,1,1,3,2,3)	544579	12	+0.00	0.002
56	9+16+ 31	(1,3,3,3,3,3,1,3,3,1, 1,1,1,3,1,1,3,2,3)	544579	12	+0.00	0.001
64	9+16+ 39	(1,3,3,3,3,3,1,3,3,1, 1,1,1,3,1,1,3,2,3)	544579	11	+0.00	0.0.01

Table 12. New results for p31108 for $B = 3$ ($\mathcal{P}_{\mathbf{PAW}}$).

W	New co-optimization method					$\Delta\mathcal{T}$ (%)	Ratio $\frac{\mathcal{E}_{new}}{\mathcal{E}_{old}}$
	B	TAM partition	Core assignment	$\mathcal{T}_{new}$ (cycles)	$\mathcal{E}_{new}$ (sec)		
16	4	3+3+ 5+5+	(1,3,3,3,2,2,3,1,3,2, 1,3,1,2,1,3,3,4,1)	1033210	27	+3.45	0.12
24	4	5+5+ 6+8+	(2,3,2,1,4,2,1,4,3,2, 4,3,1,4,2,1,3,4,2)	882182	27	+22.38	0.08
32	5	5+4+ 6+8+9	(3,5,3,5,2,2,3,3,5,3, 5,3,1,3,2,3,5,4,1)	663193	51	+12.21	0.03
40	4	3+7+ 15+15	(2,3,1,1,3,1,2,3,1,2, 1,1,2,2,1,1,1,4,1)	544579	39	+0.00	0.04
48	5	5+3+8+ 15+17	(5,5,2,3,5,5,5,2,5,3, 1,2,2,2,2,5,5,4,2)	544579	205	+0.00	0.03
56	6	5+3+5+ 8+15+20	(6,6,6,6,4,6,6,6,6,4, 3,4,6,2,3,4,6,5,4)	544579	109	+0.00	0.01
64	6	5+3+5+ 8+15+28	(6,6,6,6,4,6,6,6,6,4, 3,4,6,2,3,4,6,5,4)	544579	288	+0.00	0.26

Table 13. New results for p31108 ($\mathcal{P}_{\mathbf{NPAW}}$).

4.4 Results for SOC p93791

Circuit (core)	Number range			Scan chain lengths	
	Test patterns	Functional I/Os	Scan chains	Min	Max
Logic cores	11–6127	109–813	11–46	1	521
Memory cores	42–3085	21–396	0	–	–

Table 14. Ranges in test data for the 32 cores in p93791.

W	Exhaustive method			
	$w_1 + w_2$	Core assignment	Testing time $\mathcal{T}_{old}$ (cycles)	Exec. time $\mathcal{E}_{old}$ (sec)
16	4+12	(2,2,1,2,1,2,2,2,2,1,1,2, 2,2,2,2,1,2,2,1,2,2,2,1, 1,1,2,2,2,2,2,1)	1798740	6
24	1+23	(2,1,1,1,2,2,1,1,1,2,1,2, 2,2,1,2,2,1,2,2,1,1,2,1, 1,1,2,2,2,1)	1211740	24
32	9+23	(1,2,1,2,1,2,1,1,1,2,1,2, 2,2,1,1,1,1,2,2,1,2,2,1, 1,1,2,1,1,1,1,1)	894342	33
40	17+23	(1,2,2,1,1,2,1,1,2,1,1,1, 2,2,2,2,2,1,1,1,1,2,2,1, 1,1,2,2,2,2,1,1)	747378	30
48	2+46	(2,1,1,2,2,2,1,1,1,2,1,2, 2,2,1,2,2,2,2,2,2,2,2,1, 1,1,2,2,2,1,1,1)	622199	85
56	10+46	(2,1,1,1,1,2,1,1,1,1,1,2, 2,2,1,1,2,1,1,2,1,1,2,1, 1,1,2,1,1,1,1,1)	524203	66
64	18+46	(1,2,1,2,1,2,1,1,2,2,1,2, 2,2,1,2,2,2,2,1,2,2,2,1 1,2,2,1,1,2,2,1)	467424	71

Table 15. Exhaustive results for p93791: $B = 2$ ($\mathcal{P}_{\mathbf{PAW}}$).

SOC p93791 contains 32 cores [8]. Of these 32 cores, 18 are memory cores and 14 are scan-testable logic cores. A summary of the 32 cores is presented in Table 14.

Tables 15, 16, 17, and 18 compare the exhaustive results for p93791 with the new results for $B = 2$ and $B = 3$ (Problem $\mathcal{P}_{\mathbf{PAW}}$). Note that for both $B = 2$ and $B = 3$, the new co-optimization method results in optimum testing times for several values of W. Here too, we did not achieve a solution with the exhaustive method after two days of execution for $B = 4$. Table 19 presents results for $\mathcal{P}_{\mathbf{NPAW}}$. The testing times obtained using the new co-optimization technique are comparable to the best testing times using the exhaustive method for all values of W, and equal to the results in [8] for $W = 48$ and 56. The testing times are on average 3% larger than those reported in [8]. The CPU times of the new *Partition_evaluate*

W	New co-optimization method				$\Delta\mathcal{T}$ (%)	Ratio $\frac{\mathcal{E}_{new}}{\mathcal{E}_{old}}$
	w_1+ w_2	Core assignment	$\mathcal{T}_{new}$ (cycles)	$\mathcal{E}_{new}$ (sec)		
16	1+15	(2,1,1,1,2,2,1,1,1,1,1, 2,2,2,1,1,2,1,2,2,1,1, 2,1,1,1,2,1,1,1,2,2)	1952800	1	+8.56	0.17
24	8+16	(2,1,1,1,1,2,1,1,1,2,1, 2,1,2,1,1,1,1,2,1,1,1, 2,1,1,1,2,2,1,1,1,2)	1217980	1	+0.51	0.04
32	9+23	(1,2,1,2,1,2,1,1,1,2,1, 2,2,2,1,1,1,1,2,2,1,2, 2,1,1,1,2,1,1,1,1,1)	894342	1	+0.00	0.03
40	16+24	(1,2,2,2,2,2,1,1,2,1,1, 2,1,1,2,2,2,2,2,1,2,2, 2,2,1,2,2,2,2,2,2,1)	750311	1	+0.39	0.033
48	23+25	(1,1,2,2,2,2,1,1,1,2,2, 2,2,1,1,2,2,1,2,1,2,2, 1,2,2,2,1,2,1,1,2,2)	632474	1	+1.65	0.01
56	10+46	(2,1,1,1,1,2,1,1,1,1,1, 2,2,2,1,1,2,1,1,2,1,1, 2,1,1,1,2,1,1,1,1,1)	524203	1	+0.00	0.02
64	18+46	(1,2,1,2,1,2,1,1,2,2,1, 2,2,2,1,2,2,2,1,2,2,2, 2,2,1,2,2,1,1,2,2,1)	467424	1	+0.00	0.01

Table 16. New results for p93791 for $B = 2$ ($\mathcal{P}_{\mathbf{PAW}}$).

W	Exhaustive method			
	w_1+ $w_2 + w_3$	Core assignment	Testing time $\mathcal{T}_{old}$ (cycles)	Exec. time $\mathcal{E}_{old}$ (sec)
16	5+3+8	(3,1,1,3,1,2,1,1,3,3,2, 3,3,2,1,3,1,1,1,3,1,2, 3,2,1,1,3,2,1,3,2,1)	1771720	25
24	7+8+9	(3,1,2,1,1,2,2,2,3,3,1, 1,2,2,1,1,3,2,1,3,2,1, 3,3,2,1,1,1,1,1,2,3)	1187990	50
32	4+5+23	(1,1,1,1,2,3,1,2,1,2,1, 3,3,3,1,1,3,1,2,3,1,1, 3,1,1,1,3,2,2,1,1,2)	887751	85
40	6+12+23	(3,1,1,1,2,3,1,1,1,2,1, 3,2,2,1,1,1,1,2,3,2,1, 1,1,1,1,3,1,2,1,1,2)	698583	130
48	9+16+23	(3,2,2,2,2,3,2,2,1,2,1, 2,1,2,2,2,2,2,2,3,2,2, 3,2,2,1,1,2,2,2,2,3)	599373	210
56	10+23+23	(1,3,2,3,2,3,3,3,3,3,1, 2,2,2,2,3,2,1,2,3,3,3, 2,1,1,3,3,3,2,2,2,2)	514688	270
64	18+23+23	(3,2,3,2,2,3,2,2,2,3,3, 2,2,2,2,3,2,1,1,1,3,2, 3,1,1,2,1,3,2,2,3,2)	460328	440

Table 17. ILP results for p93791: $B = 3$ ($\mathcal{P}_{\mathbf{PAW}}$).

algorithm are between two and three orders of magnitude smaller than the CPU times of the exhaustive method. This is because *Partition_evaluate* is able to effectively prune the solution space by halting evaluation of unnecessary partitions, for which the testing time of a TAM exceeds the previous minimum value for the SOC. For example, of the 341 unique partitions for $W = 64$ and $B = 3$, only 23 were evaluated to completion. Furthermore, the new heuristic algorithm *Core_assign* makes it possible to evaluate wrapper/TAM architectures for industrial SOCs having multiple TAMs, which was not feasible using the methods in [8].

5 Conclusion

We have presented a new efficient technique for co-optimization of the wrapper/TAM architecture for industrial SOCs. The general wrapper/TAM co-optimization problem has been formulated as a progression of four problems. The first problem $\mathcal{P}_{\mathbf{W}}$ relating to wrapper design is solved using an efficient algorithm presented earlier. For the second problem $\mathcal{P}_{\mathbf{AW}}$, relating to core assignment among TAMs of fixed widths, we have presented an efficient procedure called *Core_assign* that executes significantly faster than an

W	New co-optimization method: $w_1+w_2+w_3$	Core assignment	$\mathcal{T}_{new}$ (cycles)	$\mathcal{E}_{new}$ (sec)	$\Delta\mathcal{T}$ (%)	Ratio $\frac{\mathcal{E}_{new}}{\mathcal{E}_{old}}$
16	5+5+6	(2,1,2,1,3,3,1,1, 1,2,1,2,3,3,3,1, 1,1,2,2,1,3,2,1, 1,1,3,3,1,3,1,1)	1786200	2	+0.82	0.08
24	8+8+8	(2,1,1,3,1,3,1,1, 2,1,1,2,1,1,2,1, 1,1,1,3,3,1,2,1, 1,3,2,1,1,2,3,3)	1209420	3	+1.80	0.06
32	4+5+23	(1,1,1,1,2,3,1,2, 1,2,1,3,3,3,1,1, 3,1,2,3,1,1,3,1, 1,1,3,2,2,1,1,2)	887751	2	+0.00	0.02
40	6+10+24	(3,1,1,1,2,3,1,2, 2,1,1,3,1,1,2,2, 2,1,1,3,1,1,2,2, 1,2,3,1,1,1,1,3)	741965	1	+4.60	0.01
48	9+16+23	(3,2,2,2,2,3,2,2, 1,2,1,2,1,2,2,2, 2,2,2,3,2,2,3,2, 2,1,1,2,2,2,2,3)	599373	3	+0.00	+0.01
56	10+23+23	(1,3,2,3,2,3,3,3, 3,3,1,1,2,2,2,3, 2,1,2,3,3,3,2,1, 1,3,3,3,2,2,2,2)	514688	3	+0.00	0.01
64	15+23+26	(2,1,2,1,3,3,1,1, 1,3,1,3,1,1,3,2, 2,2,1,2,1,2,2,1, 1,3,3,2,2,1,1,1)	473997	2	+2.96	0.004

Table 18. New results for p93791 for $B = 3$ ($\mathcal{P}_{\mathbf{PAW}}$).

W	B	New co-optimization method: TAM partition	Core assignment	$\mathcal{T}_{new}$ (cycles)	$\mathcal{E}_{new}$ (sec)	$\Delta\mathcal{T}$ (%)	Ratio $\frac{\mathcal{E}_{new}}{\mathcal{E}_{old}}$
16	3	5+5+6	(2,1,2,1,3,3,1,1, 1,2,1,2,3,3,3,1, 1,1,2,2,1,3,2,1, 1,1,3,3,1,3,1,1)	1786200	2	+0.82	0.08
24	3	8+8+8	(2,1,1,3,1,3,1,1, ,2,1,1,21,1,2,1, 1,1,1,3,3,1,2,1, 1,3,2,1,1,2,3,3)	1209420	3	+1.80	0.06
32	2	9+23	(1,2,1,2,1,2,1,1, 1,2,1,2,2,2,1,1, 1,1,2,2,1,2,2,1, 1,1,2,1,1,1,1,1)	894342	1	+0.74	0.03
40	3	6+10+24	(3,1,1,1,2,3,1,2, 2,1,1,3,1,1,2,2, 2,1,1,3,1,1,2,2, 1,2,3,1,1,1,1,3)	741965	1	+4.60	0.01
48	3	9+16+23	(3,2,2,2,2,3,2,2, 1,2,1,2,1,2,2,2, 2,2,2,3,2,2,3,2 2,1,1,2,2,2,2,3)	599373	3	+0.00	+0.01
56	3	10+23+23	(1,3,2,3,2,3,3,3, 3,3,1,1,2,2,2,3, 2,1,2,3,3,3,2,1, 1,3,3,3,2,2,2,2)	514688	3	+0.00	0.01
64	3	15+23+26	(2,1,2,1,3,3,1,1, 1,3,1,3,1,1,3,2, 2,2,1,2,1,2,2,1, 1,3,3,2,2,1,1,1)	473997	2	+2.96	0.004

Table 19. New results for p93791 ($\mathcal{P}_{\mathbf{NPAW}}$).

ILP model for the same problem presented earlier. The third and fourth problems in the progression, $\mathcal{P}_{\mathbf{PAW}}$ and $\mathcal{P}_{\mathbf{NPAW}}$ relate to determining a partition of TAM width and an effective number of TAMs for the SOC, such that testing time is minimized. These two problems have been solved using a new heuristic procedure called *Partition_evaluate* that quickly reaches within the neighborhood of the optimal solution to $\mathcal{P}_{\mathbf{PAW}}$ and $\mathcal{P}_{\mathbf{NPAW}}$. *Partition_evaluate* uses extensive solution-space pruning to identify an effective TAM partition for the SOC. Finally, the existing ILP model for $\mathcal{P}_{\mathbf{AW}}$ is used to optimize the core assignment and testing time for the width partition produced by *Partition_evaluate*. Experimental results for several industrial SOCs demonstrate that wrapper/TAM co-optimization can be effectively carried out in over an order of magnitude less time than exact methods based on ILP and exhaustive enumeration presented earlier.

The drawback of the heuristic methods presented in this paper are that they exhibit anomalous behavior at times. The width partition and number of TAMs returned by *Partition_evaluate* do not always provide the lowest testing time after the final (exact) optimization step is performed.

Acknowledgements

We thank Henk Hollman for his help with partition theory. We are grateful to Graeme Francis, Harry van Herten and Erwin Waterlander for their help with providing data for the Philips SOCs, and Bart Vermeulen, Harald Vranken and Graeme Francis for their comments on an earlier version of this paper.

References

[1] J. Aerts and E.J. Marinissen. Scan chain design for test time reduction in core-based ICs. *Proc. Int. Test Conf.*, pp. 448-457, 1998.

[2] M. Berkelaar. *lpsolve 3.0*, Eindhoven University of Technology, Eindhoven, The Netherlands. *ftp://ftp.ics.ele.tue.nl/pub/lp_solve*

[3] P. Brucker. *Scheduling Algorithms.* 3rd ed., Springer, Berlin, Germany, 2001.

[4] K. Chakrabarty. Design of system-on-a-chip test access architectures under place-and-route and power constraints. *Proc. Design Automation Conf.*, pp. 432-437, 2000.

[5] K. Chakrabarty. Optimal test access architectures for system-on-a-chip. *ACM Trans. Design Automation of Electronic Systems*, vol. 6, pp. 26–49, January 2001.

[6] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Co., San Francisco, CA, 1979.

[7] J. Hromkovic. *Algorithmics for Hard Problems*. Springer, Berlin, Germany, 2001.

[8] V. Iyengar, K. Chakrabarty, and E.J. Marinissen. Test wrapper and test access mechanism co-optimization for system-on-chip. *J. Electronic Testing: Theory and Applications*, vol. 18, March 2002, in print.

[9] E. Larsson and Z. Peng. An integrated system-on-chip test framework. *Proc. Design, Automation, and Test in Europe (DATE)*, pp. 138-144, 2001.

[10] J.H. van Lint and R.M. Wilson, *A Course in Combinatorics*, Cambridge University Press, 1992.

[11] E.J. Marinissen et al. A structured and scalable mechanism for test access to embedded reusable cores. *Proc. Int. Test Conf.*, pp. 284-293, 1998.

[12] E.J. Marinissen, S.K. Goel and M. Lousberg. Wrapper design for embedded core test. *Proc. Int. Test Conf.*, pp. 911–920, 2000.

[13] M. Nourani and C. Papachristou. An ILP formulation to optimize test access mechanism in system-on-chip testing. *Proc. Int. Test Conf.*, pp. 902–910, 2000.

[14] P. Varma and S. Bhatia. A structured test re-use methodology for core-based system chips. *Proc. Int. Test Conf.*, pp. 294–302, 1998.

[15] Y. Zorian, E.J. Marinissen and S. Dey. Testing embedded-core-based system chips. *IEEE Computer*, vol. 32, pp. 52–60, June 1999.