On Using Rectangle Packing for SOC Wrapper/TAM Co-Optimization

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

The testing time for a system-on-chip (SOC) is determined to a large extent by the design of test wrappers and the test access mechanism (TAM). Wrapper/TAM co-optimization is therefore necessary for minimizing SOC testing time. We recently proposed an exact technique for co-optimization based on a combination of integer linear programming (ILP) and exhaustive enumeration. However, this approach is computationally expensive for large SOCs, and it is limited to fixed-width test buses. We present a new approach for wrapper/TAM co-optimization based on generalized rectangle packing, also referred to as two-dimensional packing. This approach allows us to decrease testing time by reducing the mismatch between a core's test data needs and the width of the TAM to which it is assigned. We apply our co-optimization technique to an academic benchmark SOC and three industrial SOCs. Compared to the ILP-based technique, we obtain lower or comparable testing times for two out of the three industrial SOCs. Moreover, we obtain more than two orders of magnitude decrease in the CPU time needed for wrapper/TAM co-design.

1 Introduction

Testing has emerged as a major bottleneck in plug-and-play system-on-chip (SOC) design. In order to reduce design time, a large number of embedded cores are often stitched into an SOC. The testing of embedded cores is a formidable challenge. To facilitate the reuse of test patterns provided by the core vendor, an embedded core must be isolated from surrounding logic, and test access must be provided from the I/O pins of the SOC. Test wrappers form the interface between cores and test access mechanisms (TAMs), while TAMs transport test data between SOC pins and test wrappers [12]. Effective test wrappers and TAMs are therefore important parts of an SOC test infrastructure.

We address the problem of designing test wrappers and TAMs to minimize SOC testing time. While optimized wrappers reduce test application times for the individual cores, optimized TAMs lead to more efficient test data transport on-chip. Since wrappers influence TAM design, and vice versa, a co-optimization strategy is needed to jointly optimize the wrappers and the TAM for an SOC.

Most prior research has either studied wrapper design and TAM optimization as independent problems, or not addressed the issue of sizing TAMs to minimize SOC testing time [1, 2, 8, 13]. Alternative approaches that combine TAM design with test scheduling [5, 10, 15] do not address the problem of wrapper design and its relationship to TAM optimization.

Integrated wrapper/TAM co-optimization methodologies presented in the literature are based on fixed-width test buses [6, 7]. Related recent work combining TAM design with test scheduling is also based on the use of fixed-width test buses [9]. However, the fixed-width test bus models of [6, 7, 9] preclude the design of more flexible TAM architectures and the use of optimization algorithms for such architectures that can explore a much larger solution space. Furthermore, such architectures generally lead to inefficient usage of TAM wires. For example, in a design with a large number of cores and only a small number of fixed-width test buses, cores with widely-varying test data requirements are often assigned to the same test bus. Now, it was shown in [6] that for a given core, the testing time varies with TAM width as a "staircase" function. This implies that the testing time does not decrease with increasing TAM width until a core-specific threshold is exceeded. Hence if a core is connected to a TAM of width w, the same testing time may actually be obtained using only w' wires ($w' < w$). The remaining $w - w'$ wires, which could have been used to transport test data for another core, are not efficiently utilized.

We present a new approach to wrapper/TAM co-optimization based on a generalized version of rectangle packing. Rectangle packing is also referred to as two-dimensional packing in [4]. We first use the wrapper design method presented in [6] to design a set of wrappers for each core that eliminate the mismatch between the core's test data requirements and its TAM width. A test schedule is then determined and an effective amount of TAM width is assigned to each core in the test schedule. Finally, the wrapper corresponding to the TAM width assigned to the core is chosen. Hence, instead of assigning cores to a small number of fixed-width test buses for the entire schedule as in [6], we achieve a more flexible partitioning of the total TAM width among the cores. This partitioning is tailored to the test data needs of the specific group of cores being tested at any interval in the schedule. The efficient TAM design algorithm developed achieves over two orders of magnitude reduction in the CPU time over the exact methods in [6]. This is especially important because it enables the TAM designer to explore a significantly larger solution space than was possible in [6]. The flexible-width TAM architecture leads to a reduction in the "real" costs of SOC test—testing time and on-chip hardware (total TAM width).

The remainder of this paper is organized as follows. In Section 2, we define the wrapper/TAM co-optimization problem and formulate it as a generalized version of rectangle packing. In Section 3, we present an efficient algorithm to obtain an effective wrapper/TAM architecture and a test schedule that minimizes testing time. Finally, in Section 4, we present experimental results on one academic SOC and three industrial SOCs. Compared to the ILP-based approach of [6], we obtain lower testing times for two out of the three industrial SOCs. In addition, we obtain more than two orders of magnitude decrease in the CPU time for wrapper/TAM co-design.

2 The rectangle packing problem

The wrapper/TAM co-optimization problem that we address in this paper is as follows.

Problem $\mathcal{P}_{\mathrm{co-opt}}$: Given the test set parameters for the cores and the total TAM width W for the SOC, determine the TAM width and a

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

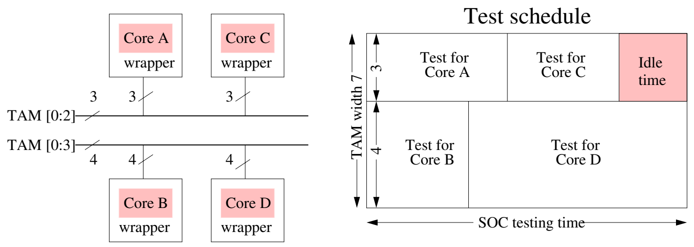

Figure 1. TAM design using TAM width partitioning [6].

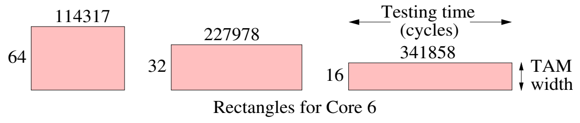

Figure 2. Example rectangles for Core 6 in Philips SOC p93791 (figure not drawn to scale).

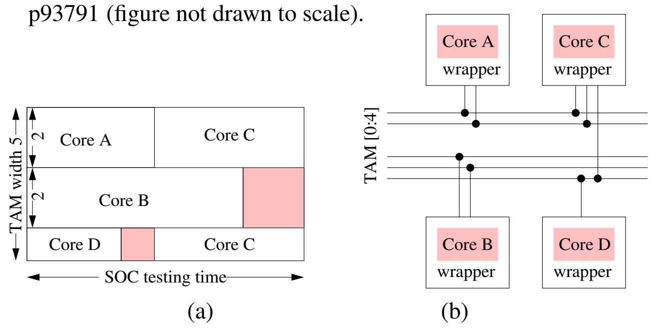

Figure 3. An illustration of (a) rectangle splitting, and (b) the corresponding TAM architecture.

wrapper design for each core, and a test schedule for the SOC such that (i) the total number of TAM wires utilized at any moment does not exceed W, and (ii) the overall SOC test completion time is minimized. □

The test parameters for each core include the number of primary inputs, primary outputs, bidirectional I/Os, test patterns, scan chains, and scan chain lengths. Unlike in [1], we assume that the number and lengths of scan chains are fixed.

To solve the wrapper/TAM design problems in [6], we modeled TAMs as fixed width test buses. The total TAM width was partitioned among a number of fixed-width test buses and each core was assigned to one of these TAMs, as illustrated in Figure 1 for a generic SOC. Since exact methods were used for optimization, solutions could be obtained for up to only three test buses. Since large SOCs often contain more than twenty cores, the inflexible TAM architecture of [6] often led to the situation where a set of cores having widely-varying test data requirements were assigned to the same test bus. This mismatch between core test data needs and TAM width led to test schedules with unnecessarily high SOC testing times. Although the overall objective of the problems in [6] and Problem $\mathcal{P}_{\rm co-opt}$ is the same (the minimization of SOC testing time for a total TAM width), there is a significant difference between the two TAM architecture models. Hence, the new wrapper/TAM design problem formulation is also different. In the new approach, there is no explicit partition of the total TAM width among fixed width TAMs to which cores must then be assigned. Instead, a more flexible TAM architecture is created in which the total TAM width is partitioned effectively among the group of cores being tested during any time interval in the schedule. This partition is allowed to vary with time, such that the test data needs of each group of cores are effectively addressed.

We next discuss the new wrapper/TAM design problem. Problem $\mathcal{P}_{\rm co-opt}$ consists of three parts: wrapper design, TAM width assignment, and test scheduling. These subproblems must be solved in conjunction to achieve the lowest testing time. We solved the problem of wrapper design for cores in [6] using an algorithm based on the Best Fit Decreasing (BFD) heuristic for the Bin Packing problem. The proposed *Design_wrapper* algorithm [6] has two optimization criteria: (i) minimizing core testing time, and (ii) minimizing the TAM width required for the test wrapper. These objectives are achieved by balancing the lengths of the wrapper scan chains, and determining the number of wrapper scan chains that are actually needed to minimize testing time. Criterion (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.

Here, we first introduce the notion of using rectangles to model core tests, and then illustrate the flexibility in TAM design and test scheduling provided by the proposed TAM model. The use of rectangles for core test representation during test scheduling has previously been studied in [3, 5, 11]. The *Design_wrapper* algorithm is used to obtain the different test application times for each core for varying values of TAM width. A set of rectangles for a core can now be constructed, such that the height of each rectangle corresponds to a different TAM width and the width of the rectangle represents the core test application time for this value of TAM width. For example, Figure 2 illustrates how the test application times for three different TAM widths for Core 6 (from example SOC p93791 from Philips) can be represented using rectangles. (The relevant details of SOC p93791 are presented in Section 4.)

We now formulate Problem $\mathcal{P}_{\rm co-opt}$ as a generalized version of the rectangle packing problem [4]. The rectangle packing problem is described as follows. Given a collection of rectangles, and a bin of fixed height and unbounded width, pack the rectangles into the bin, such that no two rectangles overlap, and the width to which the bin is filled is minimized. This problem can be generalized as follows. Consider an SOC having N cores, and let $\mathbf{R}_i$ be the set of rectangles for core i, $1 \leq i \leq N$. Now, Problem $\mathcal{P}_{\rm GRP}$ (generalized rectangle packing) can be stated as follows.

Problem $\mathcal{P}_{\rm GRP}$: Given the collection $\{\mathbf{R}_1, \mathbf{R}_2, \ldots, \mathbf{R}_N\}$ of rectangles for an SOC, select one rectangle $R_{ij} \in \mathbf{R}_i$ from each set $\mathbf{R}_i$, $1 \leq i \leq N$, and pack the selected rectangles into a bin of fixed height and unbounded width, such that no two rectangles overlap, and the width to which the bin is filled is minimized. Furthermore, during packing, each rectangle selected is allowed to be split vertically into several non-adjacent pieces, each having the same co-ordinates on the horizontal axis. □

In Problem $\mathcal{P}_{\rm GRP}$, during packing, the rectangle selected for a core can be vertically split into several non-adjacent rectangles having the same width, as illustrated for Core C in Figure 3(a). This is because it is possible to assign a group of non-contiguous TAM wires to a single core, using fork-and-merge of TAM wires as illustrated in Figure 3(b). All the pieces of the split rectangle must, however, have the same co-ordinates on the horizontal axis. Recall that in [4], rectangles are considered to be indivisible entities.

Problem $\mathcal{P}_{\rm GRP}$ relates to Problem $\mathcal{P}_{\rm co-opt}$ as follows; see Figure 4. The height of the rectangle selected for a core corresponds to the TAM width assigned to the core, while the rectangle width corresponds to its testing time. The height of the bin corresponds to the total SOC TAM width, and the width to the which the bin is ultimately filled corresponds to the system testing time that is to be minimized. The unfilled area of the bin corresponds to the idle time on TAM wires during test. Furthermore, the distance between the left edge of each rectangle and the left edge of the bin corresponds to the begin time of

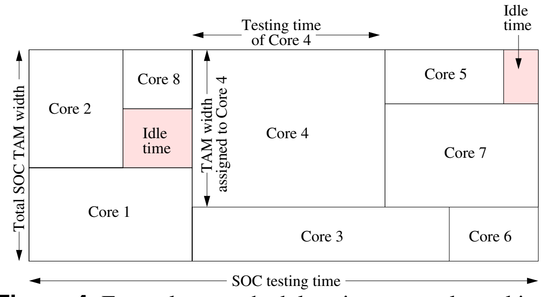

Figure 4. Example test schedule using rectangle packing.

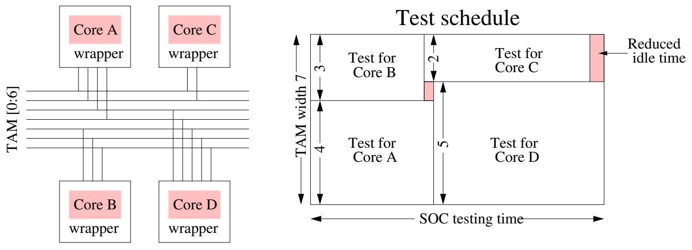

Figure 5. TAM design using generalized rectangle packing.

each core test. Thus, a one-to-one correspondence exists between the packed bin and the final test schedule.

Problem $\mathcal{P}_{\mathrm{GRP}}$ can be shown to be $\mathcal{NP}$-hard by a restriction argument. A special case of $\mathcal{P}_{\mathrm{GRP}}$, in which the cardinality of each set $\mathbf{R}_i$, $1 \leq i \leq N$ equals one, and no rectangles are allowed to be split directly corresponds to the rectangle packing problem in [4]. Since the rectangle packing problem was shown to be $\mathcal{NP}$-hard in [4] (by restriction to Bin Packing), $\mathcal{P}_{\mathrm{GRP}}$ is also $\mathcal{NP}$-hard. Furthermore, since Problem $\mathcal{P}_{\mathrm{co-opt}}$ is equivalent to Problem $\mathcal{P}_{\mathrm{GRP}}$, Problem $\mathcal{P}_{\mathrm{co-opt}}$ is $\mathcal{NP}$-hard.

We now illustrate the flexibility in TAM design offered by the proposed approach; see Figure 5. The generalized rectangle packing algorithm is used to determine the precise partition of the total TAM width among a group of cores during any interval in the schedule to minimize the testing time of the overall schedule. Note the increased flexibility in the test schedule of Figure 5 over Figure 1.

Pareto-optimal points. We achieve a significant reduction in computation time and improvement in TAM wire utilization over the method in [6] by noting that not all rectangles having values of TAM width between 1 and W, where W is the total SOC TAM width, need to be considered. It was shown in [6] that for a given core, the testing time varies with TAM width as a "staircase" function. From the values of testing time for different TAM widths for Core 6 of SOC p93791, illustrated in Figure 6, we see that the testing time decreases only when the TAM width exceeds core-specific thresholds. These threshold points are known as pareto-optimal points, and are formally defined as follows. A solution to the wrapper design problem for Core i can be expressed as a 2-tuple $(w_j, T_i(w_j))$, where w_j is the TAM width supplied to the wrapper and $T_i(w_j)$ is the testing time of Core i with the given wrapper. A solution $(w_j, T_i(w_j))$ is *Pareto-optimal* if and only if there does not exist a solution $(w_k, T_i(w_k))$, such that $w_k \leq w_j$ and $T_i(w_k) \leq T_i(w_j)$, where at least one of the inequalities is strict. Intuitively, the steps at which the testing time decreases are the Pareto-optimal points, and only rectangles corresponding to Pareto-optimal TAM width values need to be considered. For example, in Figure 6, a TAM width of 46 results in a testing time of 115850 cycles, while all TAM widths from 47 up to 64 result in the same testing time of 114317 cycles. Hence 47 is a Pareto-optimal TAM width, and rectangles of height between 48 and 64 can be ignored. The stair-

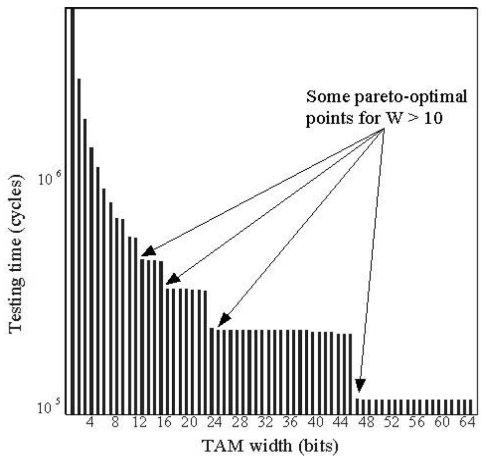

Figure 6. Relationship between testing time and TAM width for Core 6 in Philips SOC p93791.

Data structure *Schedule*

1. $width_p(i)$ /* preferred TAM width for Core i */
2. $width(i)$ /* TAM width assigned to Core i */
3. $begin(i)$ /* begin time of Core i */
4. $end(i)$ /* end time of Core i */
5. $scheduled(i)$ /* boolean indicates Core i has been scheduled */
6. $complete(i)$ /* boolean indicates test for Core i has completed */

Figure 7. Data structure for the test schedule.

case nature of the testing time variation with TAM width for cores is thus exploited to reduce the TAM width assigned to cores to the minimal value required to achieve a specific testing time. The extra TAM wires can be used for other cores in the SOC, thereby yielding a more efficient test schedule.

3 TAM optimization and test scheduling

In this section, we detail the algorithm that we have developed to solve Problem $\mathcal{P}_{co-opt}$ modeled as Problem $\mathcal{P}_{\mathrm{GRP}}$. Our algorithm first identifies a small value of "preferred TAM width" $width_p(i)$ for each core, such that the core's testing time is within a small percentage of its testing time at a maximum allowable TAM width W_{max}. (In this paper, W_{max} is chosen to be 64.) The test for each Core i is then scheduled using $width_p(i)$ bits, as long as there are enough TAM wires available. If the number of available TAM wires is insufficient to schedule any new tests, the resulting idle time is filled using several heuristics that insert tests to minimize the idle time. After the first currently-running test completes, the number of available TAM wires is incremented, and the algorithm repeats the scheduling process for the remaining tests. Next, we explain the rationale behind the heuristic decision-making in our algorithm, and show how these decisions minimize system testing time. We elaborate on each step of the algorithm in the following paragraphs.

Data structure. The data structure in which we store the TAM width and testing time values for the cores of the SOC is presented in Figure 7. This data structure is updated with the begin times, end times, and assigned TAM widths for each core as the test schedule is developed.

Preferred TAM widths. The pseudocode for our algorithm *TAM_schedule_optimizer* is presented in Figure 8. In Line 1, we compute the collection of Pareto-optimal rectangles as described in Section 2. In Line 2, we calculate the preferred TAM width values for

```
Procedure TAM_schedule_optimizer(C, $W, d, p$)

1 Compute collection R of rectangles using Design_wrapper;
2 Initialize(C, $d, p$);
3 Set $w\_avail = W; current\_time = 0$;
4 While C $\neq \emptyset${
5   If $w\_avail > 0${
6     If Core $i \in$ C can be found, such that
          $width_p(i) \leq w\_avail$ AND $T_i(width_p(i))$ is maximum{
7       If C$-\{i\} = \emptyset${
8         Set $width(i) = w$, where $w$ is the highest Pareto-optimal
              width for $i$, such that $w \leq w\_avail$;}
9       Else{ $width(i) = width_p(i)$;
10      Update($i$);}}
11    Else{ /* find a core that can use the resulting idle TAM wires */
12      Find $next\_time = end(i)$, such that
            $end(i) > this\_time$ AND $end(i)$ is minimum;
13      If Core $i$ can be found, such that $scheduled(i) = 0$ AND
            $T_i(w\_avail) + this\_time \leq next\_time$ AND
            $T_i(w\_avail) + this\_time$ is maximum{
14        Set $width(i) = w$, where $w$ is the highest Pareto-optimal
              width for $i$, such that $w \leq w\_avail$;
15        Update($i$);}
16      Else{
17        If Core $i$ can be found, such that $begin(i) = this\_time$ AND
              $T_i(width(i)) - T_i(width(i) + w\_avail)$ is maximum;{
18          Set $width(i) = w$, where $w$ is the highest Pareto-optimal
                width, such that $w \leq width(i) + w\_avail$;
19          Update($i$);}
20        Else{ /* declare TAM wires $w\_avail$ idle until $next\_time$ */
21          Set $w\_idle = w\_avail$; $w\_avail = 0$;}}}}
22    Else{
23      Calculate $next\_time = end(i)$, such that
            $end(i) > this\_time$ AND $end(i)$ is minimum;
24      Set $this\_time = next\_time$;
25      For every Core $i$, such that $end(i) = this\_time${
26        Increment $w\_avail = w\_avail + width(i)$;
27        Set $complete(i) = 1$;}}}
28 Return Schedule;
```

Figure 8. Algorithm for solving $\mathcal{P}_{\text{co-opt}}$.

each core from the input percent value p. Recall that the core testing time varies with TAM width w as a staircase function that drops rapidly at first for small values of w and less rapidly after that. For example, for Core 6 in p93791 (Figure 6), at $w = 10$ the testing time reaches within 10% of its value at $w = 64$, and at $w = 15$, the testing time is within 5% of its value at $w = 64$. However, the highest Pareto-optimal value of TAM width is $w = 47$. Hence, instead of attempting to assign the highest Pareto-optimal width to a core, a considerable savings in system TAM width can be realized by assigning a pre-calculated preferred value of width, such that the testing time of the core reaches within a small percent value p of its testing time at $w = W_{max}$. This value of p is usually between 1 and 10.

In subroutine *Initialize* (Figure 9), we initialize $width_p(i)$ to the Pareto-optimal TAM width that will provide the closest testing time to the calculated value of time T_{ip} that is within p% from $T_i(W_{max})$. (We use $T_i(w)$ to denote the testing time of Core i, when provided with a TAM width of w.) In Lines 4 and 5 of *Initialize*, we make an allowance for $width_p(i)$ to be set to the highest Pareto-optimal TAM width w_h if the difference between the value of $width_p(i)$ from Line 3 and the value of w_h is less than the input difference value d. This heuristic aids significantly in minimizing system testing time, especially when it is beneficial to assign a few ($\leq d$) extra TAM wires to a bottleneck core in the system, while the other cores receive TAM widths corresponding to p. For example, when using $p = 2$ for example SOC p34392 from Philips (presented in Section 4), we noticed

```
Procedure Initialize(C, $d, p$)

1 For each Core $i \in$ C{
2   Calculate $T_{ip} = T_i(W_{max}) + \frac{p}{100} \times (T_i(1) - T_i(W_{max}))$;
3   Set $width_p(i) = w$, such that $T_i(w) - T_{ip}$ is minimum;
4   Calculate highest Pareto-optimal width $w_h$;
5   If $w_h - width_p(i) \leq d$ then $width_p(i) = w_h$;}
```

Figure 9. Preferred widths initialization subroutine.

```
Procedure Update($i$)

1 Let $i$ be the core to be updated in the test schedule;
2 Set $begin(i) = current\_time$;
3 Set $end(i) = current\_time + T_i(width(i))$;
4 Set $scheduled(i) = 1$;
5 If $i \in$ C then C = C $-\{i\}$;
```

Figure 10. The data structure update algorithm.

that Core 18 was assigned $width_p(18) = 9$ bits, leading to a testing time of $T_{18}(9) = 622163$ cycles. The testing time for the SOC was also found to be 622163 cycles, from which we noted that Core 18 is the bottleneck core for p34392. A further study of the testing time–TAM width characteristics of Core 18 revealed that its highest Pareto-optimal TAM width is 10 bits, at which the testing time for Core 18 reaches its minimum value of 544579 cycles. Hence, providing an extra TAM wire to Core 18 reduced its testing time as well as the overall SOC testing time to $T_{18}(10) = 544579$ cycles. Thus the minimum testing time for SOC p34392 could be achieved using the heuristic in Lines 4 and 5 of *Initialize* with $d = 1$.

In our current approach, we consider incrementing the preferred TAM width by d only if it leads to the highest Pareto-optimal width. A more sophisticated heuristic that increments the preferred TAM width by d to lead to Pareto-optimal widths other than the highest, can potentially improve our current method; this needs further investigation. Additionally, note that the use of parameters p and d lead only to Pareto-optimal preferred TAM widths. Non-Pareto-optimal widths are not considered.

Assigning preferred TAM widths to cores. Next, Line 3 initializes the main rectangle packing loop. While executing the main **While** loop (Line 4), if there are $w\_avail > 0$ TAM wires available for assignment, Lines 6 to 10 will assign the TAM wires to the Core i with the longest testing time $T_i(width_p(i))$ that can be found with the condition $width_p(i) \leq w\_avail$. Furthermore, if (in Line 7) Core i is found to be the last core in **C** being scheduled, it will be assigned its highest Pareto-optimal width w, such that $w \leq w\_avail$ in Line 8. This heuristic minimizes the testing time time for Core i at no additional expense of TAM width, since it is the last core being scheduled and can thus receive all the available width without depriving other cores of TAM width. We found this heuristic to be useful in reducing the testing time of the last scheduled core in several cases, thereby reducing the testing time of the entire SOC. However, if Core i is not the last core to be scheduled, it is assigned its preferred $width_p(i)$ TAM wires in Line 9. Line 10 updates the test schedule data structure as outlined in Figure 10.

Rectangle insertion in idle time. If there is no core found in Line 6, rather than let the w_avail TAM wires remain idle, *TAM_schedule_optimizer* attempts to insert the rectangle for some unscheduled core into the available time, as illustrated in Figure 11. In Line 12, we find $next\_time$, i.e., the end time for the first test that is expected to end after $this\_time$. Since there will be an increment to

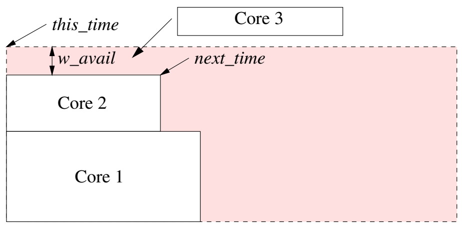

Figure 11. Inserting a rectangle to avoid idle time.

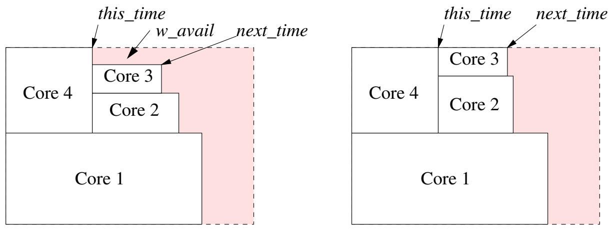

Figure 12. Increasing TAM width to fill idle time.

w_avail at $next\_time$, we are interested in filling the idle time only until $next\_time$. Filling in time beyond $next\_time$ is not prudent, because this can lead to inefficient test scheduling by assigning the few w_avail TAM wires to a large core that could instead benefit from a wider TAM width later on in the schedule. The heuristic in Line 13 attempts to find the core that fills in the most time until $next\_time$, thereby minimizing the potential idle time. Lines 14 and 15 schedule the test for the core.

Increasing TAM widths to fill idle time. If no rectangle is available to fill in the idle time, then the heuristic in Line 17 is used to determine which of the cores currently scheduled to begin at $this\_time$ will benefit the most, in terms of testing time decrease, from an extra w_avail TAM wires. If such a core can be found, then its currently-assigned $width(i)$ TAM wires are increased to the highest Pareto-optimal width less than $width(i) + w\_avail$. This heuristic is illustrated in Figure 12. In Figure 12, after Cores 2 and 3 have been assigned their preferred widths, no other core's preferred width is small enough to fit in the idle time above the rectangle for Core 3. Furthermore, there is no core available, for which a rectangle can be made to fit in the idle time, such that its end time is less than $next\_time$. Therefore, Core 2 is selected to have its width increased to $width(2) + w\_avail$. Note from Figure 12 that the width of a core (e.g., Core 1) whose begin time lies before $this\_time$ cannot be increased at $this\_time$, because its test has already begun with $width(i)$ TAM wires. This requirement for Core i is ensured by the first argument to the AND condition in Line 17.

Finally, if the heuristic in Line 17 fails to find a core whose width can be increased, the w_avail TAM wires are declared idle in Line 21. The value of w_avail is set to 0 and the loop beginning at Line 4 is repeated. When w_avail is found to be 0 in Line 5, the execution proceeds to Line 22 where the process of updating $this\_time$ and w_avail is begun. Line 23 updates $this\_time$ to $next\_time$, Line 26 increases w_avail by the width of all cores ending at the new value of $this\_time$, and Line 27 sets $complete(i)$ to 1 for all cores whose test has completed at $this\_time$. The resulting test schedule is output in Line 28.

4 Experimental results

In this section, we present experimental results for four benchmark SOCs: d695 (an academic benchmark from Duke University), p22810, p34392, and p93791 (industrial SOCs from Philips). These four SOCs are part of the *ITC'02 SOC test benchmarking initiative* [14]. The number (e.g., 93791) in each SOC name is a measure of its test complexity. This naming convention is described in [6]. The four SOCs were orginally introduced as d695, p21241, p31108, and p93791, respectively in [6] and [7]. However, in [6, 7], test data for the interconnect circuitry in the SOCs was not included. The interconnect test data has since been added, and the SOCs have been renamed; the SOCs are available at [14].

	SOC d695				**SOC p22810**			
	Method in [6]		New method		Method of [6]		New method	
W	$\mathcal{T}$ (cc)	$\mathcal{E}$ (sec)	$\mathcal{T}_{new}$ (cc)	$\Delta\mathcal{T}$ (%)	$\mathcal{T}$ (cc)	$\mathcal{E}$ (sec)	$\mathcal{T}_{new}$ (cc)	$\Delta\mathcal{T}$ (%)
16	42568	16	43723	+2.71	462210	11	452639	-2.07
24	28292	40	30317	+7.16	361571	24	307780	-14.88
32	21566	67	23021	+6.75	312659	49	246150	-21.27
40	17901	105	18459	+3.12	278359	60	197293	-29.12
48	16975	159	15698	-7.52	268472	84	167256	-37.70
56	13207	210	13415	+1.57	266800	80	145417	-45.50
64	12941	290	11604	-10.33	260638	122	136941	-47.46

Table 1. Results for d695 and p22810.

The experimental results were obtained using a Sun Ultra 10 with a 333 MHz processor and 256 MB memory.

Example SOCs. SOC d695 consists of ISCAS benchmark circuits [6]. SOC p22810 contains 6 memory cores and 22 scan-testable logic cores. SOC p34392 contains 15 memory cores and 4 scan-testable logic cores. SOC p93791 contains 18 memory cores and 14 scan-testable logic cores.

Table 1 presents results of wrapper/TAM co-optimization for SOCs d695 and p22810. We considered all possible integer values of the parameters p and d in the range $1 \leq p \leq 10$, $0 \leq d \leq 4$, and tabulated the best results. The symbols $\mathcal{T}$ and $\mathcal{E}$ are used to represent the SOC testing time (expressed in clock cycles) and the computation time (expressed in seconds), respectively, of the ILP-based algorithm [6]. The symbol $\mathcal{T}_{new}$ represents the SOC testing time (expressed in clock cycles) of the new rectangle packing algorithm. The percentage change in testing time using the new method is calculated using the formula $\Delta\mathcal{T}\ (\%) = \frac{\mathcal{T}_{new} - \mathcal{T}}{\mathcal{T}} \times 100$. The computation time of the new algorithm is less than 1 second in each case; hence these times are not mentioned in Table 1. (Note that the CPU time for the ILP solver used in [6] does not increase monotonically with W.)

The testing times for d695 obtained using the proposed method are comparable to the testing times obtained using the ILP-based method in [6]. For p22810, however, the new method yields a significantly-lower SOC testing time. This is because the problem instance size is larger and the Philips SOC has a larger number of cores; thus our rectangle packing heuristics have more room for rectangle manipulation and height-width optimization in this case. The values of $\mathcal{E}$ shown for p22810 in Table 1 are for two TAMs. This is because the ILP models for p22810 were particularly intractable, and the ILP method [6] did not run to completion for three or more TAMs, even after two days of execution. The CPU time of our new algorithm is several orders of magnitude lower than the CPU times required by the method in [6]; the execution speed-up factor can in fact be estimated from the values of $\mathcal{E}$ in Table 1, since the new algorithm takes less than 1 second to execute in each case.

Table 2 presents results of wrapper/TAM co-optimization for SOCs p34392 and p93791. The values of $\mathcal{T}$ shown for p34392 are for three TAMs. (For four TAMs, the ILP method of [6] did not provide a solution even after two days of CPU time.) For p34392, we reach the optimum (lower bound) testing time of 544579 cycles at $W = 32$. This lower bound corresponds to the time taken to test the bottleneck core, Core 18, when it is supplied with a TAM width equal to its highest Pareto-optimal point. An expression for this lower bound on the

	SOC p34392				SOC p93791			
	Method of [6]		New method		Method of [6]		New method	
W	$\mathcal{T}$ (cc)	$\mathcal{E}$ (sec)	$\mathcal{T}_{new}$ (cc)	$\Delta\mathcal{T}$ (%)	$\mathcal{T}$ (cc)	$\mathcal{E}$ (sec)	$\mathcal{T}_{new}$ (cc)	$\Delta\mathcal{T}$ (%)
16	998733	222	1023820	+2.51	1771720	25	1851135	+4.48
24	720858	325	759427	+5.35	1187990	50	1248795	+5.12
32	591027	1576	544579	-7.86	887751	85	975016	+9.83
40	544579	1081	544579	0.00	698583	130	794020	+13.66
48	544579	6198	544579	0.00	599373	210	627934	+4.77
56	544579	11331	544579	0.00	514688	270	568436	+10.44
64	544579	1125	544579	0.00	460382	440	511286	+11.06

Table 2. Results for p34392 and p93791.

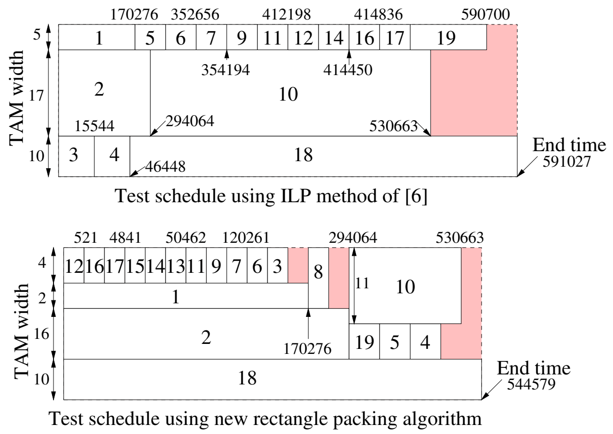

Figure 13. Test schedules for p34392 using [6] and the new method (figures not drawn to scale).

system testing time for an SOC was derived in [2]. The ILP-based method requires a total TAM width of 40 to reach this lower bound. Note that the values of $\mathcal{E}$ do not increase monotonically with W because the time taken by the ILP tool to solve the $\mathcal{NP}$-hard problems in [6] varies significantly with the problem instance; different width partitions for the same W value result in widely-varying CPU times. In Figure 13, we present the test schedules obtained for p34392 for $W = 32$ to further illustrate the difference between the way TAM width is allocated to cores by the method of [6], and by the new algorithm. The numbers between 1 and 19 in the rectangles of Figure 13 denote cores.

The new testing times for p93791 (the largest example SOC having the most cores) are on average 8% higher than the testing times obtained using the method in [6]. A careful study of the test schedules of Table 1 and the number of I/Os, test patterns and scan chain lengths of the cores in p93791 reveals that the vast differences between the test data requirements of the cores make it difficult for the proposed algorithm to optimize the system testing time using only a single value of p for all cores. We therefore added an additional heuristic to our algorithm to better allocate TAM resources to cores based on their test data volume. Rectangles for cores that have higher test data volume are packed using a lower value of p. Significantly, this new heuristic resulted in a further decrease in testing time for $W = 32$. The new values of $\mathcal{T}_{new}$ and $\Delta\mathcal{T}$ for $W = 32$ are as follows. $W = 32$: $\mathcal{T}_{new} = 940916$, $\Delta\mathcal{T}$ = +5.99%. The new decreased testing time for $W = 32$ motivates further investigation into how the value of p affects testing time for each core, and how it should be tailored to the test data needs of each individual core for larger SOCs such as p93791.

5 Conclusion

We have presented a new technique based on rectangle packing for wrapper/TAM co-optimization and test scheduling for SOCs. TAM widths have been tailored to the test data needs of cores through the use of Pareto-optimal points. We have also presented several heuristics that minimize the idle time on TAM wires, thereby leading to a fast and efficient algorithm for TAM width allocation and test scheduling. The new algorithm based on generalized rectangle packing is scalable for large industrial SOCs and completes in less than a second of CPU time. This represents several orders of magnitude improvement over exact methods for TAM optimization presented in earlier work. In addition, for two out of the three large SOCs, we obtained lower testing times for several values of W. Finally, the selection of the p, d values and the specific idle-time filling heuristics used for each execution of the algorithm can be included as user-programmable options in an industrial CAD tool implementation.

Acknowledgements

The usage of Pareto-optimal TAM widths and rectangle packing was inspired by discussions with Jan Korst and Sandeep Koranne, respectively, for which we gratefully acknowledge them. We also thank Graeme Francis, Harry van Herten and Erwin Waterlander for their help with providing data for the Philips SOCs. We thank Joep Aerts and Sandeep Goel for their useful comments on an earlier version of this paper. Finally, we acknowledge Kristian Schalter for discussions on Pareto-optimality.

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] 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.

[3] R. M. Chou, K. K. Saluja and V. D. Agrawal. Scheduling tests for VLSI systems under power constraints. *IEEE Trans. VLSI Systems,* vol. 5, no. 2, June 1997.

[4] E. G. Coffman, Jr., M. R. Garey, D. S. Johnson and R. E. Tarjan. Performance bounds for level-oriented two-dimensional packing algorithms. *SIAM J. Computing*, vol. 9, pp. 809–826, 1980.

[5] Y. Huang et al. Resource allocation and test scheduling for concurrent test of core-based SOC design. *Proc. Asian Test Symp.*, pp. 265-270, 2001.

[6] 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, pp. 211–228, March 2002.

[7] V. Iyengar, K. Chakrabarty, and E. J. Marinissen. Efficient wrapper/TAM co-optimization for large SOCs. *Proc. Design Automation and Test in Europe (DATE) Conf.*, 2002, in press.

[8] V. Iyengar and K. Chakrabarty. Test bus sizing for system-on-a-chip. *IEEE Trans. Computers*, vol. 51, May 2002, in press.

[9] S. Koranne. On test scheduling for core-based SOCs. *Proc. Int. Conf. VLSI Design*, pp. pp. 505–510, 2002.

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

[11] E. Larsson and Z. Peng. Test scheduling and scan-chain division under power constraint. *Proc. Asian Test Symp.*, pp. 259-264, 2001.

[12] 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.

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

[14] E.J. Marinissen, V. Iyengar and K. Chakrabarty. ITC 2002 SOC benchmarking initiative. *http://www.extra.research.philips.com/itc02socbenchm*

[15] 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.