An SoC Test Scheduling Algorithm using Reconfigurable Union Wrappers

Tomokazu Yoneda†, Masahiro Imanishi† and Hideo Fujiwara†
†Graduate School of Information Science, Nara Institute of Science and Technology
Kansai Science City, 630-0192, Japan
{yoneda, fujiwara}@is.naist.jp

Abstract
This paper presents a reconfigurable union wrapper that can wrap multiple cores into a single wrapper design. Moreover, we present a test scheduling algorithm to minimize a test application time using the proposed reconfigurable union wrapper. The proposed heuristic algorithm can achieve short test application time with low computational cost compared to the conventional approaches where every core has its own wrapper. Experimental results for the ITC’02 SOC Benchmarks show the effectiveness of our approach.

keywords: system-on-a-chip, test scheduling, reconfigurable union wrapper, test access mechanism

1 Introduction
In the SoC test environment, each embedded core is isolated from other logics during test of the core. Therefore, the following three components are necessary for SoC testing: 1) test pattern source and test response sink, 2) test access mechanism (TAM), and 3) wrapper [1]. The TAM propagates test patterns for a core from test pattern source to the core, and furthermore propagates the responses from the core to test pattern sink. The wrapper provides functions for cores to switch the mode of the cores: 1) normal, 2) INTEST, 3) EXTEST, and 4) BYPASS defined in IEEE 1500 standard [2]. The goal for SoC test is to develop techniques for wrapper design, TAM design and test schedule that minimizes test application time under given constraints. A number of approaches have addressed wrapper design [3, 4, 5]. Several TAM architectures have been proposed such as TestBus [6, 7], TESTRAIL [8], transparency based TAMs [9, 10, 11]. However, wrapper and TAM co-optimization problem was shown to be NP-hard in [3]. Therefore, many heuristic approaches for this problem have been proposed [4, 5, 12, 13, 14, 15, 16, 17].

Recently, the reconfigurable wrapper was proposed by Koranne [18] that allows a core to have several wrapper configurations in contrast to the previous approaches. The advantage is the increased flexibility for the test scheduling. Larsson and Peng proposed the reconfigurable power-conscious core test wrapper to increase the flexibility for power-constrained test scheduling [19].

One crucial aspect of these approaches including reconfigurable and non-reconfigurable wrapper designs is that each core has an own wrapper and a core is tested independently of other cores. Quasem et al. showed that further reductions in test application time can be achieved if multiple cores are wrapped into single wrapper design and the overlapped test application scheme is used [20]. They also proposed the reconfigurable wrapper design that includes all embedded cores in a single wrapper. In the proposed reconfigurable wrapper design, the length of each wrapper scan chain can be reconfigured (reduced) during test by bypassing scan chains which are not necessary for further test application.

However, wrapping all the cores into a single wrapper design cannot always achieve the minimum test application time due to the lack of flexibility for test scheduling. Moreover, the proposed reconfigurable wrapper can only bypass scan chains in each wrapper scan chain, and cannot reassign scan chains to different wrapper scan chains.

This paper presents a reconfigurable union wrapper that can wrap multiple cores into a single wrapper design. Moreover, we present a test scheduling algorithm to minimize a test application time using the proposed reconfigurable union wrapper. The proposed algorithm finds the optimal core groups for designing reconfigurable union wrappers, and it can achieve short test application time with low computational cost. Experimental results for the ITC’02 SOC Benchmarks show the effectiveness of our approach.

The rest of this paper is organized as follows. We discuss our motivation in Section 2. Section 3 shows a reconfigurable union wrapper design. After formulating a test scheduling problem using reconfigurable union wrappers in Section 4, we present a test scheduling algorithm in Section 5. Experimental results are discussed in Section 6. Finally, Section 7 concludes this paper.

2 Motivation
In this section, we examine the effectiveness of the reconfigurable union wrapper for minimizing test application time by using example cores shown in Fig.1. A significant amount of research has been devoted to reducing test application time. Consequently, several test scheduling algorithms have been proposed. In these approaches, every core has its own wrapper and cores are tested independently of
others by using the wrappers. Wrapper designs for core 1 and 2 using three wrapper scan chains are shown in Fig.1(a) and Fig.1(b), respectively. The time required to apply the entire test set to a core is given as follows.

\[ T = (1 + \max\{s_i, s_o\}) \cdot p + \min\{s_i, s_o\} \]  

Here, \( p \) is the number of test patterns, and \( s_i(s_o) \) is the length of the longest wrapper scan-in(scan-out) chain for the core. In this example, test time for core 1 and core 2 are 550 and 605, respectively. The total test time for these two cores is 1165 if core 1 and core 2 are tested serially by using a shared TAM with three bits.

Fig.2 shows the proposed reconfigurable union wrapper design for core 1 and core 2. The wrapper design has two test configurations. The first configuration shown in Fig.2(b) includes all scan chains, and core 1 and core 2 are tested at the same time for first 50 patterns. After applying the first 50 patterns, the wrapper is reconfigured to configuration 2 shown in Fig.2(c) where only scan chains in core 2 are connected to the wrapper scan chains. Then, core 2 is tested using configuration 2 for the remaining 50 patterns. Test time for configuration 1 and 2 are 560 and 305, respectively. The total test time for these two cores is 860 since the first scan-in in configuration 2 can be overlapped with the last scan-out in configuration 1. Consequently, we can achieve 26% test time reduction by using the proposed reconfigurable union wrapper. Therefore, we can reduce the total SoC test time if we can find the optimal groups for the proposed reconfigurable union wrappers during test scheduling.

### 3 Reconfigurable Union Wrapper Design

In this section, we present a method to design the reconfigurable union wrapper. Let \( S \) be the set of cores and \( w \) be the number of wrapper scan chains. The proposed procedure is shown in Fig.3. First, configuration ID \( i \) is set to 1 (line 1). Then, configuration \( i \) is designed by WrapperDesign procedures proposed in [3] by regarding cores in \( S \) as a single core (line 3). After that, the cores with the smallest number of test patterns are removed from \( S \) (line 4), and configuration ID \( i \) is increased by 1 (line 5). The above process is repeated until \( S \) becomes empty. An example of the reconfigurable union wrapper for core 1 and core 2 shown in Fig.1 and its configurations are shown in Fig.2.

![Figure 1. Wrapper designs using three wrapper scan chains by [3].](a) wrapper design for core 1 (b) wrapper design for core 2](a) wrapper design for core 1 (b) wrapper design for core 2)

### 4 Problem Formulation

From our preliminary experiments, we observed that we cannot always achieve minimum test time by uniting all cores in a given SoC into one group and giving one proposed reconfigurable wrapper to them. The total SoC test time can be reduced if we can find the optimal groups for the proposed reconfigurable union wrappers during test scheduling. Therefore, we formulate the test scheduling problem using reconfigurable union wrappers as follows.

**Definition 1** \( P_{\text{raw}} \): Given the maximum TAM width \( W_{\text{max}} \), a set of cores \( M \) and for each core \( m \in M \) the test set parameters including the number of test patterns \( p_m \), the number of input terminals \( l_{m_i} \), the number of output terminals \( o_{m_o} \), the number of bidirectional terminals \( b_{m_b} \), the number of scan chains \( s_m \) and for each scan chain \( k \) the length of it \( l_{m_k} \), determine the test groups, the TAM width and the...
reconfigurable union wrapper design for each group, and a test schedule such that: (1) the total number of TAM wires utilized at any moment does not exceed $W_{\text{max}}$ and (2) the overall test time is minimized.

5 Scheduling Algorithm using Reconfigurable Union Wrapper

In this section, we present an algorithm for determining a solution to the test scheduling problem using reconfigurable union wrappers. In our algorithm, a TAM $b$ is represented as a set of cores which are connected to $b$. Our algorithm determines a solution which consists of

- the set of TAMs $B$ such that every core is assigned to exactly one TAM,
- the width $w(b)$ of every TAM $b \in B$ such that the summed widths of the TAMs does not exceed $W_{\text{max}}$,
- the reconfigurable union wrapper design for each TAM $b \in B$ such that total test time $T$ is minimized.

To determine the test time $t(b,w)$ for a TAM $b$ with width $w$, we assume the existence of a procedure $\text{TestTime}(b,w)$ which uses a procedure $\text{RUW}(b,w)$ for designing a reconfigurable union wrapper for cores in $b$ presented in Section 3.

An outline of this algorithm is presented in Fig.4. The algorithm is inspired by an existing effective algorithm (TR-Architect [14]), and it has three main steps which are ASSIGN, TRANSFER and MERGE.

```
procedure TestScheduling
1 sort $M$ in the descending order based on the amount of test data;
2 for all $m \in M$ in the decided order at line 1 |
3 ASSIGN($m$);
4 TRANSFER();
5 |
6 repeat until no more improvement on $T$ is possible |
7 MERGE();
8 TRANSFER();
9 |
```

Figure 4. Scheduling algorithm

First, it sorts cores in the descending order based on the amount of test data (line 1). The amount of test data of a core is equivalent to the test time provided by the wrapper design using one wrapper scan chain. Then, a core is assigned to a TAM such that current total test time $T$ is minimized by ASSIGN procedure (line 3). Whenever a core is assigned, TRANSFER procedure is executed (line 4). The TRANSFER procedure tries to minimize $T$ by finding a bottleneck TAM $b_{\text{max}}$ which is a TAM with the longest test time and re-assigning a core in $b_{\text{max}}$ to other TAM. TRANSFER repeats the above process until no more improvement on $T$ is possible. After assigning all cores to TAMs, the algorithm performs the MERGE procedure (line 7). The MERGE procedure tries to minimize $T$ by finding a bottleneck TAM $b_{\text{max}}$ and merging $b_{\text{max}}$ with other TAM. MERGE repeats the above process until no more improvement on $T$ is possible. Then, the TRANSFER procedure is executed again (line 8). These MERGE and TRANSFER procedures are repeated until no more improvement on $T$ is possible. Each of these steps is explained in more detail in the sequel of this section.

5.1 Assignment

The procedure ASSIGN as outlined in Fig.5 determines a TAM to which the current targeted core $m$ is assigned. It consists of three main steps.

```
procedure ASSIGN($m$)
1 /*Step1: Compute $T_{ex}$ when $m$ is assigned to the existing TAM */
2 find $b_{ex} \in B$ s.t. $t(b_{ex},w(b_{ex}))$ is minimized;
3 $T_{ex} := t(b_{ex},w(b_{ex}))$;
4 /*Step2: Compute $T_{new}$ when $m$ is assigned to new TAM */
5 generate new TAM $b_{new} := \{m\}$;
6 $T_{new} := w(b_{new})$;
7 compute $T_{tar}$ by [14];
8 for all $b \in B_{new}$ |
9 compute $w(b)$ s.t. $t(b,w(b))$ is closest to $T_{tar}$;
10 $T_{new} := t(b_{new},w(b_{new}))$;
11 $W := \sum_{b \in b_{new}} w(b)$;
12 while($W > W_{\text{max}}$) |
13 find $b_{new} \in B_{new}$ s.t. $t(b_{new},w(b_{new}) - 1)$ is minimized;
14 $w(b_{new}) := w(b_{new}) - 1$;
15 $W := W - 1$;
16 if($T_{new} < t(b_{new},w(b_{new}) - 1)$) |
17 $T_{new} := t(b_{new},w(b_{new}) - 1)$;
18 |
19 /*Step3: Update TAMs */
20 if($T_{ex} < T_{new}$) |
21 $T := T_{ex}$; $b_{ex} := b_{ex} \cup \{m\}$ for $b_{ex} \in B$;
22 else |
23 $T := T_{new}$; $B := B_{new}$.
```

Figure 5. ASSIGN procedure

In Step 1 (line 1-2), for each TAM $b \in B$ which already exists in the current test schedule, it computes the test time when the core $m$ is assigned to $b$ and a reconfigurable union wrapper for $b$ is designed. Then, it finds $b_{ex}$ which is a TAM with the shortest test time $T_{ex}$ after assigning $m$. In Step 2 (line 3-18), it computes the test time $T_{new}$ when a new TAM $b_{new}$ is created and the core $m$ is assigned to $b_{new}$. When $b_{new}$ is created, for each TAM $b \in B \cup b_{new}$, the width of $b$ is updated such that test time of $b$ is the closest to the lower bound $T_{tar}$ on test time proposed in [14] for the cores currently scheduled (line 6-8). Consequently, if the total TAM width $W$ exceeds $W_{\text{max}}$, the algorithm finds a TAM $b_{min}$ such that $t(b_{min},w(b_{min}) - 1)$ is minimized and the TAM width $w(b_{min})$ is updated. This process is repeated until $W$ does not exceed $W_{\text{max}}$ (line 12-18). In Step 3 (line 19-22), two solutions created in Step 1 and 2 are compared and the solution with smaller test time is selected.

Fig.6 shows an example of the ASSIGN procedure. In this example, it tries to assign core 4 for the current test schedule shown in Fig.6(a). Fig.6(b) and (c) show the results of Step 1 and 2, respectively. Consequently, core 4 is
assigned to TAM 2 since the solution shown in Fig.6(b) has smaller test time.

5.2 Transfer

The procedure TRANSFER as outlined in Fig.7 tries to minimize the total test time $T$ by finding a bottleneck TAM $b_{\text{max}}$ and re-assigning a core in $b_{\text{max}}$ to other TAM.

```plaintext
procedure TRANSFER()
1 repeat until no more improvement on $T$ is possible{
2 Find a bottleneck TAM $b_{\text{max}}$;
3 $T_{\text{max}} := t(b_{\text{max}}, w(b_{\text{max}}))$;
4 for all core $s \in b_{\text{max}}$ {
5 for all TAM $b \in B - \{b_{\text{max}}\}$ {
6 $T_{\text{temp}} := \infty$;
7 $b_1 := b \cup \{s\};$ $b_2 := b_{\text{max}} - \{s\}$;
8 $w_{b_1} := w(b_1)$; $w_{b_2} := w(b_2)$;
9 repeat until no more improvement on $T_{\text{temp}}$ is possible{
10 if($T_{\text{temp}} > \max(t(b_1, w_{b_1} + 1), t(b_2, w_{b_2} - 1))$) {
11 $T_{\text{temp}} := \max(t(b_1, w_{b_1} + 1), t(b_2, w_{b_2} - 1))$;
12 $w_{b_1} := w_{b_1} + 1; \; w_{b_2} := w_{b_2} - 1;$
13 }
14 }
15 if($T_{\text{temp}} < T_{\text{max}}$) {
16 $T_{\text{max}} := T_{\text{temp}}$; $b_{\text{min}} := b; \; s_{\text{min}} := s$;
17 $w(b_{\text{min}}) := w(b_{\text{min}}) + w(b_{\text{max}})$;
18 }
19 }
20 if($T_{\text{max}} < T$) {
21 $b_{\text{min}} := b_{\text{min}} \cup \{s_{\text{min}}\}$; $b_{\text{max}} := b_{\text{max}} - \{s_{\text{min}}\}$;
22 $w(b_{\text{min}}) := w(b_{\text{min}})$; $w(b_{\text{max}}) := w(b_{\text{max}})$;
23 $T := T_{\text{max}}$;
24 }
25 }
26 }
```

Figure 7. TRANSFER procedure

First, it finds a bottleneck TAM $b_{\text{max}}$ and computes its test time $T_{\text{max}}$ (line 2-3). Then, for each core $s \in b_{\text{max}}$, it computes the test time $T_{\text{temp}}$ when $s$ is removed from $b_{\text{max}}$ and re-assigned to other TAM $b \in B - \{b_{\text{max}}\}$ (line 4-20). During this step, TAM width of $b_{\text{max}}$ is also reduced from $b_{\text{max}}$ and re-assigned to $b$ until no more improvement on $T_{\text{temp}}$ is possible (line 9-14). After that, if it finds a core $s_{\text{min}} \in b_{\text{max}}$ and $b_{\text{min}} \in B - \{b_{\text{max}}\}$ such that test time $T_{\text{temp}}$ is minimized when $s_{\text{min}}$ is re-assigned to $b_{\text{min}}$ (line 15-20). If the re-assignment gives smaller test time than $T$ which is the current total test time, then it updates the solution.

Figure 8. An example of TRANSFER procedure.

5.3 Merge

The procedure MERGE as outlined in Fig.9 tries to minimize the total test time $T$ by finding the bottleneck TAM $b_{\text{max}}$ and merging $b_{\text{max}}$ with other TAM.

```plaintext
procedure MERGE()
1 repeat until no more improvement on $T$ is possible{
2 Find a bottleneck TAM $b_{\text{max}}$;
3 $T_{\text{max}} := t(b_{\text{max}}, w(b_{\text{max}}))$;
4 for all $b \in B - \{b_{\text{max}}\}$ {
5 $b_{\text{temp}} := b \cup b_{\text{max}}$;
6 $w_{\text{temp}} := w(b) + w(b_{\text{max}})$;
7 if($t(b_{\text{temp}}, w_{\text{temp}}) < T_{\text{max}}$) {
8 $T_{\text{max}} := t(b_{\text{temp}}, w_{\text{temp}})$; $b_{\text{min}} := b$;
9 }
10 }
11 if($T_{\text{max}} < T$) {
12 $w(b_{\text{min}}) := w(b_{\text{min}}) + w(b_{\text{max}})$;
13 $b_{\text{min}} := b_{\text{min}} \cup b_{\text{max}}$;
14 $B := B - \{b_{\text{max}}\}$;
15 $T := T_{\text{max}}$;
16 }
17 }
```

Figure 9. MERGE procedure

First, it finds a bottleneck TAM $b_{\text{max}}$ and computes its test time $T_{\text{max}}$ (line 2-3). Then, for each core $b \in B - \{b_{\text{max}}\}$, it computes the test time when $b_{\text{max}}$ is merged with $b$ (line 4-10). After that, if it finds a TAM $b_{\text{min}} \in B - \{b_{\text{max}}\}$ such that test time $T_{\text{max}}$ is minimized when $b_{\text{max}}$ is merged with $b_{\text{min}}$ (line 15-20). If the merged TAM gives smaller test time than the current total test time $T$, then it updates the
In this example, the bottleneck TAM 3 is merged with TAM 1. Improvement on its configurations. Table 2 shows the area overhead computed by the following equation.

\[
\text{overhead} = \frac{\sum (\text{gates of additional multiplexers})}{\sum (\text{gates of 1500 wrapper cells})} \times 100 \tag{2}
\]

Please note that the area overhead is only to the logic introduced by IEEE 1500 wrapper cells, not to the overall SoC. In other words, the area overhead shows the increase ratio caused by IEEE 1500 wrapper cells, not to the overall SoC. Therefore, the area overhead will become insignificant for complex cores with hundreds of thousands of gates.

6 Experimental Results

Table 1 compares the test time results obtained using the proposed method with those using the test scheduling methods proposed by Im et al. [5], Xia et al. [16] and Zou et al. [4]. We have chosen the methods presented in [5, 16, 4] for comparison since they have been applied to the ITC'02 benchmarks [21] and give the best results to the best of our knowledge. Here, we considered the ITC'02 benchmarks SoCs to be flat. In 23 out of the 28 cases (7 TAM widths for 4 benchmarks) our method can find the solution with the smallest test time.

In Table 1, we also show the lower bound on test time proposed in [14] in order to discuss the potential advantage of the proposed reconfigurable union wrapper. We can observe that our method finds the solution with smaller test time than the lower bound in some cases. This indicates the potential advantage of the proposed reconfigurable union wrapper for test time minimization.

7 Conclusion

In this paper we have proposed a reconfigurable union wrapper and test scheduling algorithm using it. We have shown that by allowing multiple cores to have a single wrapper, test time can be reduced compared to the approaches where every core has its own wrapper. Moreover, our method finds the solution with smaller test time than lower bound proposed in [14] in some cases. This indicates the potential advantage of the proposed reconfigurable union wrapper for test time minimization.

Acknowledgments

This work was supported in part by Japan Society for the Promotion of Science (JSPS) under Grants-in-Aid for Scientific Research B(No. 15300018) and for Young Scientists B(No.18700046). The authors would like to thank Profs. Kewal K. Saluja, Michiko Inoue, Dr. Satoshi Ohtake and members of Fujiwara Laboratory (Nara Institute of Science and Technology) for their valuable comments.

References

Table 1. Test time results (#cycles).

<table>
<thead>
<tr>
<th>soc</th>
<th>approach</th>
<th>16</th>
<th>24</th>
<th>32</th>
<th>40</th>
<th>48</th>
<th>56</th>
<th>64</th>
</tr>
</thead>
<tbody>
<tr>
<td>d695</td>
<td>our method</td>
<td>40933</td>
<td>27456</td>
<td>20789</td>
<td>16935</td>
<td>14144</td>
<td>11978</td>
<td>10571</td>
</tr>
<tr>
<td></td>
<td>RA[5]</td>
<td>41442</td>
<td>27725</td>
<td>20948</td>
<td>16852</td>
<td>14182</td>
<td>12084</td>
<td>10628</td>
</tr>
<tr>
<td></td>
<td>EA[16]</td>
<td>41553</td>
<td>27982</td>
<td>21014</td>
<td>16908</td>
<td>14240</td>
<td>11988</td>
<td>10571</td>
</tr>
<tr>
<td></td>
<td>SA[4]</td>
<td>41604</td>
<td>28064</td>
<td>21161</td>
<td>16993</td>
<td>14182</td>
<td>12085</td>
<td>10723</td>
</tr>
<tr>
<td></td>
<td>Lower Bound[14]</td>
<td>40951</td>
<td>27305</td>
<td>20482</td>
<td>16388</td>
<td>13695</td>
<td>11709</td>
<td>10247</td>
</tr>
<tr>
<td>p22810</td>
<td>our method</td>
<td>420299</td>
<td>283391</td>
<td>214434</td>
<td>169891</td>
<td>143756</td>
<td>123721</td>
<td>114579</td>
</tr>
<tr>
<td></td>
<td>RA[5]</td>
<td>422345</td>
<td>284632</td>
<td>214354</td>
<td>173637</td>
<td>145781</td>
<td>126548</td>
<td>112600</td>
</tr>
<tr>
<td></td>
<td>EA[16]</td>
<td>438783</td>
<td>292824</td>
<td>226545</td>
<td>167792</td>
<td>153260</td>
<td>133094</td>
<td>117638</td>
</tr>
<tr>
<td></td>
<td>SA[4]</td>
<td>438619</td>
<td>289287</td>
<td>218855</td>
<td>175946</td>
<td>147944</td>
<td>125947</td>
<td>109591</td>
</tr>
<tr>
<td></td>
<td>Lower Bound[14]</td>
<td>419466</td>
<td>279644</td>
<td>209734</td>
<td>167787</td>
<td>139823</td>
<td>119848</td>
<td>104868</td>
</tr>
<tr>
<td>p34392</td>
<td>our method</td>
<td>924846</td>
<td>625963</td>
<td>544579</td>
<td>544579</td>
<td>544579</td>
<td>544579</td>
<td>544579</td>
</tr>
<tr>
<td></td>
<td>RA[5]</td>
<td>939855</td>
<td>625543</td>
<td>544579</td>
<td>544579</td>
<td>544579</td>
<td>544579</td>
<td>544579</td>
</tr>
<tr>
<td></td>
<td>EA[16]</td>
<td>939855</td>
<td>641514</td>
<td>544579</td>
<td>544579</td>
<td>544579</td>
<td>544579</td>
<td>544579</td>
</tr>
<tr>
<td></td>
<td>SA[4]</td>
<td>944768</td>
<td>628602</td>
<td>544579</td>
<td>544579</td>
<td>544579</td>
<td>544579</td>
<td>544579</td>
</tr>
<tr>
<td></td>
<td>Lower Bound[14]</td>
<td>932790</td>
<td>621903</td>
<td>544579</td>
<td>544579</td>
<td>544579</td>
<td>544579</td>
<td>544579</td>
</tr>
<tr>
<td>p93791</td>
<td>our method</td>
<td>1739032</td>
<td>1157239</td>
<td>868825</td>
<td>696522</td>
<td>580936</td>
<td>500200</td>
<td>438141</td>
</tr>
<tr>
<td></td>
<td>RA[5]</td>
<td>1742995</td>
<td>1157974</td>
<td>870059</td>
<td>701204</td>
<td>587907</td>
<td>509976</td>
<td>441786</td>
</tr>
<tr>
<td></td>
<td>EA[16]</td>
<td>1755490</td>
<td>1171190</td>
<td>886038</td>
<td>706820</td>
<td>600986</td>
<td>501057</td>
<td>445748</td>
</tr>
<tr>
<td></td>
<td>SA[4]</td>
<td>1754452</td>
<td>1169495</td>
<td>878493</td>
<td>718005</td>
<td>594575</td>
<td>509041</td>
<td>447974</td>
</tr>
<tr>
<td></td>
<td>Lower Bound[14]</td>
<td>1746657</td>
<td>1164442</td>
<td>873334</td>
<td>698670</td>
<td>582227</td>
<td>499053</td>
<td>436673</td>
</tr>
</tbody>
</table>

Table 2. Area overhead (%) to IEEE wrapper cells.

<table>
<thead>
<tr>
<th>soc</th>
<th>16</th>
<th>24</th>
<th>32</th>
<th>40</th>
<th>48</th>
<th>56</th>
<th>64</th>
</tr>
</thead>
<tbody>
<tr>
<td>d695</td>
<td>33.84</td>
<td>28.82</td>
<td>32.98</td>
<td>27.65</td>
<td>21.26</td>
<td>22.55</td>
<td>29.62</td>
</tr>
<tr>
<td>p22810</td>
<td>23.16</td>
<td>35.41</td>
<td>25.47</td>
<td>31.47</td>
<td>32.68</td>
<td>37.53</td>
<td>38.82</td>
</tr>
<tr>
<td>p34392</td>
<td>5.84</td>
<td>3.75</td>
<td>2.43</td>
<td>2.43</td>
<td>2.43</td>
<td>2.43</td>
<td>2.43</td>
</tr>
<tr>
<td>p93791</td>
<td>109.42</td>
<td>114.38</td>
<td>96.90</td>
<td>128.88</td>
<td>159.88</td>
<td>136.33</td>
<td>86.46</td>
</tr>
</tbody>
</table>


