Abstract

This paper addresses an efficient NoC-based SoC test scheduling method based on a rectangle packing algorithm. In order to apply the proposed ideas to NoC-based systems, we designed a new test platform suitable for NoC structures. Furthermore, we extended the proposed algorithm considering multiple test clocks and power constraints. Experimental results using some ITC’02 benchmark circuits show the proposed algorithm can reduce the overall test time by up to 50% and 20% on the average in comparison with previous works.

I. Introduction

NoC (Networks-on-Chip) is an emerging design paradigm intended to cope with a future SoC containing numerous built-in cores and can be defined as an interconnection model implemented on a chip in the form of a micro-network [1].

The test strategy is a significant factor in the practicality and feasibility of NoC-based SoCs. NoC-based SoCs have nearly the same test methodologies as common SoCs. However, some test issues incorporating the NoC characteristics remain as unresolved problems under investigation. In particular, TAM (Test Access Mechanism) is one of the most active research areas in testing NoC-based SoCs. Among earlier TAM architectures for SoCs, an on-chip test bus has been the most efficient form. However, it may be impractical to have TAM solely for the purpose of testing only in NoC-based SoCs because test costs such as for the silicon area and pin count will be much higher. Thus, the reuse of NoCs as TAMs is a very attractive and logical goal. In NoC-based TAM, all test data should come in a packet type. For this reason, it is difficult to assign TAM pins to cores by a bit scale. This constraint makes many test scheduling ideas based on common SoC structures not be directly applicable to testing NoC-based SoCs. Therefore, an efficient test scheduling method using NoC-based TAM is important to minimize the overall system test time.

In this paper, we propose a novel NoC-based SoC test scheduling reusing the NoC for data communications as TAM with power constraints. Firstly, to apply the proposed scheduling method to NoC-based structures, we designed a new NoC test platform including test packet generation, transmission and routing. This test platform helps us to implement an efficient test scheduling minimizing the test time loss due to the packet-style transmission of test vectors. Though any heuristic approach proposed for SoC tests can be applied to solve the scheduling problem on the aforementioned test platform, we focused on the geometric bin packing approaches [2–6] for their simplicity and feasibility. In addition, we extended the proposed scheduling algorithm by factoring in multiple test clocks and power constraints.
II. Related Work

The general concept of testing NoC-based SoCs was first shown in [7]. Before the built-in core test, the communication infrastructure of a NoC should be tested. The primary issues and methodologies regarding testing communication resources are well introduced in [8]. After verifying the communication resource, we can advance to the standard core test using an on-chip network as TAM. An early approach on the subject of test architectures utilizing an on-chip network was shown in [9].

Test scheduling algorithms for NoC-based SoCs can be grouped roughly into two main categories: packet-based and core-based scheduling. Packet-based method determines the order of generation and transmission of test packets for cores according to the priority of each core. Cota et al have proposed the test scheduling based on a packet-switching protocol [10]. Test vectors and responses per core are represented as a set of packets to be transmitted throughout the network, and the packets are scheduled to minimize the total test time using test parallelism. “Test parallelism” denotes that several cores are tested simultaneously, improving the test efficiency through fully utilizing the network bandwidth. Enhanced versions of this algorithm have been reported, such as the addition of power constraints [11]. Embedded processors have been used for test sources and sinks to increase test parallelism [12].

Core-based scheduling determines the test order of each core [13–14]. In this approach, the scheduler assigns each core a routing path, including an input port, an output port and corresponding channels that transport test vectors from the input to the core and test responses from the core to the output. Once the core is scheduled on this path, all resources on the path are reserved for the core test until the entire test is completed. Since the proposed idea maintains the test pipelining from a test vector input to a test response output for a CUT (Core under Test), it shows fairly good scheduling results. The term “test pipelining” means the continuous, concurrent operation of scan input of a test vector and output of a response. As the collision among the routed test data causes the test pipelining to be broken, the corresponding routing path for each CUT has to be reserved. However, the reservation of a path reduces the test parallelism.

Recently, a new test scheduling idea was proposed improving the limitation of NoC-based TAM. In [14], the idle channel width of TAM that cannot contribute to reducing the overall system test time is efficiently utilized through a combination of an on-chip clocking and parallel-serial conversion of test data. In order to preserve the test pipelining, however, the test clock generated from an on-chip PLL for a CUT should increase in proportion to the number of test vector transmitted together through a test packet. Considering the difficulty of an on-chip PLL design, we cannot help but limit the frequency and number of test clocks generated by the PLL, and thereby idle channel widths still remain.

III. NoC-Based Test Platform

(1) Test Resources and Configuration

In this study, we assumed a NoC which uses a 2-D mesh topology. A test resource denotes any specific logic required for a test operation. In this paper, a test source, sink, and a controller are mainly used for test resources. A test source can generate test vectors at a rate of up to one packet per network time step and a test sink can absorb test responses at the same rate as the test source. While a multi-source/sink mode has been used in previous studies [10–14], we adopt a single test source and sink attached directly to a router. Since test data should move on a network in a packet type,
we assumed the test source and sink include an NI internally for network routing. A test controller assures that test sources and sinks satisfy the test pipelining and parallelism by means of a predetermined test schedule. The number of test sources and sinks in a multi-source/sink mode corresponds to the difference of clock rate between an on-chip network and CUTs in a single source and sink mode. For example, if an on-chip network operates two times faster than the CUT, it produces the same effect as if there were two test sources and two sinks.

(2) Generation and Transmission of Packets

Common SoC test scheduling achieves minimal test time by assigning proper TAM width to each core. However, because NoC-based TAMs are not reconfigurable, we use the idea found in [14] to vary the size of TAM width available for scheduling. A packet can transfer multiple test vectors if the packet size is multiple times larger than the vector size [14]. The test vector size is identical to the TAM width assigned to a core. Such a packet generation scheme can reduce the overall test time by efficiently utilizing the network bandwidth in spatial domain.

Next, we propose a test packet transmission method including multiple vectors without test clock multiplications used in [14]. An example of the proposed packet transmission is shown in Figure 1. We assumed that the channel width of the network is 32-bit and CUT1, CUT2, and CUT3 are assigned to 16-, 8-, and 8-bit TAM widths, respectively. CLKN is a network clock, CLKT is a CUT test clock and all clocks have the same frequency. First, P1 for CUT1 is transmitted at cycle t1. Since P1 has two test vectors, the test source can deliver P2 at the next t2 cycle. In t3, P1 should be sent again for the test pipelining of CUT1. P3 is delivered at t4. The above process is repeated until the test of anyone of the CUTs is done. In this paper, we confine the TAM width to the power of 2, i.e. 1, 2, ..., 2^k (2^k ≤ W, W is the channel width of a network) for the simplicity of calculating of transmission cycles.

(3) Routing Strategy of Packets

Instead of the path reservation that has been used in the past, a new test packet routing method using its routing characteristics will be presented in this section.

Test vector packets from the test source to CUTs show a “one-to-many” communication pattern. In a one-to-many pattern, just one node is identified as a sender and the other nodes are receivers. If one sender transmits test packets in order and they move with XY routing, there will be no collision between test vector packets. However, the test response packets show a “many-to-one” pattern because there is one test sink receiver, but many CUT senders. Since the transmission time of each CUT is not uniform, response packet collision should be expected. In order to solve this problem, we use the global combining method [15]. Global combining removes packet collision by combining different packets into one packet. XY routing is still valid even when global combining is used.

Figure 2 illustrates the process of packet routing using global combining. All basic conditions in Figure 2 are equal to those of Figure 1. When CUT1 generates a response packet, P1, the response value will be located on the upper 16bit of P1. In the case of P2, the
response is located on the upper 8 bits within the lower 16 bits. P3 will use the other 8 bits for response data. A packet header should include information indicating the location of a real response. If P1, P2, and P3 arrive at the router, R3, simultaneously, R3 combines the input packets into one output packet, and forwards it. Since a test scheduler makes the total sum of TAM widths of CUTs tested at the same time less than the channel width of the network, it is very efficient to use global combining to solve the packet collision problem.

In spite of this routing scheme, it is possible that a test vector packet will collide with a response packet. This problem can be solved by placing a test source and sink on the opposite side of a row or column in a NoC and not installing another core in the row or column.

Fig. 2. Test Packet Routing with Global Combining

IV. Test Scheduling Based on a RP

(1) Problem Formulation

NoC-based SoC test scheduling can be formulated in terms of a rectangle packing problem if the test platform described in section III is used. The test time varies with TAM width in a staircase pattern, and the testing of a core is represented as a rectangle whose height indicates the TAM width assigned to that core; width denotes the test time of the core for the corresponding value of the TAM width. Thus, we can obtain a number for TAM width and the test time combinations for the same core. Taken as a whole, a test scheduler picks up just one rectangle from the candidate rectangle set of each core, and then packs it into a bin of a fixed height and an unlimited width until the bin is filled with rectangles of all cores embedded in a SoC, while minimizing the overall width of the bin without overflowing the bin’s height.

We now take up the scheduling problem in terms of a rectangle packing modeling. First, we assumed that a NoC-based SoC N includes m cores and has the channel width W. Also, let \( w_i \) be the TAM width assigned to core \( i \) embedded to \( N \) and \( t_i \) be the test time of core \( i \) for \( w_i \). All rectangles of core \( i \), \( R_i \), are represented as a ordered pair such that

\[
R_i = (w_i(k), t_i(k)), \quad 1 \leq i \leq m, \quad 1 \leq 2^k \leq W \tag{1}
\]

where \( k \) denotes the number of TAM widths available for a time-division transmission. Note that \( R \) is Pareto-optimal. The scheduling problem can be summarized as selecting one rectangle \( r_i \) from \( R_i \) (1 \( \leq i \leq m \)), packing the selected rectangles into a bin of height \( W \) and unbounded width without overlapping between the \( r_i \) rectangles and minimizing the width of the bin.

Up to this point, the scheduling problem has had the same clock rate for the on-chip network and the core test clocks. However, as a single source/sink is used in this paper instead of multi-source/sinks, we can extend the problem to a multiple bin packing problem. Let the network clock frequency be \( f_N \) and the test clock frequency of the cores be \( f_T \). Further, let \( f_N \) be \( n \) times faster than \( f_T \). Note that \( n \) is an integer value. Since a test scheduler can operate \( n \) times more within a period of test clock and the overall test time is measured by the test clock, we regard \( n \) as the number of bins to be packed by the core rectangles. In this multiple bin packing problem, we determine that a rectangle cannot be separated into multiple bins for a simple calculation of a packet transmission time.
(2) Power Constraints

During the testing, the peak power cannot be over the power limit of system. Generally, there are two sources of power consumption in a NoC-based SoC: the communication resources such as routers and channels, and the cores. In this paper, we used the power consumption model introduced in [11]. However, we limited ourselves to considering the power consumption at a core, as the power consumption of communication resources is difficult to estimate at this time. The power consumption of a core $i$, $P_i$, can be represented as

$$P_i = C_L \cdot V_{dd}^2 \cdot \frac{1}{T} \cdot \left( (\sigma_{gf} + 1) \cdot nbf_i + \sigma_{gf} \cdot nb_g \right)$$  \hspace{1cm} (2)$$

where $C_L$ is the load capacitance, $V_{dd}$ is the voltage applied to the core, $T$ is the test clock period, and $\sigma$ is the switching factor. $nbf_i$ and $nb_g$ denote the number of active flip-flops and gates in the core, respectively. To simplify the power calculation, we assume $P_i$ does not vary with the number of the wrapper scan chains and is the same over the entire time the core is tested.

(3) Test Scheduling Procedure

Before beginning the scheduling, we designed test wrappers for embedded cores and found $R$ using the One-Element Exchange algorithm [16]. Then, the heuristic procedure to solve the scheduling problem is advanced. While many ideas have been proposed, we chose and extended TAM_optimizer [3] for its simplicity and feasibility. In TAM_optimizer, first a scheduler calculates $w_{prefer}$ for each core through heuristic ideas. Then, cores assigned to $w_{prefer}$ are scheduled in succession on the basis of their test time. The scheduler also supplements cores to idle room or assigns additional TAM widths available to cores scheduled earlier to reduce the overall test time. While the basic methods are nearly the same as TAM_optimizer, some parts are added or modified. We now explain them in detail.

- Data Structure: The data structure of a core is shown in Figure 3. In the data structure, $bpos(i)$ denotes the position of bin including core $i$, and $clk_mode(i)$ is the test clock rate of core $i$ relative to the normal test clock. Note that we assumed that test clock frequencies of cores are uniform at the start-up. $pwr(\Delta)$ is the estimated power parameter of a core $i$.

    ![Fig. 3. Data Structure of Core](image)

- Calculation of $w_{prefer}$: The calculation procedure of $w_{prefer}$ is presented in Figure 4. Let $T(w)$ be the test time of core $i$ where a TAM width $w$ and $A_i(k)$ indicate the result of $w_i(k) \cdot T(w_i(k))$, $(1 \leq k \leq W)$. In our algorithm, $w_{prefer}(i)$ is set to the highest $w_i$ in $R$ such that the difference between $A_i(k)$ and $A_i(0)$ is less than the $p\%$ of $A_i(0)$. Furthermore, we assign a core the TAM width providing the corresponding core with the best test time if $T(w_{prefer})$ of the core is more than the $q\%$ of the total sum of $T(w_{prefer})$ of all embedded cores. The search space to find the best $p$ and $q$ is selected on an experimental basis. These heuristic approaches can make the proposed method applicable to various test circuits.

    ![Fig. 4. Preferred TAM Width Calculation](image)
- Multiple bin packing: In order to solve a multiple bin packing problem, we use an incremental packing on the basis of a cur_time. The cur_time indicates the time trying to put a new rectangle into the bin with available space. First, the bin having the minimum cur_time among n bins is chosen to pack rectangles. If no rectangle is available to fill the bin in the cur_time, the cur_time of current bin moves to the shortest end time in cores scheduled earlier within the bin. In this way, n bins are packed step by step.

- Consideration of multi-clock: In the domain of a bin packing problem, the increase of a test clock causes the width of a rectangle to decrease, but the height to increase in proportion to the decrease in width. For example, let the rectangle $r_i$ operate $f_T$ initially. If we assign $2f_T$ to $r_i$ as a test clock, then the width and height of $r_i$ becomes $t/2$ and $2w_i$, respectively. In this paper, we restrict the maximum test clock rate to $2f_T$ and consider two multi-clock modes similar to [14]. One is the case that the test clock is predetermined. Only the cores fulfilling the condition that $T(W) = T(W/2)$ can be tested by the faster clock $2f_T$. In the other case, we assign the clock $2f_T$ to a core, as long as there are enough TAM lines available during the scheduling of the core. If TAM lines are short for the faster clock, the normal test clock $f_T$ is tried instead.

- Incorporating power parameter: To consider power constraints, before packing rectangles, the scheduler always check whether the overall power consumption of cores tested at a time is over the power budget. As the power consumption is directly proportional to test clock frequency, both constraints should be considered together.

The pseudo-code of the proposed scheduling procedure is shown in Figure 5.

V. Experimental Results

We simulated four ITC'02 benchmark circuits [17] to evaluate the proposed scheduling algorithm. All of our simulations were conducted on a SUN UltraSPARC III with 1.2-GHz processors.

Table 1 displays the experimental results where a common clock was used for the core test. Column 2 of the table shows the test configuration method related to the number of test sources and sinks. In order to make comparisons with previous results, we assumed that the network clock frequency $f_N$ could increase up to four times as fast as the core test clock frequency $f_T$, i.e., $n = f_N / f_T$ and $n$ is an integer. Columns 3 and 5 show the results of core-based test scheduling without on-chip clocking by Liu et al. [14]. The network channel width is either 32 or 16 bits. Table 2 contains the scheduling results derived from two multi-clock modes. We confined the test clock of all cores to $f_T$ or $2f_T$ under the multi-clock mode. In case 1 in Table 2, we assigned a $2f_T$ clock to core $i$ if $T_i(W) = T_i(W/2)$. On the other hand, the test clock $f_T$ or $2f_T$ was dynamically chosen according to the TAM width available in case 2.

In Table 2, we compared the test times of the proposed method with those of [14] using on-chip clocking. Our algorithm significantly improves on earlier methods in all cases regardless of circuit sizes, channel width $W$, and network speed $n$ (Tables 1 and 2). However, there are some cases in which the overall test time is not decreased in spite of the increase of the network clock rate. This is because the test time reaches the lower bound due to bottleneck cores.

Table 3 contains the results with power constraints. As power consumption information of benchmark circuits is not available, we replace $nbff$ in Equation (2) by the sum of the numbers of scan flip-flops, inputs, outputs, and
bidirectional ports for power calculation. To evaluate the scheduling results including power constraints, we assumed that a system has an overall power limit $P_{\text{max}}$, and each core $i$, $(1 \leq i \leq m)$, has a separate power limit $P_i$. We set the power limit $P_{\text{max}}$ to 50% of $\Sigma P_i$ to compare the previous results in [14]. As seen in Table 3, the proposed algorithm shows good performance in most cases, since 50% bound has small influence on the test parallelism. However, owing to the unbalance between fixed test clock rates and the power consumption of embedded cores of p22810, scheduling results are not so good in case 1 mode. Therefore, it is efficient to select test clocks dynamically in circuits including cores whose size and power consumption vary extensively. In addition, the computation time of our algorithm is no more than one second in most cases regardless of power constraints. This is a very encouraging indicator for the practicality and feasibility of the proposed algorithm.

Fig. 5. Pseudo-Code of Test Scheduling Procedure

```plaintext
Procedure of Test Scheduling

1. Compute $R_i$ from wrapper design of cores;
2. until $p > 50\%$ do
3. until $q > 50\%$ do
4. Initialize $C[n]$, cur_time[$\text{bin\_pos}$] = 0, next_time[$\text{bin\_pos}$] = 0, $w_{\text{perm}}[\text{bin\_pos}] = W$;
5. Calculate $w_{\text{req}}$ of $C[n]$;
6. until all cores are tested do
   Select TAM position, $p$, that has the lowest $\text{cur\_time}$;
   /* Start of rectangle packing heuristics for a bin $p$ */
   If $[w_{\text{req}}(p) > 0]$ do
      If core $i$ can be found, such that ($\Sigma P_i \leq P_{\text{max}}$ AND $w_{\text{perm}}(i) + 2 \leq w_{\text{req}}(p)$ AND $T(w_{\text{req}}(i))/2$ is maximum) OR ($\Sigma P_i \leq P_{\text{max}}$ AND $w_{\text{perm}}(i) \leq w_{\text{req}}(p)$ AND $T(w_{\text{req}}(i))$ is maximum) do
         set $\text{bin\_pos}(i)$ AND $\text{clk\_mode}(i)$;
         Rectangle packing process with assigning $w_{\text{req}}(i)$ AND $\text{clk\_mode}(i)$ to cores
      Else do
         If core $i$ can be found, such that ($\text{bin\_pos}(i) = p$) AND ($\text{scheduled}(i) = \text{YES}$ AND $\text{complete}(i) = \text{NO}$)
            AND $\text{end\_time}(i) > \text{cur\_time}(p)$ AND $\text{end\_time}(i)$ is minimum do
               $\text{next\_time}(p) = \text{end\_time}(i)$;
            else do
               If core $i$ can be found, such that ($\text{scheduled}(i) = \text{NO}$) AND
               ($\Sigma P_i \leq P_{\text{max}}$ AND $T(w_{\text{req}}(i))/2 + \text{cur\_time}(p) \leq \text{next\_time}(p)$ AND $T(w_{\text{req}}(i))/2 + \text{cur\_time}(p)$ is maximum) OR
               ($\Sigma P_i \leq P_{\text{max}}$ AND $T(w_{\text{req}}(i)) + \text{cur\_time}(p) \leq \text{next\_time}(p)$ AND $T(w_{\text{req}}(i)) + \text{cur\_time}(p)$ is maximum) do
                  set $\text{bin\_pos}(i)$ AND $\text{clk\_mode}(i)$;
                  find $w(i)$, where $w(i)$ is the highest Pareto-optimal width, such that $w(i) \leq w_{\text{req}}(p) \text{clk\_mode}(i)$
                  Rectangle packing process with rectangle insertion in idle time
               Else do
                  If core $i$ can be found, such that ($\text{bin\_pos}(i) = p$) AND ($\text{scheduled}(i) = \text{YES}$ AND $\text{complete}(i) = \text{NO}$)
                   AND $\text{begin\_time}(i) \leq \text{cur\_time}(p)$ AND $(w_{\text{req}}(i) \text{clk\_mode}(i)) \leq w_{\text{req}}(p)$ AND
                   $(T(w_{\text{req}}(i)) - T(w_{\text{req}}(i) - w_{\text{req}}(p) \text{clk\_mode}(i))) / \text{clk\_mode}(i)$ is maximum do
                      find $w(i)$, where $w(i)$ is the highest Pareto-optimal width, such that $w(i) \leq w_{\text{req}}(p) \text{clk\_mode}(i)$
                      Rectangle packing process with increasing TAM widths to fill idle time
                   else do
                      set $w_{\text{req}}(p) = 0$;
                   end do
            end do
         else do
            move to next and update information
         end do
      end do
   end do
   /* End of rectangle packing heuristics for a bin $p$ */
   If max $\text{cur\_time}[\text{bin\_pos}]$ is less than current $\text{Best\_test\_time}$ do
      Update $\text{Best\_test\_time} = \text{cur\_time}[\text{bin\_pos}]$;
      Increase $q$ by $5\%$;
   end do
   Increase $p$ by $5\%$;
end do
7. return $\text{Best\_test\_time}$;
VII. Conclusion

In this paper, we proposed a new test scheduling algorithm for NoC-based SoCs using the rectangle packing solution used in testing common SoCs. In order to implement the proposed approach on NoC structures, the test resource configuration was designed and test methods such as test packet generation and routing were developed. Furthermore, we extended the algorithm by considering power constraints and improved the test efficiency with multiple test clocks. Experimental results show the proposed algorithm is very efficient and feasible for testing NoC-based systems.

References


Table 1. Experimental Results without Power Constraints (Single-Test Clock Mode)

<table>
<thead>
<tr>
<th>Benchmark</th>
<th># of IO or n</th>
<th>Single-test clock mode</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>W = 32</td>
<td>W = 16</td>
<td>Proposed</td>
</tr>
<tr>
<td>D695</td>
<td>2/2 or 2</td>
<td>18869</td>
<td>13732</td>
<td>26200</td>
</tr>
<tr>
<td>10cores</td>
<td>3/3 or 3</td>
<td>13412</td>
<td>9869</td>
<td>17807</td>
</tr>
<tr>
<td></td>
<td>4/4 or 4</td>
<td>10705</td>
<td>9869</td>
<td>16197</td>
</tr>
<tr>
<td>G1023</td>
<td>2/2 or 2</td>
<td>25052</td>
<td>14794</td>
<td>28635</td>
</tr>
<tr>
<td>14cores</td>
<td>3/3 or 3</td>
<td>17925</td>
<td>14794</td>
<td>19620</td>
</tr>
<tr>
<td></td>
<td>4/4 or 4</td>
<td>16489</td>
<td>14794</td>
<td>17925</td>
</tr>
<tr>
<td>P22810</td>
<td>2/2 or 2</td>
<td>271384</td>
<td>138990</td>
<td>331672</td>
</tr>
<tr>
<td>30cores</td>
<td>3/3 or 3</td>
<td>180905</td>
<td>102965</td>
<td>16197</td>
</tr>
<tr>
<td></td>
<td>4/4 or 4</td>
<td>150921</td>
<td>102965</td>
<td>16197</td>
</tr>
</tbody>
</table>

Table 2. Experimental Results without Power Constraints (Multi-Test Clock Mode)

<table>
<thead>
<tr>
<th>Benchmark</th>
<th># of IO or n</th>
<th>Multi-test clock mode (Case 1)</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>W = 32</td>
<td>W = 16</td>
<td>Proposed</td>
</tr>
<tr>
<td>D695</td>
<td>2/2 or 2</td>
<td>15570</td>
<td>11978</td>
<td>22779</td>
</tr>
<tr>
<td>10cores</td>
<td>3/3 or 3</td>
<td>10705</td>
<td>9869</td>
<td>15223</td>
</tr>
<tr>
<td></td>
<td>4/4 or 4</td>
<td>10705</td>
<td>9869</td>
<td>14945</td>
</tr>
<tr>
<td>G1023</td>
<td>2/2 or 2</td>
<td>16286</td>
<td>9822</td>
<td>19888</td>
</tr>
<tr>
<td>14cores</td>
<td>3/3 or 3</td>
<td>10941</td>
<td>7550</td>
<td>13338</td>
</tr>
<tr>
<td></td>
<td>4/4 or 4</td>
<td>9163</td>
<td>7397</td>
<td>11172</td>
</tr>
<tr>
<td>P22810</td>
<td>2/2 or 2</td>
<td>124787</td>
<td>7397</td>
<td>180977</td>
</tr>
<tr>
<td>30cores</td>
<td>3/3 or 3</td>
<td>99345</td>
<td>72981</td>
<td>11172</td>
</tr>
<tr>
<td></td>
<td>4/4 or 4</td>
<td>608809</td>
<td>470011</td>
<td>933119</td>
</tr>
<tr>
<td>P93791</td>
<td>2/2 or 2</td>
<td>124787</td>
<td>7397</td>
<td>180977</td>
</tr>
<tr>
<td>32cores</td>
<td>3/3 or 3</td>
<td>479403</td>
<td>341858</td>
<td>732888</td>
</tr>
<tr>
<td></td>
<td>4/4 or 4</td>
<td>133091</td>
<td>259263</td>
<td>502876</td>
</tr>
</tbody>
</table>

Table 3. Experimental Results with Power Constraints (W = 32)

<table>
<thead>
<tr>
<th>Benchmark</th>
<th># of IO or n</th>
<th>Single-test clock mode</th>
<th>Multi-test clock mode (Case 1)</th>
<th>Multi-test clock mode (Case 2)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>50%</td>
<td>Proposed</td>
<td>50%</td>
</tr>
<tr>
<td>D695</td>
<td>2/2 or 2</td>
<td>18869</td>
<td>13732</td>
<td>16615</td>
</tr>
<tr>
<td>10cores</td>
<td>3/3 or 3</td>
<td>13412</td>
<td>9869</td>
<td>15048</td>
</tr>
<tr>
<td></td>
<td>4/4 or 4</td>
<td>10705</td>
<td>9869</td>
<td>13644</td>
</tr>
<tr>
<td>G1023</td>
<td>2/2 or 2</td>
<td>16286</td>
<td>9822</td>
<td>16286</td>
</tr>
<tr>
<td>14cores</td>
<td>3/3 or 3</td>
<td>17925</td>
<td>14794</td>
<td>12313</td>
</tr>
<tr>
<td></td>
<td>4/4 or 4</td>
<td>16489</td>
<td>14794</td>
<td>12087</td>
</tr>
<tr>
<td>P22810</td>
<td>2/2 or 2</td>
<td>271384</td>
<td>143785</td>
<td>187061</td>
</tr>
<tr>
<td>30cores</td>
<td>3/3 or 3</td>
<td>181169</td>
<td>130853</td>
<td>134998</td>
</tr>
<tr>
<td></td>
<td>4/4 or 4</td>
<td>156810</td>
<td>122280</td>
<td>119202</td>
</tr>
<tr>
<td>P93791</td>
<td>2/2 or 2</td>
<td>611991</td>
<td>484922</td>
<td>608809</td>
</tr>
<tr>
<td>32cores</td>
<td>3/3 or 3</td>
<td>482352</td>
<td>351217</td>
<td>479403</td>
</tr>
<tr>
<td></td>
<td>4/4 or 4</td>
<td>389935</td>
<td>308707</td>
<td>389935</td>
</tr>
</tbody>
</table>