Test-Wrapper Optimization for Embedded Cores in TSV-Based Three-Dimensional SOCs*

Brandon Noia[1], Krishnendu Chakrabarty[1] and Yuan Xie[2]
[1]*Department of Electrical and Computer Engineering, Duke University, Durham, NC 27708*
[2] *Department of Computer Science and Engineering, Pennsylvania State University, University Park, PA 16802*
[1] *Email: brn2@duke.edu, krish@ee.duke.edu* [2] *Email: yuanxie@cse.psu.edu*

***Abstract*— System-on-chip (SOC) designs comprised of a number of embedded cores are widespread in today's integrated circuits. Embedded core-based design is likely to be equally popular for three-dimensional integrated circuits (3D ICs), the manufacture of which has become feasible in recent years. 3D integration offers a number of advantages over traditional two-dimensional (2D) technologies, such as the reduction in the average interconnect length, higher performance, lower interconnect power consumption, and smaller IC footprint. Despite recent advances in 3D fabrication and design methods, no attempt has been made thus far to design a 1500-style test wrapper for an embedded core that spans multiple layers in a 3D SOC. This paper addresses wrapper optimization in 3D ICs based on through-silicon vias (TSVs) for vertical interconnects. Our objective is to minimize the scan-test time for a core under constraints on the total number of TSVs available for testing. We present two polynomial-time heuristic solutions. Simulation results are presented for embedded cores from the ITC 2002 SOC test benchmarks.**

I. Introduction

System-on-chip (SOC) designs are common for today's integrated circuits, and they are often comprised of a number of embedded cores. In order to test the cores at a system level, a test-access mechanism (TAM) must be included on the chip. The purpose of the TAM is to allow test data to be transported between the cores and the chip input and output (I/O) pins. Test wrappers, such as the IEEE Std. 1500 wrapper [1], interface the TAM to the embedded cores and connect the I/O of the cores to other cores.

Modular testing is advocated for core-based SOCs because it facilitates test reuse and allows the cores to be tested without complete knowledge about their internal structural details. In order to reduce cost and test time, test wrapper and TAM optimization are critical. Although a lot of research [2]–[4] has been directed at modular testing for SOCs, including the IEEE Std. 1500 for wrapper design, it has largely been targeted at two-dimensional (2D) integrated circuits (ICs). The integration of embedded cores in multi-layer three-dimensional (3D) ICs is leading to new modular test challenges.

The promise of 3D IC technology lies in the numerous benefits it can potentially provide over traditional 2D ICs [5], [6]. Due to the relentless increase in chip complexity, interconnects have become longer in 2D ICs, leading to increased circuit delay and power consumption. 3D ICs will lead to a reduction in the average interconnect length and help obviate the problems caused by long global interconnects [7]–[9]. Since die can be stacked in a 3D environment, on-chip data bandwidth can be increased as well. Furthermore, since 3D ICs can scale "up" instead of "out", higher packing density and smaller footprint can be achieved.

Although a number of 3D integration methods have been proposed in the literature, in this work we focus on through-silicon via (TSV) vertical interconnects, as it offers the promise of the highest vertical interconnect density among the proposed technologies. Using TSV technology, 3D ICs are created by placing multiple device layers together through wafer or die stacking, and these are then connected using vertical TSVs [8]. Fig. 1 shows an example of a 3D IC using TSV interconnects.

3D SOC design can be done at two levels of granularity:

1) Coarse-granularity partitioning: With this approach, each embedded core in the SOC is still a 2D design.
2) Fine-granularity partitioning: With this approach, each core in the SOC chip is partitioned into multiple layers [8]. For example, a repartitioning of the Intel Core 2 processor across four die-stacked tiers showed significant reductions in latency, resulting in an overall 47.9% increase in frequency and 47.0% performance improvement [10].

Even though fine-granularity 3D SOC design provides the above benefits, it introduces additional challenges on how to test such true 3D cores. Scan-chain design in such multi-layer cores can also span across layers so that the wire-length for scan chains can be reduced (and therefore reduce test power and testing time) [11]. Test-wrapper design for 3D SOC systems with true 3D cores is more challenging compared to the case of coarse-granularity 3D SOC design [12], because scan chains for each individual core can now span multiple layers and the test wrapper must interface with input/output and scan-chain terminals located on any layer.

Though the manufacture of 3D ICs is now feasible, design-automation and testing tools are not yet fully mature for commercial exploitation. These tools need to be able to exploit the benefits of 3D technologies, while taking into account the various design-related tradeoffs. For example, in a TSV-based 3D IC, the number of available TSVs for test access is limited because of available chip area. Most TSVs are likely to be dedicated for functional access, power/ground, and clock routing.

Testing of 2D ICs and optimization of test-access architectures have been well studied [2] [13] [4]. Optimization methods have included ILP [2], rectangle packing [14] [2],

*This work was supported in part by a Master's Scholarship from the Semiconductor Research Corporation (SRC) and NSF 0903432

978-1-4244-5028-2/09/$25.00 ©2009 IEEE

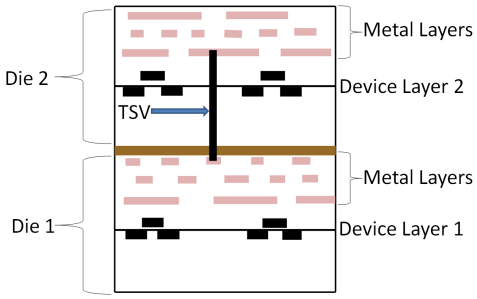

Fig. 1. An example of a 3D IC manufactured by die stacking with TSV connections.

iterative refinement [15], and other heuristics [4] [16]. However, these methods were developed for 2D IC technologies, and the added design problems related to 3D technologies were not considered.

Some early work has been reported recently on testing of 3D ICs [12], [17]. However, no attempt has been made thus far to design a 1500-style test wrapper for an embedded core that spans multiple layers in a 3D SOC. The goal of this paper is to address wrapper optimization in 3D ICs that use TSVs for vertical interconnects and fine-granularity partitioning. Our objective is to minimize the scan-test time for the core under constraints on the total number of TSVs available for testing. The embedded core is assumed to have full scan, and the scan chains are "hard" in the sense that they cannot be redesigned for wrapper optimization.

The wrapper optimization problem is known to be NP-complete for 2D ICs [2], [13]. Therefore, it follows from the "method of restriction" that this optimization problem is also intractable for 3D ICs. Bin design methods that have been adapted for 2D wrapper design cannot be directly used for 3D ICs because of the need to incorporate constraints on the number of TSVs. Moreover, the possibility of scan chains on multiple layers adds an additional level of complexity for optimization.

In this paper, we address wrapper chain optimization for embedded cores in TSV-based 3D SOCs. To the best of our knowledge, this work is the first attempt to study the test-wrapper design for 3D SOC systems with true 3D cores. 3D wrapper chain optimization differs from that in 2D ICs in that scan-chains can span multiple layers, core inputs and outputs can be on any layer, and that additional constraints on the number of TSVs exist. Furthermore, all of the chip pins are at the lowest layer [8], necessitating that wrapper chains begin and end on the lowest layer. This, coupled with TSV restrictions, adversely affects access to core elements on the highest layers.

The rest of this paper is organized as follows: Section II uses a simple example to motivate this work and briefly outlines prior work. Section III presents two heuristic approaches for solving the wrapper optimization problem. Section IV gives experimental results for several cores from the ITC 2002 SOC test benchmarks. Finally, Section V presents conclusions drawn from this work.

II. Problem Formulation

A. Motivational Example

Figure 2 shows a simple example to motivate this work. The TAM width used to access a core determines the number of wrapper chains that must be created. An upper limit on the total number of TSVs that can be used for routing all of the wrapper chains restricts how many TSVs each chain can utilize. The length of the longest wrapper chain determines the number of clock cycles needed to load in and read out test data. Therefore, the objective is to reduce the length of the longest wrapper chain (and thus scan time) while utilizing no more TSVs than the global TSV limit. In order for loading and reading test data in a pipelined fashion (and thus as efficiently as possible), functional inputs must be placed first in a wrapper chain and functional outputs must be placed last, with the scan-chains in the middle.

There are two layers to the 3D IC of Fig. 2. Part (a) shows the placement of the functional I/Os and the scan elements. All pins to the IC are on layer 0, such that all wrapper chains must begin and end on layer 0. There are two scan chains, one with 3 registers and one with 5. The 5-register scan chain spans both layers, with its scan-in on layer 0 and scan-out on layer 1. For this example, let us assume a global limit of 3 TSVs. Parts (b) and (c) show 2 possible solutions using two wrapper chains. As can be seen, both solutions use 3 TSVs but the wrapper chains in part (c) are balanced, while those of part (b) are unbalanced. The solution of part (c) leads to less test time. The balancing of the lengths of wrapper chains for large cores is a challenging design task. Optimization methods are needed for determining the best wrapper designs under a set of design and technology constraints.

B. Problem Formulation

The goal of wrapper optimization is to minimize the test time for each core. The optimization problem addressed in this paper is as follows. Given a core (placement of inputs, outputs, scan-ins, and scan-outs, as well as lengths of scan-chains), the TAM width, and the 3D design constraint of a maximum number of TSVs to be used, determine the optimal placement of core elements into wrapper chains such that the length of the longest wrapper chain is minimized.

We have developed two faster heuristic approaches to produce near-optimal solutions. These two models extend the bin design method presented in [13] for 2D wrapper design. The first heuristic presented is less close to optimality but faster for cores with many scan-chains, while the second heuristic is slower for cores with many scan-chains but produces better results under tight TSV constraints.

III. 3D Wrapper Optimization

Scan chains can span multiple layers and their scan-in and scan-out can be on different layers. TSVs internal to a scan chain are not counted against the TSV constraint. The lowest layer of the IC is layer 0, followed by layers 1, 2, etc.

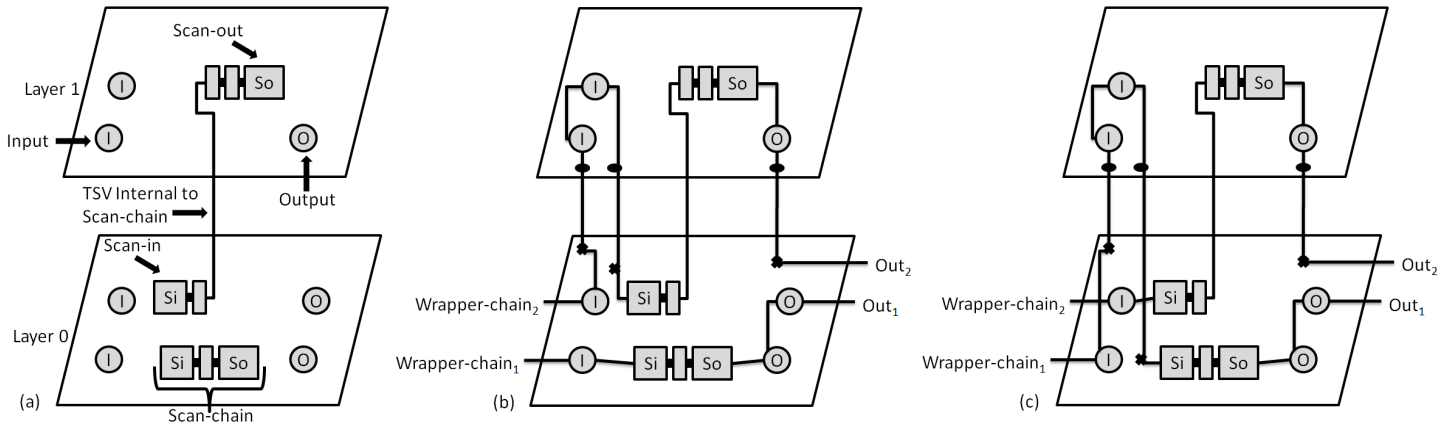

Fig. 2. A 3D IC with two possible wraper-chain solutions.

A. Basic Heuristic Method

The goal of this heuristic method is to achieve near-optimal sorting of elements across a specified number of wrapper chains in polynomial time. We are given the following: TAM width M, a set of core elements $S = \{s_1, s_2, \ldots, s_n\}$, and a maximum number of TSVs, $TSVmax$, which can be used for all the routing of the wrapper chains. For the purpose of the heuristic, an element is a data structure. Each element also contains two layer values corresponding layer of the first register and layer of the last register of the element. For inputs and outputs, which reside on only 1 layer and contain only 1 register, these two values are equal. For scan-chain elements, these two layers represent the scan-in and scan-out and can be on any layer. Lastly, an element has a type value which delineates the element as being an input, output, our scan-chain. Each element will be placed in only one wrapper chain. Fig. 3 outlines the element data structure.

Data structure *Element*

1. $Length$ /*Number of Registers*/
2. Lin /*Layer of First Register*/
3. $Lout$ /*Layer of Last Register*/
4. $Type$ /*Element Type*/

Fig. 3. The element data structure.

The heuristic approach to this problem is based on the bin-design method for 2D wrapper chains [13]. In our approach, each bin corresponds to a wrapper chain and is a data structure. Each bin contains a variable which defines the maximum number of TSVs that the wrapper chain can utilize. A size ($Size$) is associated with a bin that stores the length of the wrapper chain (the number of registers contained in the wrapper chain). The length of all wrapper chains at the start of the algorithm is 0. The number of TSVs used at any given time by the wrapper chain is also stored. This number must always be less than or equal to the TSV limit. Lastly, the bin contains a set of all of the elements $E = \{e_1, e_2, \ldots, e_b\}$ in the wrapper chain.

For this heuristic, we are given the following: a TAM width M, a set of core elements $S = \{s_1, s_2, \ldots, s_n\}$, a set of layers $Lin = \{lin_1, lin_2, \ldots, lin_n\}$ corresponding to the layer on which a scan-in, functional input, or functional output resides for set S, a set of layers $Lout = \{lout_1, lout_2, \ldots, lout_n\}$ corresponding to the layer of the scan-out (or equal to the corresponding L_{in} value for inputs and outputs) for set S, and a set of types $T = \{t_1, t_2, \ldots, t_n\}$ corresponding to the type of each element in set S.

Algorithm 1 BinDesign($\{c_1, \ldots, c_M\}$, S, Lin, $Lout$, T)

```
Initialize n elements; Initialize M bins;
Sort(elements); Sort(bins);
for i = 1 to n do
   for j = 1 to M+1 do
      if j == M+1 then
         /*No solution can be found for this enumeration*/
         return -1;
      end if
      if InsertElement(bin_j, s_i) then
         break;
      end if
   end for
   Sort(bins);
end for
return Size(bin_M);
```

The heuristic begins by partitioning the number of TSV in $TSVmax$ between M wrapper chains. The algorithm for doing this is not shown here explicitly since it is a straightforward enumerative procedure (without repetitions). This algorithm produces a set $C = \{c_1, c_2, \ldots, c_M\}$ of a possible partitioning of $TSVmax$ across M wrapper chains. These combinations are given as an argument to the next phase, the bin design algorithm.

The bin design algorithm is shown in Algorithm 1. The purpose of the algorithm is to create all elements and bins, and then to attempt to put elements into bins until all elements are in a bin or it fails to put an element in a bin. It begins by initializing n element data structures such that their variables correspond to the values in sets S, Lin, $Lout$, and T. Likewise, M bins are initialized with $TSVlimit$ equal to the respective limits of set C. It then proceeds by sorting the elements and bins. Elements are sorted depending first on the value of $Length$ and second by Lin. Those elements with

the longest $Length$ are placed first, and between elements of the same $Length$ those with the highest Lin are placed first. Sorting the bins is based first on the value of $Size$ and then on $TSVlimit$. Those with larger $Size$ values are placed first, and where bins are of equal size, those with the smallest $TSVlimit$ are placed first. Since all bins begin at $Size$ = 0, they are first ordered by $TSVlimit$.

The order of elements and bins is important, as it determines the order in which they are considered by the algorithm. For the bins, $Size$ is prioritized, and we attempt to place the elements in the smallest bins first. Adding elements to the smallest chains first reduces the size of the largest chains, one of the goals of our heuristic. Among bins of equal size, we attempt to place elements in those with the lowest $TSVlimit$ first. Since those bins with higher $TSVlimit$ values are capable of holding a wider array of elements without violating their constraints, it is imperative to first attempt to utilize bins with less leeway. Elements are ordered such that large scan-chains are considered first. Since there tend to be more I/O than scan chains, and I/O have a $Length$ of 1, they are used to balance out wrapper chains that may have large size differences caused by placing scan-chains. Elements on higher layers are considered next, since they have the least leeway in which bins they can be placed.

After initialization and sorting, the bin design algorithm attempts to place elements into bins based on the sorted priorities of both elements and bins. It does this using the insert element algorithm shown in Algorithm 2. If the element was successfully placed, InsertElement returns true, otherwise it returns false. If false is returned, the bin being checked is incremented and InsertElement is run to see if the element can be placed in the next bin. If the algorithm has gone through all the bins and has been unable to place the element in a bin without violating the TSV limit of each bin, then the enumeration of C is considered to be infeasible and a value indicating that no solution could be found is returned. If true is returned by InsertElement, then the bins are resorted (since $Size$ and $TSVlimit$ changed for the bin the element was placed in) and the algorithm attempts to insert the next element. The bin design algorithm returns the $Size$ of the longest bin.

The insert element algorithm is shown in Algorithm 2. This algorithm takes as its arguments an element s_{in} that the algorithm will try to put in the bin Bin_j . If it can insert s_{in} into Bin_j without using more TSVs than its $TSVlimit$, then s_{in} is added to E (the set of elements in the wrapper chain represented by Bin_j) and the algorithm returns true. Otherwise, it returns false.

InsertElement begins by defining the variable $used$, which keeps track of how many TSVs the wrapper chain needs to connect all elements in the chain. It copies E to a new set, the $testSet$, which also contains s_{in}. The number of elements in $testSet$ is stored in the variable $elementsToConsider$, which begins as totaling the number of elements in E (which equals b) plus 1 since s_{in} is also added to $testSet$.

Since inputs must appear first in the wrapper chain, InsertElement first connects all elements of $Type$ = input. The minimum number of TSVs necessary to connect all inputs in a wrapper chain, regardless of the number of inputs, is simply equal to the value of Lin for the input on the highest layer. This fact is shown conceptually in Fig. 4. Here, the minimal number of TSVs used for a wrapper chain in which the highest input is on layer 3 is 3. Thus, InsertElement looks at the value of Lin for all elements of $Type$ input, storing the highest value of Lin in the variable $hiLayer$. The number of TSVs utilized at this point (the value of $used$) is set to be equal to $hiLayer$.

InsertElement next considers scan chains, as they succeed inputs in the wrapper chain. The variable $currentLayer$ keeps track of the layer on which the last element placed in the wrapper chain resided. $layerDif$ contains the difference between the layer $currentLayer$ and the Lin of the next scan-chain to be inserted in the wrapper chain. $layerDif$ is

Algorithm 2 InsertElement(Bin_j, s_{in})

```
/*All functions are performed on Bin_j*/
used=0;
copy E to testSet;
Add element s_in to testSet; elementsToConsider = b+1;
/*Stage 1 - Consider Inputs*/
hiLayer = 0;
for i = 1 to elementsToConsider do
   if Type(testSet_i) == input then
      if Lin(testSet_i) > hiLayer then
         hiLayer = Layer(s_i);
      end if
   end if
end for
used += hiLayer;
/*Stage 2 - Consider Scan-chains*/
currentLayer = hiLayer; layerDif = 999;
while testSet contains at least 1 scan-chain do
   for i = 1 to elementsToConsider do
      if Type(testSet_i) == scan-chain then
         if |Lin(testSet_i)-currentLayer| < layerDif then
            layerDif = |Lin(testSet_i)-currentLayer|;
            bestElement = i;
         end if
      end if
   end for
   used += layerDif; layerDif = 999;
   currentLayer = Lout(testSet_bestElement);
   remove testSet_bestElement from testSet;
   elementsToConsider--;
end while
/*Stage 3 - Consider Outputs*/
hiLayer = 0;
for i = 1 to elementsToConsider do
   if Type(testSet_i) == output then
      if Lin(testSet_i) > hiLayer then
         hiLayer = Lin(testSet_i);
      end if
   end if
end for
used += hiLayer + |currentLayer-hiLayer|;
if used <= TSVlimit then
   Insert s_in in E;
   return TRUE
end if
return FALSE
```

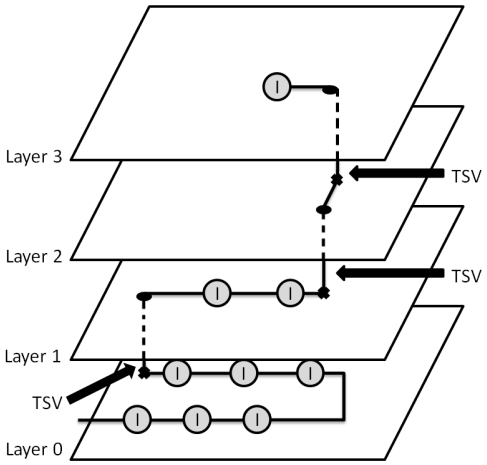

Fig. 4. Conceptual view of the number of TSVs used for inputs in a wrapper chain.

initialized to an arbitrarily high value. The while statement ensures that all elements of $Type$ scan-chain are inserted into the wrapper chain. For all scan-chains in the testSet, the smallest difference between $currentLayer$ and the Lin of a scan-chain is determined, and the index of that scan-chain is saved in $bestElement$. We aim to find the smallest $layerDif$ in order to reduce the number of TSVs used between elements. $layerDif$ is added to $used$ to reflect any extra TSVs that may have been used connecting elements. $layerDif$ is returned to its arbitrarily high value, and $currentLayer$ is changed to the value of the current scan-chain's $Lout$ to reflect the layer on which the next trace will be placed. The element contained in $bestElement$ is removed from the $testSet$ so that it is not reconsidered, and $elementsToConsider$ is decremented to reflect the removal.

After all of the scan-chain elements have been placed into the wrapper chain, InsertElement places outputs. As with the inputs, the number of TSVs used for the outputs only depends on the highest layer on which an output resides. Thus, the algorithm checks the $testSet$ for the highest Lin for an element of $Type$ output and stores it in $hiLayer$. $used$ is incremented by the number of TSVs used to connect all the outputs back to layer 0 plus the difference between the scan-out of the last scan-chain used and the output on the highest layer. This is the minimum number of TSVs that can be used to connect all of these elements, as shown in Fig. 5. Fig. 5 shows two possible solutions for an example wrapper chain. The one on the left is the minimal solution resulting in 3 TSVs, one to connect the last scan-out to the highest output and then connect other outputs down to layer 0. The example on the right shows a suboptimal solution using a different method and resulting in 5 TSVs.

After all elements have been placed in the wrapper chain, InsertElement checks to see if the $TSVlimit$ has been violated. If not, then s_{in} is inserted into E and the algorithm returns true. The bin's $Size$ and $TSVused$ variables are also updated, which for conciseness are not shown in the algorithm. $Size$ is simply the sum of the $Length$ variables for all elements in E, and $TSVused$ is equal to $used$ at the end of the algorithm. If the $TSVlimit$ has been exceeded,

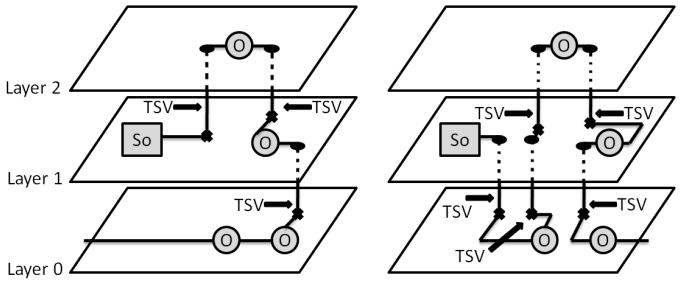

Fig. 5. An optimal and suboptimal solution to connecting outputs in a wrapper chain.

then s_{in} is discarded and the algorithm returns false.

B. Heuristic Method with Look Ahead

In Algorithm 2, the method of placing scan chains in a wrapper chain is the main obstacle towards achieving near-optimal results. The problem with Algorithm 2 will be explained using the simple example shown in Fig. 6. In Part (a), the scan-out of a previous scan chain is shown, with three other scan chains still to be connected. Determining that the two closest scan-ins are on the same layer as the scan-out, Algorithm 2 will select the first of the closest scan-chains. As can be seen by the next series of connections in Part (b), labeled in the order in which they were made, this leads to a suboptimal result using one TSV. Part (c) shows a more effective solution for connecting the scan chains, and it results in the use of no TSVs.

A solution to this problem is a method that we will refer to as *look-ahead*. The goal of look ahead is to determine, given a layer on which a preceding input or scan-out resided, which next scan-chain will result in the fewest used TSVs. It does this by creating a set $R = \{R_1, R_2, \ldots, R_q\}$ of scan-chains that are closest in layer to the preceding element, and then checking to see which scan-chains' scan-out is closest in layer to the next selectable scan-chain. This is what we mean by look ahead, and in this case we look ahead by 1 element.

To use look ahead, we must alter how InsertElement deals with scan-chains. Algorithm 3 shows the section of the InsertElement algorithm that considers scan-chains modified for use with look ahead. Whereas before we only attempted to determine one element that is closest in layer to the previous element, now we wish to place all of the closest elements into set R. If an element is found with a layer closer than $layerDif$, then R is cleared and the element is copied into R. If another element is found with the same $layerDif$ as the current minimum, then the element is added to R without first clearing R, since we want a set of all the closest elements. To determine $bestElement$, we now use the LookAhead algorithm. The remainder of the modified InsertElement is carried out as before.

The LookAhead algorithm takes as its arguments the set of elements R, the $testSet$, and the number of elements $elementsToConsider$ in $testSet$. For each prospective element in R, the algorithm checks all elements in the $testSet$ to determine which element in $testSet$ is closest in layer to the element in R. First, LookAhead checks to see if the layer

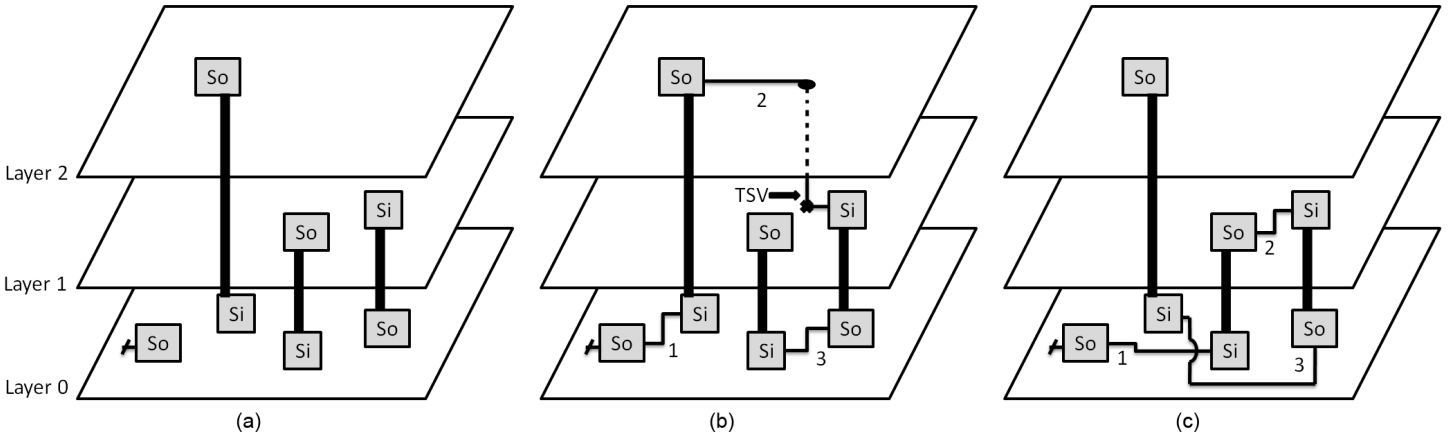

Fig. 6. A conceptual view of the shortcomings of Algorithm 2.

Algorithm 3 Stage 2 of InsertElement modified for look ahead

```
/*Stage 2 - Consider Scan-chains*/
currentLayer = hiLayer; layerDif = 999;
while testSet contains at least 1 scan-chain do
  for i = 1 to elementsToConsider do
    if Type(testSet_i) == scan-chain then
      if |Lin(testSet_i)-currentLayer| < layerDif then
        layerDif = |Lin(testSet_i)-currentLayer|;
        clear R; copy testSet_i to R;
      end if
      if |Lin(testSet_i)-currentLayer| == layerDif then
        copy testSet_i to R;
      end if
    end if
  end for
  bestElement = LookAhead(R, testSet, elementsToConsider)
  used += layerDif; layerDif = 999;
  currentLayer = Lout(testSet_bestElement);
  remove testSet_bestElement from testSet;
  elementsToConsider--;
end while
```

difference is less than dif and that the element R_i is not, in fact, the same element as $testSet_j$. If this is the case, then choosing R_i as the next element is likely to produce better results than choosing another element in R, since fewer TSVs would be used to connect to the next scan-chain. It should be noted that if more than one element has a second element with layer difference dif, then LookAhead chooses the first element it found for $bestElement$, just as the umodified $InsertElement$ of Algorithm 2 had.

C. Analysis of Heuristics

The BinDesign algorithm relies on the use of enumeration, and we produce a complete set of unique enumerations. For example, if $TSVmax = 4$ and $M = 2$, the set of unique enumerations is {[0 4], [1 3], [2 2]}. This set grows quickly as M and $TSVmax$ increases. For the sake of our analysis of algorithms that follows, we focus on complexity for only a single enumeration.

The BinDesign algorithm will attempt to place all n elements in M wrapper chains. Assuming that all elements are successfully placed, and given a worst-case scenario in which the element is always placed in the last bin considered, the algorithm is of complexity $O(M \cdot n)$. Thus, BinDesign runs in polynomial time in n and M.

TABLE I

RESULTS ON WRAPPER OPTIMIZATION FOR CORE 7 OF SOC D281 WITH THREE LAYERS.

M	$TSVmax$	Shortest Wrapper (No LA)	Shortest Wrapper (LA)	CPU time (No LA) (minutes)	CPU time (LA) (minutes)
2	12	N/A	N/A	≈0.00	≈0.00
	14	N/A	N/A	≈0.00	0.13
	16	1633	1623	0.22	0.98
	18	1064	1064	1.42	3.13
	22	1064	1064	2.03	3.63
3	12	N/A	N/A	≈0.00	≈0.00
	14	N/A	N/A	≈0.00	0.33
	16	1633	1347	0.25	1.07
	18	710	752	3.67	7.52
	22	710	710	6.73	10.42
4	12	N/A	N/A	≈0.00	≈0.00
	14	N/A	N/A	0.02	0.67
	16	1633	1347	0.30	1.52
	18	532	611	7.05	11.47
	22	532	545	16.17	20.70
5	12	N/A	N/A	0.02	0.02
	14	N/A	N/A	0.05	1.02
	16	1633	1347	0.37	2.13
	18	470	486	11.98	14.48
	22	426	427	32.28	33.20
6	12	N/A	N/A	0.02	0.03
	14	N/A	N/A	0.07	1.33
	16	1633	1347	0.47	2.73
	18	459	441	16.75	18.18
	22	355	364	53.83	46.37

The unmodified InsertElement of Algorithm 2 determines the number of TSVs that would be used if an element was placed in a given bin. In a worst-case scenario, all elements n would be scan-chains, as scan-chains require the most computation to place. For the number of scan-chains (n in our worst case), InsertElement checks to see which scan-chain is closest in layer to the previous element and then removes that scan-chain from the set of scan-chains that we consider. This results in $\sum_{i=0}^{n-1} n - i$, or $\frac{1}{2}n^2 + \frac{1}{2}n$ computations. Therefor, InsertElement has a complexity of $O(n^2)$. When combined with the BinDesign algorithm, the entire wrapper optimization algorithm without look ahead has a complexity of $O(n^3)$ per enumeration.

With a look ahead of one element, complexity increases. The complexity of the modified InsertElement algorithm is

the same as its unmodified counterpart. The difference is that a set R of all elements closest in layer to the previous element is presented as an argument to the LookAhead algorithm instead of simply choosing the first of the elements. Thus, the complexity of the LookAhead algorithm must be considered. LookAhead searches through the entirety of the $testSet$ a number of times equivalent to the number of elements in R each time LookAhead is called. Assuming a worst-case scenario, in which R is the entire $testSet$, then this ammounts to $\frac{1}{2}n^2 + \frac{1}{2}n$ iterations as with the InsertElement algorithm. This gives the entire wrapper optimization algorithm with look ahead a complexity of $O(n^5)$ per enumeration.

TABLE II
RESULTS ON WRAPPER OPTIMIZATION FOR CORE 13 OF SOC P93791 WITH FOUR LAYERS.

M	$TSVmax$	Shortest Wrapper (No LA)	Shortest Wrapper (LA)	CPU time (No LA) (minutes)	CPU time (LA) (minutes)
2	18	N/A	N/A	≈0.00	0.03
	20	N/A	4835	≈0.00	0.10
	22	4875	4835	0.03	0.13
	24	4835	4838	0.12	0.20
	34	4835	4835	0.18	0.33
3	18	N/A	N/A	0.02	0.07
	20	3371	5008	0.05	0.12
	22	3328	3362	0.12	0.28
	24	3257	3253	0.30	0.57
	34	3226	3225	1.02	1.50
4	18	N/A	N/A	0.05	0.15
	20	N/A	5008	0.12	0.30
	22	2598	2548	0.25	0.68
	24	2477	2462	0.70	1.53
	34	2432	2452	3.37	4.57
5	18	N/A	N/A	0.08	0.25
	20	N/A	2107	0.20	0.52
	22	2520	2059	0.50	1.13
	24	2042	2014	1.10	2.47
	34	1963	1975	8.58	11.22
6	18	N/A	N/A	0.15	0.40
	20	N/A	2107	0.37	0.83
	22	2520	1885	0.88	1.87
	24	1820	1694	1.83	4.17
	34	1645	1645	22.07	26.93
7	18	N/A	N/A	0.42	0.72
	20	N/A	2107	0.97	1.62
	22	2520	1885	2.18	3.77
	24	1820	1694	4.83	8.02
	34	1426	1428	89.73	96.12
8	18	N/A	N/A	1.47	1.82
	20	N/A	2107	3.68	4.45
	22	2520	1885	8.80	10.6
	24	1820	1694	20.07	24.03
	34	1242	1242	535.35	634.60

It should be noted that these worst-case complexities would result in significantly worse run times than would be seen in realistic examples. This is because there tend to be relatively few scan-chains, even in complex cores, when compared to the number of inputs and outputs. For example, core 7 of SOC d281 from the ITC'02 SOC test benchmarks contains 1510 elements, only 20 of which are scan-chains.

IV. EXPERIMENTS AND RESULTS

A. Experimental Setup

The optimization heuristics were run on cores from various SOCs presented in the ITC'02 SOC test benchmarks. We present results from four cores: cores 4 and 13 of SOC p93791, core 5 of SOC h953, and core 7 of SOC d281. For SOC p93791, core 4 consists of 23 scan chains, 15 inputs, and 30 outputs, while core 13 has 46 scan chains, 111 inputs, and 31 outputs. Core 5 of SOC h953 has 4 scan chains, 19 inputs, and 13 outputs, and core 7 of SOC d281 contains 20 scan chains, 700 inputs, and 790 outputs. The heuristics were coded in C++. The layers on which inputs, outputs, scan-ins, and scan-outs resided were determined randomly.

B. Results for Heuristics

TABLE III
RESULTS ON WRAPPER OPTIMIZATION FOR CORE 4 OF SOC P93791 WITH THREE LAYERS.

M	$TSVmax$	Shortest Wrapper (No LA)	Shortest Wrapper (LA)	CPU time (No LA) (minutes)	CPU time (LA) (minutes)
2	12	N/A	N/A	≈0.00	≈0.00
	13	N/A	132	≈0.00	≈0.00
	14	89	87	≈0.00	≈0.00
	18	77	79	≈0.00	0.02
	20	77	77	≈0.00	0.02
3	12	N/A	N/A	≈0.00	≈0.00
	13	N/A	69	≈0.00	≈0.00
	14	84	69	≈0.00	0.02
	18	51	53	0.03	0.05
	20	51	52	0.03	0.07
4	12	N/A	N/A	≈0.00	≈0.00
	13	N/A	59	≈0.00	0.02
	14	79	59	0.02	0.03
	18	39	43	0.07	0.12
	20	39	42	0.08	0.13
5	12	N/A	N/A	≈0.00	≈0.00
	13	N/A	59	≈0.00	0.02
	14	79	59	0.02	0.03
	18	37	37	0.12	0.15
	20	35	33	0.12	0.23
6	12	N/A	N/A	≈0.00	0.02
	13	N/A	59	≈0.00	0.03
	14	79	47	0.03	0.05
	18	31	31	0.18	0.25
	20	28	31	0.32	0.40
7	12	N/A	N/A	≈0.00	0.03
	13	N/A	59	0.02	0.05
	14	79	47	0.07	0.10
	18	30	31	0.43	0.50
	20	30	29	0.87	0.97
8	12	N/A	N/A	≈0.00	0.07
	13	N/A	59	0.08	0.12
	14	79	47	0.18	0.22
	18	30	27	1.37	1.48
	20	25	27	3.32	3.43

Table I shows results from core 7 of SOC d281, Table II shows results from core 13 of SOC p93791, Table III shows results from core 4 of SOC p93791, and Table IV shows results for core 5 of SOC h953 for both heuristic methods presented in Sections III-A and III-B. Results are shown for different values of M and $TSVmax$. Column 1 lists the TAM width M. Column 2 gives $TSVmax$, the upper limit on the number of TSVs that can be used. The third and fourth columns present the shortest wrapper chain achieved through all enumerations of possible TSV distributions by the heuristic approach without and with look ahead, repectively. The value 'N/A' means that the algorithm was unable to determine a feasible solution for the given values of M and $TSVmax$. This would occur when an element cannot be placed in any wrapper chain without violating that wrapper chain's TSV limit for all enumerations. The fifth and sixth columns present the run time in minutes to determine a solution for the heuristics without and with look ahead, respectively. This CPU time includes the time required by

TABLE IV

RESULTS ON WRAPPER OPTIMIZATION FOR CORE 5 OF SOC H953 WITH THREE LAYERS.

M	$TSVmax$	Shortest Wrapper (No LA)	Shortest Wrapper (LA)	CPU time (No LA) (minutes)	CPU time (LA) (minutes)
2	8	N/A	N/A	≈0.00	≈0.00
	10	386	386	≈0.00	≈0.00
	12	258	258	≈0.00	≈0.00
	16	258	258	≈0.00	≈0.00
3	8	N/A	N/A	≈0.00	≈0.00
	10	249	249	≈0.00	≈0.00
	12	247	247	≈0.00	≈0.00
	16	241	241	≈0.00	≈0.00
4	8	N/A	N/A	≈0.00	≈0.00
	10	249	249	≈0.00	≈0.00
	12	132	132	≈0.00	≈0.00
	16	129	129	≈0.00	≈0.00
5	8	N/A	N/A	≈0.00	≈0.00
	10	249	249	≈0.00	≈0.00
	12	132	132	≈0.00	≈0.00
	16	129	129	≈0.00	≈0.00
6	8	N/A	N/A	≈0.00	≈0.00
	10	249	249	≈0.00	≈0.00
	12	132	132	≈0.00	≈0.00
	16	129	129	0.03	0.03
7	8	N/A	N/A	≈0.00	≈0.00
	10	249	249	≈0.00	≈0.00
	12	132	132	0.02	0.02
	16	129	129	0.12	0.12
8	8	N/A	N/A	≈0.00	≈0.00
	10	249	249	0.02	0.02
	12	132	132	0.05	0.05
	16	129	129	0.47	0.47

the enumeration method to produce all unique enumerations for a given M and $TSVmax$.

As expected, the run time of the heuristic that includes look-ahead is typically greater than that without look-ahead. We further observe that the increase in time, while not insignificant, is also not very large. Since the LookAhead algorithm only runs on a very small subset of the total elements, even across many enumerations the added complexity does not incur a significant time increase. $O(n^5)$ is very much an upper bound, and actual results are far better than worst-case scenarios.

We conclude that the heuristic method utilizing look-ahead is superior to that without when dealing with tighter TSV constraints. Table II shows that, a large majority of the time, the look-ahead heuristic is capable of determining a solution given TSV constraints under which the other heuristic is unable to produce results. Furthermore, Table II and particularly Table I reveal that, under tight TSV constraint in which both methods produced results, the look-ahead heuristic performed significantly better. Under looser TSV constraints, it is difficult to determine which heuristic will give more optimal results. Different placement decisions lead to different optimizations such that both methods perform near to each other, but one method may be slightly better or worse than the other.

As the TAM width and number of TSVs available increase, run time also increases. The number of enumerations that must be considered increases significantly for large M and $TSVmax$ values. Furthermore, time needed to produce all enumerations becomes a larger contributor to the total run time. The number of layers used by the core has no bearing on test time but obviously increases the number of TSVs needed to produce results, which directly affects CPU time.

V. CONCLUSIONS

We have presented several optimization techniques for minimizing test time by reducing wrapper chain length in 3D core-based SOCs. We have considered constraints on the TAM width and the number of TSVs available for testing. We have provided two heuristic approaches, one that requires less time to run and one that produces better results under tight TSV restrictions. Results have been presented for four representative cores from the ITC'02 test benchmarks. Results show that the heuristic methods solve the problem of 3D wrapper optimization in a timely manner, producing varying degrees of near-optimality. This early work on wrapper optimization is expected to pave the way for design-fortestability tools for emerging 3D core-based ICs.

REFERENCES

[1] *IEEE Std. 1500: IEEE Standard Testability Method for Embedded Core-Based Integrated Circuits*. *IEEE Press*, New York, 2005.

[2] V. Iyengar, K. Chakrabarty, and E. J. Marinissen, "Test Wrapper and Test Access Mechanism Co-optimization for System-on-chip", *JETTA*, vol. 18, pp. 213-230, 2002.

[3] Q. Xu and N. Nicolici, "Resource-constrained System-on-a-chip Test: a Survey," *IEE Proc. Comp. Dig. Tech.*, vol. 152, no. 1, pp. 67-81, 2005.

[4] E. Larsson, K. Arvidsson, H. Fujiwara, and Z. Peng, "Efficient Test Solutions for Core-based Designs," *TCAD*, vol. 23, no. 5, pp. 758-775, 2004.

[5] K. Banerjee et al., "3-D ICs: a Novel Chip Design for Improving Deep-submicrometer Interconnect Performance and Systems-on-chip Integration," *Proc. IEEE*, vol. 89, no. 5, pp. 602-633, 2001.

[6] R. Weerasekera et al., "Extending Systems-on-chip to the Third Dimension: Performance, Cost and Technological Tradeoffs," in *ICCAD*, 2007.

[7] W. R. Davis et al., "Demystifying 3D ICs: the Pros and Cons of Going Vertical," *IEEE Design and Test of Computers*, vol. 22, no. 6, pp. 498-510, 2005.

[8] Y. Xie, G. H. Loh, and K. Bernstein, "Design Space Exploration for 3D Architectures," J. *Emerg. Technol. Comput. Syst.*, 2(2):65-103, 2006.

[9] G. Loh, Y. Xie, and B. Black, "Processor Design in 3D Die Stacking Technologies," *IEEE Micro* Vol 27, No. 3, 2007, pp.31-48

[10] K. Puttaswamy and G. H. Loh, "Thermal Herding: Microarchitecture Techniques for Controlling Hotspots in High-Performance 3D-Integrated Processors," *IEEE High Performance Computer Architecture*, pp. 193-204, 2007.

[11] X. Wu, P. Falkenstern, K. Chakrabarty, and Y. Xie, "Scan-chain Design and Optimization for 3D ICs,"' *ACM Journal on Emerging Technologies in Computing Systems*, vol. 5, Article 9, July 2009.

[12] X. Wu, Y. Chen K. Chakrabarty, and Y. Xie, "Test-access Mechanism Optimization for Core-based Three-dimensional SOCs," *ICCD* 2008.

[13] E. J. Marinissen, S. K. Goel, and M. Lousberg, "Wrapper Design for Embedded Core Test," *Proc. Int'l Test Conf.*, pp. 911-920, 2000.

[14] Y. Huang et al., "Optimal Core Wrapper width Selection and SOC Test Scheduling based on 3-D Bin Packing Algorithm,"' In *International Test Conference*, pp. 7482, 2002.

[15] S. K. Goel and E. J. Marinissen. "SOC Test Architecture Design for Efficient Utilization of Test Bandwidth,"' *ACM Transactions on Design Automation of Electronic Systems*, 8(4):399429, 2003.

[16] Q. Xu and N. Nicolici, "Resource-Constrained System-on-a-Chip Test: A Survey, *IEE Proceedings: Computers and Digital Techniques*. vol. 152, pp. 67-81, Jan. 2005.

[17] L. Jiang, L. Huang, and Q. Xu, "Test Architecture Design and Optimization for Three-dimensional SoCs,"' *DATE*, pp. 220-225, 2009.