www.ietdl.org

Published in IET Computers & Digital Techniques
Received on 10th November 2009
Revised on 29th March 2010
doi: 10.1049/iet-cdt.2009.0111

Special Issue on Three-dimensional Integrated Circuits Design

ISSN 1751-8601

Test-wrapper optimisation for embedded cores in through-silicon via-based three-dimensional system on chips

B. Noia K. Chakrabarty

Department of Electrical and Computer Engineering, Duke University, Durham, NC 27708, USA
E-mail: brn2@duke.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 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 study addresses wrapper optimisation in 3D ICs based on through-silicon vias (TSVs) for vertical interconnects. The authors objective is to minimise the scan-test time for a core under constraints on the total number of TSVs available for testing. The authors present an optimal solution based on an integer linear programming model as well as two polynomial-time heuristic solutions. Simulation results are presented for embedded cores from the ITC 2002 SOC test benchmarks.

1 Introduction

System-on-chip (SOC) designs are common for today's integrated circuits, and they are often comprised of a number of embedded cores [1]. 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 the 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 [2], 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 optimisation are critical. Although a lot of research [3–5] 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 [6, 7]. Owing 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 [8–10]. 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 [9]. 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. Course-granularity partitioning reduces the design effort (because each IP core itself does not need any modification), and the test wrapper design for an individual core can follow the conventional approach in 2D IC design.
2. *Fine-granularity partitioning:* With this approach, each core in the SOC chip is partitioned into multiple layers [9]. Compared to coarse-granularity partitioning, this approach not only reduces the global interconnects among cores, but also reduces global and local interconnects within a single core, bringing with it improvements in power and performance. For arithmetic logic, a barrel shifter implemented

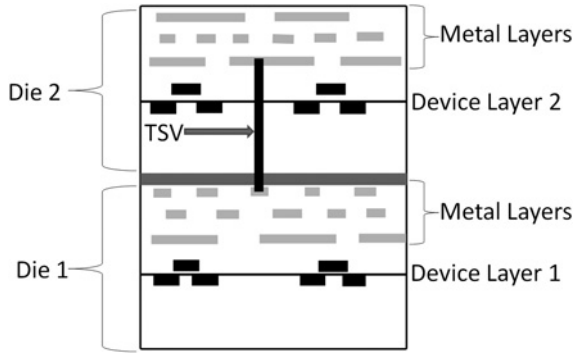

Fig. 1 *Example of a 3D IC manufactured by die stacking with TSV connections*

in a 3D architecture showed a 9% reduction in latency and an 8% reduction in energy consumption [11]. For an entire core, a repartitioning of the Intel Core 2 processor across four die-stacked tiers was examined. In this case, both the two cores and the L2 cache were spread across all four tiers. Significant reductions in latency were reported, including reductions of 52% for the reorder buffer, 53% for the register file and 36% for the ALU and bypass. This resulted in an overall 47.9% increase in frequency and 47.0% performance improvement [12]. Furthermore, 3D-integrated arithmetic units scale better than their 2D counterparts, having lower latency degredation and smaller increases in energy consumption with increases in issue widths and operating temperatures [13]. The fine-granularity partitioning approach can also result in footprint reduction for each individual core and bring new opportunities for design innovation.

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) [14]. Test-wrapper design for 3D SOC systems with true 3D cores is more challenging compared to the case of coarse-granularity 3D SOC design [15], because scan chains for each individual core can now span multiple layers and the test wrapper must interface with I/O and scan-chain terminals located on any layer.

Although the manufacture of 3D ICs is now feasible, design-automation and the 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 optimisation of test-access architectures have been well studied [3, 5, 16]. Optimisation methods have included ILP [3], rectangle packing [3, 17], iterative refinement [18] and other heuristics [5, 19]. 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 [15, 20]. 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 optimisation in 3D ICs that use TSVs for vertical interconnects and fine-granularity partitioning. Our objective is to minimise 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 optimisation.

The wrapper optimisation problem is known to be NP-complete for 2D ICs [3, 16]. Therefore it follows from the 'method of restriction' that this optimisation 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 optimisation.

In this paper, we address wrapper chain optimisation 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 optimisation differs from that in 2D ICs in that scan-chains can span multiple layers, core I/Os 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 [9], 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 organised as follows: Section 2 uses a simple example to motivate this work and briefly outlines prior work. Section 3 presents an integer linear programming model and two heuristic approaches for solving the wrapper optimisation problem. Section 4 presents experimental results for several cores from the ITC 2002 SOC test benchmarks. Finally, Section 5 presents conclusions drawn from this work.

2 Problem formulation

2.1 Motivational example

Fig. 1 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 utilise. 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 utilising 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 in the 3D IC of Fig. 2. Fig. 2*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 three registers and one with five. The five-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 three TSVs. Figs. 2*b* and *c* show two possible solutions using two wrapper chains. As can be seen, both solutions use three TSVs but the wrapper chains in Fig. 2*c* are balanced, whereas those in Fig. 2*b* are

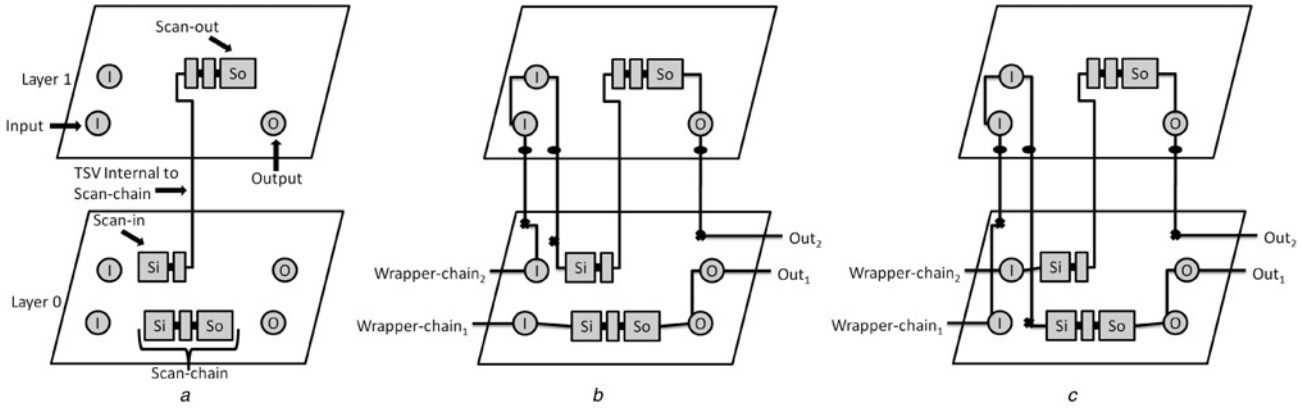

Fig. 2 *3D IC with two possible wrapper-chain solutions*

unbalanced. The solution in Fig. 2*c* leads to less test time. The balancing of the lengths of wrapper chains for large cores is a challenging design task. Optimisation methods are needed for determining the best wrapper designs under a set of design and technology constraints.

2.2 Problem formulation

The goal of wrapper optimisation is to minimise the test time for each core. The optimisation problem addressed in this paper is as follows. Given a core (placement of I/Os, 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 minimised.

This paper approaches the problem in two ways. First, an integer linear programming (ILP) model formulation is presented. Since the wrapper design problem is NP-compete [3], CPU time of the ILP model for realistic cores would be too high. Thus, we have developed two faster heuristic approaches to produce near-optimal solutions. These two models extend the bin design method presented in [16] for 2D wrapper design. The first heuristic presented is less close to optimality but faster for cores with many scan-chains, whereas the second heuristic is slower for cores with many scan-chains but produces better results under tight TSV constraints. We also use a simplified form of the ILP model to obtain lower bounds on the test time, which allows us to evaluate the effectiveness of the heuristic methods.

3 3D wrapper optimisation

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.

3.1 ILP formulation

In a linear programming model, a linear objective function is minimised subject to a set of linear constraints. For integer linear programming, as in the solution we present, all variables are integers. The 3D wrapper optimisation problem is defined as follows: Given a set $S = \{s_1, s_2, \ldots, s_n\}$ of n elements for a 3D core, a set $l = \{l_1, l_2, \ldots, l_n\}$ of input layers on which each element resides, a set $L = \{L_1, L_2, \ldots, L_n\}$ of lengths for each element, a global TSV constraint TSVlimit, and a TAM width M, determine an assignment for cores to wrapper chains such that testing time is minimised. The elements correspond to functional I/Os, scan-in terminals and scan-out terminals. Functional I/Os have a length of one and scan-out elements have a length of zero. The length L_i for a scan-in element equals the length of the corresponding scan-chain.

We first use a directed-graph model to formulate the optimisation problem. The vertices of the graph represent the elements (I/Os, and scan-chains) in a 3D core. There are two edges between every pair of vertices. Each edge is weighted with the number of TSVs that are needed to connect the elements corresponding to the two vertices. If two elements should not be connected in a wrapper chain, the weight is made prohibitively large. A simple example of a graph for 3D wrapper optimisation is shown in Fig. 3. In this example, there are four elements: an input, an output, and the scan-in and scan-out of a single scan-chain. The input can only connect to other inputs or scan-ins. Thus, the edges connecting an input to another input or a scan-in is weighted as the positive difference (d in the example) between the layer on which the first element (the input) resides and the layer on which the second element (the scan-in) resides. A scan-in element can only connect to its respective scan-out, and since TSVs internal to scan-chains are not counted against the global TSV limit, this edge is weighted as 0. Scan-outs can only connect to the scan-ins of other scan-chains or to outputs, and these edges have weights d. Finally, outputs can only connect to other

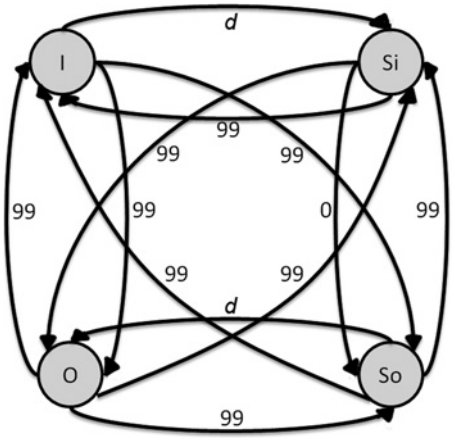

Fig. 3 *Directed-graph model for deriving the IPL-based solutions*

doi: 10.1049/iet-cdt.2009.0111

outputs, also with weight d. All other edges are weighted with prohibitively large values (99 in this case), since inputs can never connect to outputs, outputs can never connect to scan-ins etc.

We define the weights between vertices as w_{ik}, where i is the first element and k is the second, connecting element. The full matrix of weights can be created using the given set l. It should be noted that $w_{ik} = 0$ when $i = k$, since an element cannot connect to itself. We further define the binary variable $y_{ikj} = 1$ if element k immediately follows element i in wrapper chain j. Using these variables, the number of TSVs used by all the elements across all the scan-chains can be calculated by

$$\sum_j \sum_{i,k} y_{ikj} w_{ik} \tag{1}$$

That is, the number of TSVs used is equal to the sum of the weights w_{ik} of all connections between subsequent elements y_{ikj} summed across all wrapper chains j. This must be equal to or less than the global TSV constraint TSVlimit.

The set S of elements, as was seen in Fig. 3, consists of functional I/Os, scan-ins, and scan-outs. For the purposes of the ILP solution, s_1 is a dummy input of length $L_1 = 0$ and layer $l_1 = 0$. Likewise, s_n is a dummy output of length $L_n = 0$ and layer $l_n = 0$. The weights for the dummy I/O are calculated as for any other I/O. The dummy I/O will be utilised to bound each wrapper chain and ensure that the number of TSVs necessary to reach from layer 0 to the first element and from the last element back to layer 0 are considered. This is necessary because all pins on the IC reside on layer 0.

We need to ensure that all elements are placed into one and only one wrapper chain and that the dummy I/O exist in every wrapper chain. We define the binary variable $x_{ij} = 1$ if element i is included in wrapper chain j. To achieve element placement in the wrapper chains, we define the constraints

$$\sum_j x_{ij} = 1, \quad 2 \le i \le n-1 \tag{2}$$

$$x_{1j} = 1 \quad \forall j \tag{3}$$

$$x_{nj} = 1 \quad \forall j \tag{4}$$

Equation (2) guarantees that, for element i, $x_{ij} = 1$ for only one wrapper chain j and 0 for all others. Thus, each element (except for the two dummy elements) must exist in only one wrapper chain. Constraints (3) and (4) ensure that the dummy I/O are placed in every wrapper chain j.

We must establish that every element in a wrapper chain has a successor except for the dummy output, as this should be the last element in every wrapper chain. That is, if an element i is placed in wrapper chain j ($x_{ij} = 1$) then there must be one and only one element k also in wrapper chain j that succeeds element i ($y_{ikj} = 1$). We can describe this constraint as

$$x_{ij} - \sum_k y_{ikj} = 0 \quad \forall i \ne n, j \tag{5}$$

By summing over all successor elements k for each wrapper chain j, equation (5) ensures that if $x_{ij} = 1$, then there is one corresponding $y_{ikj} = 1$. The dummy output ($i = n$) is left out of this constraint.

Similar to (5), we must establish that every element in a wrapper chain has a predecessor except for the dummy input, as this should be the first element in every wrapper chain. The predecessor and successor in a wrapper chain will correspond to each other. We model this similarly to (5) as

$$x_{ij} - \sum_k y_{kij} = 0 \quad \forall i \ne 1, j \tag{6}$$

The dummy input ($i = 1$) is left out of this constraint. The mathematical programming model (Model 1) is shown in Fig. 4. Note that the min–max objective function can be linearised using standard techniques [21].

As established thus far, the ILP model cannot prevent loops from forming within a wrapper chain. To prevent loops from forming, we introduce the integer variable u_{ij}, which represents the position of element i in wrapper chain j. When element k immediately follows element i in wrapper chain j ($y_{ikj} = 1$), we want u_{kj} to take the value of the position of k relative to i, such that $u_{ij} + 1 = u_{ik}$. If there were a loop in the wrapper chain, then the value of u_{ij} would increase infinitely. Thus, we want to place a constraint on the maximum value of any given u_{ij}. The position of any element should never exceed n, the total number of elements, so we make this our limit. These constraints are described as

$$y_{ikj}(u_{ij} + 1 - u_{kj}) = 0 \quad \forall i \ne k, k, j \tag{7}$$

$$u_{ij} \le n \quad \forall i, j \tag{8}$$

Since (7) contains the product of two variables, it needs to be linearised. We do this by adding a new integer variable, z_{ijikj} that takes the value of $u_{ij}y_{ikj}$. We constrain the behaviour of z_{ijikj} as

$$z_{ijikj} \ge 0 \quad \forall i, k, j \tag{9}$$

$$z_{ijikj} - n y_{ikj} \le 0 \quad \forall i \ne k, k, j \tag{10}$$

$$z_{ijikj} - u_{ij} \le 0 \quad \forall i \ne k, k, j \tag{11}$$

$$u_{ij} - z_{ijikj} + n y_{ikj} \le n \quad \forall i \ne k, k, j \tag{12}$$

Now, we can express (7) linearly using z_{ijikj} as follows

$$z_{ijikj} + y_{ikj} - z_{kjikj} = 0 \quad \forall i \ne k, k, j \tag{13}$$

Objective:

$$\text{Minimise} \max_{1 \le j \le M} \sum_{i=1}^{n} x_{ij} \cdot L_i$$

Subject to:

$$\sum_j \sum_{i,k} y_{ikj} \cdot w_{ik} \le TSVlimit$$

$$\sum_j x_{ij} = 1 \qquad 2 \le i \le n-1$$

$$x_{ij} - \sum_k y_{ikj} = 0 \qquad \forall i \ne n, j$$

$$x_{ij} - \sum_k y_{kij} = 0 \qquad \forall i \ne 1, j$$

$$x_{1j} = 1 \qquad \forall j$$

$$x_{nj} = 1 \qquad \forall j$$

Fig. 4 *Mathematical programming model for 3D wrapper optimisation with loops (Model 1)*

Objective:

$$\text{Minimise} \max_{1 \le j \le M} \sum_{i=1}^{n} x_{ij} \cdot L_i$$

Subject to:

$$\sum_{j} \sum_{i,k} y_{ikj} \cdot w_{ik} \le TSVlimit$$

$$\sum_{j} x_{ij} = 1 \qquad 2 \le i \le n-1$$

$$x_{ij} - \sum_{k} y_{ikj} = 0 \qquad \forall i \ne n, j$$

$$x_{ij} - \sum_{k} y_{kij} = 0 \qquad \forall i \ne 1, j$$

$$x_{1j} = 1 \qquad \forall j$$

$$x_{nj} = 1 \qquad \forall j$$

$$z_{ijikj} \ge 0 \qquad \forall i, k, j$$

$$z_{ijikj} - n \cdot y_{ikj} \le 0 \qquad \forall i \ne k, k, j$$

$$z_{ijikj} - u_{ij} \le 0 \qquad \forall i \ne k, k, j$$

$$u_{ij} - z_{ijikj} + n \cdot y_{ikj} \le n \qquad \forall i \ne k, k, j$$

$$z_{ijikj} + y_{ikj} - z_{kjikj} = 0 \qquad \forall i \ne k, k, j$$

Fig. 5 *Complete mathematical programming model (Model 2) for 3D wrapper optimisation*

The complete ILP model (Model 2) is summarised in Fig. 5. The linearisation of the min–max objective function is not shown explicitly. This model is very complex, and is thus impractical for use with large cores. For this reason, we use the incomplete ILP model that allows loops to form in order to find lower bounds on wrapper lengths for cores. This allows us to evaluate heuristic methods with a model that is significantly less complex and useable for medium-sized cores. The ILP model can also be used to obtain lower bounds quickly using LP-relaxation [22].

3.2 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, the first corresponding to the layer of the scan-in and the second to the layer of the scan-out belonging to the scan-chain represented by the element. For I/Os, which reside on only one layer and contain only one 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 that delineates the element as being an I/O, our scan-chain. Each element will be placed in only one wrapper chain. Fig. 6 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. 6 *Element data structure*

The heuristic approach to this problem is based on the bin-design method for 2D wrapper chains [16]. 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 utilise. 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 $\text{Lin} = \{\text{lin}_1, \text{lin}_2, \ldots, \text{lin}_n\}$ corresponding to the layer on which a scan-in, functional I/O resides for set S, a set of layers $\text{Lout} = \{\text{lout}_1, \text{lout}_2, \ldots, \text{lout}_n\}$ corresponding to the layer of the scan-out (or equal to the corresponding L in value for I/Os) 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.

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 Fig. 7. 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 initialising n element data structures such that their variables correspond to the values in sets S, Lin, Lout and T. Likewise, M bins are initialised 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

```
BinDesign({c_1, ... , c_M}, S, Lin, Lout, T)
  Initialise n elements; Initialise 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);
```

Fig. 7 *Bin design algorithm*

doi: 10.1049/iet-cdt.2009.0111

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 prioritised, 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 utilise 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 initialisation 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 Fig. 8. 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 Fig. 8. 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. 9. Here, the minimal number of TSVs used for a wrapper chain in which the highest input is on layer 3 is three. 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 utilised 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

```
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. 8 *InsertElement algorithm*

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 initialised 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

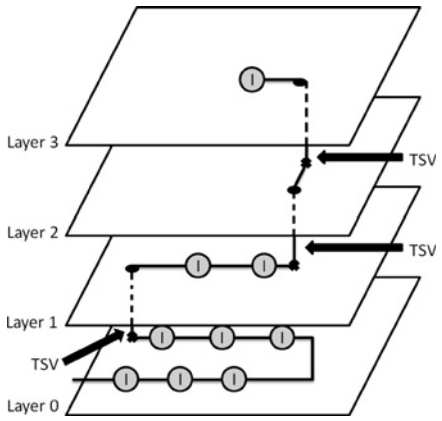

Fig. 9 *Conceptual view of the number of TSVs used for inputs in a wrapper chain*

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. 10. Fig. 10 shows two possible solutions for an example wrapper chain. The one on the left is the minimal solution resulting in three 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 five TSVs.

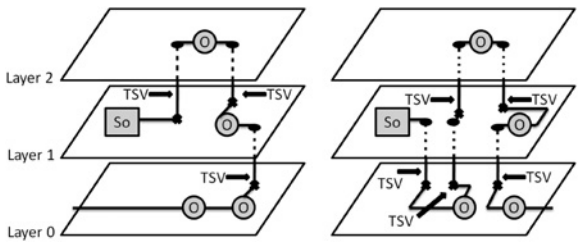

Fig. 10 *Optimal and suboptimal solution to connecting outputs in a wrapper chain*

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

3.3 Heuristic method with look ahead

In Fig. 8, the method of placing scan chains in a wrapper chain is the main obstacle towards achieving near-optimal results. The problem with Fig. 8 will be explained using the simple example shown in Fig. 11. In Fig. 11*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, Fig. 8 will select the first of the closest scan-chains. As can be seen by the next series of connections in Fig. 11*b*, labelled in the order in which they were made, this leads to a suboptimal result using one TSV. Fig. 11*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

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
```

Fig. 12 *Section of the InsertElement algorithm*

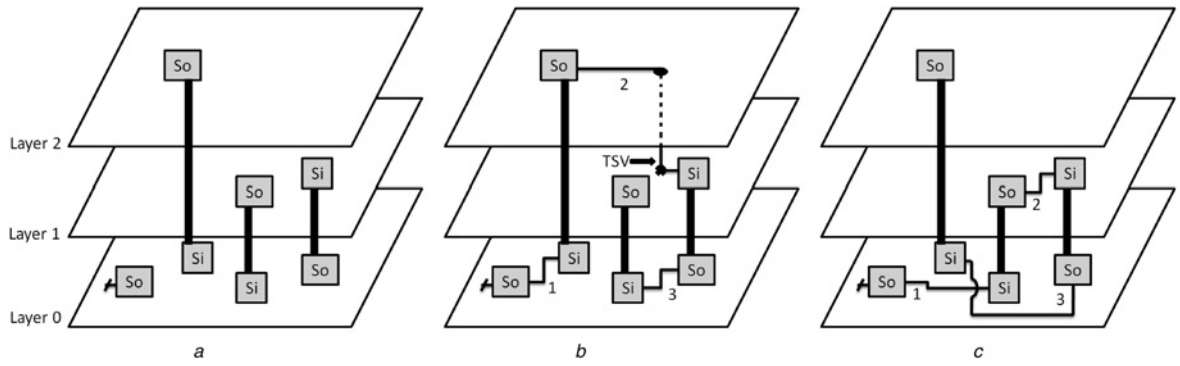

Fig. 11 *Conceptual view of the shortcomings of Fig. 8*

doi: 10.1049/iet-cdt.2009.0111

Table 1 Results on wrapper optimisation for a fictitious core with three layers

M	TSVmax	Shortest wrapper (no LA)	Shortest wrapper (LA)	Shortest wrapper (ILP)	Shortest wrapper (subtour)
2	5	N/A	N/A	N/A	9
	6	9	9	9	9
M	TSVmax	CPU time (No LA), min	CPU time (LA), min	CPU time (ILP), min	CPU time (subtour), min
2	5	0.00	0.00	42.68	0.01
	6	0.00	0.00	42.93	0.01

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 one element.

To use look ahead, we must alter how InsertElement deals with scan-chains. Fig. 12 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 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 Fig. 8 had.

3.4 Analysis of heuristics

The BinDesign algorithm relies on the use of enumeration, and we produce a complete set of unique enumerations. For

Table 2 Results on wrapper optimisation for core 7 of SOC d281 with three layers

M	TSVmax	Shortest wrapper (no LA)	Shortest wrapper (LA)	CPU time (no LA), min	CPU time (LA), min	Lower bound
2	12	N/A	N/A	0.00	0.00	1064
	14	N/A	N/A	0.00	0.13	
	16	1633	1623	0.22	0.98	
	18	1064	1064	1.42	3.13	
	20	1064	1064	1.88	3.40	
	22	1064	1064	2.03	3.63	
3	12	N/A	N/A	0.00	0.00	710
	14	N/A	N/A	0.00	0.33	
	16	1633	1347	0.25	1.07	
	18	710	752	3.67	7.52	
	20	710	710	5.78	9.13	
	22	710	710	6.73	10.42	
4	12	N/A	N/A	0.00	0.00	532
	14	N/A	N/A	0.02	0.67	
	16	1633	1347	0.30	1.52	
	18	532	611	7.05	11.47	
	20	532	556	12.98	16.42	
	22	532	545	16.17	20.70	
5	12	N/A	N/A	0.02	0.02	426
	14	N/A	N/A	0.05	1.02	
	16	1633	1347	0.37	2.13	
	18	470	486	11.98	14.48	
	20	458	479	23.95	22.25	
	22	426	427	32.28	33.20	
6	12	N/A	N/A	0.02	0.03	355
	14	N/A	N/A	0.07	1.33	
	16	1633	1347	0.47	2.73	
	18	459	441	16.75	18.18	
	20	419	435	38.30	29.15	
	22	355	364	53.83	46.37	

Table 3 Results on wrapper optimisation for core 13 of SOC p93791 with four layers

M	TSVmax	Shortest wrapper (no LA)	Shortest wrapper (LA)	CPU time (no LA), min	CPU time (LA), min	Lower bound
2	18	N/A	N/A	0	0.03	4835
	20	N/A	4835	0	0.10	
	22	4875	4835	0.03	0.13	
	24	4835	4838	0.12	0.20	
	28	4835	4835	0.17	0.28	
	32	4835	4835	0.18	0.32	
	34	4835	4835	0.18	0.33	
3	18	N/A	N/A	0.02	0.07	3223
	20	3371	5008	0.05	0.12	
	22	3328	3362	0.12	0.28	
	24	3257	3253	0.30	0.57	
	28	3247	3228	0.65	1.13	
	32	3226	3225	0.93	1.42	
	34	3226	3225	1.02	1.50	
4	18	N/A	N/A	0.05	0.15	2418
	20	N/A	5008	0.12	0.30	
	22	2598	2548	0.25	0.68	
	24	2477	2462	0.70	1.53	
	28	2435	2457	2.03	3.05	
	32	2432	2452	2.95	4.03	
	34	2432	2452	3.37	4.57	
5	18	N/A	N/A	0.08	0.25	1934
	20	N/A	2107	0.20	0.52	
	22	2520	2059	0.50	1.13	
	24	2042	2014	1.10	2.47	
	28	1963	1994	4.18	5.82	
	32	1963	1975	7.03	9.00	
	34	1963	1975	8.58	11.22	
6	18	N/A	N/A	0.15	0.40	1612
	20	N/A	2107	0.37	0.83	
	22	2520	1885	0.88	1.87	
	24	1820	1694	1.83	4.17	
	28	1645	1646	8.26	10.77	
	32	1645	1645	16.7	20.72	
	34	1645	1645	22.07	26.93	
7	18	N/A	N/A	0.42	0.72	1382
	20	N/A	2107	0.97	1.62	
	22	2520	1885	2.18	3.77	
	24	1820	1694	4.83	8.02	
	28	1427	1459	21.00	24.48	
	32	1426	1428	56.65	62.55	
	34	1426	1428	89.73	96.12	
8	18	N/A	N/A	1.47	1.82	1209
	20	N/A	2107	3.68	4.45	
	22	2520	1885	8.80	10.6	
	24	1820	1694	20.07	24.03	
	28	1316	1314	92.23	97.15	
	32	1250	1252	288.17	347.53	
	34	1242	1242	535.35	634.60	

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.

The unmodified InsertElement of Fig. 8 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 $(1/2)n^2 + (1/2)n$ computations. Therefore, InsertElement has a complexity of $O(n^2)$. When combined with the BinDesign algorithm, the

doi: 10.1049/iet-cdt.2009.0111

Table 4 Results on wrapper optimisation for core 4 of SOC p93791 with three layers

M	TSVmax	Shortest wrapper (no LA)	Shortest wrapper (LA)	CPU time (no LA), min	CPU time (LA), min	Lower bound
2	12	N/A	N/A	0.00	0.00	77
	13	N/A	132	0.00	0.00	
	14	89	87	0.00	0.00	
	16	77	79	0.00	0.02	
	18	77	79	0.00	0.02	
	20	77	77	0.00	0.02	
3	12	N/A	N/A	0.00	0.00	51
	13	N/A	69	0.00	0.00	
	14	84	69	0.00	0.02	
	16	51	55	0.02	0.03	
	18	51	53	0.03	0.05	
	20	51	52	0.03	0.07	
4	12	N/A	N/A	0.00	0.00	39
	13	N/A	59	0.00	0.02	
	14	79	59	0.02	0.03	
	16	44	46	0.03	0.05	
	18	39	43	0.07	0.12	
	20	39	42	0.08	0.13	
5	12	N/A	N/A	0.00	0.00	31
	13	N/A	59	0.00	0.02	
	14	79	59	0.02	0.03	
	16	39	38	0.05	0.08	
	18	37	37	0.12	0.15	
	20	35	33	0.12	0.23	
6	12	N/A	N/A	0.00	0.02	26
	13	N/A	59	0.00	0.03	
	14	79	47	0.03	0.05	
	16	42	35	0.08	0.13	
	18	31	31	0.18	0.25	
	20	28	31	0.32	0.40	
7	12	N/A	N/A	0.00	0.03	22
	13	N/A	59	0.02	0.05	
	14	79	47	0.07	0.10	
	16	41	34	0.17	0.23	
	18	30	31	0.43	0.50	
	20	30	29	0.87	0.97	
8	12	N/A	N/A	0.00	0.07	20
	13	N/A	59	0.08	0.12	
	14	79	47	0.18	0.22	
	16	41	34	0.52	0.58	
	18	30	27	1.37	1.48	
	20	25	27	3.32	3.43	

entire wrapper optimisation 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 $(1/2)n^2 + (1/2)n$ iterations as with the InsertElement algorithm. This gives the entire wrapper optimisation algorithm with look ahead a complexity of $O(n^5)$ per enumeration.

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 I/Os. For example, core 7 of SOC d281 from the ITC'02 SOC test benchmarks contains 1510 elements, only 20 of which are scan-chains.

4 Experiments and results

4.1 Experimental setup

The ILP models and optimisation 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, as well as one fictitious core. The heuristics were coded in C++ and were compiled and ran using Microsoft Visual Studio Professional 2008. The ILP models were ran using XPRESS-MP [23]. The layers on which I/Os, scan-ins, and scan-outs resided were determined randomly.

Table 5 Results on wrapper optimisation for core 5 of SOC h953 with three layers

M	TSVmax	Shortest wrapper (no LA)	Shortest wrapper (LA)	CPU time (no LA), min	CPU time (LA), min	Lower bound
2	8	N/A	N/A	0.00	0.00	258
	10	386	386	0.00	0.00	
	12	258	258	0.00	0.00	
	14	265	265	0.00	0.00	
	16	258	258	0.00	0.00	
3	8	N/A	N/A	0.00	0.00	241
	10	249	249	0.00	0.00	
	12	247	247	0.00	0.00	
	14	241	241	0.00	0.00	
	16	241	241	0.00	0.00	
4	8	N/A	N/A	0.00	0.00	129
	10	249	249	0.00	0.00	
	12	132	132	0.00	0.00	
	14	130	130	0.00	0.00	
	16	129	129	0.00	0.00	
5	8	N/A	N/A	0.00	0.00	103
	10	249	249	0.00	0.00	
	12	132	132	0.00	0.00	
	14	129	129	0.00	0.00	
	16	129	129	0.00	0.00	
6	8	N/A	N/A	0.00	0.00	86
	10	249	249	0.00	0.00	
	12	132	132	0.00	0.00	
	14	129	129	0.02	0.02	
	16	129	129	0.03	0.03	
7	8	N/A	N/A	0.00	0.00	74
	10	249	249	0.00	0.00	
	12	132	132	0.02	0.02	
	14	129	129	0.05	0.05	
	16	129	129	0.12	0.12	
8	8	N/A	N/A	0.00	0.00	65
	10	249	249	0.02	0.02	
	12	132	132	0.05	0.05	
	14	129	129	0.17	0.12	
	16	129	129	0.47	0.47	

4.2 Results for heuristics and ILP models

Table 1 gives results for a small, fictitious core containing three scan-chains of length 4, 3, and 2, 4 inputs, and 4 outputs. Table 2 shows results from core 7 of SOC d281, Table 3 shows results from core 13 of SOC p93791, Table 4 shows results from core 4 of SOC p93791 and Table 5 shows results for core 5 of SOC h953 for both heuristic methods presented in Sections 3.2 and 3.3. 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 the enumeration method to produce all unique enumerations for a given M and TSVmax. The seventh column indicates the lower bound determined by ILP Model 1.

Table 1 demonstrates well the complexity of the ILP model if an exact solution is attempted. While the model that produces loops (subtour) and the two heuristic methods produce results in less than a second, the complete ILP model requires over 42 minutes. A minimum wrapper length of 9 is the optimal solution to the core given a width of 2, and while the heuristic and complete ILP methods produce this result when TSV constraints allow, the ILP model that includes loops will produce this result unless the TSV limit is prohibitively small.

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 utilising look-ahead is superior to that without when dealing with tighter TSV constraints. Table 3 shows that, a large majority of the time, the look-ahead heuristic is capable of determining a solution

doi: 10.1049/iet-cdt.2009.0111

given TSV constraints under which the other heuristic is unable to produce results. Furthermore, Table 3 and particularly Table 2 reveal that, under tight TSV constraint in which both methods produced results, the look-ahead heuristic performed significantly better. The reason for this is that the look-ahead method makes early optimisation decisions better suited to reducing TSV costs than with no look-ahead, but at the expense of run time and possibly future decisions that would more greatly reduce test time. Under looser TSV constraints, it is difficult to determine which heuristic will give more optimal results. Different placement decisions lead to different optimisations such that both methods perform near to each other, but one method may be slightly better or worse than the other. Both methods do approach, if not hit, the lower bound as determined by the LP model.

As the TAM width and number of TSVs available increase, run time also increases. The number of enumerations that must be considered increases significantly at large M and TSVmax values. Furthermore, the amount of time necessary to produce all enumerations begins to become a larger factor in 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.

5 Conclusions

We have presented several optimisation techniques for minimising 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, and an optimal ILP model. Results have been presented for four representative cores from the ITC'02 test benchmarks and for a fictitious core. Results show that the heuristic methods solve the problem of 3D wrapper optimisation in a timely manner, producing varying degrees of near-optimality. This early work on wrapper optimisation is expected to pave the way for design-for-testability tools for emerging 3D core-based ICs.

6 Acknowledgments

The authors thank Prof. Yuan Xie from Pennsylvania State University for valuable discussions. This work was supported in part by a Master's Scholarship from the Semiconductor Research Corporation (SRC). A preliminary version of this paper was published in Proc. IEE Conf. Computer Design, 2009.

7 References

1 Marinissen, E.J., Iyengar, V., Chakrabarty, K.: 'A set of benchmarks for modular testing of SOCs'. Proc. Int. Test Conf., Baltimore, USA., October 2002, pp. 519–528
2 IEEE Std. 1500: 'IEEE standard testability method for embedded core-based integrated circuits'2005)
3 Iyengar, V., Chakrabarty, K., Marinissen, E.J.: 'Test wrapper and test access mechanism co-optimisation for system-on-chip', *J. Electron. Test. Theory Appl.*, 2002, **18**, pp. 213–230
4 Xu, Q., Nicolici, N.: 'Resource-constrained system-on-a-chip test: a survey', *Proc. Comp. Dig. Tech.*, 2005, **152**, (1), pp. 67–81
5 Larsson, E., Arvidsson, K., Fujiwara, H., Peng, Z.: 'Efficient test solutions for core-based designs', *J. Technol. Comput. Aided Des.*, 2004, **23**, (5), pp. 758–775
6 Banerjee, K., *et al.*: '3-D ICs: a novel chip design for improving deep-submicrometer interconnect performance and systems-on-chip integration', *Proc. IEEE*, 2001, **89**, (5), pp. 602–633
7 Weerasekera, R., *et al.*: 'Extending systems-on-chip to the third dimension: performance, cost and technological tradeoffs'. Proc. ICCAD, 2007, pp. 212–219
8 Davis, W.R., *et al.*: 'Demystifying 3D ICs: the pros and cons of going vertical', *IEEE Des. Test Comput.*, 2005, **22**, (6), pp. 498–510
9 Xie, Y., Loh, G.H., Bernstein, K.: 'Design space exploration for 3D architectures', *J. Emerg. Technol. Comput. Syst.*, 2006, **2**, (2), pp. 65–103
10 Loh, G.H., Xie, Y., Black, B.: 'Processor design in 3D die stacking technologies', *IEEE Micro.*, 2007, **27**, (3), pp. 31–48
11 Puttaswamy, K., Loh, G.H.: 'The impact of 3-dimensional integration on the design of arithmetic units'. Proc. IEEE Int. Symp. on Circuits and Systems, 2006
12 Puttaswamy, K., Loh, G.H.: 'Thermal herding: microarchitecture techniques for controlling hotspots in high-performance 3D-integrated processors'. *IEEE High Perform. Comput. Archit.*, 2007, pp. 193–204
13 Puttaswamy, K., Loh, G.H.: 'Scalability of 3D-integrated arithmetic units in high-performance microprocessors'. Proc. Design Automation Conf., 2007, pp. 622–625
14 Wu, X., Falkenstern, P., Chakrabarty, K., Xie, Y.: 'Scan-chain design and optimization for 3D ICs', *ACM J. Emerg. Technol. Comput. Syst.*, 2009, **5**, (9)
15 Wu, X., Chen, Y., Chakrabarty, K., Xie, Y.: 'Test-access mechanism optimisation for core-based three-dimensional SOCs'. Proc. ICCD, 2008
16 Marinissen, E.J., Goel, S.K., Lousberg, M.: 'Wrapper design for embedded core test'. Proc. Int. Test Conf., 2000, pp. 911–920
17 Huang, Y., *et al.*: 'Optimal core wrapper width selection and SOC test scheduling based on 3-D bin packing algorithm'. Proc. Int. Test Conf., 2002, pp. 74–82
18 Goel, S.K., Marinissen, E.J.: 'SOC test architecture design for efficient utilization of test bandwidth', *ACM Trans. Des. Autom. Electron. Syst.*, 2003, **8**, (4), pp. 399–429
19 Xu, Q., Nicolici, N.: 'Resource-constrained system-on-a-chip test: a survey'. Proc. Computers and Digital Techniques, 2005, vol. 152, pp. 67–81
20 Jiang, L., Huang, L., Xu, Q.: 'Test architecture design and optimisation for three-dimensional SoCs'. Proc. DATE, 2009, pp. 220–225
21 Williams, H.P.: 'Model building in mathematical programming' (John Wiley, 1985, 2nd edn.)
22 Bertsimas, D., Tsitsiklis, J.: 'Introduction to linear optimisation' (Athena Scientific, 1997)
23 FICO. Xpress-MP: Available at http://www.fico.com/en/Products/DMTools/Pages/FICO-Xpress-Optimization-Suite.aspx, accessed November 2009