System-level Design Space Exploration for Three-Dimensional (3D) SoCs

Qiaosha Zou, Yibo Chen, Yuan Xie*
Computer Science and Engineering
The Pennsylvania State University
University Park, PA 16802
{qszou, yxc236, yuanxie}@cse.psu.edu

Alan Su
Synopsys Inc.
Hsinchu, Taiwan
Alan.Su@synopsys.com

ABSTRACT

Three-dimensional (3D) ICs promise to overcome barriers in integration density and interconnect scaling by leveraging fast, dense inter-die vias, thereby offering benefits of improved performance, higher memory bandwidth, smaller form factors, and heterogeneous integration. 3D integration provides additional architectural and technology-related design options for future system-on-chip (SoC) designs, making the early design space exploration more critical. This paper proposes a system-level design partition and hardware/software co-synthesis framework for 3D SoC integration. The proposed methodology can be used to explore the enlarged design space and to find out the optimal design choices for given design constraints including form factor, performance, power, or yield.

Categories and Subject Descriptors

B.7.2 [**Integrated Circuits**]: Design Aids

General Terms

Design

1. INTRODUCTION

To further improve integration density and to tackle the interconnect challenge as technology continues scaling, researchers have been pushing forward three-dimensional (3D) IC stacking [5, 13]. In a 3D IC, multiple device layers are stacked together with direct vertical interconnects through substrates. 3D ICs offer a number of advantages over traditional two-dimensional (2D) designs, such as higher packing density, smaller footprint, and shorter global interconnects. Consequently, 3D IC designs have drawn a lot of attentions from both academia and industry in recent years.

We have seen a recent trend of moving design abstraction to a higher level in order to handle the increasing design complexity, with an emphasis on **Electronic System Level (ESL)** design methodologies. Electronic System Level is an established approach built upon high-level abstracted languages such as C/C++, and is now being used increasingly in System-on-Chip (SoC) design. From its genesis as an algorithm modeling methodology with "no links to implementation", ESL is evolving into a set of complementary methodologies that enable embedded system design, verification, and debugging through to the hardware and software implementation of custom SoC systems [2].

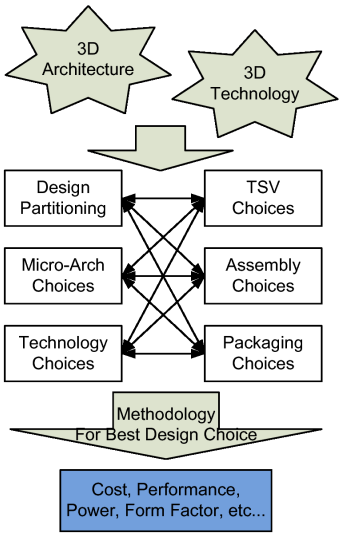

Figure 1: Design space exploration in both technology and architecture options for 3D SoCs.

In conventional system-level exploration, designers consider trade-off in the way hardware and software components of a system work together to exhibit a specified behavior, given a set of performance goals and technology options. In the scenario of 3D SoC integration, the stacking strategies and 3D-related technology options will further complicate the design space exploration, as shown in Fig. 1. A system-level design space exploration methodology that helps make the decisions at the early stage of 3D SoC design is therefore of great importance.

This paper describes a methodology that explores the system-level design space of 3D SoCs and finds out the design options leading to minimal implementation cost under given design constraints. The paper is organized as follows. A system-level synthesis framework incorporating task

*Zou, Chen and Xie's work was supported in part by NSF 0903432, 1017277, 1017391, and SRC grants.

CODES+ISSS'11, October 9–14, 2011, Taipei, Taiwan.

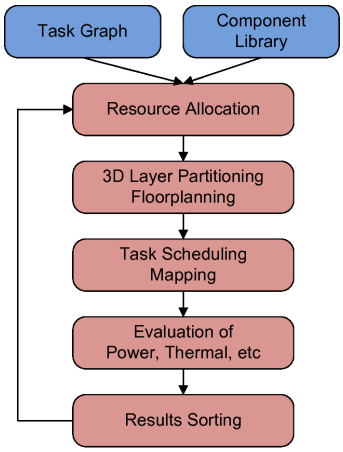

Figure 2: System-level Exploration Framework for 3D SoCs.

scheduling, resource allocation, layer assignment and performance evaluation is proposed in Section 2 to achieve the goal. In Section 3 experiment analysis and case studies show that the proposed methodology is able to produce the optimal design choices for the given 3D SoCs design instances.

2. SYSTEM-LEVEL SYNTHESIS FRAMEWORK FOR 3D ICS

This section introduces the basic flow of the proposed system-level synthesis framework. Resource allocation, task scheduling and layer assignment methodologies are presented later in this section.

2.1 Architecture Synthesis Framework

Fig. 2 shows the design flow of our proposed architecture synthesis framework. Similar to 2D synthesis flow, the synthesis tool takes a task graph and a component library as inputs, and outputs the optimal 3D architecture suggestions. A task graph is a directed graph with edges oriented from source task cell to destination task cell showing data flow direction. A component library contains corresponding hardware and software implementations for each cell.

The resource allocation step assigns suitable component to each task based on functionality requirement. After resource allocation, chip area, power consumption and basic resource cost can be evaluated based on selected components. According to the functionality and estimated area of components, 3D partitioning is deployed to determine the optimal partitioning of system components of the 3D IC. While different partitioning methods lead to different system cost and interconnect delay, 3D floorplanning helps decide the layer assignment of each component block and generates chip layout of each layer. Combining 3D partitioning and floorplanning in the synthesis flow can help optimize chip area and mitigate heat dissipation problem which is introduced by chip stacking. Accurate interconnect information that is obtained from the partitiong/floorplanning stage, is fed to the next stage for fine-grained task scheduling. Finally, the generated synthesis results are presented in a sorted order, based on the cost function that is given according to designer's preference.

2.2 Resource Allocation

Some cells in the task graph can be implemented either by hardware or software. As the number of tasks implemented by software increases, the average cost is reduced. However, in terms of performance, software implementations are usually inferior to their hardware counterparts. On the other hand, hardware implementations enlarge chip area and increase power consumption. Moreover, heat dissipation problem in 3D ICs becomes severe as more hardwares are used. With a cost function that integrates all these aspects, the proposed resource allocation algorithm balances the usage between hardware and software and achieves minimal costs for the hardware/software co-synthesis.

In this work, a genetic algorithm is used as the optimization method for resource allocation. The advantage of genetic algorithm is that it can provide several optimized solutions in a short time by removing solutions with high cost and leaving other solutions to generation evolution[9]. At the beginning of the algorithm, a set of architectures with required population size is generated by a random allocation strategy.

Fig. 3 shows the outline of the proposed random resource allocation strategy. The allocation begins with searching along task edges from the source cell. For each task cell, the algorithm tries to assign it to the existing allocated components. If none of the allocated components is compatible with the current task in terms of both functionality and timing, a new component is allocated, by randomly picking up a compatible component from the component library. The corresponding timing information is then extracted for use in later resource allocation and also task scheduling. The allocation strategy can reduce task stalling time and lower system cost by increasing each component's utilization.

```
ResourceAllocation(taskgraph)
    ▷ initialization
 1  source cell start time = 0;
    ▷ main loop for resource allocation
 2  while (!end of task graph)
 3        do search component list;
 4           if (comp function == cell function &&
             comp available time < cell start time)
 5              then obtain comp pointer;
 6              else  random select comp from comp lib;
 7                    obtain comp pointer;
 8           insert cell to comp task list;
 9           task finish time = start time + comp delay;
10           comp available time = task finish time;
11           next cell start time =
             task finish time + connection delay;
```

Figure 3: Outline of the Random Resource Allocation Strategy

The randomly allocation is performed in multiple runs to generate an initial set of architectures. After this, the evolution process begins. All the generated architectural solutions are sorted based on the cost that is calculated from the cost function. Among all the architecture solutions, the first one third of solutions are treated as good chromosomes

and reserved for the next generation. The second part of architectures are selected to participate in evolution. The last part of solutions are discarded and new architectures are generated to replace them. For the architectures that are selected for evolution, they need to go through mutation task by task.

As chromosomes with high-quality genes and DNAs are kept during generation evolution, system cost of each population gradually falls into a considerable small range. This leads to the convergence condition of the algorithm. Cost standard deviation is computed every time after evolution, and if the standard deviation falls below a user-set threshold, the convergence condition is met and the evolution process terminates. In case the standard deviation can't converge, the algorithm terminates after a certain number of iterations.

2.3 Layer Assignment

Layer assignment is closely related to the stacking strategy that is decided by 3D partitioning. Designers can choose their preferred stacking strategy, or partitioning granularity of components in this step. Issues and concerns such as balanced I/O pins, optimal TSV numbers can also be tackled during 3D partitioning[10]. After partitioning, 3D floorplanning takes the layer assignments of each component as input, and generates virtual layout of each layer. Various design issues, such as interconnect routing, power/ground network[7] and thermal dissipation[4] are taken into account in 3D floorplanning. The partitioning and floorplanning step has significant impact on the cost of 3D ICs. For example, if designers choose a logic-to-memory stacking strategy, then during resource allocation, once a software implementation is allocated, the memory space that is allocated to this software is located in a different layer. Memory operation delay that was originally determined by memory bandwidth, is now constrained by TSV bandwidth between layers. In contrast, if logic-to-logic stacking is chosen, hardware implementations with higher memory bandwidth requirements are put closer to on-chip memory in the same layer.

2.4 Task Scheduling

After the 3D partitioning and floorplanning of each components, the task finishing time is validated through scheduling to see if the task can meet the timing constraint. Task cell execution time and communication delay together contribute to final task completion time. During resource allocation, the execution time of each task cell is evaluated. Communication delay including bus delay, port delay and TSVs delay (if two cells are in different layers) are computed during scheduling after all the routing and bandwidth information are available. The number of TSVs used in design has impact on bandwidth between layers and thus interconnect delay potentially reduces with more TSVs are integrated. An ASAP(As-Soon-As-Possible) scheduling strategy is applied to get the total completion time of this task. If the synthesized architecture can satisfy the timing constraint, the solution is marked as feasible and accepted by the synthesis tool. The total latency of this task is also a factor that influences solutions sorting.

2.5 Cost Function

In architecture synthesis, the cost function is a key metric to sift final architecture solutions. Different design requirements lead to different cost functions. It is impractical to find a universal cost function that satisfy every aspect of consideration. Based on the rationale above, we use a flexible cost function formulation. The cost terms and weights for each term are determined by designers. Equation 1 shows the general form of the cost function.

$$Cost = (\omega_1 X_1 + \omega_2 X_2 + \omega_3 X_3 \ldots + \omega_n X_n) + Cost_{3Dstacking} \quad (1)$$

Terms in parentheses of Equation 1 is decided by user. ω_n is the weight of each design factor. X_n represents the factor that needs to be considered during design, such as area constraint, fabrication cost, power consumption, etc. The last term in this equation stands for system cost of 3D IC stacking. In our work, we mainly consider wafer-to-wafer bonding for 3D integration and TSVs for inter-layer communications. The cost of chip that is built using wafer-to-wafer bonding is given in equation 2 [6]. The goal of this synthesis tool is to minimize total cost defined by Equation **??** given by user. By choosing suitable factors and adjusting their associated weights, designers can easily create cost function that best describes their design requirements.

$$C_{w2w} = \frac{\sum_{i=1}^{N} C_{die_i} + (N-1)\, C_{bonding}}{\left(\prod_{i=1}^{N} Y_{die_i} Y_{bonding}^{N-1}\right)} \quad (2)$$

Table 1: Case Study Specification

Library size	23	
Task size	43	
TSV size	$50\mu m^2$	
Base Cost	small	0.4
	medium	0.8
	large	1.67
	CPU	32.29
	Memory	32.29
3D Cost	die yield	0.95
	bonding yield	0.95
	bonding cost	5

3. ANALYSIS AND CASE STUDY OF 3D ESL EXPLORATION

Based on the synthesis flow presented above, we take a GSM edge algorithm for base station as the target application and analyze the performance of our proposed 3D ESL tool. The target application contains complex data process steps including data ciphering and deciphering, data encoding and decoding, data formatting, etc. Most of the task cells can be implemented by either software or dedicated hardware. Table 1 lists specification of this case study, including the component library size, the component cost assumptions, and 3D stacking cost assumptions. Component library is classified by component functionality with 23 component classes. Among these classes, excluding CPU and memory, 6 functions can only be implemented by software and 4 only by hardware. The basic cost of components is calculated from the silicon area of components and divided into four levels. 3D system cost using wafer to wafer bonding is given with the assumption that die yield and bonding yield are 0.95, bonding cost is 5.

Table 2: Architecture Synthesis Results

Case	System Cost	Chip Area	Finish Time	HW/SW ratio
2D	60.30	60266	4201733	12/20
	62.59	67266	4299533	15/17
	62.77	68766	4201733	14/18
3D 100 TSVs	61.46	65000	4195733	12/20
	63.75	72000	4291533	15/17
	63.94	73500	4195733	14/18
3D 1000 TSVs	72.71	110000	4193633	12/20
	74.99	117000	4288733	15/17
	75.18	118500	4193633	14/18

In total 43 task cells need to be allocated for this application. The suggested solution given by our 3D ESL tool shows that 32 components are allocated, including one on-chip memory component and one DSP. A two layer logic-to-memory stacking architecture with single bus is used in this application. All the hardware implementations are put in the bottom layer, while the on-chip memory is put in the second layer. Since the target application is memory-intensive, memory bandwidth is the bottleneck. Logic-to-memory stacking can then increase available memory bandwidth by TSVs connections. In this application, system cost (including both 3D integration cost and component cost), application finish time and chip area are taken into consideration. The weight for system cost is 0.5. Chip area and application delay are equally weighted. Since the system cost is the main considered factor, the optimal design can be built with 12 components implemented in hardware and 20 in software. We compared the results with 2D implementation, 3D implementation with 100 TSVs and 3D with 1000 TSVs. Table 2 shows the result of 3 suggested architecture designs from each case. The final system cost, estimated chip area and completion time for each architecture are listed in Columns 2-4, respectively.

Since the on-chip memory size from given component library is relatively small, building 3D chips doesn't have much area benefits. The chip area even increases when TSVs number is large. It can be seen that these three cases have almost the same suggestions when we put more weight on the resource cost. As we can see from Table 2, compared with 2D implementation, 3D implementation can gain better performance with less chip area. Performance has been further improved with larger number of TSVs, but the area increases. These results still stand when we consider chip area as the main design metric, since chip area is contributed by hardware implementation. If we consider application finish time as the main design metric. The optimal design can be obtained by using 1000 TSVs 3D implementation. The hardware/software ratio is 19/13, with system cost 98.06. However, if 100 TSVs are used, the cost is decreased by 11.46% with very small delay increment.

4. RELATED WORK

Recently we have seen a recent trend of moving design abstraction to a higher level, with an emphasis on *Electronic System Level (ESL)* design methodologies. Ariki et al. [1] proposed a model-based SoC design flow using ESL environment. Su et al. [12] and Schafer et al. [11] presented case studies of ESL design methodologies on GSM edge algorithm and complex image processing systems, respectively. Nevertheless, the major research on ESL is targeting at conventional 2D SoC architecture.

Cost analysis for 3D ICs have been addressed in several existing literatures. Mercier et al [8] first looked at the yield modeling of 3D IC stacking regarding stacking yield loss. Dong et al [6] proposed system-level cost analysis and design exploration for 3D ICs, by estimating the implementation cost of different stacking options, given the gate count of a design. Chen et al [3] extended the cost analysis of 3D ICs by considering the testing cost of different design choices.

5. CONCLUSION

With the adoption of 3D IC technology, designers have more choices in the design space, which makes it harder to reach optimal design options only by human effort. For design space exploration, a system-level 3D SoC design and hardware/software co-synthesis framework is proposed in this paper. The proposed framework aims at providing designers optimal architectures in a short time with user-defined design goals.

We demonstrated that our approach can generate optimal design choices combining 3D partitioning and floorplanning with task allocation and scheduling in system-level synthesis. Analysis of the influence of 3D integration on the synthesized results is presented. A real world case study using the proposed framework is performed showing the effectiveness of our proposed methodology.

6. REFERENCES

[1] D. Araki, A. Nakamura, and M. Miyama. Model-based SoC design using ESL environment. In *SOCC*, 2010.

[2] B. Bailey, G. Martin, and A. Piziali. *ESL Design and Verification: A Prescription for Electronic System Level Methodology*. Morgan Kaufmann/Elsevier, 2007.

[3] Y. Chen, D. Niu, Y. Xie, and K. Chakrabarty. Cost-effective integration of three-dimensional (3D) ICs emphasizing testing cost analysis. In *ICCAD*, pages 471–476, 2010.

[4] J. Cong, J. Wei, and Y. Zhang. A thermal-driven floorplanning algorithm for 3D ICs. In *ICCAD*, 2004.

[5] W. R. Davis, J. Wilson, and et al. Demystifying 3D ICs: the pros and cons of going vertical. *IEEE Design & Test of Computers*, 22(6):498–510, 2005.

[6] X. Dong and Y. Xie. System-level cost analysis and design exploration for three-dimensional integrated circuits (3D ICs). In *ASP-DAC*, 2009.

[7] P. Falkenstern, Y. Xie, Y.-W. Chang, and Y. Wang. Three-dimensional integrated circuits (3D IC) floorplan and power/ground network co-synthesis. In *ASP-DAC*, 2010.

[8] P. Mercier, S. R. Singh, K. Iniewski, B. Moore, and P. O'Shea. Yield and cost modeling for 3D chip stack technologies. In *CICC*, 2006.

[9] K. Ohmori. High-level synthesis using genetic algorithm. In *IEEE Intl. Conf. on Evolutionary Computation*, 1995.

[10] S. Sawicki, G. Wilke, M. Johann, and R. Reis. A cells and I/O pins partitioning refinement algorithm for 3D VLSI circuits. In *ICECS*, 2009.

[11] B. C. Schafer, A. Trambadia, and K. Wakabayashi. Design of complex image processing systems in ESL. In *ASP-DAC*, 2010.

[12] A. P. Su. Application of ESL synthesis on GSM edge algorithm for base station. In *ASP-DAC*, 2010.

[13] Y. Xie, G. H. Loh, and et al. Design space exploration for 3D architectures. *J. Emerg. Technol. Comput. Syst.*, 2(2):65–103, 2006.