ABSTRACT

HU, JIANCHEN. System-level Pathfinding Flow for Three Dimensional Integrated Circuits. (Under the direction of Dr. W. Rhett Davis).

The limited performance improvement of transistors in ultra-deep-submicron technologies is making it more difficult to achieve computing performance increases from scaling alone. Solutions to address this challenge come from combined system-level and physical-level optimizations. At the system-level, more complex on-chip systems are implemented by integrating various processing elements with interconnections to increase parallel processing capability without the need to push transistors to high operating frequencies or advanced technologies. At the physical-level, three dimensional integrated circuits (3D-ICs) increase chip density and reduce wire delay by stacking multiple ICs vertically with through-silicon-vias (TSVs). Both techniques enlarge the system design space by introducing more design parameters such as the number of processing elements and types, interconnection topologies, IC stacking schemes, and heterogeneous technologies. Furthermore, the design parameters at different levels of abstraction interact with each other, which necessitates a system-level “pathfinding” design flow to evaluate the design parameters fast at early design stage. However, most state-of-the-art pathfinding flows focus on the register-transfer and gate levels of abstraction for system modeling and are hard to integrate with tools that focus on the transaction and instruction levels. In this work, we present an electronic system-level (ESL) pathfinding flow that integrates transaction-level and physical-level evaluation by using ESL models and interfaces, allowing fast, physically aware system design evaluation.
System-level Pathfinding Flow for Three Dimensional Integrated Circuit

by
Jianchen Hu

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Computer Engineering

Raleigh, North Carolina
2014

APPROVED BY:

_______________________________  ______________________________
Dr. W. Rhett Davis  Dr. Eric Rotenberg
Committee Chair

_______________________________  ______________________________
Dr. Greg T. Byrd  Dr. Gregory Reeves
DEDICATION

To my parents and grandma
BIOGRAPHY

Jianchen Hu was born on 4th Sep. 1984 in Beijing, China. He received his Bachelor’s degree in Electrical Engineering in Tsinghua University in 2007. In fall 2007, he began his graduate studies in the Electrical and Computer Engineering Department in North Carolina State University, Raleigh, NC. From June 2008, he has been working in the Methodologies for User-friendly System-on-a-chip Experimentation (MUSE) group of Prof. Rhett Davis in the field of network-on-chips in multiprocessor system issue.

From fall 2009, he works as a Ph.D student with Dr. Rhett Davis. His research interests include transaction-level modeling, computer architectures and 3D specific EDA tools design.
ACKNOWLEDGMENTS

First I would like to express my love to my parents who support me through my studies.

I would also like to thank my advisor Dr. Rhett Davis for the great Ph.D research experience offered. He always guides me for my research work and supervises me with his knowledge, cutting-edge idea and enthusiasm of research.

I would like to thank Dr. Eric Rotenberg, Dr. Greg Byrd and Dr. Gregory Reeves who serve me as my committee members.

Special thanks to Shivam Priyadarshi who work with me and help me a lot with my research work in detail.

I would like to thank all my colleagues and friends at North Carolina State University: Yi Lou, Christopher Mineo, Zhuo Yan, Zhenqian Zhang, Wenxu Zhao, Harun Demircioglu, for all the informative and constructive discussion and good time together.
TABLE OF CONTENTS

LIST OF TABLES ........................................................................................................ vii
LIST OF FIGURES ...................................................................................................... viii

Chapter 1 Introduction ............................................................................................... 1
  1.1 Motivation ........................................................................................................... 1
  1.2 Contribution ....................................................................................................... 5
  1.3 Organization ...................................................................................................... 6

Chapter 2 Related Works ......................................................................................... 8
  2.1 Introduction ....................................................................................................... 8
  2.2 Network-on-Chip Introduction and Design Space ............................................. 10
  2.3 Transaction-level Modeling .............................................................................. 15
  2.4 System-level CAD flow ..................................................................................... 18
    2.4.1 RTL-centric flow ......................................................................................... 18
    2.4.2 TLM-centric flow ....................................................................................... 20
  2.5 Thermal Analysis tools ..................................................................................... 24
  2.6 Summary .......................................................................................................... 24

Chapter 3 System-level Pathfinding Flow ............................................................... 26
  3.1 Introduction ....................................................................................................... 26
  3.2 Pathfinding Flow Descriptions ........................................................................ 28
    3.2.1 Overview ................................................................................................... 28
    3.2.2 Front-end Part Description ........................................................................ 31
    3.2.3 Back-end Description ................................................................................. 32
  3.3 Transaction-level Simulation with GEMS framework ..................................... 40
    3.3.1 Event-driven Simulation ............................................................................ 40
    3.3.2 Adding Interconnect Model ....................................................................... 42
  3.4 Summary .......................................................................................................... 44

Chapter 4 High-level Power Model and Characterization .................................... 46
  4.1 Introduction ....................................................................................................... 46
  4.2 Current High-level Power Model ..................................................................... 48
    4.2.1 CACTI, the cache & memory model ......................................................... 48
    4.2.2 ORION: On-chip router power model ...................................................... 50
    4.2.3 McPat: Processor power model .................................................................. 50
  4.3 Power Model Characterization Flow ............................................................... 51
  4.4 Power Model Characterization for crossbar used in on-chip router ................ 54
    4.4.1 Crossbar design description ....................................................................... 55
    4.4.2 testbench generation and characterization ................................................. 57
    4.4.3 Result Analysis and Comparison with Orion 2.0 ...................................... 61
  4.5 Summary .......................................................................................................... 66

Chapter 5 Physical-level Analysis .......................................................................... 67
  5.1 Introduction ....................................................................................................... 67
  5.2 ESL Floorplan based physical analysis framework ......................................... 70
  5.3 Global routing analysis .................................................................................... 72
5.3.1 Global routing introduction............................................................... 72
5.3.2 Graph Model and Routing Algorithms................................................. 73
5.3.3 Adjusted Hadlock Algorithm for 3D global routing............................... 76
5.3.4 Power estimation for global wires ....................................................... 78
5.4 Summary ............................................................................................. 78
Chapter 6 Case Study ................................................................................. 80
6.1 Introduction ........................................................................................ 80
6.2 Multi-core case .................................................................................... 81
   6.2.1 System Overview ........................................................................ 81
   6.2.2 Core Model Integration and Multiprogram Support ....................... 83
   6.2.3 Core and Cache Power Estimation .............................................. 86
   6.2.4 Transaction-level simulation result ............................................. 87
   6.2.5 Physical-level simulation result .................................................... 89
6.3 NoC test case ....................................................................................... 94
   6.3.1 System Overview ........................................................................ 95
   6.3.2 Core and Interconnect Power Estimation ..................................... 100
   6.3.3 Transaction-level simulation result & analysis .............................. 101
   6.3.4 Physical-level simulation result & analysis .................................. 105
6.4 Summary ............................................................................................. 111
Chapter 7 Conclusion ............................................................................... 113
REFERENCES ........................................................................................ 115
APPENDICES ............................................................................................ 120
Appendix A Pathfinder3D ........................................................................ 121
   A.1 Design File Format ........................................................................ 121
   A.1.2 Stimulus File Format ............................................................... 123
   A.1.3 Interface File Format ............................................................... 123
LIST OF TABLES

Table 4.1 High-performance and low-power configurations for caches in CACTI 6.5 .................49
Table 4.2 Design and characterization parameters for crossbar power characterization...............61
Table 6.1 Core power consumption for each stage ........................................................................86
Table 6.2 Cache model parameters for power and area estimation in CACTI .............................87
Table 6.3 L2 miss rate of quad-core system running MCF and BZIP benchmark with different L2 size .................................................................88
Table 6.4 2D and 3D temperature and global wire analysis result for quad-core case with (a) 256KB and (b) 1024KB L2 bank .................................................................................................92
Table 6.5 Power and area estimation for elements used in NoC case ........................................101
Table 6.6 2D and 3D temperature and global wire result for Mesh and Ring topologies ..............108
LIST OF FIGURES

Figure 2.1 Circuit-switching Scheme ................................................................. 11
Figure 2.2 Two types of topologies for NoCs. (a) Mesh and (b) Flattened butterfly ............... 13
Figure 2.3 NoC with Ring topology ..................................................................... 14
Figure 2.4 Pathfinding flow presented in Rako’s work .................................................. 17
Figure 2.5 Physical-analysis flow used in SpyGlass3D .................................................. 19
Figure 2.6 TLM-2.0 Architecture .......................................................................... 22
Figure 3.1 System-level Evaluation Flow for 3D-IC ....................................................... 30
Figure 3.2 Cross section of the first three tiers of the FreePDK3D45 technology .................. 34
Figure 3.3 Block communication with event-driven simulator ......................................... 42
Figure 3.4 Example to add interconnections to GEMS framework ................................. 44
Figure 4.1 High-level Power Model Characterization Flow ............................................ 53
Figure 4.2 Schematic of Logic Gate Based Crossbar .................................................... 57
Figure 4.3 Test Bench Generation Flow ..................................................................... 60
Figure 4.4 Characterization result for 7-port crossbar with 3 different data input pattern ........ 62
Figure 4.5 Characterization result for 7-port crossbar with time and port dependent request pattern . 63
Figure 4.6 Power model format for 7-port 16-bit corssbar ............................................. 64
Figure 4.7 Power estimation of crossbar from Orion 2.0 ............................................... 65
Figure 5.1 Physical-level analysis scope .................................................................... 69
Figure 5.2 Pathfinder3D: ESL floorplan-based analysis engine ........................................ 71
Figure 5.3 Global routing and detailed routing scheme ................................................. 73
Figure 5.4 Different graph model. (a) Actual layout (b) Grid model (c) Checker Board model and (d) Channel intersection model ......................................................... 75
Figure 5.5 Pseudo code for adjusted Hadlock algorithm for global routing in 3D-IC ............ 77
Figure 6.1 Quad-core test case ............................................................................... 82
Figure 6.2 Attaching detailed core model to TLM simulation framework ........................... 84
Figure 6.3 Simple TLB for address conversion ........................................................... 85
Figure 6.4 CPI vs. power for 4-core case running BZIP and MCF benchmarks ................... 89
Figure 6.5 Physical floorplan for 4-core system with 1-tier and 2-tier configuration ............. 90
Figure 6.6 Temperature distribution of 4-core system with 256KB L2 bank ....................... 91
Figure 6.7 Wire length distribution comparison between 2D and 3D floorplan for (a) 256KB L2 bank and (b) 1024KB L2 bank ................................................................. 93
Figure 6.8 Router architecture for packet-switched based network ........................................ 96
Figure 6.9 Parallel transactions in Ring topology ..................................................................... 97
Figure 6.10 (a) 8X8 Mesh topology and (b) 4X4X4 Mesh topology ........................................... 99
Figure 6.11 Pareto-efficiency fro 64-node NoCs ................................................................. 104
Figure 6.12 Power efficiency of a 64-node NoC with different 3D partitioning ...................... 105
Figure 6.13 Temperature distribution for 4X4X4 Mesh topology ........................................... 106
Figure 6.14 Global wire distribution comparison between 8X8 and 4X4X4 Mesh NoCs .......... 107
Figure 6.15 Global wire distribution of 4X4X4 Mesh NoC for 1-tier and 4-tier cases .............. 108
Figure 6.16 Pareto-efficiency for 64-node NoC with temperature limitation ......................... 110
Figure 6.17 Pareto-efficiency for 64-node NoC with global wire length limitation ................. 111
Figure A.1 Design file format of Pathfinder3D ....................................................................... 122
Figure A.2 Stimulus File Format in Pathfinder 3D ................................................................. 123
Figure A.3 Interface file in Pathfinder3D .............................................................................. 124
Chapter 1

Introduction

1.1 Motivation

The limited performance improvement of transistors in ultra-deep-submicron technologies is making it more difficult to achieve computing performance increases from scaling alone [1]. As the size of transistors shrink, the delay and power consumption are not scaled down at a pace of Moore’s Law. This situation is further aggravated by the growing complexity of interconnects. The average length of global wires is determined by chip size and tends to remain fixed as technology scales, but the delay of a unit length wire is rising [2]. Meanwhile, the increasing on-chip system complexity coming from high chip integration with advanced technology nodes raises the manufacturing cost due to lithography and patterning integration [3]. The chip yield and variation in electric characteristics of transistors further increase cost for advanced technologies. Furthermore, the slowdown in supply voltage scaling makes it more difficult to reduce power consumption.

To address these challenges, new techniques at both system-level and process-level are implemented. At system-level, increasing parallelism by integrating multiple processing cores and caches on a single chip improves the system performance without pushing cores into higher operating frequency and advanced process. The chip multiprocessors (CMP)
solution has been accepted as commercial offerings today [4] [5]. Chip multiprocessors also get the benefit of heterogeneity by applying mixed type (fast/slow hybrid) of cores to run different applications [6]. To further increase parallel processing ability and meet the requirement of on-chip communication, network-on-chip (NoC) architectures allow integrating more processing elements (generic cores, DSP processors or other dedicate function units) on a single chip with network interconnections [7]. An NoC is considered a scalable architecture since the interconnections with different topologies distribute on-chip traffic contentions and congestions compared to shared-bus-based interconnections. Moreover, the regularity of NoC structure makes it easy to extend the number of on-chip cores by repeating the same unit architecture [8].

At the manufacturing process-level, three dimensional integrated circuits (3DIC) address some of these challenges by stacking different ICs vertically [9], [10]. Circuits in different tiers of a 3DIC can communicate with each other through different types of Through Silicon Vias (TSVs), which can largely reduce the total wire length and routing congestion compared to a conventional 2D implementation [11], [12]. The reduction of global wire length results in power and delay reduction of interconnections. Another benefit from 3D integration is high memory bandwidth by building on-chip memories and connecting with cores by using TSVs. In that sense the conventional off-chip latency between cores and memories is converted to on-chip latency between tiers, and wide core-to-memory interfaces can be implemented within the TSV capacity [13]. Since tiers can be manufactured independently in different technologies, it gives flexibility to optimize the design by evaluating different technologies and stacking schemes [14]. Moreover, as a result of shrinking die area, global
wire lengths can be reduced, leading to cost reductions from reduced metallization layers, and heterogeneous integration can reduce the cost by implementing non-critical components in cheap technologies [15].

Both CMP/NoC architecture and 3D integration provide a rich variety of choices to system designers at both the architecture-level (core types, on-chip cache size, network topologies) and the physical-level (homogenous vs. heterogeneous partitioning of a design, number of stacking layers in 3D design, integration of heterogeneous technologies, and different types of 3D bonding methods). In [16], an NoC architecture is applied with 3D integration to get both benefits from architectures and 3D integration. However, the large system design space necessitates a huge amount of effort to model the system and identify its critical parameters. In such cases, it is necessary to develop system-level computer-aided design (CAD) flows that help designers to evaluate various designs and make the best design decisions.

The CAD flows for system-level design and evaluation can be divided into two categories: register transfer-level (RTL) centric and electronic system-level (ESL) centric flow. The RTL centric flow evaluates designs based on detailed gate netlists, generating accurate estimates of power, area and other metrics. It is usually used by designers to verify and optimize a design at the circuit implementation phase. The detailed netlist requirement does not help much for making architectural decisions. On the other side, the ESL centric flow manages design complexity by modeling and simulating the target system at a higher level of abstraction, reducing modeling effort and increasing evaluation speed. However, an ESL centric flow is usually lacking physically aware features, which complicate 3D-IC
designs since both benefits (wire reduction, heterogeneous technologies, etc.) and costs (temperature rise, TSV impact) are tightly dependent on physical-level parameters. Fast evaluation of the impact of physical decisions on ESL behavior is required to effectively explore the rapidly expanding design space. The term “pathfinding” is used for this kind of approach. The pathfinding flow integrates system-level behavior and physical-level analysis while providing useful feedback for computer architects and valuable input for circuit designers, targeting different communities with the intent to close the gap between high-level design evaluation and circuit-level design implementation. In this work, we present a pathfinding flow that integrates TLM simulator and ESL physical analysis tools to allow fast system designs evaluation with the impact of physical-level metrics such as temperature and global wires.

A necessary part of any pathfinding flow is the power model. Accurate power estimation is critical for ESL performance evaluation. Furthermore, ESL power-estimates are tightly coupled to physical-level temperature analysis, due to the exponential dependence of leakage power on temperature. There are two types of power models: characterized models and analytical models. Analytical models predict block power based on estimated capacitance by using the assumption of certain circuit structures. They are often integrated within ESL centric flow to provide fast power estimation. However, as more physical-level factors are taken into account in analytical power models, the task of fitting model parameters becomes more difficult, and users tend to find their estimates diverging more and more from actual implementations. Characterized models predict power by breaking designs into standard cells, and power data for each standard cell is simulated at circuit level. Such an approach provides
accurate power number but requires RTL design to be able to characterize the power. In this work, we characterize power to obtain a higher level, abstract power model for the ESL simulator, so that the RTL and gate levels can be avoided with minimum reduction in accuracy.

1.2 Contribution

Our main contribution in this work is to develop a pathfinding flow for fast evaluation at system-level with the consideration of physical-level impact in 3D-ICs. The flow integrates transaction-level simulation to capture run-time dependent system behavior and physically aware analysis to estimate temperature and global routing result. The transaction-level simulation framework allows user to build various systems, exploring design space in terms of architectural configurations. An open-source ESL framework named Pathfinder3D [17] is developed for physical-level analysis used in our pathfinding flow. This framework provides convenient method to examine the impact of 3DIC manufacturing options such as stacking schemes and wafer technologies. Furthermore, it sets up an ESL floorplan based prototype which bypasses the requirement of detailed gate netlist and makes it easy to map the high-level system description to physical-level. A set of physical-level tools are built around this prototype for thermal and global routing analysis. By using this pathfinding flow, users can easily evaluate the benefits of 3D technologies without the detailed design, and compare various systems to find best architectures based on the simulation result of TLM simulator and physical-level analysis result.
Another contribution in this work is the power model characterization flow. Analytical model used to be considered an easy-to-use and scalable approach for power estimation. However, as technologies keep advancing, more physical/technology parameters are needed to be considered to perform power analysis which makes analytical power model hard to build and use at system-level. In this work, a characterization flow is presented for power model generation. We incorporate standard cell power estimation flow to get accurate dynamic and static power number by using gate level library but characterize the power model at ESL. It is a fast approach to read transaction-level simulation output and provide power number back feedback to high-level analysis as well as physical-level analysis such as temperature estimation. The one-time building effort to characterization is paid off once the power model is set up and been used by various systems.

1.3 Organization

We organize the work as follows. Chapter 2 reviews various RTL-centric and ESL-centric CAD flow. In chapter 3, the pathfinding flow is presented in detail, describing how we build system at high-level and generate information for physical-level analysis. Meanwhile, the thermal analysis tool that is used in this work to get temperature analysis result is introduced [18]. High-level power model characterization flow is presented in chapter 4, which describes how we use transaction-level parameter to characterize power based on detailed netlist. And we compare our characterized model to analytical model to illustrate the necessity of characterization. In chapter 5, the floorplan based pathfinder3D framework is presented to show how we perform physical analysis without knowing detailed
circuit design. And a global routing tool, it is presented to show how physical analysis can be performed based on ESL floorplan rather than detailed gate-level netlist. Two case studies are shown in chapter 6. The multi-core case shows how users examine 3D benefits for different system configuration without detailed design. And the NoC case illustrates the importance to have physically aware feature pathfinding flow to make architectural design decisions. Chapter 7 concludes the work.
Chapter 2

Related Works

2.1 Introduction

Developing a CAD flow for system-level design exploration and evaluation becomes more and more difficult due to the increasing system complexity and technologies shrinking. Several key issues need to be addressed for system-level CAD flow. First, the CAD flow needs to choose the right design parameters as well as performance metrics for evaluation. Complex system contains hundreds of design parameters and it is impossible to capture all of them before actually implementing the design with limited time and effort invested. Thus picking critical sub-set of input parameters for system evaluation is important to reduce the design effort in the follow-up design phases. Another important feature for the flow is to capture the run-time dependent effect of a system. Power and performance of a system not only depend on the hardware architectures but also applications [19] [20]. Identifying the performance variation due to different applications helps system designers to make design decisions and apply various optimizations [21]. The trade-off between modeling/simulating effort and the accuracy of predicted performance needs to be considered in system-level CAD flow, especially for complex system such as CMP or NoC. In that sense, selecting appropriate level of abstraction for modeling and simulating is important to get accurate
result with fast simulation speed. In 3D-ICs, those challenges become more difficult for system-level researchers and designers to evaluate and optimize designs. 3D integration with multiple tiers and 3D floorplans make it more difficult to find out key design parameters. The technology heterogeneity and different bonding/stacking schemes also need to be evaluated with other system-level parameters. 3D integration also requires the flow to support more dynamic features. For example, the temperature rise in 3D-ICs increases leakage power consumption and leads to higher temperature, in that case a dynamic thermal analysis need to be performed to figure out an appropriate thermal management method [22].

The system-level CAD flow can be divided into two categories: RTL-centric and TLM-centric flow, focusing on different scopes and objectives. The RTL-centric flow targets detailed design optimization. It predicts accurate performance estimation to reflect subtle design changes by incorporating the commercial physical design flow. However, such an approach requires detailed RTL design netlist as an input, which takes too much time to generate for complex system evaluation. On the other hand, TLM-centric flow targets system-level design space exploration for computer architects to compare complex systems with large amount of architectural configurations. Modeling speed and the capability to model large systems are considered to be the most important features in this kind of flow. The divergent objectives make the two types of flow developed in parallel and used by computer architects and circuit designers in their research scopes respectively.

However, the emerging 3DIC technology adds various physical design choices into system designs. Evaluating the impact of 3D integration on various designs at system-level is essential to make design decisions and reduce the design effort. In that case, integrating
physically aware analysis which is usually in RTL-centric flow to TLM-centric flow for fast system-level evaluation is the solution to that problem. In this chapter, we first introduce the basis of TLM simulation, and then the works of different types of CAD flows for system-level evaluation.

2.2 Network-on-Chip Introduction and Design Space

As the number of processing cores increases in multi-core system, the conventional bus-based or dedicated wire-based on-chip interconnections cannot provide enough bandwidth for the traffic among cores. To address the bottleneck of on-chip communication bandwidth, Network-on-Chip (NoC) is implemented by connecting on-chip processing elements with network structures. There are many factors that affect the network performance such as routing and arbitration algorithms, network buffer size, router structures, etc. In this work, we will examine two system-level parameters of NoCs: switching schemes and network topologies.

NoCs can be divided into two categories according to the switching schemes: packet-switched based NoCs and circuit-switched based NoCs. Both of them brings the concept from computer networks. A key feature of packet-switched based networks is that the transactions can be stored and routed at network routers based on its destination information and the routing algorithm. In a multi-hop network, transactions generally are not sent to the destination node directly, but routed at each node independently. In that case, buffers are required in each network node due to possible traffic contentions and congestions. This point-to-point switching scheme distributes the routing into each node, thus transactions
without interactions can be routed and transmitted simultaneously. The parallel transactions feature increases the bandwidth by improving the channel utilization. By implementing the same router structure and routing algorithm for each network node, it’s easy to scale the NoC to large number of nodes [36].

Different from packet-switched NoCs, circuit-switched based NoCs establish dedicated “circuit” for data transfer between source and destination nodes which provide more end-to-end features compared to packet-switched based NoCs. Typically data transfer in circuit-switched networks can be divided into 3 phases: connection request phase, data transfer phase, and connection release phase. Figure 2.1 shows the 3 phases switching scheme.

![Figure 2.1 Circuit-switching Scheme](image)

In connection request phase, the source node sends out a request to the destination via routers, and waits for a grant signal. The request packet can be routed as the packet-switching
scheme. Once the destination node accepts the request, an acknowledgement (ACK) packet would be sent back to the source node. The request and ACK packets reserve a dedicated channel for following data packets so that other transactions cannot use the same channel. The \textit{data transfer} phase begins after the channel has been established. Multiple data packets are sent out through the channel without the need of routing and storing in intermediate nodes. After all data packets are sent out, the \textit{connection release} phase starts with a tail packet sent to the destination. Similar to connection request phase, the tail packet releases the channel so that other transactions can use it. Compared to packet-switching, circuit-switching does not require data buffers for each network node thus reduce area and power of the network. Several implementations make use of these advantages to reduce the network cost \cite{37-39}. However, circuit-switching requires extra phase to establish and release channel which reduces the effective bandwidth by introducing extra latency. And the time to setup and release channel increases as the number of network nodes grows, making the circuit-switched based NoCs hard to scale up.

The network topology represents the arrangement of the network nodes and the links connecting them. We introduce several widely used topologies here for the case study we used later in this work.

The first one is mesh topology. As fig. 2.2 (a) shows, mesh has regular shape and links which make it easy to implement. However, the nodes in mesh topology can only connect to the adjacent nodes which limits the performance. Once the mesh grows larger, nodes in the center of the network need to process more traffic than the edge and corner ones. To solve the traffic contention problem in large scale of NoCs, another topology called flattened
butterfly is introduced [40]. The main difference between mesh and flattened butterfly is nodes in flattened butterfly topology have direct access to all the nodes in the same row and column. As fig. 2.2 (b) shows, each node in a flattened butterfly topology has 6 ports regardless of the locations. Such high-radix nodes greatly improve the bandwidth of NoC since the maximum number of hops required for data transfer reduces from 6 to 2 compared to mesh topology. However, such high-radix routers consume more power and area. And increasing radix can also slow down the operating frequency. Moreover, only on-chip traffic centric applications can utilize the extra possible bandwidth provided by flattened butterfly, otherwise the bandwidth benefit is wasted due to not enough on-chip transactions.

![Figure 2.2 Two types of topologies for NoCs. (a) Mesh and (b) Flattened butterfly topology]
The third topology is a ring topology. Different from mesh and flattened butterfly, it is usually used in circuit-switched based rather than packet-switched based ones. As fig. 2.3 shows, nodes in the ring topology connected with each other through a shared ring bus, and a centralized arbiter controls all the transaction on the ring. Compared to other topologies, ring topology has simple router architecture since no buffer and arbitration unit is needed. Since buffers consume large percentage of router power, the ring topology helps reduce the power as well as area [41]. However, ring topology has the worst scalability. Increasing the size of the ring not only increases average traffic length but reduces channel utilizations.
2.3 Transaction-level Modeling

There are two critical aspects in hardware description and modeling: the notion of timing and the connection between function blocks. Based on different modeling approaches of the two aspects, we have two conventional modeling styles: RTL modeling and functional-level modeling. RTL model specifies all details of port connections between logic blocks. Each pin connection needs to be specified and the type of input/output need to match. In functional-level model, there are usually no ports between blocks and data transfer is implemented by using function calls. Timing in RTL is modeled in detail at cycle count. In contrast, functional-level model does not hold any timing information.

Transaction-level is a level of abstraction in between the RTL and functional-level. Transaction-level modeling is an approach to model digital systems where details of communication among modules are separated from the details of the implementation of the functional units or the communication architecture [23]. As such, TLMs usually do not include pin-accurate detail like RTL models, but aggregate many input/output signals into ports and channels. Communications among channels are implemented by calling functions. Such approach avoids implementing detailed interfaces between block and simplifies data transfer. TLMs also support two modeling styles in terms of timing annotation method: loosely-timed and approximately-timed. Loosely-timed modeling style allows few timing points compared to approximately-timed and cycle-accurate model. The latency of a transaction is predefined at the beginning of the transaction. And the functions in modules can be executed ahead of simulation time, which means the module behavior does not
depend on simulation time strictly. Approximately-timed modeling provides more timing points in a transaction. It breaks a transaction into several phases. The module could annotate delay in each phase, which was implemented by using timeout or timed event notification. Approximately-timed model supports pipelined structure implicitly.

The timing feature and simplified port declaration allows TLM to support hardware modeling and simulation by using software-based environment such as C++ or SystemC. Simulation environment and flexible timing annotation methods make TLMs can be set up quickly and run much faster than RTL models [24].
Figure 2.4 Pathfinding flow presented in Rako’s work [25]
2.4 system-level CAD flow

2.4.1 RTL-centric flow

One RTL-centric pathfinding flow is presented in [25] as figure 2.4 shows. In that flow, first different component models (typically adders, multipliers, muxes of different sizes and datapath widths) are synthesized and performance parameters are extracted. Meanwhile, a high-level C++ based system description is generated. Then a High-level Synthesis (HLS) process translates the system description into completed RTL by using the performance parameters. After that, the RTL netlist is synthesized to gate-level netlist via standard physical design flow. In the end, a physical prototype is generated by using gate-level netlist. The prototype is simplified from completed layout, but still keeps place and route information as well as parasitic capacitances to estimate power, delay and area of the design. The result feeds back to HLS phase and it will regenerate RTL netlist to optimize timing. Such flow links physical-level analysis to system-level from two aspects. 1) The performance extraction phase enables “technology aware netlist generation”. 2) The feedback from physical prototype to HLS allows designers to optimize architectures based on physical-level analysis result. However, the HLS process is highly restricted and the example shown in that work only makes small part of their design in C. And the design parameters and performance metrics are limited. The optimization focuses on operating frequency, area and power consumption which are all “circuit-level” parameters, the run-time/simulation based parameters are not examined. The detailed netlist requirement and lacking of architectural-
level parameters analysis prohibits the flow from being used by system architects. Furthermore, it does not provide enough support for various 3D integration options.

Figure 2.5 Physical-analysis flow used in SpyGlass3D [26]
The SpyGlass3D is another RTL-centric flow focusing on physical-level analysis of 3DIC as figure 2.5 shows [26]. This flow provides physical-level performance estimation and optimization by extending standard 3D-IC design flow. The analysis starts with a 2D RTL netlist and get initial floorplan by using typical 2D physical design flow. Then the blocks which will move to another tier are marked as point objects, and floorplans in different tiers are generated based on point objects. To specify the pad and TSV locations, a pad clustering process is used to aggregate TSVs. Then another floorplan with specification of internal and external pad information is generated. The physical-level analysis focuses on different partitioning schemes and wire length optimization. It shows that good partition for 3DIC can reduce the total wire length by 50% to get better power saving and cut off the maximum length by 30% for better performance. As an evaluation flow, the work requires detailed physical design netlist which makes it hard to use for system-level designers since it is hard to integrate with other system-level tools due to the detail requirement.

2.4.2 TLM-centric flow

Different from RTL-centric flow, TLM-centric flow focuses on modeling complex system at architectural-level. A toolset for multi-core system modeling and simulation named GEMS is presented in [27]. It provides a framework to simulate multi-core with memory hierarchies and coherent protocols. By including a texture description of cache protocols, users can easily build multi-core systems with different cache structures to compare the system performance. There are two major contributions that make GEMS popular. First, it maintains an event-driven kernel that supports the parallel nature of hardware. The system-
level configuration such as number of cores, caches, cache hierarchy levels and cache-to-core connection configurations can be modified without coding work but only texture specification. That allows computer architects to explore large design space fast and easily.

Second, GEMS provides good timing estimation across the system, the latency of each transaction can be simulated in detail due to the time scheduler. We consider GEMS as a transaction-level simulator according to these important features above. Since GEMS focuses on cache hierarchies only, a CPU simulator called M5 is integrated which provides features that many CPU-centric simulators lack, such as full-system (including OS) simulation and I/O support [28]. The combined simulator GEM5 enlarges the simulator scope [29]. Setting up and simulating a system in GEM5 is nothing more than writing a python script that instantiates and connects various components with each other. Some other tools targeting peripheries in CMP scope are integrated into GEMS-GEM5 framework. For example, a cycle-accurate memory simulator is integrated to provide accurate memory latency [35]. One of the GEMS-GEM5 framework limitations is the framework targets CPU-centric system only such as multi-core or many-core with various cache hierarchies, but not for generic SoCs which also require system-level simulation for design space exploration. It is easy to add new cores and caches, but the core-cache backbone is hardly changeable. And there is no built-in support for physical-level analysis such as area and temperature.

SystemC is another TLM framework [30]. It was created largely in response to the need for TLM. Based on C++, SystemC provides a series of classes and methods to help build TLMs. The objective of SystemC is to set up a platform that allows users to build their own systems fast at transaction-level. The basic layer of SystemC provides an event-driven
simulation kernel. This kernel works with events and processes in an abstract manner, coordinating events and switching between processes, thereby allowing SystemC to simulate the implicitly parallel hardware features. Modules and processes describe the abstraction of structural information, while interfaces and channels represent the abstraction for communications. Data is transferred between modules through interfaces and channels. Since SystemC is built on top of C++, all the C++ features can be implemented to speed up modeling and increase code reusability.
To further help establishing a transaction-level model based system, a standard for transaction-level modeling with SystemC is presented in TLM-2.0 standard [31]. Fig 2.6 shows the architecture of TLM-2.0. There are several key contributions in TLM-2.0 to help standardize transaction-level modeling. First one is the standard transaction class *generic_payload*. As the name implies, this transaction class defines a data structure for communications between function blocks. It contains basic information for data transfer such as command type, address, ID, data pointer as well as an extension class to specify custom attributes. TLM-2.0 also defines sockets which are not only aggregations of signals and ports but also interfaces and transaction methods. Each module with appropriate type of socket could bind with each other by using port-export binding method, and sends transactions by calling the relative method of the socket. Implementing sockets further reduce the bonding effort between blocks when designers try to build or modify the top level module of large systems. To model a variety of bus standards which are used in most of the SoCs, TLM-2.0 introduces a base protocol. The base protocol specifies a set of rules regarding to data transfer such as transaction phases and transaction statuses. By using (part of) the protocol feature, it is easy to model different bus protocols without requiring specific extensions. TLM-2.0 standard reduces modeling effort by aggregating ports, signals, data structures to a higher level of abstraction and help designers to build large complex systems by reusing codes from other sources. However, the systemC platform is still focusing on architectural-level simulation only without the support of any physical-level analysis.
2.5 Thermal Analysis tools

Both of the TLM-centric flows mentioned above are lacking of physical-level analysis. A thermal analysis tool called HotSpot [32] is used by system-level designers to estimate temperature. Building the thermal model at ESL, HotSpot takes a transient power trace as input for which it typically relies on an external detailed cycle-accurate performance/power simulator. The detailed architectural simulation approach works well if only the processor alone is considered, but this approach is not feasible for simulation of System-on-chip (SoC) containing several processors, memory and bus, etc. in realistic simulation time. To explore the impact of 3DIC, HotSpot is extended later in [33]. However, it has several 3D specific limitations. For example, currently it does not explicitly support modeling of TSVs and microbumps [34].

2.6 Summary

In this chapter, we discussed the challenges to create CAD flow for system-level design exploration and evaluation. The complex system and 3DIC technologies add more difficulties at both architectural-level and physical-level. Large amount of architectural configurations, run-time dependent requirements, different 3D integration schemes and limited modeling/simulation time budget make conventional CAD flows hard to capture all the necessary parameters and metrics for complex systems.

There are two types of system-level CAD flows: RTL-centric and TLM-centric flow. The RTL-centric flows usually take the detailed design netlist as input, and perform detailed
physical-level analysis such as area estimation and wire length and distribution estimation. The objective of RTL-centric flows is design optimization at circuit-level, targeting at circuit designers and manufacturers. On the other hand, the TLM-centric flows focus on complex system modeling which captures application dependent parameters as well as architectural parameters. The objective of the TLM-centric flows is to evaluate large amount of architectures with minimum effort and allow computer architects to find out the best design at an early stage. RTL-centric flow is hard to use by computer architects due to the huge effort to build detailed design before the evaluation. TLM-centric flow does not contain physical-aware feature which makes it hard to detect the impact of physical-level parameters especially in 3DIC. Due to the challenges of system-level design with 3DIC, integrating system-level and physical-level analysis into a unified CAD flow is necessary.
Chapter 3

System-level Pathfinding Flow

3.1 Introduction

3D integration enlarges the system-level design space by adding more physical-level choices such as number of stacking tiers, heterogeneous technologies for different tiers, and different 3D bonding methods. Those physical-level choices affect the system-level design decisions and necessitate a system-level physically aware pathfinding flow. Such a flow should be able to evaluate systems at physical-level without the need of detailed gate netlist. To be specific, an system-level pathfinding flow for 3DIC should be able to (a) explore design spaces at appropriate level of abstractions for different 3D related physical parameters such as stacking schemes, floorplans and wafer technologies. And (b) shows the feedback added by physical choices and help system designer to optimize the design. Furthermore, the pathfinding flow should provide valuable input for circuit designer or manufacturer to guide their work.

Current pathfinding flows for 3D-IC evaluation have several problems. Firstly, the physical-level pathfinding flow requires too much design details. For example, in [25] a completely RTL netlist is required to perform the simplified physical design flow to get parasitic capacitances. The capacitances are used to get power data via detailed RTL
simulation. Then the power data can be used for temperature or power distribution analysis. In [26], a completely gate-level netlist is required to explore different partitioning and floorplanning schemes. These approaches require huge effort to perform system-level analysis and prevent computer architects to evaluate their designs in architectural-level. Secondly, the system-level pathfinding flow is lacking of flexibility to build various systems. For example, in [27] people can easily build a multi-core system with the built-in feature such as memory hierarchy and scheduler. However, it is extremely difficult to add different modules or features since it usually requires hard-coded and time consuming. Thirdly, most of the pathfinding flows are lacking of integration of architectural-level and physical-level analysis tools. Lots of physical-level analysis tools start from RTL or gate-level design netlist, which is hard to obtain from system-level CAD flow. On the other side, system-level CAD flow does not provide enough area, power and floorplanning information to perform physical-level analysis. The difficulty of integrating system-level and physical-level analysis together prevents architectural and physical level co-optimization for complex system at early design stage, which is more and more important due to the time-to-market pressure and coupled design space in 3D-ICs.

In this chapter, we present a pathfinding flow for fast system-level 3D-IC evaluation. Due to the complexity of 3D systems, the entire pathfinding flow is based on ESL instead of RTL, yet still provides enough physical-level analysis to evaluate the impact of 3D integration. The flow we presented incorporates a transaction-level simulator for system modeling and simulation which can accelerate the system-level pathfinding process. An ESL floorplan based toolset is used for physical-level evaluation such as thermal and routing analysis. The
pathfinding flow we presented here can build and evaluation system fast while providing useful information about power, performance and physical-level metric such as temperature for system designers.

### 3.2 Pathfinding Flow Descriptions

#### 3.2.1 Overview

Fig. 3.1 shows the proposed pathfinding flow, which is divided into two parts: the front-end part and the back-end part. In the front-end part, an ESL system description is created to evaluate system performance and provide input to the back-end part. To capture the dynamic effects of underlying application on power and performance, a transaction-level simulator is implemented. The simulator can show the performance in terms of number of transactions per cycle for each IP block by running a benchmark. Furthermore, a high-level power model is used to calculate the dynamic and static power based on IP configurations and performance results from a TLM simulation. The back-end part of the flow starts with an ESL floorplan obtained from system level description using an area model of components. Users must specify technology information of each layer/tier of the stack (the “wafer technologies”), information about TSV-based stacking of different tiers (a “stack technology”) and material properties. To speed-up the thermal simulation, layers in the wafer technologies are collapsed into a reduced set of layers, called composite layers. We feed the composite layers, rough floorplan and power information to our in-house thermal simulator to generate the static and dynamic temperature profiles. Meanwhile, a global routing tool performs global routing for
wires connected between blocks, and TSVs for 3D integration. The routing tool generates global wire length and distribution as well as total number of TSVs, which represent the cost for actual implementation of the designs. The system description can be easily changed based on obtained power, performance and thermal results and new iteration can be performed. This flow facilitates fast design space exploration to find optimal 3D design without doing detailed implementation of all the design choices.
Figure 3.1 System-level Evaluation Flow for 3D-IC
3.2.2 Front-end Part Description

The front-end part of the flow consists of a system description, a transaction-level simulation framework, and a high-level power model to calculate power based on TLM simulation results.

A. System description

As a first step, designers are required to create high-level system description for a set of design parameters using functional and timing models of component IP blocks. This step includes two phases namely: IP configuration and logical partitioning. In the IP configuration phase, designer set appropriate parameters for each IP block (core type, cache size, etc.) and in the logical partition phase, designer specifies how the IP blocks are connected to each other and how the data is transferred among the blocks. If the system has caches and memories, the cache policy and cache protocol are also determined in this phase.

B. Transaction-level simulator

It is important to capture the dynamic effects of underlying application on power and performance; hence a transaction-level simulator is used. TLM based simulation can greatly reduce the simulation time compared to RTL simulation with acceptable timing accuracy [42]. The authoring of TLM simulations is the most time-consuming part of this flow, and therefore users will most often need to reuse existing TLM frameworks. The TLM simulator used in this work is the GEMS framework, which maintains a global event queue to manage
all the blocks. Each block can schedule an event for the subsequent block. The global event queue triggers all the appropriate events scheduled for current cycle. To ensure functional and timing correctness, all the outputs of a block are available to other blocks in the next cycle. Thus all the blocks can be triggered out-of-order. We get switching activity results from this event-driven simulator in terms of cycle counts to estimate dynamic power.

C. Power Model

A high-level power model is needed for estimating power consumption. Computer architects typically use analytical models to predict power. However, it can be difficult to tune these models to accurately represent the expected power of a block after it has passed through a physical design flow. We therefore chose to characterize the blocks by extracting power models after carrying them through a complete physical design flow. This process is time consuming, but it is faster than the alternative, which is to assemble the complete system RTL before beginning a pathfinding study. By characterizing sub-blocks, we avoid the effort of designing glue-logic and modifying block interfaces such that they are compatible. The power model will be introduced in detail in Chapter 4.

3.2.3 Back-end Description

The backend of the proposed flow primarily consists of composite model extraction, ESL floorplanning, thermal, and global routing simulation. It takes material properties, wafer and stacking technologies descriptions as inputs. These data can be obtained from vendor PDKs. In order to permit delivery of this flow and avoid the intellectual-property restrictions on
vendor kits, we developed a free, open-source design kit compiler for stacked dies in a predictive 45nm technology, which we call the FreePDK3D45 [43]. The kit represents a five-tier stack of FreePDK45 predictive technology allowing the designers to maintain a complete 3D conception of their design. The kit supports three kinds of through-silicon-vias (TSV): a) down via b) up via and c) through-tier via, as illustrated in Figure 3. The kit is intended for use in demonstrating and debugging new OpenAccess-based design tools for 3DICs. As such, the kit contains the basics of what is needed to perform schematic entry, SPICE simulation, layout, DRC, and LVS checks.

Using FreePDK3D45 as an underlying framework, we have a developed an open source toolset to deliver the path-finding flow proposed here [17]. The tool currently supports thermal evaluation of TSV-enabled digital architectures as well as global routing analysis. Here we describe composite-model extraction, rough floorplanning, thermal analysis and routing aspects of the flow.

A Composited Model Extraction

Pathfinder3D’s technology-file format allows the specification of a new heterogeneous 3D stack with changes to only a few lines of code [17]. The materials section allows specifying a list of material names, their thermal conductivities, densities and specific heat capacities. Single-wafer manufacturing technologies and their associated layers, materials, and thicknesses can be specified in wafer-technology section. The stack technology section allows instantiation of wafer-technology tiers plus additional glue layers. Using this approach, a complex layer stack can be described very concisely.
Figure 3.2 Cross section of the first three tiers of the FreePDK3D45 technology.

The complete list of cross-sectional layers generated by the FreePDK3D45 technology is more than 100, which is too much detail for a system-level thermal analysis. Hence this tool represents each tier in the 3D stack with a reduced set of composite layers namely a)
substrate b) active and c) metal. This division is chosen because of the differing thermal properties of each portion. First, substrates typically have much higher conductivity than other portions and are therefore modeled separately. Second, the active layer is modeled separately, because it is typically the place where the majority of heat is generated. Third, the metallization layers can be collectively viewed as a metal-insulator composite and are typically where most of the temperature rise occurs in a 3D stack. Thermal conductivity in such composite materials can be difficult to predict, because the conductivities of materials and insulators differs by 100 to 1000. Accurate prediction requires a detailed description of the structure and a set of complex matrix solutions to find the thermal conductivity (k) to be used with Fourier’s law of heat conduction in the form of a 3x3 matrix tensor.

In the absence of detailed layout information, Pathfinder3D constructs a basic unit cell that depends on the metal densities for each layer (as a 0 to 1 range) and its routing direction (horizontal, vertical or cut). Performing the exact matrix calculation of the conductivity tensor for the basic unit cell is straight-forward and described in detail in Pathfinder3D tutorial [17]. However, the exact conductivity tensor is currently not computed, because it is time consuming to code and debug. In the meantime, Pathfinder3D calculates upper and lower bounds on the diagonal elements of the conductivity tensor (kx, ky, and kz). The off-diagonal elements tend to be small, since most of the wires tend to follow the x, y, and z axes.

Here we describe how these conductivity bounds are calculated. The two simplest thermal conductivity calculation models can be called the parallel and orthogonal models. If a composite material consists of a matrix of one material with cross-sections in another material running parallel to the direction of heat conduction, then the material can be viewed
as a parallel combination of conductances. If a composite material consists of a series of material segments that are orthogonal to the direction of heat conduction, then the material can be viewed as a series combination of conductances. Most materials will not directly fall into these two categories. However, there are a large number of structures (including rectangular meshes) for which we can apply a combination of the parallel and orthogonal equivalent calculations. The order in which we apply them has different assumptions on how heat flows and can therefore be viewed as bounds on the equivalent conductivity. The upper bound can be calculated considering a unit cell consisting of \( N \) orthogonal segments, each with \( M_j \) parallel cross-sections. In this case, a parallel equivalent can be assumed for each segment, and an orthogonal equivalent can be assumed for the segments collectively. The equivalent parallel-orthogonal conductivity can be calculated as follows:

\[
\kappa_{eq\_par\_orth} = \frac{1}{\sum_{i=1}^{N} L_i \left( \sum_{j=1}^{M_i} \frac{A_{i,j}}{k_{i,j}} \right)},
\]

assuming,

\[
\sum_{i=1}^{N} L_i = 1, \forall i \sum_{j=1}^{M_i} A_{i,j} = 1.
\]

This model differs from the exact equivalent conductivity, because it assumes perfect heat spreading between the segments. In reality, heat spreads gradually through the material when a temperature gradient is applied. Therefore, \( \kappa_{eq\_par\_orth} \) can be viewed as an upper-bound on the conductivity of the composite material. The lower bound of equivalent thermal conductivity can be calculated considering a unit cell consists of \( N \) parallel cross sections,
each with $M_j$ orthogonal segments. In this case an orthogonal equivalent can be assumed for each cross-section, and a parallel equivalent can be assumed for the cross sections collectively. The equivalent orthogonal-parallel conductivity can be computed as follows:

$$
k_{eq\_orth\_par} = \sum_{i=1}^{N} \left( \frac{A_i}{\sum_{j=1}^{M_i} L_{i,j} k_{i,j}} \right),
$$

(3)

assuming

$$
\sum_{i=1}^{N} A_i = 1, \forall \sum_{j=1}^{M_i} L_{i,j} = 1.
$$

(4)

This model differs from the exact equivalent conductivity, because it assumes no heat spreading between the cross-sections. Therefore, $k_{eq\_orth\_par}$ can be viewed as a lower-bound on the conductivity of the composite material. In Pathfinder3D, these approximations are applied to the basic unit cell that is constructed for each technology. The upper-bound, $k_{eq\_par\_orth}$ is used for $k_x$ and $k_y$ values, while the lower-bound, $k_{eq\_orth\_par}$ is used for the $k_z$ value.

The tool also considers the effect of TSVs in equivalent thermal conductivity calculation. TSVs are essentially cut-layers, but depending on the start and stop layers defined for each TSV in the technology file, they are likely to coincide with a wafer-technology layer (i.e. horizontal or vertical routing layer or another cut layer). In these cases, the density of the TSVs is added to the density of the wafer-technology layer, and the wafer-technology routing direction is assumed. The new “metal” conductivity is calculated as the parallel equivalent of
the wafer-technology routing layer and the TSV. This is akin to an upper-bound on the conductivity impact of a TSV.

Pathfinder3D also calculates the equivalent specific heat capacity of metal-insulator composite using the similar approach used for equivalent thermal conductivity calculation. This parameter is used in transient thermal simulation.

B. Rough Floorplan

In the pathfinding flow, a rough floorplan and power profiles of floorplan blocks are required for the thermal simulations. Pathfinder3D allows users to specify textual description of their floorplan. They can define the dimensions of basic building blocks of each tier in the 3D stack as macrocells. Each macrocell can have multiple sockets representing ports of the block. Users can then replicate these macrocells as different instances at various locations in their corresponding tiers by specifying the coordinates and the connections among instances. The instances connection information can be imported from the system description defined in the front end. After that, a simple routing tool is applied to estimate global wire length and repeaters are added for those wires based on a minimum delay insertion algorithm. This allows estimating interconnect power of the design. In this way a full floorplan can be constructed. Currently users can specify the total power value for each instance in the floorplan in a separate power stimulus file.

The tool reads the floorplan and creates an Open Access layout. In layout, each macrocell is mapped with three layers namely; substrate, active and metal-insulator composite associated to macrocell’s wafer technology. Furthermore, the tool also reads the power stimulus
file and assigns the total power of each instance to the active layer of that instance. Currently power is uniformly assigned to complete area of macrocell. At this stage, the inputs to the thermal simulator are ready.

C. Thermal simulation

The Pathfinder3D toolset consists of a physical thermal extractor, WireX [12]. WireX reads the layout generated from rough floorplan and creates a linear resistive and capacitive thermal netlist. It uses the thermal conductivity, specific heat capacity and other material properties of the composite model to generate this netlist. WireX meshes the layout and discretizes it in cuboids. Each cuboid is modeled using a thermal resistor from the center of cuboid to each face with a thermal capacitor from center terminal to the thermal ground. The user is able to control the fidelity of mesh. Usually for fast thermal simulation in pathfinding phase the resolution of mesh is kept low. For steady state thermal analysis, the extractor generates a thermal modified nodal admittance matrix (Y) and power vector (J). Then, \( Yv = J \) is solved by a linear sparse matrix solver to get the temperature vector (v).

The WireX tool currently supports only one style of boundary condition, which is a perfect heatsink on one face of the stack and adiabatic on all other faces. The perfect heatsink is assumed to be connected to the first tier specified in the stack technology. All temperatures will be reported as rise above the heatsink. For absolute temperatures, the temperature calculation must take the package and physical heatsink in account. The current model is accurate assuming the substrate of first tier is a perfect heat spreader which is not always an
accurate assumption. However, users have flexibility to model portions of the package and heatsink as composite layers in the Pathfinder3D technology file.

The transient thermal simulation requires a transient solver. Currently Pathfinder3D does not include any transient solver. However, it can generate a netlist compatible with HSPICE or the open source circuit simulator fREEDA [13 from 3DIC]. fREEDA provides various techniques [44] to speed-up the transient electro-thermal simulation which can be very useful for pathfinding studies.

*D Global routing and wire length estimation*

Aside from thermal analysis, pathfinder3D also provides a global routing tool. The tool reads the floorplan and blocks connection information, performs trial route under area/capacity constraints. In the case of 3D integration, the TSVs are also taken into account. An important benefit of 3DIC is the global wire reduction due to small chip area and vertical logics. In that case, it is important to evaluation the benefit before implementing detailed design.

**3.3 Transaction-level Simulation with GEMS framework**

**3.3.1 Event-driven Simulation**

In GEMS framework, different modules are managed by a scheduler via event queue. The event queue schedule mechanism is the kernel of this simulator. Figure 3.3 shows how the simulator model parallel blocks like hardware by using event queue. Assume that there
are two blocks A and B are connected with each other, and A would like to send data to B. To simulate that behavior, first, we need to build the connection between A and B. In the design partitioning phase, both blocks are instantiated and the consumer of A is pointed to B. During the simulation, the scheduler will check the event queue every cycle, and in cycle N, block A is scheduled and to be triggered. Then A is executed and generates data for B, meanwhile, A will send a schedule request to the event queue to schedule its consumer B. Assume that it takes m cycles for A to generate the data and send it to B due to link latency, A would schedule B to trigger B m cycles later. The events in event queue are checked every cycle, and in cycle N+m, B is executed. B reads the data generated by A m cycles before. In that example, both A and B perform their function upon triggered. Since the schedule request and trigger block cannot happen during the same cycle, the execution order of blocks in the same cycle does not matter. In case a block is scheduled multiple times to execute at the same time, all schedule requests are merged and the target block executes once in that cycle. The event-drive simulator makes it easy to build complex system. Designers can add new blocks by using this event queue mechanism while do not need to arrange the execution order for each block as normal high-level simulator required.
3.3.2 Adding Interconnect Model

GEMS framework is built for multi-core and implements various cache protocols to address the coherency problems. It also has some simple network to simulate the interconnect behavior. However, the simple network provided within GEMS framework does not maintain a detailed timing model and cannot capture the behavior for interconnect-centric system such as Network-on-Chip (NoC). In this chapter, we present an approach to add interconnect to GEMS for a circuit-switched NoC.

First, we need to identify the interface between interconnections and other blocks such as cores and caches. In GEMS framework, cores, caches and interconnections are sub-systems in which multiple copies such as cache banks, routers can be instantiated. Those sub-systems are connected via “message buffer” storing the data for exchange between them. Thus to
create a new interconnection sub-system, we need 1) create the network sub-system including routers and arbiters inside the network. 2) Create message buffers connecting blocks outside the interconnect sub-system.

After setting the interface up, the next step is port bonding and consumer setting. As we showed before, all blocks need to “schedule” their outputs to the event queue so that they can be triggered correctly. In our case, interconnects can be divided into two parts: data-path and control-path. Data packets in data-path are transferred from router to router or router to core/cache. Each router should set its output block as consumer and schedule the consumer if there is pending data ready to send out. Meanwhile, a shared queue is created for the 2 routers to store the data. In control-path, request and grant signals are transferred instead of data. We have a centralized arbiter which controls all the traffic inside the interconnections. All routers should set the arbiter as their consumer and schedule the arbiter if there is a pending data to send. That will cause multiple schedule event occurs at the same time, but since we can merge the event as mentioned above, the arbiter can be triggered correctly without request loss. And since the arbiter decides which router(s) can send data, it also needs to set all routers as its consumer and schedule the routers based on arbitration result. Figure 3.4 shows the above structures.
3.4 Summary

In this chapter, a pathfinding flow for system-level design evaluation. The flow is divided into two parts: front-end part and back-end part. The objective of front-end part is to generate system description for simulator and back-end part. In this flow, we incorporate a TLM simulator from GEMS framework. The TLM simulator can build various systems with minimum effort while maintaining timing information for performance evaluation. To extend
GEMS framework, we present an example on how to build custom on-chip interconnections in that framework. The simulator provides power and system performance for architectural-level evaluation and serve as input data to the back-end part. In the back-end part, we integrate a floorplan based physical-level framework. Different from the detailed gate-level netlist, floorplan use block level description which is easy to be extracted from front-end part. We describe the thermal and routing analysis used in this flow, temperature and routing results can be generated in back-end part with the input from front-end part, and feedback to front-end part for system-level evaluation.
Chapter 4

High-level Power Model and Characterization

4.1 Introduction

Power consumption is an important metric for system design especially in 3D-ICs. Since the power for each block not only represents the energy consumed during certain amount of time, but also affects chip temperature. In our pathfinding flow, power is the connection between the front-end part and the back-end part. The transaction-level simulator generates power number for each ESL block and feeds the data into the ESL floorplan to perform thermal simulation. In that case, a fast and easy-to-use power generating method is required to make the front-end and back-end evaluation flow working together.

Generally, there are two approaches to generate power number: high-level abstracted model centric approach and detailed design netlist centric approach. Both approaches are widely used but focus on different applications. The netlist centric approach is used in circuit design and manufacturing. To get the power number, first an RTL netlist is required to describe the design. Then the netlist is synthesized and converted to standard cells. Those standard cells are created and simulated at transistor level based on certain technologies to get accurate power number. After generating standard cells netlist, place & route flow is performed to get the actual layout of the design. For power estimation, not only the logic
cells but wires and capacitances contribute. In the place & route flow, all the wires connecting cells are created and routed, the wire length and shape can be used to estimate wire resistance and capacitance. And the crosstalk capacitances can be extracted via parasitics extraction. Furthermore, an RTL test bench is required to capture switching activities for all input, output and internal nodes. With all the information about cells, wires and parasitic capacitances, we can get accurate power number through simulation and analysis. Such a process is used for detailed circuit design. Though the most of above can be done automatically with current commercial tools, the detailed RTL netlist for design and test bench require huge amount of time and effort. Since power estimation in system-level is to help designer to make choices over different architectures, the netlist centric approach is not appropriate. Hence we focus on the high-level model centric approach.

In the netlist centric approach, we build the power model at the standard cells level. The entire design is break down to standard cells and the power of standard cells is pre-characterized by using HSPICE simulation. Moreover, we follow the standard approaches to perform the parasitic extraction and switching activities annotation with commercial EDA tools. However, in high-level power estimation flow, there are several difficulties preventing such standardization.

First, the large design space from architectural-level down to physical-level makes it hard to build standard power model. Power is affected by both architectural-level and physical-level parameters such as different technologies and number of pipelines for a core. In that case, people build their own power models focusing on parameters in different abstract levels, making it hard to integrate multiple power models together. The second difficulty is the
limited data to generate power model. In standard cell models, the most of existing libraries contain all kinds of standard cells such as basic logic gates and flip-flops that are used to implement complex designs. And it is convenient to add new cells by using Marcos along with an HSPICE model. However, in system-level design and architectural research, implementing all sub-systems requires too much effort. Hence people usually find some existing models to estimate power. Since accurate commercial models are usually not shared between companies, designers have to develop their own power models with limited data. Another difficulty in building high-level power model is the trade-off between accuracy and flexibility. The goal of standard cells power model is to provide accurate power number for designers to optimize the design. But the building effort and scalability must be considered in building system-level power model. Circuit designers and manufacturers would like to see accurate power model which can provide valuable estimations to guide the circuit/board design. On the other side, computer architects who explore different systems require the power model to be easy to use and can be tuned for different configurations with minimum effort.

4.2 Current High-level Power Model

4.2.1 CACTI, the cache & memory model

One widely used power model is the cache/memory model CACTI [45], which is popular in academic since implementing cache or memory in detail requires huge effort and without industrial circuit libraries, it cannot get comparable performance. CACTI provides simple
interfaces to users who want to get power, delay and area for memories or cache (with or without tag) blocks. Users can specify architectural parameters such as cache size, block size and associativity, or technology parameters such as threshold voltage. For all cache data and tag blocks, either “high performance” or “low power” type can be specified. The power number is figured out based on capacitances. Since cache and memory are designed with repeatable units, the total capacitance can be estimated based on the cache architectures. Such an approach can generate power number for various configurations and integrate with other high-level analysis tools. But the accuracy problem in CACTI makes it good to compare different cache configurations only. Table 4.1 shows the difference by using high performance and low power configuration for different cache sizes. 70 times difference between the two configurations for leakage power can cause huge different results in thermal analysis. And with high-performance configuration, the cache behavior is negligible and power consumption is dominated by cache size.

<table>
<thead>
<tr>
<th>Cache Size</th>
<th>Leakage Power HP (mW)</th>
<th>Leakage Power LP (mW)</th>
<th>Dynamic Energy HP (nJ)</th>
<th>Dynamic Energy LP (nJ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>128KB</td>
<td>155</td>
<td>2</td>
<td>0.21</td>
<td>0.09</td>
</tr>
<tr>
<td>256KB</td>
<td>280</td>
<td>3.5</td>
<td>0.253</td>
<td>0.12</td>
</tr>
<tr>
<td>512KB</td>
<td>528</td>
<td>7</td>
<td>0.336</td>
<td>0.18</td>
</tr>
<tr>
<td>1024KB</td>
<td>1140</td>
<td>16</td>
<td>1.18</td>
<td>0.81</td>
</tr>
<tr>
<td>2048KB</td>
<td>2357</td>
<td>32</td>
<td>1.67</td>
<td>1.14</td>
</tr>
<tr>
<td>4096KB</td>
<td>4505</td>
<td>61.9</td>
<td>2.57</td>
<td>3.44</td>
</tr>
</tbody>
</table>
4.2.2 ORION: On-chip router power model

Orion is a power model for on-chip routers that connect cores and caches in network-on-chip [46]. Similar to CACTI, ORION also models both architectural and technology parameters, such as router radix, buffer depth, technology nodes. Power estimation in ORION follows similar approach as CACTI. On-chip routers are divided into several sub-blocks such as crossbar and arbiter, and capacitances are estimated based on the circuit structures of sub-blocks. To explore the router power under different technologies, ORION applies scaling factors across different technologies. The capacitance for all basic transistors and logic gates are scaled according to different scaling factors. ORION provides an easy way to enlarge the modeling space. However, it uses too many linear scaling approaches. The router power with different radix is scaled linearly, and power consumption with different technologies is scaled by using the scaling factor directly, which reduce the accuracy of the power model.

4.2.3 McPat: Processor power model

Different from CACTI and Orion, McPat [47] focuses more on processor models. Since processors are more complex than memories and routers, they require more complex power model. McPat provides a unified power, area and timing estimation framework, and quickly developed as the most popular power extension for performance simulators. It uses a XML interface for users to specify design parameters from the architectural-level such as core type, frequency, cache size, multithread, down to the technology-level such as threshold voltage.
Based on the configurations, McPat select circuit-level analytical model to estimate power. However, the complex interface in McPat is not fit with other simulators such as GEMS or GEM5. To overcome the difficulties, several assumptions need to be made while integrating McPat with other simulators, thus generating errors.

4.3 Power Model Characterization Flow

High-level power models are usually built based on analytical model like CACTI or ORION. Analytical model does not require simulation to get power number. Instead, it predicts total capacitance which is derived from certain type of circuit structures. Such approaches reduce the effort to build model, but lose accuracy. The standard cell based power model is characterized model. Cells are created and simulated at transistor level. Power consumption of cells is characterized with size and load based on simulation result. The characterized power model creates an accurate power library but requires the design to be break up into standard cells to match the power model.

Though the analytical power model is considered easy to build and extend, as more complex target systems come, it is difficult to tune to accurately estimate the expected power of a block after it has passed through a physical design flow. On the other side, characterized model maintains simple interfaces to users while keeping good accuracy. The standard cell level model is not appropriate for system-level design due to the low abstract level of the power model. Designers are required to build detailed netlist to match the power model which prevents fast estimation at system-level.
In this chapter, we present an approach to build high-level power model by using characterization. We characterize the power model at ESL instead of standard cell level to match the transaction-level simulation framework. By characterizing power for ESL blocks, we avoid the effort of designing glue-logic and modifying block interfaces such that they are compatible. It is a time consuming process compared to analytical model, but can achieve more accuracy. Figure 4.1 shows the presented flow for power characterization.
Figure 4.1 High-level Power Model Characterization Flow
The inputs of the flow are divided into design parameters and characterization parameters. Design parameters specify how the design should be implemented in RTL. And the characterization parameters specify run-time dependent input pattern for testbench. The characterization parameters link TLM simulation results to the power model, helping the evaluation flow to figure out the power consumption fast based on the TLM output. An example of such a parameter is the injection rate of blocks, which affects the dynamic power consumption and can be derived from the TLM simulation result. The testbench generator maps the characterization parameters to actual RTL input patterns. TLM simulator usually generates statistics such as cache miss rate or block injection rate which cannot be directly used as the input for detailed design. In that case, the testbench generator will create a set of RTL testbench to simulate all possible input combinations based on the TLM output. Next, a standard cell netlist is generated with extracted parasitics from commercial physical design tools as the standard cell centric approach does. After we have got the netlist with parasitics, it is simulated with the characterized testbench and detailed switching activities are recorded. The power consumption is calculated by using switching activities and standard cell technology library. At last, a lookup table is generated with block information and power consumption indexed by characterization parameters.

4.4 Power Model Characterization for crossbar used in on-chip router

In this section, we present a power characterization example for a crossbar used in on-chip routers. Crossbars (Xbar) are designed to transfer multiple packets simultaneously by
using combinational logics. It is a critical part in terms of area and power consumption especially when the number of input and output ports increased. A crossbar is usually built with basic units so that it can be scaled up for different number of input and output ports. The high-level analytical power model Orion 2.0 also provides power estimation for crossbars. We will compare the power number generated from characterization flow with the estimation result from Orion 2.0 to show the importance to have a characterized model for ESL power estimation.

### 4.4.1 Crossbar design description

Crossbar is used for on-chip data transfer between function blocks. Since crossbar does not have any control logic, it is always implemented within a router which helps to decode and arbitrate incoming transactions. Compared to the shared bus which connects all on-chip blocks and provides a shared data channel, crossbar supports high-speed on-chip communication better due to the parallel transfer feature. Assuming there are two packets are sent from different sources simultaneously, targeting different destinations. Those two transactions can be transferred at the same clock cycle with crossbar, which increases the communication bandwidth. However, the bandwidth increases at a cost of area and power, especially for systems with lots of on-chip router and data transfers. In that case, estimate power for different type of crossbar is important to schedule power budget before actually implementing the detailed design.

In this section, a logic gate based crossbar is implemented and characterized. The crossbar is used in a fully-pipelined router which is a part of our target system in chapter 6.
Figure 4.2 shows the structure of crossbar. Assume that we have an \( N \) input \( M \) output crossbar, and each input has a request signal attached. The request signal is generated by arbitration blocks in the router, to indicate the crossbar which input data is valid and should be forward to output port. The request signal should guarantee that for each output port \( m \) (\( 0 < m < M \)), the related request signals \( req \left( n, m \right) \) (\( 0 < n < N \)) only have at most one request valid while all the other \( req \left( n, m \right) \) should remain 0. The contention case is handled by control blocks in the router, so the \( req \) signal should always perform correct patterns. Filtered input data will be transferred to output port via logic OR gate. Logically we have an \( N \) input OR gate for each output port. However, when the crossbar is implemented in RTL, usually only 2-input gate is used to reduce load capacitance as well as delay. So for a \( N \) input \( N \) output crossbar, the order of gate number is \( O(N^2) \), which means the power consumption increases quadratic as the radix increases and necessitates the power analysis for crossbar.
4.4.2 **testbench generation and characterization**

As the characterization flow shown in figure 4.1, the input can be divided into design parameters and characterization parameters. For the crossbar, the design parameters are number of input/output and the data size for each input/output. Each combination of design parameters results in an independent crossbar design for synthesis and parasitics extraction. The characterization parameter selected is the injection rate. Injection rate represents the load
of the crossbar, measured as how many data packets are transferred through crossbar per cycle. The injection rate is unified with number of input/output port, which makes it a number from 0 to 1.

Using a single parameter (injection rate) to characterize power is very convenient for computer architects who model complex on-chip systems. However when it comes down to a detailed RTL simulation, we need to decide how to translate the injection rate to an actual RTL input pattern for both data and request signals. Since the goal of injection rate is to estimate power under different load, the input data should be a constraint randomized pattern to match the injection rate. That means if we set injection rate to 1, not only we need to ensure that all input ports send packets to output ports at the same time, but the data bits should flip from one to another packets to reflect the maximum load case. For the injection between 0 and 1, there are two ways to implement: 1) part of the input/output ports are activated to represent the load. 2) All input/output are activated for certain amount of time to represent the load. We try both approaches to see if there is significant difference between the two. Meanwhile, considering the nature of data communication, it happens that one input port always send data to one output port and the data is almost fixed. To simulate such “streaming” effect of data transfer and examine how it affects power consumption, we need to set up test bench for those streaming data transfer cases.

The complexity of RTL testbench needed for a single characterization parameter necessities the testbench generator which help translating the TLM parameters to RTL input. Figure 4.3 illustrate how the generator works to build a set of testbench for power estimation. At the first step, a testbench base is created which contains the framework and input/output
port definition. Next the input patterns for data and request are created based on injection rate respectively. Data input can be random, fixed, or flipping between all “0” and all “1” each cycle, to simulate different switching cases. Request input should ensure no contention request pattern is generated, and generate valid requests depends on the injection rate. Then the generated input pattern is inserted into the testbench base. This process iterates several times for different combination of injection rates and data switching cases.
Figure 4.3 Test Bench Generation Flow
4.4.3 Result Analysis and Comparison with Orion 2.0

Based on the power characterization flow above, we generate the power model for crossbar. The design parameters and simulation environment are listed in table 4.2. We use a 7-port (7 input and 7 output ports) crossbar, each port contains 16-bit data, simulated with Nangate opencell 45nm technology under 1GHz clock and 1.1V supply voltage.

Table 4.2 Design and characterization parameters for crossbar power characterization

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of input/output</td>
<td>7</td>
</tr>
<tr>
<td>Data size</td>
<td>16-bit</td>
</tr>
<tr>
<td>Frequency</td>
<td>1GHz</td>
</tr>
<tr>
<td>Supply Voltage</td>
<td>1.1V</td>
</tr>
<tr>
<td>Technology</td>
<td>45nm</td>
</tr>
<tr>
<td>Load</td>
<td>From 0 to 1</td>
</tr>
</tbody>
</table>

Figure 4.4 shows the characterization result for the crossbar. The power consumption includes both dynamic and static power, estimated under different injection rates. Input data and request are set to all “0”s in the cycle of no input data case. Three types of data pattern are applied at injection rate from 0 to 1 with a step of 0.05. In the first pattern, input data switching between “0” to all “1” for each input port, representing the maximum power consuming case. In the second pattern, random data are applied to represent average power consumption. In the last pattern, we use fixed data input in which only 1 bit is set to high for each input port. The fixed data pattern shows low power case for different injection rates.
When injection rate equals to 0, the power consumption for all data pattern is about 1.2mW. Most comes from the dynamic power consumed by clock related gates and cells. The static power is 0.0625mW which contributes less than 10% for total power. As the injection rate, the random data pattern shows the average power consumption as expected which maximum and fixed data pattern represent upper and lower bound respectively. The maximum switching pattern consumes 38% more power compared to random data pattern. In that case, while creating the text power model, it is important to take the data switching schemes as one of the input parameters.

Figure 4.4 Characterization result for 7-port crossbar with 3 different data input pattern
To explore different request characterization effects, we compare two approaches that interpret injection rate in different ways. 1) Either all requests generate valid input or no request is injected, which is called the time-dependent scheme. 2) Each input port has a chance to generate input request, while the chance equals to the current injection rate. Figure 4.5 shows the differences for the two different request patterns. And we can see that the different way to interpret injection rate do not affect the power result. In that sense, we can safely use 1) in our approach to simulate different injection rates.

Figure 4.5 Characterization result for 7-port crossbar with time and port dependent request pattern
From the result of fig. 4.4 and 4.5, we figure out which characterization parameters to be used in our power model other than injection rate. A two dimensional look-up table is generated indexed by injection rate and data pattern (max, avg and min) as figure 4.6 shows. The table provides a simple interface to high-level simulator, help it pulling power number fast at system-level.

```
zbar7x16:
|
| injection rate:
| [0 0.05 0.1 0.15 0.2 0.25 0.3 0.35 0.4 0.45 0.5 0.6 0.7 0.8 0.9 1],
| switching scheme:
| [max, avg, min],
| dynamic power:
| [0.001 0.0012 0.0015 0.0019 0.0022 0.0025 0.0028 0.0031 0.0034 0.0037 0.004 0.0043 0.0046 0.0049 0.0052 0.0055 0.0058 0.0061]
| [0.0012 0.0013 0.0015 0.0017 0.002 0.0022 0.0025 0.0027 0.0029 0.0032 0.0034 0.0036 0.0038 0.004 0.0042 0.0044]  
| [0.0012 0.0013 0.0015 0.0016 0.0018 0.002 0.0022 0.0024 0.0026 0.0028 0.003 0.0032 0.0034 0.0036 0.0038 0.004 0.0042 0.0044]
| }
| leakage: 6.24e-05,
|
```

Figure 4.6 Power model format for 7-port 16-bit crossbar

To compare our power estimation result with other high-level power models, we use the same configuration in Orion 2.0 and get the power number from that. To minimize the error from technologies, we tune technology parameters defined in Orion 2.0 to match our library. Figure 4.7 shows the estimation result for Orion with the same crossbar configuration. Orion provides a linear estimation for different injection rates, which is different from our simulation result since the registers and other clock dependent gates are not considered in Orion. In that case, if injection rate is set to 0, power predicted with Orion only contains
static part. Moreover, Orion simply scales the power number from “max” case to “avg” case by halving the number. But the simulation result shows only 38% increased power when goes from “avg” to “max” rather than 100%. The power number predicted by Orion is 5 – 10 times higher than simulation, which may cause two problems. 1) When building complex system with routers, the large power consumption of crossbar may affect the design choice and result in incorrect architecture. 2) Temperature estimation based on the power number may result in higher temperature rise than the actual case.

Figure 4.7 Power estimation of crossbar from Orion 2.0
4.5 Summary

High-level power model is a critical part in a system-level pathfinding flow. Power numbers predicted by system-level simulation feed into physical-level analysis, in that sense power model is the link between the front-end part and the back-end part. Most high-level power models are implemented based on analytical models to reduce the modeling effort at a cost of accuracy. However, the increasing system complexity requires more and more parameters to be evaluated, which increases the building effort of the analytical power model.

In this chapter, we present a characterization based power model generation flow. This flow incorporates the detailed physical design approach to get accurate power number. Different from the standard cell centric flow, we characterize power at ESL. A set of testbench are generated based on design and characterization parameters, and a RTL design with extracted parasitics is required for simulation. At last, a look-up table is generated which contains both static and dynamic power for different characterization parameters. With the one-time building effort, we create an easy-to-use model for power estimation in the early design stage. And such a model can be integrated with TLM simulator easily.

We present an example of to illustrate the power model creation by using crossbars. In this example we show how the flow generates testbench and estimate power, as well as the power model format. We compare the result to analytical power model and show the necessity to build power models with characterization flow.
Chapter 5

Physical-level Analysis

5.1 Introduction

One of the key features of pathfinding flow is to integrate physically aware prototype in the ESL design flow. In 3D-IC, most of the benefits from 3D integration are physical-level related, such as global wire length reduction, area reduction and stacking with heterogeneous technologies, which necessitate the physical-level analysis. In the pathfinding flow for 3D-IC design and evaluation, the physical-level analysis should provide physical-level metrics to guide system-level design and to illustrate architectural-level design trade-offs which affected by physical-level choices. There are two challenges need to be addressed before implementing physical-level analysis: 1) what kind of physical-level analysis we should provide in a pathfinding flow. The principle is to select physical-level metrics which affect architectural-level design choices. 2) The level of abstraction to build analysis tools and the interface with front-end part in pathfinding flow.

Fig. 5.1 shows the scope of physical-level analysis for system evaluation of 3DICs. Designs are converted into physical-level representatives by applying 3D integration configurations such as number of stacking tiers and bonding schemes. After that, various analyses can be performed based on the physical-level representative. Figure 5.1 also
illustrates why it is important with 3D technology. First, more physical options can be implemented in 3D-IC thus requires more physical-level analysis. For example, thermal analysis becomes critical for 3D-ICs due to the heat dissipation issue with multiple stacking tiers and leakage current increase under high temperature [48]. So that performing thermal analysis at early design stage helps to avoid thermal runaway. Secondly, 3D integration introduces lots of factors affecting chip cost. For example, global wire reduction in 3DIC may reduce the number of metal layer needed to build the chip, and then reduce wafer cost. The number of TSVs may increase the cost of 3D bonding process. Considering that cost saving issue is critical in 3DICs [49], physical-level cost related analysis is needed to help making system design decisions like whether 3D integration should be used, and what design options should be adopted.

Currently, most of the physical-level analyses are implemented based on conventional physical design flow. The purposes of such analysis are: 1) generate detailed layout for manufacturing. 2) Estimate the impact of power delivery network and process variation to validate designs. To get the best simulation accuracy, detailed design netlist and technology information are required to perform physical-level analysis. However, it is not fit for pathfinding flow due to the high building effort and slow evaluation speed. In this chapter, we integrate a floorplan based physical-level analysis framework [17] and build global routing analysis tool based on that framework. The advantage of such approach is to avoid detailed design netlist requirement yet still provide valuable information for system designers to examine trade-offs for different systems.
Figure 5.1 also reveals two important parts of physical-level analysis: thermal analysis and place & route analysis. For thermal analysis, we incorporate a thermal analysis tool called WireX [18]. It performs steady state temperature estimation based on floorplan and power consumption of blocks, which is suitable for pathfinding flow. For global routing
analysis, we develop a floorplan based routing tool performing trial routing and estimate global wire length, distribution as well as power consumption.

5.2 ESL Floorplan based physical analysis framework

To perform physical-level analysis, a physical-level representative is required, upon which the physical-level analysis tools can predict temperature and routing result. Conventional physical design flow use gate-level floorplan which contains all the physical details. However, the level of accuracy beyond the need to evaluate system at early stage and it is hard to implement in a pathfinding flow. Instead, we use an ESL floorplan as the physical representative and apply various physical-level analyses.

Figure 5.2 shows the structure of pathfinder3D and the related analysis tools. Pathfinder3D reads a texture floorplan description and generates layout by using OpenAccess (OA) database [50]. The ESL floorplan describes the design by using marcocells. The marcocells in the ESL floorplan represent the corresponding blocks in the ESL system-level design such as cores and caches. Thus, it is convenient to transfer the design description from architectural-level to ESL floorplan. The floorplan description also specifies the block configuration such as height and width, location and located tier. Marcocells connection information is included in the floorplan description. Each marcocell is defined with port and interface information, which specify the number of ports in the marcocell as well as the port size and location. Port can be placed on the edge of each marcocell (North, South, East or West), once the layout is generated, wires are uniformly distributed along each edge of blocks as long as the wire pitch can pass the design rules.
The ESL floorplan-based analysis engine influences pathfinding flow and it is extendable for various physical-level analysis tools. As we mentioned in Chapter 3, a thermal extractor
based on OA are incorporated for temperature estimation. In this work, we develop simple global routing tool to generate other important metrics such as global wire length, distribution and power consumption.

5.3 Global routing analysis

5.3.1 Global routing introduction

Routing is the process of finding the geometric properties of all nets connected blocks under the constraint of feasibility and performance. Nets must be placed in the routing regions and must not intersect with each other, namely they follow certain spacing rules. And minimizing total nets length and area is the primary goal for routing performance optimization.

The routing process can be divided into two steps: global routing and detailed routing. The goal of global routing is to generate a loose route for each wire, and assign rough routing region without specifying the actual layout of wires. In detailed routing step, actual geometric layout of each wire should be figured out within the assigned routing region under the constraint of spacing rules. Fig 5.3 illustrates the different results of global and detailed routing step [51]. Since the purpose to perform routing in pathfinding flow is performance estimation, we implement global routing process in this work and integrate it with pathfinder3D framework.
5.3.2 Graph Model and Routing Algorithms

The global routing problem is a typical graph problem. The routing region and their relationship are modeled as graph. In that case, before applying different routing algorithms, we discuss several graph models here. Fig. 5.4 shows three different graph models: grid model, checker board model and channel intersection model. The sample floorplan is presented in 5.4(a) including 3 blocks. 5.4(b) shows the grid model representative which build a \((h \times w)\) grid with vertices and edges to represent the actual floorplan. The black vertices imply the block location which cannot be used to route. Wires will be routed from a start vertex to the destination vertex through edges. Each edge and vertex contains capacity.
information as a constraint for routing. The checker board model is shown in 5.4(c) which is greatly simplified compared to grid model. The vertices represent blocks and edges shows possible connections between blocks. Different from grid model, the vertices and edges in checker board model are weighted based on the block size and possible routing capacity between two blocks. The channel intersection model is shown in 5.4(d). In this kind of model, edges represent possible routing channels and vertices represent channel intersections. It is similar to the checker board model but considering more details about routing channel so that the routing result will be more accurate. However, identifying channels and figure out weight for each channel requires extra effort. The channel intersection model is the best one for regular floorplan such as netlist at standard cell level or gate level, but not for custom floorplan in ESL. The grid model captures the area information of floorplan as well as blocks at a cost of higher memory usage and slow simulation speed, especially for high radix grid. In our pathfinding flow, we have an ESL floorplan which describe system design. A typical ESL floorplan usually contains tens to hundreds of blocks which is much less than the standard cell or gate level floorplan (10M to 500M cells or gates). Thus the drawback of grid model is minimized and the capability to capture area information helps to estimate wire length and power.
Routing algorithm is applied on graph models. Basically there are three types of algorithm for global routing: maze algorithm, line-probe algorithm and steiner tree algorithm. Steiner tree algorithm is used for multi-pin nets which beyond our pathfinding flow since we target the signal wires between blocks that are all two-pin nets.

Maze routing algorithm is used to find a path between a pair of points in a planar rectangular grid graph. The area available for routing is represented as unblocked vertices, whereas, the obstacles are represented as blocked vertices. The objective of a maze routing algorithm is to find a path between source and target vertices without using any blocked
vertex. The process of finding a path begins with the exploration phase, in which several paths start at the source, and are expanded until one of them reaches the target. Once the target is reached, the vertices in the path are retraced to identify the path. The retrace phase can be easily implemented as long as the information about the parentage of each vertex is kept during the exploration phase. Several methods of path exploration have been developed [52][53][54]. One big problem with maze routing algorithm is the large memory requirement since all grid nodes need to store information in maze routing algorithm. Line-probe algorithm is developed to solve the problem by using line segments instead of grid nodes [55]. However, the line-probe algorithm cannot guarantee to find the minimum routing path. Since the grid scale is relative small due to limited number of blocks in ESL floorplan, we prefer maze routing algorithm since it provides less wire length.

5.3.3 Adjusted Hadlock Algorithm for 3D global routing

In this work, we develop a C++ based routing tool by using Hadlock algorithm to perform global routing. Hadlock algorithm is a maze routing algorithm which labels the current exploring point by the detour number instead of the distance from the source. In this way, the search can prefer the direction toward the target rather than the direction away from the target. Thus search time is reduced compared to [52] while still being able to provide minimum routing path. And we add 3D routing option on top of that. Fig. 5.5 shows the pseudo code for our algorithm.
1.  \(\text{plist} = \text{source}; \text{nlist} = \text{EMPTY}; \text{detour} = 0; \text{path exists} = \text{FALSE};\)

2.  \(\text{while \ plist \ is \ not \ EMPTY \ do}\)

3.  \(\text{for \ each \ vertex \ vi \ in \ plist \ do}\)

4.  \(\text{for \ all \ vertices \ vj \ neighboring \ vi \ do}\)

5.  \(\text{if \ } B[vj] = \text{UNBLOCKED \ then}\)

6.  \(D[vj] = \text{DETOUR}-\text{NUMBER}(vj); \text{INSERT (vj ,nlist)};\)

7.  \(\text{if } vj = \text{target then path exists} = \text{TRUE}; \text{exit while};\)

8.  \(\text{if \ nlist \ = \ EMPT}Y \ then\)

9.  \(\text{path exists} = \text{FALSE}; \text{exit while};\)

10.  \(\text{detour} = \text{MINIMUM}-\text{DETOUR( nlist )};\)

11.  \(\text{for \ each \ vertex \ vk \ in \ nlist \ do}\)

12.  \(\text{if } D[vk] = \text{detour \ then \ INSERT(vk ; \ plist)};\)

13.  \(\text{DELETE (nlist; \ plist)};\)

14.  \(\text{if \ path \ exists} = \text{TRUE \ then \ RETRACE (L; \ P )};\)

15.  \(\text{else \ increase \ maximum \ metal \ layer, \ repeat \ 1};\)

Figure 5.5 Pseudo code for adjusted Hadlock algorithm for global routing in 3D-IC

The UNBLOCKED function justifies if \(vj\) can be used for routing or not. It takes wire pitch and TSV capacity as input, if any of these reach the limit, the vertex is blocked. DETOUR measures how far the current vertex from target. In 3D routing, we ensure that only route wire across tier to reduce DETOUR to minimize TSV number.
The global routing tool can read the layout generated by Pathfinder3D, using the port information to set up the grid. Each grid node has limited capacity for wires and TSVs. The total wire length, maximum wire length and wire distribution is generated after routing.

### 5.3.4 Power estimation for global wires

For global wires between blocks, repeaters are inserted to reduce the wire latency. However, repeaters increase the power consumption. In that case, estimate inserted buffer based on wire routing result is important for power estimation. The number of repeaters and the total capacitance is estimated based on [56]. Assume a long wire with length L, the optimum number of inserted buffer can be estimated by using unit resistance and capacitance of wire and repeater. We extrapolate the resistance and capacitance of unit wire and repeater from ITRS. The wire resistance and capacitance are set to 1.5ohm/um and 15.8fF/um, to match the data of metal-6 in 45nm, which is typical a global wire metal layer.

\[ m_{\text{opt}} = L \sqrt{\frac{0.38rc}{0.69R_d C_d (\gamma + 1)}} \]  \hspace{1cm} (1)

### 5.4 Summary

Physical-level analysis is the key feature of pathfinding flow. The challenge of integrating physical-level analysis into system-level design flow is to find out the appropriate level of abstraction so that it could easily link with TLM simulator.
In this chapter, we introduce the physical-level analysis framework used in this work. The ESL floorplan centric flow provides simple interface to system-level flow and simplifies system description translation between system-level and physical-level.

Global routing analysis predicts wire length and distribution which is an important cost factor for 3DIC. We develop a global routing tool integrated with Pathfinder3D. The routing tool estimates global wire length, distribution and TSV number for designers to evaluate their design. Taking ESL floorplan as the input with several simple parameters, this routing tool generates wire estimation results while totally avoid detailed design netlist.
Chapter 6

Case Study

6.1 Introduction

To illustrate the pathfinding flow, a target system is required to perform the physically aware system design exploration and evaluation at early design stage. The target system should contain both architectural-level and physical-level parameters and provide the option to go vertical, showing the trade-offs between 2D and 3D integration.

In this chapter, we present two target systems as the case studies to illustrate our pathfinding flow. The first one is a multi-core system including cores, caches and on-chip interconnect. The large design space in multi-core system makes it a good research topic for computer architects. And it is also a good candidate to implement 3D technology since the multi-core system can take the advantage of less global wires and higher density in 3D-IC. In this study, we will focus on how the high-level parameters affect the physical-level metric by changing the L2 cache size and running different applications. Meanwhile, the comparison between 2D and 3D floorplaning is shown, which provides a fast approach to compare temperature and global wire length with ESL floorplan.

The second case is a 64-node Network-on-chip. NoC is more complex compared to multi-core at system-level – more cores, varies network topologies and stacking choices. The
complexity makes the NoC performance more sensitive with physical floorplan. On one side, different network topologies result different on-chip interconnect, which largely affects the power of global wires. On the other side, different global wire power and temperature will affect the architectural-level choices especially for NoCs with 3D integration. Thus providing early-stage physically aware evaluation is necessary to get power/performance feedback from physical-level and adjust the architectures. In this work, we focus on the topologies and switching schemes of NoCs and show how the temperature and routing analysis affect architecture choices.

6.2 Multi-core case

6.2.1 System Overview

In this case study, a quad-core system is presented as Figure 6.1 shown, which contains 4 cores, 4 L2 cache banks and a crossbar that connect core and cache together. Each core has an exclusive L1 Cache, including instruction and data parts. Since there is no core model in GEMS framework, we use 1-wide FabScalar out-of-order core [57]. A detailed FabScalar C++ model is incorporated in our TLM simulator to get the instruction level simulation result. A synthesizable RTL model of FabScalar can be use to build the power model.
The GEMS framework includes a cache hierarchy for cache behavior simulation. And the related CACTI [45] model is used to get power and area estimation. We assume there is 16KB L1 data and instruction cache for each core. And the L2 Cache is divided into 4 banks, which are shared by all cores. The bank size is from 128KB to 4096KB to test the impact of cache size on performance and power. A crossbar based interconnect is implemented to allow parallel transactions between core (L1) and L2 banks. The L2 cache connects to the off-chip memory which is not implemented in detail since we focus on the on-chip part. We assume a fixed latency (100-cycle) for the L2 to memory transactions to simulate the off-chip delay.
In order to capture the application dependent effect, we run different benchmarks from SPEC200. To illustrate the impact of benchmarks on multi-core system with different L2 bank sizes, we select two benchmarks: MCF and BZIP. The BZIP benchmark has relative low cache miss rate which means it tend to read/write the same or adjacent memory address, while the MCF benchmark requires large cache sizes to reduce the miss rate. Since MCF is a large-scale minimum cost problem which contains lots of “network nodes”, the compulsory miss (cache miss due to first time read from memory for certain address block) is much higher than BZIP. During simulation, we run one billion instructions for each benchmark with a skip of 2 to 3 billion instructions at the beginning to ensure both benchmarks have already skip their “set up” phase and perform correct properties.

6.2.2 Core Model Integration and Multiprogram Support

To incorporate the FabScalar core model into our TLM framework, several changes need to make. As figure 6.2 shows, the core model is attached to the GEMS framework. The core model contains a function-level simulator and a cycle-accurate simulator. The function-level simulator reads and executes pre-compiled benchmarks and generates C++ level instructions feeding to the cycle-accurate simulator via debug buffers. A TLM wrapper is built to connect the cycle-accurate instruction-level simulator and GEMS framework by using a function interface. Memory requests are generated in the TLM wrapper for all memory operation instructions (Load/Store) and sent through the function interface. Thus, we model accurate cache latency by using GEMS cache hierarchy.
There are two advantages to do this: 1) we do not need to hard-coded the detailed core model with our simulation framework, which gives lots of flexibilities to switch core model and reduces the amount effort to build up system. 2) We separate timing and functional part so that speeds up the total simulation speed.

![Figure 6.2 Attaching detailed core model to TLM simulation framework](image)

To support multi-core system, two problems need to be solved in this simulation framework. First, the FabScalar core model use independent virtual memory, which may cause memory access ambiguity in the case of shared L2. Second, the L2 cache bank allocation is based on address in GEMS framework. Since we run a small part of benchmarks, we should ensure that all L2 bank are utilized. To address these problems, a simple TLB is implemented to convert virtual memory address used in the core model to physical memory.
address used in GEMS cache hierarchy. As fig 6.3 shows, the higher bits part of virtual address and core ID are used to generate a TLB page number. The TLB is implemented by using a standard map structure. Each core ID and virtual address combination creates a TLB page number. The page number and the lower bits part of virtual address concatenate to get physical address which is used in cache hierarchy. Since the 16-bit page number is generated by using 32-bit data, it is possible that the page number get overflowed after large number of address conversion. In that case, the old map is evicted.

Figure 6.3 Simple TLB for address conversion
6.2.3 Core and Cache Power Estimation

The core power of FabScalar is characterized based on different instruction types. Due to the complexity to build fully testbench and huge simulation time to run the real benchmark at RTL, the characterized power model is used to feed in high-level simulation framework. We get the power consumption per instruction for each core pipeline stage as Table 6.1 shown. By using the instruction level core simulator, we can identify the type of each incoming instruction and decide which pipeline stage it fit. For example, a load/store instructions will not fall into exesimpleint or exefloat (floating point or integer execution unit), but exeldst (calculate load/store address) and LSQ (Load/Store Queue).

<table>
<thead>
<tr>
<th>Pipeline stage</th>
<th>Energy (nJ)</th>
<th>Pipeline Stage</th>
<th>Energy (nJ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static energy</td>
<td>0.0082</td>
<td>exesimpleint</td>
<td>0.004832</td>
</tr>
<tr>
<td>Clocktree</td>
<td>0.00687577</td>
<td>execomplexint</td>
<td>0.231204</td>
</tr>
<tr>
<td>fetch1</td>
<td>0.114874</td>
<td>exefloat</td>
<td>0.231204</td>
</tr>
<tr>
<td>fetch2</td>
<td>0.012061</td>
<td>execntrl</td>
<td>0.004932</td>
</tr>
<tr>
<td>decode</td>
<td>0.182172</td>
<td>exeldst</td>
<td>0.004932</td>
</tr>
<tr>
<td>rename</td>
<td>0.052226</td>
<td>LSQ</td>
<td>0.00743366</td>
</tr>
<tr>
<td>dispatch</td>
<td>0.183945</td>
<td>commit</td>
<td>0.0744157</td>
</tr>
<tr>
<td>issue</td>
<td>0.0808791</td>
<td>pipereg</td>
<td>0.158406</td>
</tr>
<tr>
<td>regread</td>
<td>0.176005</td>
<td>bypass</td>
<td>0.002244</td>
</tr>
</tbody>
</table>
The cache model used in this experiment is from CACTI 6.5, which gives static power and read access energy used to calculate dynamic power. We set the data array of caches “commercial model” while all other parts are “itrs-hp” to maximize performance. Since most of the cache is data array, it gives good power consumption and still keeps high speed.

Table 6.2 Cache model parameters for power and area estimation in CACTI

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache Bank Size</td>
<td>128KB to 4096KB</td>
</tr>
<tr>
<td>Block Size</td>
<td>64-bit</td>
</tr>
<tr>
<td>Associativity</td>
<td>8</td>
</tr>
<tr>
<td>-read-write port</td>
<td>1 (shared)</td>
</tr>
<tr>
<td>Input/output bus width</td>
<td>64-bit</td>
</tr>
<tr>
<td>Technology</td>
<td>45nm</td>
</tr>
</tbody>
</table>

6.2.4 Transaction-level simulation result

Before running the benchmark and get power/performance result, we first try to find out the appropriate cache size range. We set L2 cache from 128KB/bank to 4096KB/bank and run 2 benchmarks to record the cache miss, the result is shown in table 6.3.
Table 6.3 L2 miss rate of quad-core system running MCF and BZIP benchmark with different L2 size

<table>
<thead>
<tr>
<th></th>
<th>MCF</th>
<th>BZIP</th>
<th>Hybrid</th>
</tr>
</thead>
<tbody>
<tr>
<td>128KB</td>
<td>70.3%</td>
<td>47.0%</td>
<td>55.3%</td>
</tr>
<tr>
<td>256KB</td>
<td>66.5%</td>
<td>32.8%</td>
<td>51.6%</td>
</tr>
<tr>
<td>512KB</td>
<td>65.3%</td>
<td>21.8%</td>
<td>48.3%</td>
</tr>
<tr>
<td>1024KB</td>
<td>63.6%</td>
<td>11.4%</td>
<td>38.1%</td>
</tr>
<tr>
<td>2048KB</td>
<td>49.9%</td>
<td>3.14%</td>
<td>9.47%</td>
</tr>
<tr>
<td>4096KB</td>
<td>4.5%</td>
<td>2.36%</td>
<td>3.69%</td>
</tr>
</tbody>
</table>

We can find that, for BZIP benchmark, the cache miss rate reduced linearly as the cache size increasing while the MCF does not. The reason is MCF requires large number of memory access and the access address are not tend to repeat often, in that case, small cache cannot hold the useful data until next read/write coming. However, such a large L2 is not practical in real case, noticed that we have 4 L2 banks, 16MB L2 in total is too large for current technology. From the experiment above, 2MB L2 bank and 4MB L2 bank are good for BZIP and MCF benchmark without taking power/performance into account.

We run the full simulation for the 2 benchmarks and measure the performance and power for each configuration. In a multi-core system, the performance is measured by Cycle per Instruction (CPI), which describes how fast the system runs a benchmark. Power is characterized from characterized model (cores and routers) as described before and analytical model (cache). The result is shown in Figure 6.4 below. The result shows the power consumption increasing a lot with cache size since the leakage of L2 is significant. In that
case, small L2 bank size (128KB) configuration achieves best power efficiency though the performance is low, and 1024KB L2 bank is balanced size for both benchmarks.

![Figure 6.4 CPI vs. power for 4-core case running BZIP and MCF benchmarks](image)

**6.2.5 Physical-level simulation result**

The physical level analysis includes global wire length estimation and temperature analysis with different floorplans. And we compare 2D floorplans against 3D ones to see how much benefit we can get by going to vertical and how much temperature penalty for that change. The floorplan of 2D and 3D case is shown in figure 6.5, each core is 2mm x 1.6mm
while the L2 is 2mm x 2.4mm for each bank (128KB case), a centralized router in the middle which connect all cores and L2 banks. In 3D case, we try to move the core & cache to avoid high power block over high power block case, so that to reduce the temperature on tier two which is far from heat sink.

Figure 6.6 shows the temperature map of 4-core system with 256KB L2 cache, the highest temperature rise is 16.15K, on the corner of the chip. Since the heat cannot spread out from the edge of the chip due to package, and the high power part (core) is on the corner of the chip too. We run the same analysis for both 256K and 1024K L2 bank for 2D and 3D configuration and perform wire length estimation for both cases shown in table 6.4.

![Physical floorplan for 4-core system with 1-tier and 2-tier configuration](image-url)
Figure 6.6 Temperature distribution of 4-core system with 256KB L2 bank

From the table we can see that 3D floorplan largely reduces the global wire length, including total wire length and maximum global wire length. Compared with 2D and 3D configurations, 3D floorplan can reduce the wire length by 20% - 50%. That is one of the major benefit when people consider to make chip 3D. Reducing total wire length means less power consumption on global wires and more routing resource available. Reducing the maximum wire length means shorter critical path at block level. In 3D configuration, cores and caches are closer to the router in the middle of the chip, and the 2-tier floorplan gives more routing area for global wires. We also notice that the wire length reduction varies with different L2 size. The reason is large L2 bank takes more area than small ones, approximately 8x since the area is proportional with the size. In order to not make the core-over-core case happen, cores in tier 2 are far away from each other, which increase the wire
length. Moreover, larger chip area provides more routing resource which makes the wire length reduction not that significant as small chip case.

Table 6.4 2D and 3D temperature and global wire analysis result for quad-core case with (a) 256KB and (b) 1024KB L2 bank

<table>
<thead>
<tr>
<th>L2 Bank</th>
<th>Total Wire length (mm)</th>
<th>Maximum Wire length (um)</th>
<th>Maximum Temperature rise(K)</th>
</tr>
</thead>
<tbody>
<tr>
<td>256KB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2D</td>
<td>12111</td>
<td>3775</td>
<td>16.15</td>
</tr>
<tr>
<td>3D</td>
<td>6406</td>
<td>2524</td>
<td>21.7</td>
</tr>
</tbody>
</table>

(a)

<table>
<thead>
<tr>
<th>L2 Bank</th>
<th>Total Wire length (mm)</th>
<th>Maximum Wire length (um)</th>
<th>Maximum Temperature rise(K)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024KB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2D</td>
<td>24217</td>
<td>8996</td>
<td>15.6</td>
</tr>
<tr>
<td>3D</td>
<td>19230</td>
<td>6556</td>
<td>32.0</td>
</tr>
</tbody>
</table>

(b)

The temperature penalty for 3D floorplan is about 5 to 15 K depends on the floorplan. We can see that with small L2 bank size, the temperature rise for 3D is less than the large L2 case. That falls into 2 reasons: 1) Larger L2 has higher power consumption, especially for leakage. We don't have thermal issue for 2D case since it also consumes more area. But in 3D the high power element far from heat-sink creates the high temperature rise. 2) high power element stacking with each other leads to higher temperature rise. Although we
manage to avoid core-over-core case, the cache-over-core floorplan still has the same problem compared to the small cache case.

Figure 6.7 Wire length distribution comparison between 2D and 3D floorplan for (a) 256KB L2 bank and (b) 1024KB L2 bank
Figure 6.7 shows the detailed wire length distribution for the two L2 bank size cases. From that figure, we can see the 3D floorplan redistributes the global wires, reduces the number of long wires. For 256KB L2 bank case, the redistribution removes wires longer than 3000um and reduces total wire length by 50%. For 1024KB L2 bank case, the number of wires that longer than 5000um are significantly reduced in 3D floorplan. Reducing the number of long wires can help reduce the critical path and the number of inserted buffer, which contributes the link power.

6.3 NoC test case

Compared to multi-core systems, NoCs provide more bandwidth of the inter-PE communications which is highly required for parallel applications. Since NoCs contain more PEs than multi-core, usually smaller and simpler PEs are implemented instead of generic processors to reduce the chip power and area consumption. NoCs are benefit from the capability of distributing large task to achieve high system performance in terms of bandwidth with low operation frequency of PEs. In contrast, due to large number of on-chip PEs, the interconnect part is much more complex than multi-core. In a quad-core system, cores and caches are connected through a single crossbar or shared bus. However, such structure is not fit for NoCs. The critical path delay and area increases exponentially as the number of PE increasing. Instead, indirect interconnect is implemented in NoCs to link all PEs, providing a scalable network.

With stacking multiple tiers, it is possible to integrate many PEs on a single chip with small area budget. However the temperature rise of chip may exceed the thermal budget. And
the complex interconnection structure with 2D and 3D floorplans may affect the global wire result. In this section, we will examine different NoCs with 2D and 3D topologies to show the trade-offs of power, performance, temperature and global routing result.

### 6.3.1 System Overview

In this study, a 64-node NoC is examined. Each node contains a simple core to send and receive data and a router that connects with other nodes. In order to reduce the modeling effort, we focus on the on-chip interconnection part. We explore two important architectural parameters for NoC: topologies and switching schemes. We select three types of topologies: Mesh, Flattened butterfly and Ring, combining with two switching schemes: packet switching and circuit switching. Packing switching scheme is implemented with Mesh and Flattened butterfly topologies while circuit switching scheme is implemented with Ring topology.

For packet switching, a fully pipelined router is used which is divided into four parts: *input queue, decoder, arbiter* and *crossbar*. Thus we get a four-stage-pipeline as Fig. 6.8 shows. An incoming transaction is stored in the input queue, which is implemented as a First-In-First-Out (FIFO). The decoder parses the header of the transaction and figures out the right output port for that. Since most of the NoC nodes are connected indirectly, a routing algorithm is needed. We use XY routing with minimum path to avoid deadlock [58]. Then the arbiter handles the traffic confliction. In this work, we use round-robin priority in arbiter. Each input port is assigned a priority, the highest priority port can transfer data first, and others will be stalled until it has finished. Once the transfer finished, the priority of the
corresponding input port would be reset. The crossbar part has no control logic but just data transfer. Different from shared-bus, crossbar allows parallel transfer as long as they do not interact with each other. For example, input port #1 can send data to output port #2 while input port #2 transferring data to output #1.

The router structure and routing mechanism need to change for circuit-switching schemes. We follow the 3 phase protocol for circuit-switching scheme: connection request, data transfer and connection release.

In connection request phase, the core sends out a request to the centralized arbiter via the router, and waits for a grant signal. The arbiter receives and grants requests as long as no contention and conflicts occur. Similar to the arbitration block in the packet-switched network, the centralized arbiter grants one request and cancel other conflict requests. However, rather than reserving the output port, the centralized arbiter will reserve the whole path and hold other conflicting requests. Fig. 6.9 shows the case of four concurrent
transactions. To simplify routing, we ensure that: (a) All routers use minimum path. The largest number of hops for any transaction is $P/2$, where $P$ is the number of cores. (b) Transactions cannot change their channel in the network. Thus, the arbiter can maintain a bit vector to monitor which part of the channel is available and which part is used. Once a request is granted, the IP block sends out the transaction and enters the data transfer phase. Since the path has already been reserved by the arbiter, the control bits for the crossbar are set for the transaction and ensure that the transaction can be transferred to the destination without blocking. When the tail flit of the transaction reaches the destination, the router sends an ACK signal to the arbiter and enters the connection release phase. The arbiter then can reset the relative bit vector and allocate the channel to other requests.

Figure 6.9 Parallel transactions in Ring topology
We examine both 2D and 3D NoC for the above topologies with corresponding switching schemes. Fig. 6.10 (a) shows a 2D mesh topology with 8 columns and 8 rows, noted as 8X8 Mesh. Once the topology goes to 3D, extra nodes are built in different tiers. Fig. 6.10 (b) shows the 4-tier version of Mesh (4X4X4). We follow the same extension approach for Ring topology. In that case, we have multiple centralized arbiters for the 3D ring stack. Since we don't have rows and columns, we use “Ring-sizeXtier” to represent 2D and 3D rings. One thing should be noticed is in 3D topologies, all routers have 1 to 2 more port which may increase the power and area.
Figure 6.10 (a) 8X8 Mesh topology and (b) 4X4X4 Mesh topology
6.3.2 Core and Interconnect Power Estimation

Although we do not apply real benchmarks on NoCs, we still need a core model for power and area estimation. Here the OpenSparc T2 inorder core is selected in this study [59]. OpenSparc T2 is an open-source processor with a complete synthesizable RTL netlist so that can be used to get the power number and area. It is not annotated due to the huge amount of effort to build complete test bench for a commercial core, but still good estimation for the physical analysis such as thermal and global routing in this study. T2 core has a 16KB L1 instruction cache and 9K data cache, which is evaluated with CACTI in this study.

For the interconnect part, we use the flow described in chapter 4 to get power and area. The area and power listed in table 6.5 shows that there are differences for different router. That because: 1) Routers have different radix for different topologies. The Flattened butterfly topology requires higher radix router than Mesh and Ring topologies, so as power and area. 3D topology also requires two more input and output ports to connect other nodes in different tiers. 2) Ring topology does not require buffers in network, so that we take the input buffer and decoder stage out for this topology. Without buffers, the power and area reduce. The power number listed in table 6.5 with the assumption of 50% injection rate.
Table 6.5 Power and area estimation for elements used in NoC case

<table>
<thead>
<tr>
<th>Element</th>
<th>Area (mm²)</th>
<th>Power (mW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core</td>
<td>0.61</td>
<td>49.8</td>
</tr>
<tr>
<td>L1 Cache</td>
<td>0.10</td>
<td>46.4</td>
</tr>
<tr>
<td>Router (Mesh)</td>
<td>0.084</td>
<td>8.0</td>
</tr>
<tr>
<td>Router (Ring)</td>
<td>0.018</td>
<td>2.5</td>
</tr>
<tr>
<td>Router (FBFLY)</td>
<td>0.326</td>
<td>31.7</td>
</tr>
<tr>
<td>Router (Mesh 3D)</td>
<td>0.10</td>
<td>16.4</td>
</tr>
<tr>
<td>Router (Ring 3D)</td>
<td>0.029</td>
<td>4.7</td>
</tr>
<tr>
<td>Router (FBFLY 3D)</td>
<td>0.193</td>
<td>19.4</td>
</tr>
</tbody>
</table>

6.3.3 Transaction-level simulation result & analysis

In the transaction-level simulation, we assume each node keep sending transaction to other nodes. To speed up the simulation, all transactions are assigned random destination with a uniform distribution to ensure that all network nodes receive data. The multi-flit feature is used in our NoC, each transaction contains 1 head flit, 1 tail flit and 14 data flit. The 16-flit transaction requires more buffers for packet-switched NoCs and increases bandwidth for circuit-switched NoCs.

The aggregated bandwidth is used to measure system performance of NoCs, which represents the maximum processing capability of the network. We consider bandwidth rather than average transaction latency because: 1. Bandwidth directly shows how many transaction
the network can process. 2. The inter-PE communication is much slower than the communication between cores and their local cache due to the protocols and traffic contention. An effective data transfer can take more than 100 cycles due to coherent protocol and other handshake mechanism. In that case latency doesn't reflect the actual processing ability. Eq. 6.1 shows the definition of aggregated bandwidth.

\[
BW = \frac{N_{\text{flit}}}{N_{\text{cycle}}} \times f_{\text{clk}} \times n
\]  

(6.1)

The bandwidth is calculated based on the total flit number \((N_{\text{flit}})\) processed by the network and the total processing time \((N_{\text{cycle}})\) in terms of cycle count. \(f_{\text{clk}}\) is the clock frequency in GHz and \(n\) is the flit size in byte. Thus, we get the aggregate bandwidth in terms of GB/s. If we fix the flit size, we can see that different type of networks will affect both flit per cycle and clock speed and achieve different bandwidth. The bandwidth above is also called actual bandwidth since the total number of flit is from simulation which is limited by various factors. For instance, the running application may not have that much data to change between PEs, making the network idle most of the time. Or the heavy routing congestion and limited internal buffer cannot guarantee all transactions start/finish on time. We use the ratio of actual bandwidth and theoretical bandwidth to represent how far the NoC from the ideal case. The theoretical bandwidth is defined as Eq. 6.2. \(N_{\text{port}}\) is the number of input (or output) port for each NoC node. The equation describes a case that all nodes can send/receive data all the time.

\[
BW = \sum (N_{\text{port}} \times n) \times f_{\text{clk}}
\]  

(6.2)
Fig. 6.11 shows the pareto-efficiency for different topologies. From the figure we can observe several outcomes: (1) Single-crossbar topology cannot provide enough bandwidth when the number of cores is greater than 16. In a high radix router, the clock speed reduces greatly, and the bandwidth reduces as shown in Eq. 6.1. (2) Circuit-switched network is a low power design compared to packet-switched network. Routers without queues reduce power consumption. To compensate the overhead of circuit-switch feature, a ring topology with centralized arbiter is implemented. Since all the requests are handled by the arbiter, the connection request and release is minimized. (3) 3D topology can improve bandwidth and reduce power. Compared to 2D mesh, 3D mesh uses two additional ports per router, providing more links in the network and further distributing the contention. For ring topology, stacked ring with small size avoids long hop transactions which have large overhead for establishing and releasing connections. For the flattened butterfly, bandwidth increases because the 3D topology decreases the router radix. When the router radix goes high, the arbitration block becomes the critical path, and the critical path delay is sensitive to number of port since the time complexity of the arbitration algorithm is $O(N^2)$. The 3D topology can reduce the router radix and then increase the clock speed. Although the flit/cycle decreases a little, the aggregate bandwidth increases in that way. 3D topology can also reduce link power for all types of networks since the inter-tier wire length is less than 1% compared to the inter-core global wire length.
To further explore the benefit of 3D topologies, we look at the energy efficiency for a 64-node MESH with 1-tier (8X8X1), 2-tier (8X4X2) and 4-tier (4X4X4). The power efficiency measures how much bandwidth the network can achieve with certain amount of power, defined by total bandwidth divided by total simulation cycles. It shows that we can achieve 12% more power efficiency by going from 2D to two-tier 3D and 50% more power efficiency by going from 2D to four-tier 3D. Most of the benefits in this example come from the reduced number of hops in the higher-radix network topology rather than 3D integration.
If we were to compare a 4x4x4 network in a single tier, we would likely see a more modest improvement compared to an 8x8x1 network.

![Figure 6.12 Power efficiency of a 64-node NoC with different 3D partitioning.](image)

### 6.3.4 Physical-level simulation result & analysis

In NoC, all the nodes are identical in shape, and the size varies from 0.7mm x 0.7mm to 1.0mm x 1.0mm depends on different topologies. The temperature penalty in 3D case is more significant compared to multi-core case because 1) we have 4-tier design as 3D case which makes the heat hard to dissipate in top tier. 2) all nodes consume power so there is no “low
power element” and stacking cores on top of cores cannot be avoided. Fig. 6.13 shows the temperature distribution for 4X4X4 Mesh topologies.

Figure 6.13 Temperature distribution for 4X4X4 Mesh topology

Compared to the multi-core case, 3D topologies do not reduce the maximum global wire length and keep the same total wire length as 2D case. The main reason is nodes in 3D topologies have more port and wires, which increases total number of wires. And the smaller area causes more routing congestions which makes the maximum wire length increased. But the redistribution of wires helps to reduce long global wires as Fig. 6.14 shows. In 2D mesh topology, most of the wires fall into 600um range while in 3D case wires distribute from 100um to 900um. That shows an interesting result which 3D floorplan cannot reduce the wire length with complex floorplan as much as simple one like the quad-core case we shown above. The quad-core case shows at least 20% wire length reduction, but less than 10% with NoC for best cases. Since we put more and more blocks on a single chip, such early-stage physical-level estimation is necessary. However, if we eliminate the impact of router radix by applying the same logical topology on both 2D and 3D floorplan, we can see the significant
global wire reduction in 3D floorplan as fig. 6.15 shows. The total wire length reduces by 70% due to the long wires in the 2D floorplan.

Figure 6.14 Global wire distribution comparison between 8X8 and 4X4X4 Mesh NoCs
Figure 6.15 Global wire distribution of 4X4X4 Mesh NoC for 1-tier and 4-tier cases

Table 6.6 2D and 3D temperature and global wire result for Mesh and Ring topologies

<table>
<thead>
<tr>
<th></th>
<th>Total Wire length (mm)</th>
<th>Maximum Wire length (um)</th>
<th>Maximum Temperature rise(K)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2D MESH</td>
<td>12179</td>
<td>664</td>
<td>7.85</td>
</tr>
<tr>
<td>3D MESH</td>
<td>11993</td>
<td>942</td>
<td>73.9</td>
</tr>
<tr>
<td>2D RING</td>
<td>6862</td>
<td>649</td>
<td>6.21</td>
</tr>
<tr>
<td>3D RING</td>
<td>9768</td>
<td>896</td>
<td>57.8</td>
</tr>
</tbody>
</table>

Figure 6.16 shows the temperature result for different topologies. We can see that a huge temperature rise with 3D topology, especially for Flattened butterfly. In that case, the power issue is amplified here which force designer to consider low power design (such as Ring topology in this case) rather than performance. If we put a temperature limitation in the
power/performance result, we can see that the former “best” points only available with large thermal budget (90K temperature rise) like figure shows. In that case, if the design is used for low-power application, a 2D configuration with 10K temperature rise limit is the best choice. Figure 6.17 shows wire length result in the same form as 6.16, it is clear that flattened-butterfly topology has much longer wires compared to other topologies due to high router radix and longer connections since the router needs to connection all other routers in the same row and column. Ring topologies get least wire length due to low router radix as well as small area, which gives the advantages in terms of cost. From 6.16 and 6.17 we can see that after taking cost factor (temperature and wire length) into account, the 3D ring topology becomes one of the best choice overall especially if there is a hard requirement for temperature. By adding physical-level analysis into high-level system evaluation, we can see the affection of design choices. Since it is time consuming to implement detailed design, the high-level physical analysis is important for designer to evaluate among various architectures and make the decision fast and then move on to detailed design.
Figure 6.16 Pareto-efficiency for 64-node NoC with temperature limitation
6.4 Summary

In this chapter, two test cases are presented to demonstrate the pathfinding flow and the necessities of physical-level analysis during early design stage. Both of the two test cases are complex on-chip system with different processing elements, which are good candidates for 3D integration. We add temperature and routing analysis for the 2D and 3D configuration for each case, providing trail result to help system designer evaluate different architectures. Without detailed RTL design, the pathfinding flow can still provide physical information and add valuable comparison result across different design configurations.
In the multi-core test case, we show that how much benefit can be get from 3D integration. We set up a quad-core system with different L2 bank size, running ALU incentive benchmark (BZIP) and memory operation incentive benchmark (MCF) to find out a good L2 configuration for the system. We integrate a detailed core model into the TLM simulator to get accurate latency from memory operation instructions. Then we create an ESL floorplan for 2D and 3D configurations. By using the power and area information from TLM, we estimate the temperature and global wire distribution without detailed RTL netlist, and shows 3D floorplan can reduce the global wire length and redistribute long wires to minimize critical path as well as inserted buffer, at a cost of 10-15K temperature rises.

In the Network-on-Chip test case, two types of networks with various topologies are compared together. The packet-switched and circuit-switched NoCs incorporate different router architectures as well as routing algorithm. And three types of topologies are implement which enlarge the design space. Our pathfinding flow shows the approach to build and compare these different NoCs together fast at system-level. Meanwhile, adding temperature and routing analysis result to the pareto-efficiency figure shows the different best design choice under cost constraints. Ring topology with circuit-switching scheme shows more value with the temperature and global wire length limitation especially in 3D floorplan. With those analysis results, designers can reduce the design space and build detailed design based on several good candidates according to the evaluation.
Chapter 7

Conclusion

Complex CMP/NoC and 3D-IC technology are two solutions for the design challenges of today’s digital IC. Those two solutions are located at system-level and physical-level respectively but for the same objective. Furthermore, the two solutions are interactive with each other. Physical-level configurations such as number of tiers and floorplanning affect the architectural-level design choices such as number of cores, cache size and network topologies. In that case, building a pathfinding flow integrating both system-level and physical-level analysis can speed up the design evaluation while avoiding detailed RTL design netlist at the early design stage.

In this work, we presented a pathfinding flow for system-level design evaluation. It integrates a TLM-centric front-end part and an ESL-centric back-end part. Various systems can be simulated by using a transaction-level simulator to reduce the modeling effort. The TLM simulator is implemented in the GEMS framework. We extended the simple interconnection part to support different type of NoCs. To illustrate the run-time dependent feature, we integrate a detailed core model to execute pre-compiled benchmarks and generate related instructions in debug mode. The TLM simulator provides system power and performance to the back-end part. In order to avoid the requirement of RTL netlist, an ESL-centric analysis engine Pathfinder3D is developed for physical-level analysis. In
Pathfinder3D, floorplan are created with marcocells which contain geometric information. Then the block-level layout is generated by using a texture floorplan description and OpenAccess API. A global routing tool is developed based on this ESL floorplan provided by Pathfinder3D. By using this tool, system level designer can perform wire length and distribution estimation without knowing the details gate-level design. It is easy to generate an ESL floorplan by using the system-level description in the front-end part.

In a pathfinding flow, power model is the link between system-level and physical-level analysis. The level of abstraction and the estimation method and format are key challenges for high-level power model. In this work, we presented a power model characterization flow which specifies an approach to generate power model with good accuracy yet provide simple interface to TLM simulator.

Two test cases are presented to demonstrate the proposed pathfinding flow. In the quad-core case, we evaluate different benchmarks and different floorplans for 2D and 3D schemes, showing the pathfinding flow can explore the design space at both high-level and physical-level. In the NoC case, a comparison among different topologies, switching schemes and floorplans helps designers to make architectural decision. The trade-offs between performance and cost can be evaluated fast at system-level by adding physical-level metrics such as temperature and global wire length.

3D-IC and multi-core/network-on-chip systems necessitate the use of pathfinding flow for early-stage design exploration and evaluation. In this work, we show a possible approach for such evaluation flow to address these design challenges.
REFERENCES


[13] Tezzaron Octopus 8-Port DRAM for Die-Stack Applications


[51] Algorithms for VLSI Physical Design Automation


[59] OpenSPARC T2, Available at: http://www.oracle.com/technetwork/systems/opensparc/opensparc-t2-page-1446157.html
APPENDICES
Appendix A

Pathfinder3D

In this appendix, we show the format used in Pathfinder3D to represent ESL floorplan. To generate a layout and run thermal and global routing analysis, users need to specify the design file, stimulus file and interface file.

A.1 Design File Format

Figure A.1 shows the design file format of Pathfinder3D, which contains macrocells and instances. The macrocells specify several attributes for “standard blocks” used in the ESL floorplan. It contains geometric parameters such as height, width and tiers to help identify the actual shape of the macrocell. The block layers specify routing resource related to that macrocell. When the layout is generated, several metal layers are marked unavailable for routing within the shape of macrocell. Then the routing tool will read the layout and perform global routing under the constraint of available routing area. Sockets specify the port of the macrocells. It contains port name, port type and port direction. The port type is defined in interface file with wire details, which help to generate pins in the layout and passing the location information to the routing tool to perform global routing.

The instances specify actual ESL blocks in the layout, which contains the locations and connections. Locations specify the left-bottom coordinate of the instances. And connections show how the instances connect with each others.
Figure A.1 Design file format of Pathfinder3D
A 1.2 Stimulus File Format

The stimulus file specifies the power consumption for each instance in the design file as Fig. A.2 shows. It is used for thermal analysis.

```
{"stimulus:{
  power:[
    [CA0 [1020.15e-3] ],
    [CA1 [1020.15e-3] ],
    [CB0 [1020.12e-3] ],
    [CB1 [1020.163e-3] ],
    [LA0 [241.158e-3] ],
    [LA1 [241.13e-3] ],
    [LB0 [241.158e-3] ],
    [LB1 [241.13e-3] ],
    [BUS0 [20.00e-3]]
  ]
}
```

Figure A.2 Stimulus File Format in Pathfinder 3D

A 1.3 Interface File Format

Interface file specify the socket type in detail as Fig. A.3 shows. Once we specify a socket type in the design file, it is translated into detailed wires to generate layout, the interface specify every wires in bit so that we get pin-accurate port information which can be used for global routing. With the interface file, users can easily change actual pin connections without the need to modify designs.
Figure A.3 Interface file in Pathfinder3D