Efficient Data-Transfer Schemes for On-Chip Interconnection Networks

Kenichiro Anjo

2004

Abstract

System-on-Chip (SoC) is a Large Scale Integrated circuit (LSI) that integrates various functions into a chip. SoCs have been received attentions due to the wide range of applications, ability to integrate functions, cost reduction, and performance improvement. However, according to the increase of transistor density, a methodology to design SoCs becomes very complicated and time-consuming. For ease of chip designs, SoCs are designed by reusing Intellectual Property (IP) cores that have already been designed and verified. Another approach is to use a programmable device which does not require chip fabrication. A key to success in both approaches in terms of design time, performance, and chip cost is on-chip interconnection networks, or on-chip buses and Network-on-Chips (NoCs). In this thesis, how to design an on-chip bus and an NoC efficiently is discussed, especially taking both performance and hardware cost into account.

Firstly, novel cost- and performance-efficient implementation techniques for the on-chip bus with bus wrappers are addressed. Four major features are included: (i) a wrapper interface for small latency cycles, (ii) a write buffer switching technique to optimize wrapper hardware cost and performance, (iii) a retry technique to connect fast and slow slaves to the same bus with small performance overhead and live lock avoidance, and (iv) a bit-width conversion technique to reduce hardware cost. Simulated results with a traffic assumed in an SoC show that the throughput is improved by 14 % and the Read and Write latency are reduced by 16 % and 11 %, respectively, compared with the conventional wrapper bus. Furthermore, 50 % hardware is reduced in the proposed bus for a 5-master and 7-slave configuration in a CPU-based SoC in 0.15 μm CMOS. It works at 200-MHz clock frequency and occupies 3.3-mm^2 area.

Next, a novel data transfer scheme for NoCs in programmable devices is described. A novel routing technique for achieving smaller hardware cost and higher performance is discussed. Routing information is transferred in parallel to data, which is different from the conventional packet data-transfer. This removes cycle penalty for transferring header and hardware overhead for handling packet structure. The proposed routing technique uses static analysis results of communication patterns in applications and only assigns routing labels to the pairs of communicating nodes. For reducing the required number of bits for routing information, a local label which is only valid in a channel between neighboring routers, is addressed. Local labels allow reusing a label value inside a network and results in reduction of the number of label values. The presented results show that the hardware amount for a router is reduced by 46 % from the conventional distributed routing router using global addresses.

Acknowledgement

I would like to thank my advisor, Professor Dr. Hideharu Amano for his generous support and encouragement for mostly 9 years. He kindly accepted my difficult position as a working student. And he has given me many excellent and useful ideas and advices.

I gratefully thank Professor Dr. Fumio Teraoka, Professor Dr. Tadahiro Kuroda and Associate Professor Dr. Nobuyuki Yamasaki for their overall comments to my thesis for improvement. It was quite rushing request, but they accepted and responded my request courteously.

I really appreciate my superior in NEC Electronics Corporation, Dr. Masato Motomura, for his admission of my request to become a working student. I could not achieve this work without his understanding.

I also would like to thank my ex-superiors in Silicon Systems Research Laboratories of NEC Corporation, Dr. Masakazu Yamashina, Dr. Masayuki Mizuno and Muneo Fukaishi for their advices from many aspects. With their kindness support, I learned mentality as a researcher and how to proceed researches.

I would like to thank Atsushi Okamura for his many advices to drive NECoBus project. I also leaned so many technical issues about chip designs.

I greatly thank Noriko Mizushima, Tomoharu Kajiwara, Yasuaki Kuroda, and Masafumi Ohmori for their cooperation to the NECoBus project, and for their contribution of verification environment and layout.

I really really thank Dr. Michihiro Koibuchi, Yutaka Yamada, and Akiya Jouraku for their support for Black-Bus project. They accepted and understood my difficult position as a working student, and kindly helped many technical details about the researches on Network-on-Chip.

I would like to thank Katsumi Togawa as my co-worker. He always supported all aspects of my work.

I gratefully appreciate Hiroaki Inoue, Fukukyo Sudoh, Michitaka Okuno, Dr. Yuichiro Shibata, Dr. Noriaki Suzuki for their encouragement to my activities.

I would like to thank Assistant Professor Dr. Rizwan Bashirullah for his advice and encouragement to my activity. I luckily met him in CICC 2002, and since then, the discussion I had with him is very valuable and inspired my activity on Network-on-Chips.

I also would like to thank Dr. Adrian Ong for his friendship since we worked together in Lucent Bell Laboratories. His mind and leadership as a researcher and an engineer really inspired me.

Last, but not least, I wish to thank my family with whole my love, Tomoko and Yumei for their continued support and patience to my activity. This work is not achieved without their understandings and encouragement.

Yokohama, Japan
January 2005
Kenichiro Anjo

Contents

List of Figures

List of Tables

Chapter 1 Introduction

Semiconductor process scaling enables larger number of transistors embedded on a chip. This realizes wider range of applications and even higher performance. However, using large number of transistors requires longer period for designing and verifying circuits. Also, the required cost for chip fabrication becomes high, and is considered as an even higher risk of chip reworking. Thus, an organized and standardized method to design System-on-Chips (SoCs) in shorter turn-around-time (TAT) and a design methodology with lower bug risks become much important, as well as with small chip cost which is always expected.

There are two possible solutions to achieve shorter TAT and lower bug risks. A well-known way is reusing Intellectual Property (IP) cores, which were designed and verified already. IP cores are designed to meet a certain communication protocol so that they can be connected with an interconnection network. Then, SoCs can be constructed by simply connecting IP cores with the network. As on-chip interconnection networks for SoCs, on-chip buses [Arm99] [Ibm99] have been widely used to connect IP cores. This scheme results in shorter TAT and lower bug risks since IP cores which were designed and verified already are reused.

Another solution is using programmable devices, such as Field Programmable Gate Array (FPGA) or processor-arrays. According to the progress of CMOS process technologies, these programmable devices are becoming more cost-effective approach than fabricating Application Specific Integration Circuits (ASICs) by users' own expenses. The programmable devices have array-type architecture which embeds Configurable Logic Blocks (CLBs) or processing elements (PEs) in the shape of array and connects them by an interconnection

network. This network is called a Network-on-Chip (NoC), which is a programmable interconnection to realize any required connections within the array.

In both these solutions, the on-chip interconnection networks are key factors for success. In SoCs, the interconnection network is required to provide shorter TAT as well as smaller hardware amount to suppress chip cost. On the other hand, in programmable devices, programmable switches have been used as an interconnection network. The programmable switches determine interconnect programmability between CLBs/PEs, and providing sufficient programmability requires large hardware amount of the chip, that is expensive chip cost.

On-chip buses

An on-chip bus has been widely used as an interconnection network on a chip because its structure is simple and the number of connected nodes is easy to scale, especially when it is small. Each core has bus protocol logic, such as arbitration logic, and logic for issuing commands or responding to commands. And thus, an SoC can be simply integrated by connecting these IP cores with an on-chip bus. However, there are two major obstacles against encouraging easy reuse of IP cores in bus-connected SoCs. One problem is that an on-chip bus protocol is not an explicit rule to connect IP cores each other, because it has optional functionalities, such as split transactions, burst transfers, etc. Multiple vendors and designers create different IP cores for the same on-chip bus, but they may support different options of the bus specification. So, IP cores cannot be connected directly, and they are required to be modified so that they can communicate each other properly with an on-chip bus. The other problem is that several powerful on-chip bus specifications exist, and thus reusing IP cores between these buses are not achievable. So, each IP core is required to support multiple interfaces according to on-chip bus protocols, and this could be a burden for IP core vendors.

Wrapper-based buses

From the standpoint of IP core reusability, Virtual Socket Interface Association (VSIA) defined an interface protocol called Virtual Component Interface (VCI) [VSI01] which could be a satisfactory condition for IP cores to communicate each other, as an explicit rule for connecting cores. The idea is using bus wrappers, which bridge from the defined interface protocol to the physical on-chip bus protocol. The interface protocol is point-to-point and independent of any physical bus structures. The major motivation of using bus wrappers is separating bus protocol logic from IP cores. IP cores are designed to comply simply with a wrapper interface protocol to encapsulate complicated bus logic into bus wrappers. Bus wrappers are typically prepared in advance, and a wrapper-based bus can be generated by duplicating and connecting these wrappers.

Network-on-Chips

Programmable devices employ Network-on-Chips (NoCs) as their interconnection networks. There are various types of NoCs, such as programmable switches used in fine-grain architectures like FPGAs, or interconnection networks using network routers for processor-array type SoCs in future. There is no obvious definition of NoCs. Although in some published documents, this paper assumes NoCs do include interconnection network in FPGAs.

The programmable switch is a programmable crossbar, where required communication paths can be configured by configuration data kept inside configuration memories in each CLB/PE. Each switch is used only for a logical single communication path from a source to a destination. In a programmable device, multiple switches are prepared in a boundary of neighboring CLBs or PEs to achieve sufficient flexibility. Thus, the programmable switches tend to consume large amount of hardware.

An NoC using network routers is an interconnection for future generation of programmable devices. The targeted architecture is coarse grain, and each computation node can be microprocessors, FPGAs, processor-arrays, etc. This

NoC has borrowed interconnection schemes of System-Area-Networks (SANs) used in PC clusters or parallel computers. Routers used in this NoC handle network packets and route packets to a certain destination according to routing tags embedded in the packets or stored in routing tables.

Difficulties in on-chip interconnection networks

As on-chip buses for SoCs, wrapper-based buses look a viable approach for common use. A challenge is achieving small cost, which is always expected and required in SoC designs. The conventional wrapper-based approach uses a bus wrapper which includes FIFOs for data buffers, and thus results in large amount of hardware. A design scheme of a wrapper-based bus which can achieve practically small hardware without sacrificing performance and connectivity, becomes important.

As an NoC for programmable devices, achieving small hardware is a primary requirement. Conventional programmable switches used in current programmable devices potentially become a dominant factor in overall hardware amount. Although conventional work on router-based NoCs can be considered as a replacement, they did not focused on achieving practical hardware amount.

Overview of this research

This research studies the challenges involved in design techniques for wrapper-based buses and Network-on-Chips, especially to achieve practical cost as well as good performance.

Firstly, design techniques for wrapper-based buses are described. Novel wrapper interfaces are proposed to achieve better performance and lower cost. The interface is optimized so that entire bus protocol can be optimized for better performance. And several wrapper implementation techniques to minimize buffer resources in a wrapper-based bus are proposed. The effectiveness of these techniques is proved by simulation results and logic synthesis results. Furthermore, to show reliability and a practical example of the proposed

techniques, a CPU-based SoC which includes the designed bus is implemented in 0.15 μm CMOS processes.

Next, a data-transfer scheme targeted for programmable devices is shown. This includes a novel routing technique for router-based NoCs. This data-transfer approach does not use packet structure for avoiding hardware and performance overhead in handling packet structure. And, the routing scheme assumes static analysis of communication patterns and employs local labels for specifying destinations in networks, differently from the conventional distributed routing using a destination node address. The designed NoC router has simple structure, and thus smaller hardware as well as better performance. The effectiveness of these techniques is proved by application trace analysis, cycle-based simulation, and logic synthesis.

This thesis is organized as follows. In Chapter 2, the backgrounds of this research are described. Chapter 3 summarizes previous work of this research regarding on-chip buses and network-on-chips, and clarifying the motivation and the positioning of this thesis are clarified. In Chapter 4, the design techniques for wrapper-based buses are proposed. Chapter 5 addresses the data-transfer scheme for NoCs. Then, future work is discussed in Chapter 6 and finally, Chapter 7 concludes this research.

Chapter 2 Backgrounds

2.1. System-on-Chips (SoCs)

2.1.1. SoC generation

Advanced Complimentary Metal Oxide Semiconductor (CMOS) process technology has reached generation of sub-0.1 μm gate-length transistors. According to a report of International Technology Roadmap for Semiconductors (ITRS) for 2004 [Itr04], the gate density becomes more than 100 M transistors/cm^2 in sub-0.1 μm CMOS technologies. A chip size ranges up to several hundred mm^2, depending on chip costs required by users, and thus, embedded number of transistors is several hundred millions in a single chip. This large number of transistors allows many usable functions for consumers in a single chip, such as microprocessors, memories, circuits for application-specific functions like image processing, networking, etc. A chip which integrates multiple of these functions is called System-on-Chip (SoC) to differentiate from a legacy single-chip single-function device.

2.1.2. Key criteria in designing SoCs

In this subsection, key criteria in designing SoCs are summarized. The criteria listed here are turn-around-time (TAT), cost, performance and power.

2.1.2.1. Turn-around-time (TAT)

One of the important criteria in designing sub-0.1 μm CMOS SoCs is turn-around-time (TAT). The TAT of a chip can be defined as a period from the time to start designing an architectural concept and to the time when chips are shipped. The period comprises phases of architectural concept design, coding, verification, layout, fabrication, testing, and assembly. In the era of SoCs, all these steps have been getting complicated and each period has been becoming longer. The issues in these steps are summarized here.

Architectural design

While functions and requirements of an SoC range widely, an SoC architecture must be considered and verified from many aspects, such as throughput, latency, power consumption, cost etc. The architectural design is very important because direction and strategy of a chip are mostly defined. Once coding design starts, it requires lots of overhead to change the top-level strategy. In this phase, the detailed circuit-level operation is not really essential, where block diagrams and data-flows of the chip are more important.

Coding and verification

Once the top-level functions are mostly defined, each IP core is actually coded by designers. Hardware description is typically coded with Register Transfer Level (RTL) hardware description language (HDL), and recently with C-level hardware description language for large-scale or algorithmic designs.

After functions are coded, the design is verified with testbench. As testbench, RTL verification environment has been used, or recently testbench generation tool using dedicated software language is used for effective generation of test patterns. Verification is a time-consuming work especially for a large-scale SoC, since combinations of multiple functions are also verified as well as each function. In sub-0.1 μm CMOS generation, easing verification phase is a key issue to reduce TAT.

Also, after coding and verifying the design, the design is synthesized with logic synthesis tool into gate level. This requires AC timing requirements as limitation for the synthesis, and the tool will synthesize the design to meet the requirements.

Layout

After generating netlist in the coding phase, the design is actually laid out by place and route tools for back-end designs. The rough design flow includes power and ground wiring design, floor planning, placing cells, routing wires, and clock distribution design. Finally, the laid out data is statically analyzed using Static Timing Analyzer (STA) tool. If the timing does not meet, the coded design must be modified moving back to the coding phase.

This layout phase is also a time consuming phase for designing an SoC, because this requires complicated steps for each function block and top-level SoC layout. Since the sub-0.1 μm CMOS device has many issues to consider, such as crosstalk, static and dynamic IR drops, long wiring delays, and etc. These device issues are all verified in Design Rule Checking (DRC) and STA tools. Once one of these checks is violated, the layout should be re-tried to solve the problem, and this iteration tends to spend long design time, especially in designing large-scale SoCs.

Fabrication, chip testing, assembly

The laid out design is then taped-out, and the fabrication phase starts. The fabrication period ranges from one to several months, depending on process generations. After fabrication, chips are tested with LSI testers. Typically, during chip fabrication, LSI test patterns are prepared reusing verification patterns. After the chips are confirmed for correct operations under various environmental conditions, they are assembled into a device package.

In complicated sub-0.1 μm CMOS processes require longer period for fabrication because process steps are increased. Also, the LSI testing requires longer time to confirm correct operations since the number of functions to confirm is increased.

2.1.2.2. Cost

Another important criterion is cost. Cost is mostly classified into three types: design cost, Non-Recurring Engineering (NRE) cost and chip cost.

Design cost

Design cost is the cost which is spent in the design phase, such as human resource cost for designs and infrastructure cost. The human resource costs for coding, verification, logic synthesis, layout and chip testing must be increased to follow the increased functionalities of the device when the number of transistors is large.

The infrastructure cost includes license fees for CAD tools to design chips and computer servers required for running these tools. The CAD tools range widely, and they are tools for RTL simulation, logic synthesis, design verification, placing and routing, clock-tree synthesis, static timing analyzer, etc. There exist various licensing styles, but in some cases they are shared within a company or some universities. In those cases, fees are split by each project according to the total time of spent licenses. The infrastructure cost is increased to suppress design time when the number of transistors increases and the chip design becomes complex.

NRE cost

NRE cost is the required cost for producing Application Specific Integrated Circuits (ASICs). What is included in NRE varies up to chip vendors, but typically it includes costs for creating mask sets and fabrication processes, at least. The cost required for producing mask sets has been increasing while transistor size is getting smaller, and the required mask writing and defect inspection system become more and more complicated.

Chip cost

Chip cost is mainly determined by die cost and package cost. Die cost is mostly determined by die area, which is related with functions to embed into a

chip. The package cost is determined by number of I/Os and frequencies to achieve in these I/Os. Number of I/Os affects pin counts of a package, and data frequencies of I/Os determine package types, such as Flip-Chip BGA for high-speed data-rate or Tape BGA for average data-rate.

2.1.2.3. Performance

Performance is classified into throughput and latency cycles. Throughput is calculated by multiplying bit-widths of data and clock frequency in Hz. Thus, the throughput is expressed with the unit of bit/s or Byte/s. It is an index to show how much data is flowed in a certain boundary, such as internal boundary between function blocks or an off-chip interface.

Latency cycles are the latency required in certain accesses, such as Read or Write operations of a CPU or an I/O transaction. It is simply calculated by counting from the beginning cycle of a Read/Write command to the clock timing to complete access. The latency cycles affects CPU utilization which is degraded by the Read access latency.

2.1.2.4. Power consumption

Power consumption consists of static and dynamic power dissipation. Static power dissipation is the product of device leakage current and supply voltage. In sub-0.1 μm CMOS device, supply voltage becomes less than 1.0V, but device leakage current increases due to thin thickness of transistor gate oxide.

Dynamic power dissipation is the consumed power by switching current of CMOS transistors. The average dissipated power is proportional to the energy required to charge and discharge the circuit capacitance, and the proportional factor is equal to switching frequency. In sub-0.1 μm CMOS device, according to the increase of transistor operation frequencies and capacitive loads, the static power is increased. The source of power reduction is the transistor supply voltage, which is decreased less than or equal to 1.0V. How those factors affect depends on chip architectures.

Static power dissipation is more process dependent, since leakage currents of transistors are the dominant factor. Controlling back-bias voltage to change threshold of transistors is an example to suppress device leakage with circuit-level techniques.

2.2. IP-core based SoC

2.2.1. Concept

According to the increase of number of transistors, an SoC consists of various functional blocks. These blocks have been naturally considered for reusing, once they are developed. They are called Intellectual Property (IP) cores, and play important roles in designing SoCs, especially in reducing design and verification time.

An example block diagram of an IP-core based SoC is shown in Figure 2-1. It consists of various function blocks, such as a microprocessor, an MPEG decoder, internal memories, I/O interfaces etc. As shown in this diagram, IP cores are connected with an interconnection network that is an on-chip bus. Although the interconnection network can be certain dedicated interconnects between specific cores, the on-chip bus is more simple and easy to scale number of IP cores.

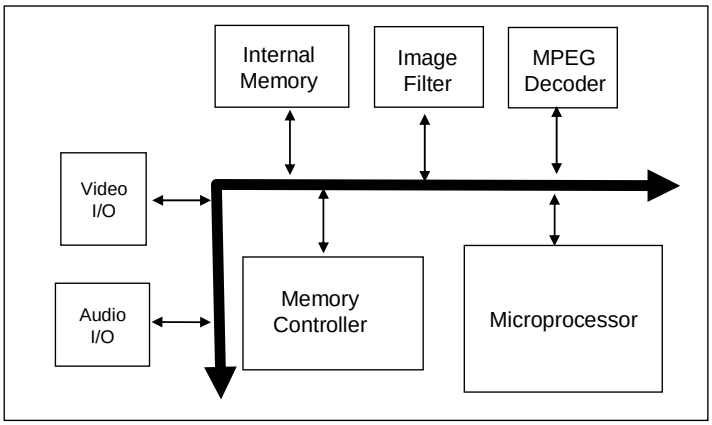


Figure 2-1 example of SoC architecture

2.2.2. Difficulties

Since the on-chip bus is a backbone of IP-based SoCs, its structure is very important to ensure IP core connectivity. Key criteria in IP-based SoCs vary up to requirements.

- An important criterion is achieving shorter TAT by reusing IP cores than developing the device from the scratch. This results in reducing design cost.
- Another is achieving good performance with small chip cost. Achieving small cost is always required in any types of designs. However, amount of hardware and degree of performance are tradeoff in most cases. Thus, the balance must be considered and chosen according to the requirement.

2.3. Programmable device

2.3.1. Concept

Chip developers design their own functions as ASICs to meet their own requirements. However, in sub-0.1 μm CMOS process generations, required costs, especially design cost and NRE cost, become large and many developers give up designing ASICs since they desire cheaper and lower-risk approaches. Those developers start designing applications on programmable devices instead of ASICs.

Programmable devices, which are Field Programmable Gate Array (FPGA), Complex Programmable Logic Device (CPLD), or processor-arrays, have been raising their market shares as alternative solutions against ASICs. Those programmable devices do not require any expensive NRE to be paid for chip fabrication vendors, and users just buy programmable devices and program the design to the chip by themselves. Furthermore, if the design includes any bugs and they are found after designing, they can change the design and re-program it to the chip. So, these programmable solutions have advantages against ASICs, from the standpoint of NRE cost.

Compared with conventional programmable solutions, such as microprocessors, Digital Signal Processors (DSPs), FPGA directly realizes hardware circuit itself. Thus, FPGA can achieve better performance than those processor-based approaches.

2.3.2. Architecture

2.3.2.1. FPGA architecture

Figure 2-2 shows an architecture example of FPGA. This is the architecture of Xilinx Virtex II series [Xil04], which can realize 40K to 8M gates in a single device. FPGA consists of Configurable Logic Block (CLB) array, which includes Look Up Table (LUT) to realize programmable gates. LUT is a mechanism to emulate gate operation, and Virtex II series have 4-input LUTs. The 4-bit input data are converted to output as 1-bit data by looking up a table. This is the emulation of 4-input complex gates. How output data and 4-bit input data are related is programmed as table data by its configuration.

Figure 2-3 shows an internal architecture of CLB. And I/Os and DLLs are not shown in this figure. CLB consists of a programmable interconnect and four Slices. The programmable interconnect is crossbar switches which can be configured to specify sets of a destination and a source CLB to communicate each other. Fast Interconnect allows fast data transmission to neighboring CLBs with low flexibility. Switch Matrix is an interconnection to communicate with other CLBs anywhere. Slice is complex of LUTs, multiplexers, and registers. Slices can be concatenated to realize cascaded operations with cascaded data input and output port, such as a shifter.

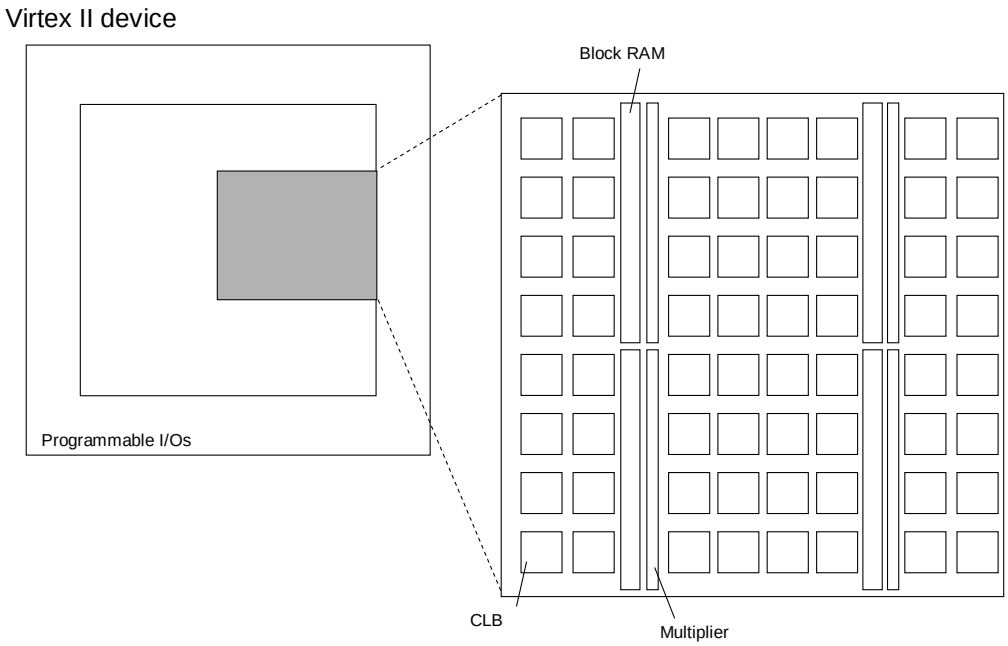


Figure 2-2 example of FPGA architecture

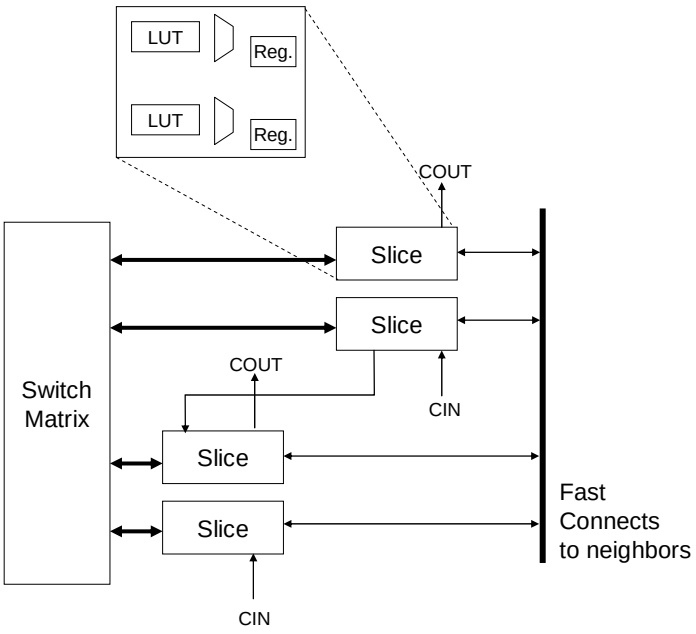


Figure 2-3 CLB architecture of Virtex II

2.3.2.2. Processor-array architecture

Another programmable solution is processor-array, differently from the LUT-based approach. Figure 2-4 shows an example of the processor-array architecture, Dynamically Reconfigurable Processor (DRP) developed by NEC Electronics [Mot02]. The processor-array type programmable device consists of processing elements (PEs) and internal memories. The specific unit for DRP is State Transition Controller (STC) which controls an instruction pointer of each processing element.

Figure 2-5 shows an internal architecture of the PE. It includes Arithmetic Logic Unit (ALU) and Data Management Unit (DMU) as calculation engines, registers to latch outputs of ALU and DMU, and crossbar switches as programmable interconnection. The PE receives an instruction pointer from the STC, and gets an instruction to execute using the pointer as a memory address. The read instruction is decoded and the PE configures the operations of ALU and DMU, programmable switch connections, and other modes supported.

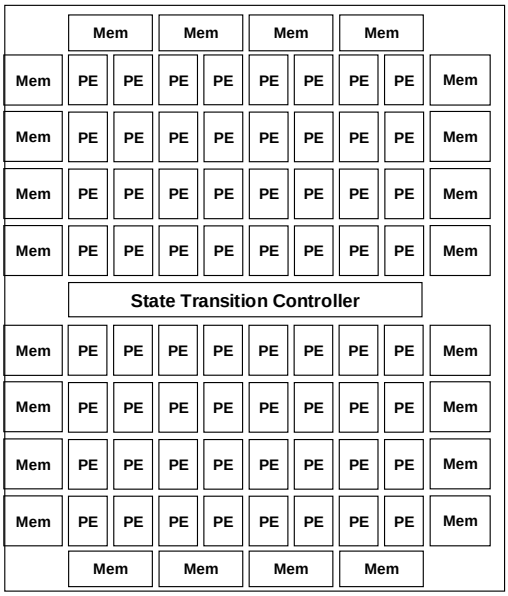


Figure 2-4 example of processor-array architecture

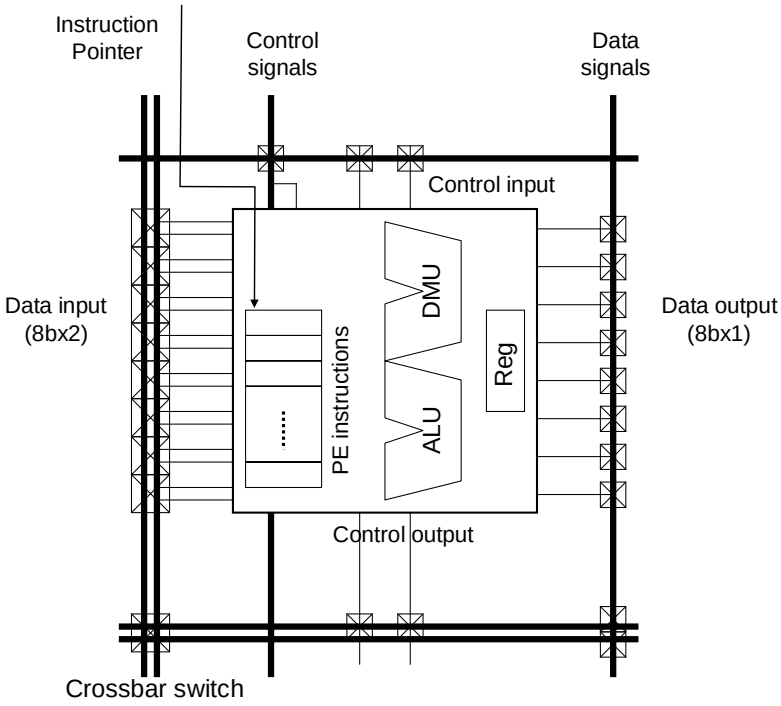


Figure 2-5 Processing element architecture

2.3.3. Difficulties in programmable device

Programmable devices have two major advantages over SoCs, which are NRE cost reduction and lower bug risks. However, there are some difficulties in producing them.

One major problem is large chip area, which results in expensive chip cost. In programmable devices, to emulate any types of hardware logic, they have flexible programmable switches as interconnection networks. And also, number of transistors required by programmable devices to emulate certain logic is more than 10 times than the raw logic used in ASICs.

Another difficulty is performance. The operation frequency of a circuit in ASICs is several times faster than that in programmable devices. This is because circuits are realized by combination of raw cell logic gates, although they are emulated by LUTs in programmable devices. Especially in sub-0.1 μm CMOS processes, wiring delay of CMOS devices become significant in overall delay in a chip, due to narrowed pitch and width of interconnects [Itr04].

Chapter 3 Related work

As described in the previous chapter, the key factor for IP-based SoCs and programmable devices is on-chip interconnection networks. In this chapter, related work of this research is summarized, from the standpoint of on-chip buses and Network-on-chips.

3.1. On-chip buses

Firstly, standard on-chip buses which have been used for System-on-Chip (SoC) designs are described in this section. There have been many buses, used in some companies' internal SoC designs, or as globally standardized specifications. Here, bus specifications called AMBA and CoreConnect, which have been widely used in SoC designs, are shown, as representatives.

3.1.1. AMBA

Advanced Microcontroller Bus Architecture (AMBA™) [Fly97][Arm99] is a bus specification for a system interface of ARM processor cores, proposed in 1997. The bus specification so far had been a local bus of a microprocessor, which needed to be tightly coupled with its load-store pipeline structure [Arm94]. To use a processor local bus as a general purposed SoC bus, there had been several problems.

1) In the era of SoCs, function blocks which had been already designed and verified should be reused to reduce design time. However, the processor local bus did not allow non-microprocessor designers who did not know microprocessor structure to integrate SoCs.

2) The processor local bus is very simple and flexible, thus the designers could enhance its specification. However, it is too simple and lack of practical functionality. Specifically, multiple bus masters had not been supported in the specification, thus integrating DMA controllers and memory controllers is problematic.
3) All the circuits must have been synchronized with a CPU internal clock. This causes complexity of the circuits and too much power consumption.

The motivations of AMBA were to solve these problems.

1) Apart from the processor local bus, a bus specification is defined for the purpose of function block reuse. This improves design time.
2) This specification allows multiple bus masters to issue transactions, for better performance.
3) The clocking structure was defined and it consists of two buses which have higher and slower clock frequencies. This approach achieves reduced power compared with the SoCs with a processor local bus. This results in less power consumption.

An SoC structure with AMBA is shown in Figure 3-1. It consists of Advanced High-performance Bus (AHB) and Advanced Peripheral Bus (APB). AHB is a multiple-master and multiple-slave bus for high performance data transfers. Slow function blocks are connected to APB, which is a single-master and multiple-slave bus. APB is connected to AHB with an AHB-APB bridge, which is the only master for APB. This bridge includes functions of an AHB slave and an APB master. An AHB master can access an APB slave through the bridge.

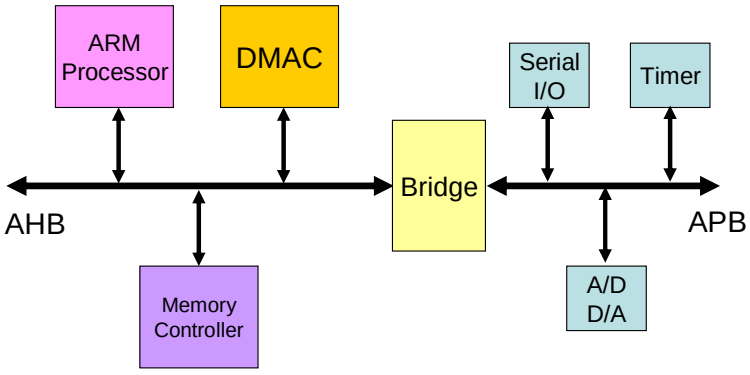


Figure 3-1 SoC structure with AMBA

3.1.2. CoreConnect

Another powerful on-chip bus is CoreConnect [Ibm99], developed by IBM Corporation and released in 1999. An SoC structure using CoreConnect is shown in Figure 3-2. This includes Processor Local Bus (PLB) for high performance transactions, and On-chip Peripheral Bus (OPB) for connecting slow function blocks. An OPB bridge connects PLB and OPB. The basic structures of PLB and OPB are similar to those of AMBA. The distinguished feature is that CoreConnect includes a Device Control Register (DCR) bus. Using the DCR bus, a microprocessor can set configurations and read status of function blocks. This is not performance critical, thus separated from PLB.

Positioning of CoreConnect is almost the same as AMBA. It improves on design time, performance, and power consumption.

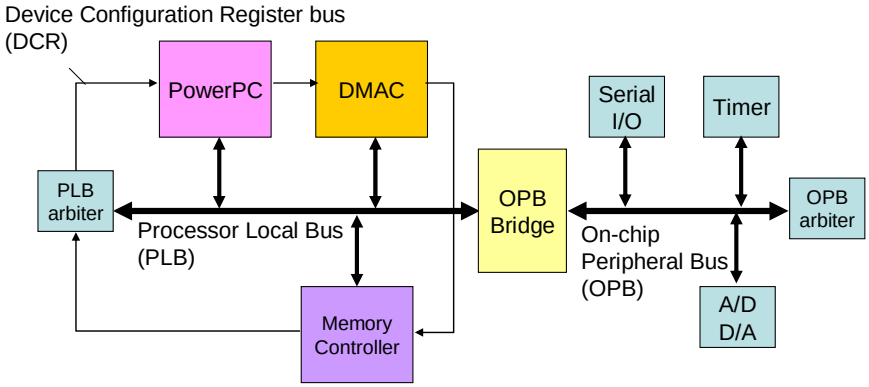


Figure 3-2 SoC structure with CoreConnect

3.2. Wrapper-based buses

Next, wrapper-based buses, started from an idea of “Interface-based design” [RS97], are described in this section. Wrapper-based buses are proposed for easing SoC designs, by reusing already developed function blocks, called Intellectual Property (IP) cores. This section shows the idea of “interface-based design” firstly, and then describes two conventional interfaces called Virtual Component Interface (VCI) and Open Core Protocol (OCP). Then, several implementations on bus wrappers and techniques for better performance are shown.

3.2.1. Interface-based design

The idea of “Interface-based design” was proposed as a new methodology for designing SoCs in short time-to-market [RS97]. The goals of this methodology are to encourage reusing IP cores, to achieve short design time, and to ease system-level designs. This method separates communication behavior from Intellectual Property (IP) cores. The design process started from developing abstract system-level functionality first, and then is refined incrementally to design more detailed parts, like its signal transition.

Figure 3-3 shows a refinement model of IP core designs from a system level down to an implementation level. Firstly, SoC designers connect IP cores with an abstract communication method, like send or receive functions. Then, this method is replaced with a behavioral block which includes communication wrappers to encapsulate detailed communication protocols. Finally, communication wrapper hardware is connected to each IP core, and the signals are visible to SoC designers. By using this methodology, the initial behavior-level design will be easier because designers do not have to consider detailed implementation information such as bus signals.

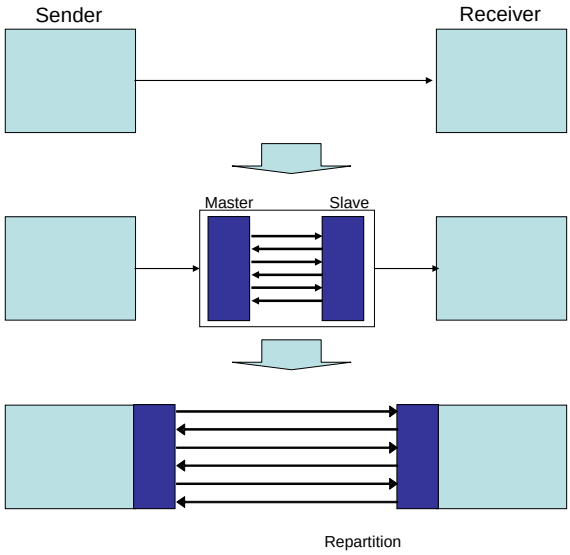


Figure 3-3 Refinement model of an IP core

3.2.2. Wrapper interfaces

Based on the idea of interface-based design, wrapper-based buses have been utilized instead of conventional on-chip buses to ease IP core development and increase its reusability. Wrapper-based buses separates complicated communication logic from IP cores, and bus wrappers are attached to IP cores for controlling bus protocols. There are two major wrapper interfaces, called Virtual Component Interface (VCI) and Open Core Protocol (OCP).

3.2.2.1. Virtual Component Interface (VCI)

Virtual Socket Interface Association (VSIA) defined a universal wrapper-interface called Virtual Component Interface (VCI) [VSI01]. The motivation of VCI is to define a general-purpose interface, such that IP cores in the shape of Virtual Components (VCs) of any origin, can be connected to SoCs developed by any chip integrator. In this manner, VCs are not limited to one-time usage by their designers, and can be reused over and over by other designers. They did not try to define a new standard bus, since removing conventional buses is not a practical strategy.

Because designers typically stick to their own buses for a long time and going into a new specification is not easily accepted, VCI is defined. VCI is an interface, rather than a bus, which is a point-to-point connection between a bus wrapper and a VC. And also it allows direct connection of VCs. VCI includes three levels of specification, Peripheral VCI (PVCI), Basic VCI (BVCI), and Advanced VCI (AVCI). These specifications classify VCs by performance and functionalities. VCI reduces design time and improves connectivity compared with conventional on-chip buses.

Figure 3-4 shows a block diagram of a VCI-based system. IP cores which initiate and receive transactions are called “Initiator VC” and “Target VC” respectively. VCI is used both in bus interfaces for Initiator VC and Target VC. Initiator VC is an initiator of VCI transaction, and Target VC is a target of VCI transaction, where both VCI is a point-to-point connection between VC and a bus wrapper. To convert transactions from VCI to an on-chip bus, wrappers

which bridging protocols are used. For Initiator VC, Initiator Wrapper which receives transactions from Initiator VC and initiates transactions on the bus is used. For Target VC, Target Wrapper is used.

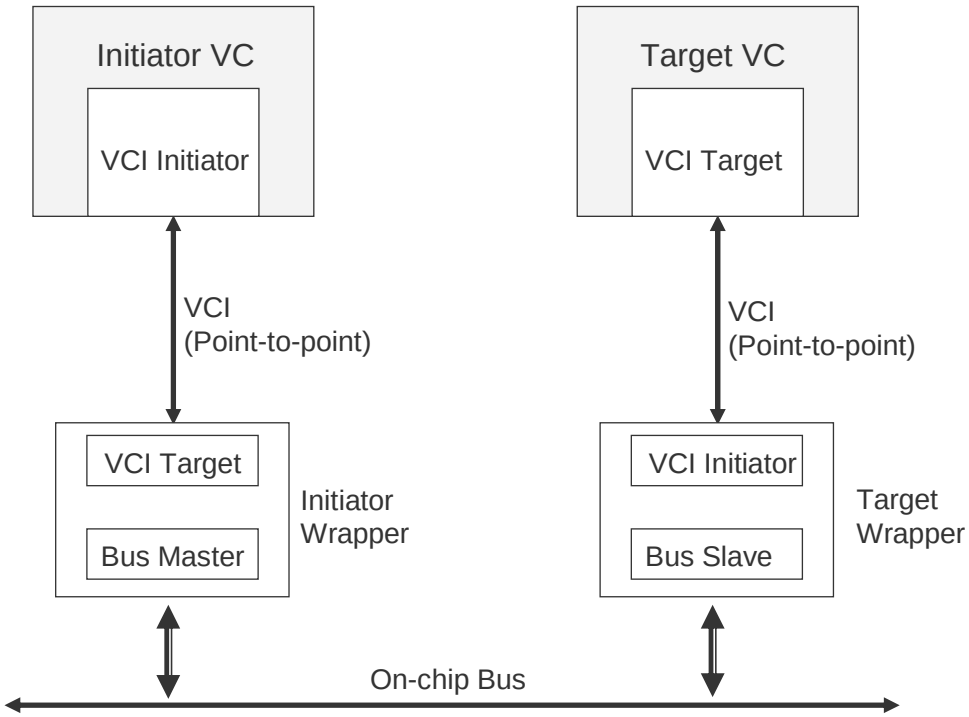


Figure 3-4 Bus connection with VCI

3.2.2.2. Open Core Protocol (OCP)

SONICS Inc. has defined Open Core Protocol (OCP) [Son00] as a communication interface for IP cores. The motivation of this interface is encouraging reuse of IP cores without any chip reworks. The interface is independent of interconnect implementation details, such as bus bit-width or control timings. The distinguished features from VCI are as follows.

1) The OCP definition is not classified as Peripheral, Basic, or Advanced, like VCI. Thus the single interface can be used for any purposes.

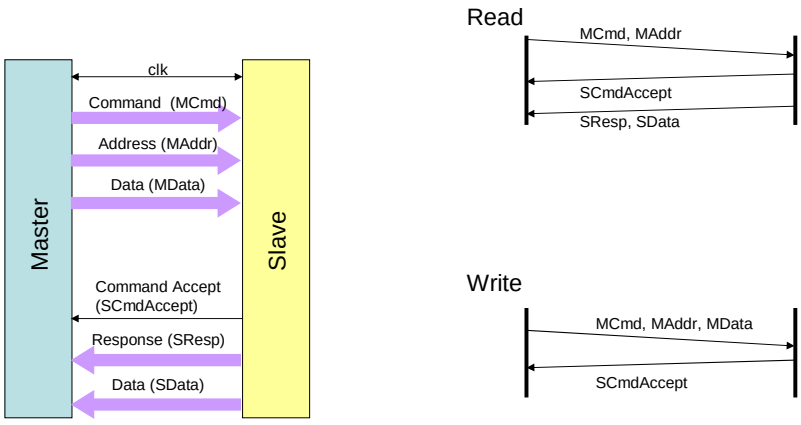
2) OCP supports control procedures and test interface signals for manufacturing.

3) A practical design has been developed, with their proprietary bus, called Silicon Backplane [Son02]. Thus, development environment for OCP already exists, while VCI mainly focused on specification definition.

Since VCI did not work well as a standardized interface, OCP was chosen as next version of VCI in 2003.

Figure 3-5 shows a basic signal definitions and protocols of OCP. In Figure 3-5 (a), signals required for communications between a master and a slave of OCP, are illustrated. The same clock signal is delivered to both the master and the slave. All the signals except for the clock signal are unidirectional from the master to the slave, or from the slave to the master. Figure 3-5 (b) shows basic protocol charts for Read and Write transactions of OCP. The master activates MCmd and MAddr signals for specifying requested commands. If a command is a Write request, Write data is also transmitted on MData. The slave responds to the request transmission by activating SCmdAccept. If a command is a Read request, SResp and SData signals are transmitted to send back the read data.

Beyond this basic description, further advanced features, such as simultaneous and out-of-order data transmission by tagging command IDs to request and response, are supported.



(a) (b)

Figure 3-5 Signal definition and protocol of OCP

3.3. Wrapper-based bus implementations

3.3.1. Silicon Backplane

SONICS Inc. developed Silicon Backplane, as an on-chip interconnect using OCP. To mainly focus on multimedia applications, Silicon Backplane is distinguished with a feature which guarantees performance, especially throughput, by employing a Time Division Multiplexed Access (TDMA) protocol.

Figure 3-6 shows communication architecture of Silicon Backplane. Master and slave cores, designed with an OCP interface, are connected by Silicon Backplane and OCP wrappers. Silicon Backplane is generated automatically by the delivered toolkit. When generating Silicon Backplane between cores, the requirements for inter-core communication performance are specified as parameters. The time slots are statically assigned to these communications so that the given requirements are satisfied. Since the arbitration cycles are statically determined, no parts which have dynamic latency cycles exist in this interconnect. For removing dynamic latency cycles, an OCP wrapper splits a burst data from an OCP initiator into small chunk of data, called threads, and each thread is transmitted within one cycle. This requires FIFO buffers in wrappers, whose size should be a maximum length supported by the OCP specification. So, from the chip cost aspect, it requires large amount of hardware required by FIFO buffers in all the wrappers.

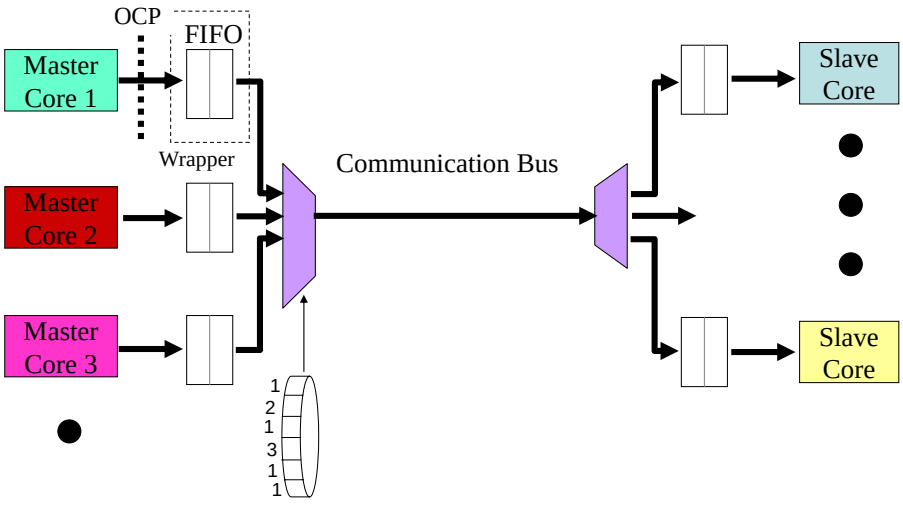


Figure 3-6 Communication architecture of Silicon Backplane

3.4. Techniques for improving wrapper-based buses

Although the wrapper-based bus architectures can increase IP core reusability, conventional publications pointed out increase of access overhead, especially of latency cycles. Here two techniques for improving access latency cycles are summarized.

3.4.1. Prefetching in slave wrappers

Bus wrappers enable to retarget IP cores to different SoCs. However, it addresses longer Read latency cycles, because additional interactions between bus wrappers and IP cores are required. To achieve better performance with wrapper interfaces, an implementation technique which keeps local copies of accessed registers in bus wrappers, has been proposed [LV02]. By prefetching register data in a slave wrapper as shown in Figure 3-7, no interaction on a slave wrapper interface is needed for a Read operation. The latency to send back the Read data to the master becomes at least 2-cycle faster.

This scheme improves specifically latency cycles of Read operations. However, creating data copies in bus wrappers requires additional hardware overhead, resulting in increase of chip cost.

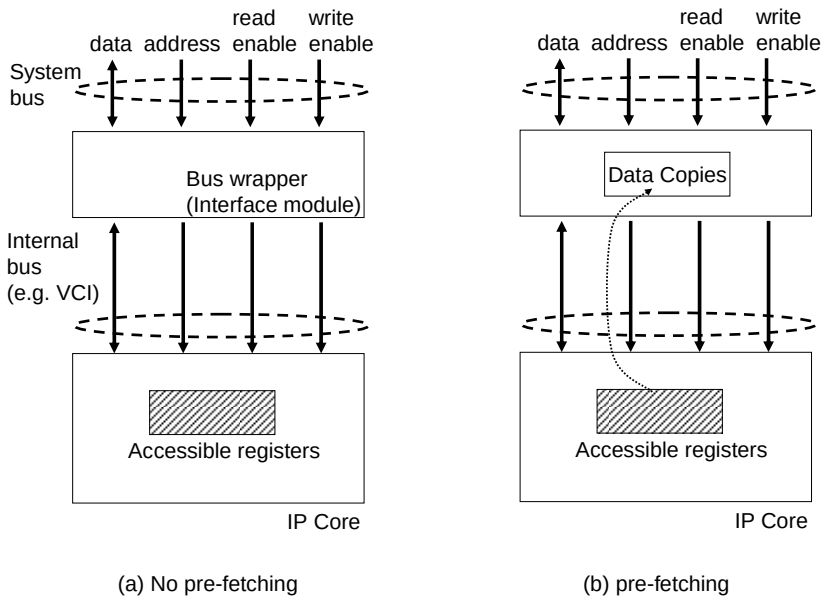


Figure 3-7 Lysecky's prefetching structure

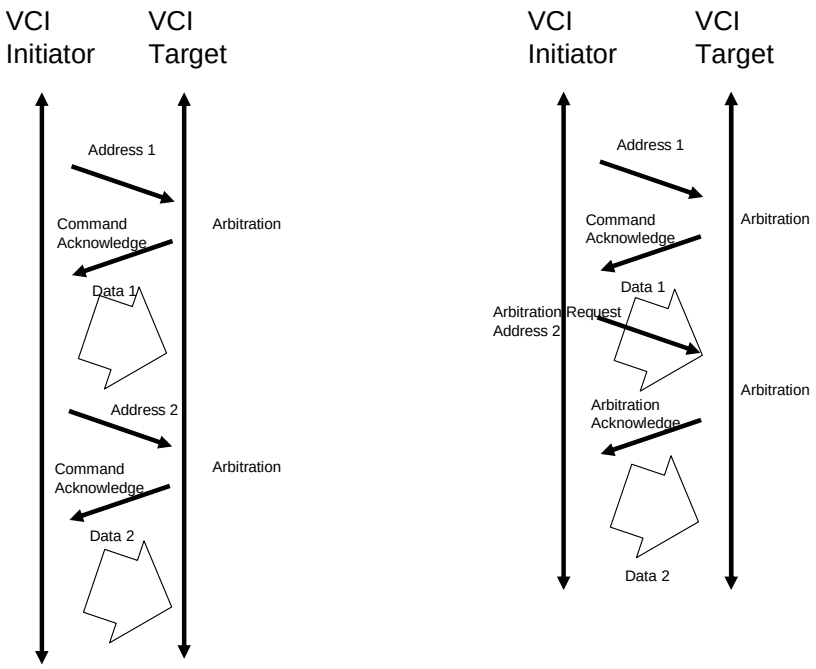
3.4.2. Arbitration hiding mode of advanced VCI

The Advanced VCI (AVCI) specification includes a technique called “arbitration hiding mode”. This is a feature of an interface protocol which allows to process a current data-transfer and a next request issue in a pipeline manner. Typically a request phase of a bus protocol includes an arbitration which consumes several latency cycles. Thus, this pipelining processing of issuing a next request and transferring current data results in better throughput by hiding arbitration cycles in a request.

Figure 3-8 shows comparison of protocol charts when (a) without and (b) with the arbitration hiding mode, in Write transaction cases. Note that a VCI initiator and a VCI target of this figure correspond to the ones in Figure 3-4.

As shown in Figure 3-8 (a), without the arbitration hiding mode, a Write transaction is initiated by the VCI initiator, which may be a master IP core. It issues a Write address and a command. Then, the VCI targets which may be a master bus wrapper, receives and processes the command. It requests arbitration and communicates with a slave. After establishing the communication path, the VCI target sends a command acknowledgement back to the initiator, and the initiator transfers Write data. Without the arbitration hiding mode, the VCI initiator must wait for an end of the previous Write data transaction to submit an address transfer of the next Write transaction.

With the arbitration hiding mode described in Figure 3-8 (b), Arbitration Command Valid and Arbitration Address signals specify an advanced arbitration request for a next request, in the meantime when a current request is in process. Arbitration Command Acknowledge signals indicates that an acknowledgement for the next request issued after bus arbitration. This overlapped arbitration reduces latency cycles for a request.



(a) Without arbitration hiding mode (b) With arbitration hiding mode

Figure 3-8 Protocol chart comparison between cases with and without arbitration hiding mode, in Write cases.

3.5. Advanced on-chip buses

Before summarizing NoCs, several advanced on-chip bus architectures which triggered researches and developments for NoCs are described. The major limitation of conventional on-chip buses is that it can only process one transaction at a time, and thus several efforts to improve on-chip bus performance have been proposed. In this section, Multi-layer AHB and Lotterybus are shown.

3.5.1. Multi-layer AHB

Multi-layer AHB [Arm01] is an interconnection which has multiple AMBA AHB buses. This enables parallel data-transfers between multiple pairs of a master and the slave in a system. Figure 3-9 shows an example structure of the multi-layer AHB. Masters and slaves are connected to an interconnect matrix which includes multiple layers of the AHB buses. These AHB buses are

separate each other, and this structure allows multiple masters to issue transactions for different slaves at a time. When multiple masters try to access to the same slave, a master in higher priority completes transaction first and another next. Further variations can be constructed, such that local slaves are connected only to a specific layer, or only some slaves are shared by some layers.

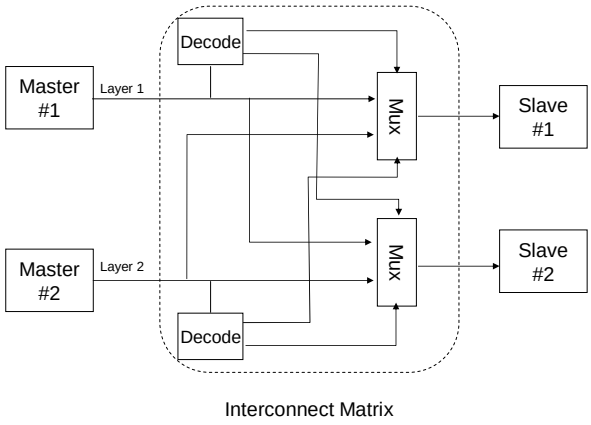


Figure 3-9 Structure of multi-layer AHB

3.5.2. Lotterybus

Lahiri pointed out problems and limitations of the conventional static priority shared bus and TDMA bus [LRL02]. A shared bus with static priority has difficulties in controlling bandwidth allocation in fine grain, IP core by IP core. Thus, the TDMA architecture is not suitable for IP cores which require low latency cycles and high throughput. The proposed Lotterybus includes a dynamic arbitration control scheme. Differently from the TDMA bus, time slots are not assigned statically to each core, and Lotterybus arbitrates and assigns time slots dynamically to the cores.

This work showed that many difficulties are included in conventional on-chip buses, and experiments through computer network architectures are useful to interconnection network designs in SoCs.

3.6. Network-on-Chips

Network-on-Chips (NoCs) are communication architectures used in programmable devices, or router-based interconnection networks which could be used in future SoCs. NoCs used in programmable devices are programmable switches. Some proposals on NoC architectures for future array-type SoCs utilize routers which have been used in interconnection networks used in System Area Network (SAN) or parallel computers. In this section, several NoC architectures are described and summarized as below.

3.6.1. Programmable switch

Most programmable devices such as FPGA, CPLD, and processor arrays, employ programmable switches [RB91] as interconnection networks. As an example, programmable switch architecture used in FPGA is shown in Figure 3-10. CLBs are connected with communication box (CBOX) and switch box (SBOX). CBOX and SBOX are programmable crossbars, which consists of multiple of multiplexers and configuration memory to store configuration data for specifying connections. In CBOX and SBOX, the actual switch architecture is not a full crossbar, and they reduce switching flexibilities by removing multiplexer inputs.

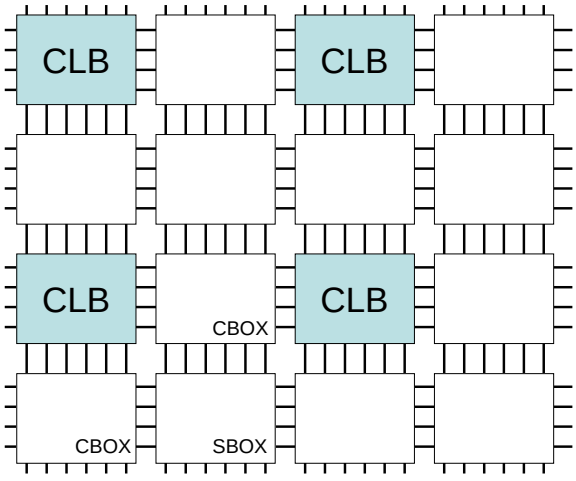


Figure 3-10 Programmable switches in FPGA

3.6.2. SPIN

In [GG02], as a replacement of conventional shared bus structures, Scalable, Programmable, Integrated Network (SPIN) is proposed. SPIN employs a fat-tree topology because of its cost efficiency [Lai85], as shown in Figure 3-11 (a). A simple packet structure is used for inter-core communication, and link and upper layers can be designed on top of it, such as stream data-flow communications or address-space accesses. The packet structure, presented in Figure 3-11 (b), includes a single-flit header, variable body flits and a tail flit. The header includes an 8-bit destination node number to indicate up to 256 nodes. By adding a tail flit, the end of a packet is specified and a variable-sized packet is provided.

Figure 3-12 shows a router architecture designed for SPIN. Since a fat-tree topology for 16 nodes requires 8 input ports for child and parent paths, and it supports two shared output buffers, a 10x10 crossbar is necessary.

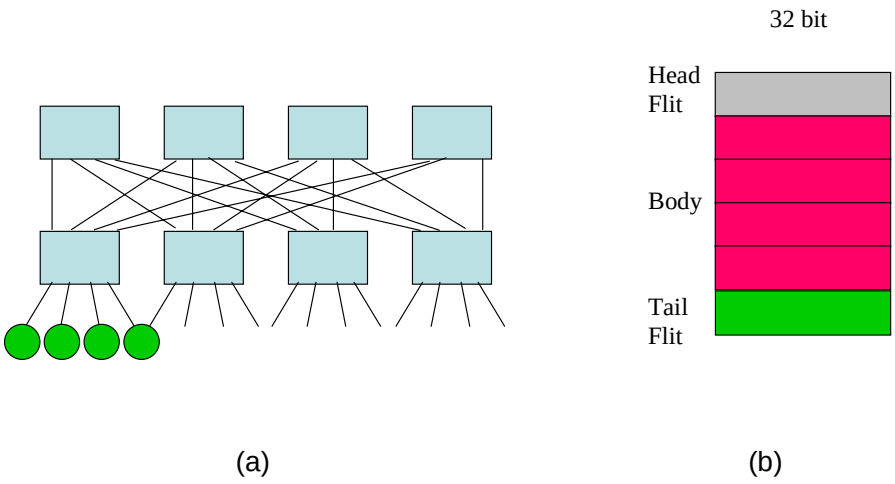


Figure 3-11 Network topology and packet structure of SPIN

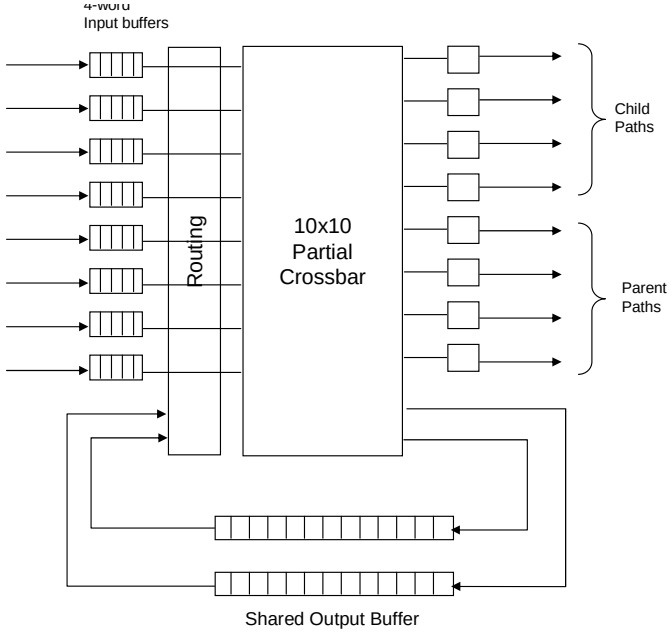


Figure 3-12 Router architecture of SPIN

3.6.3. Dally's proposal

Dally and Towles showed design criteria for NoCs and showed an example NoC structure [DT01]. NoC is applied as an interconnection so that it can treat the increased number of IP cores on a chip as a replacement of on-chip wiring. Controlling electrical parameters such as crosstalk and parasitic capacitance is made easy, since the architecture is better-structured than the conventional global wirings.

The described example includes same-sized IP cores connected by an interconnection network. Their proposed topology of this network is folded 2-D torus for 16 nodes, as shown in Figure 3-13 (a). This employs a variable-sized packet structure as shown in Figure 3-13 (b). Conventional virtual channel flow-control is applied, for removing packet blocking in channels. A large number of buffer spaces is consumed: 10K bits for each input controller in a router.

Figure 2-1 shows router architecture in this proposal. A router in a node includes five input and five output controllers. One of the ports is used to connect internal core logic. This figure only shows a west input controller and paths from west for simplicity. The output controller only has an output

multiplexer and output buffers to store data coming from the inputs. This architecture is similar to a typical router which has been used in System Area Networks of PC clusters, or parallel computers [DYN02].

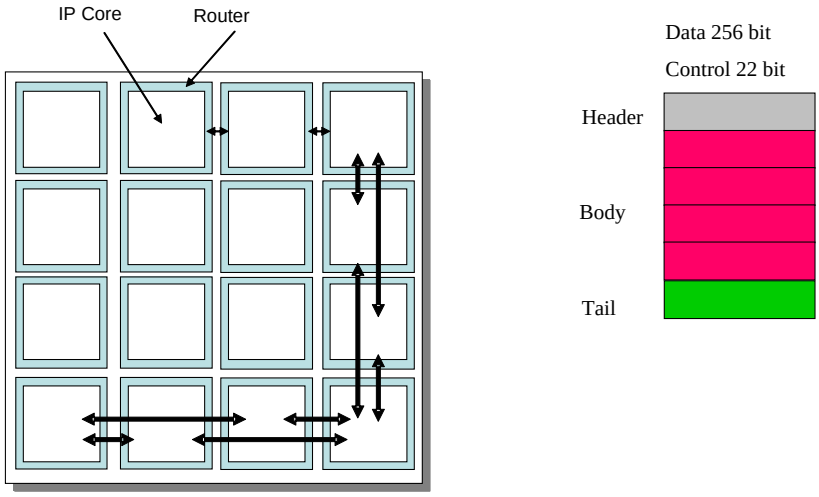


Figure 3-13 Network architecture and packet structure of Dally's proposal

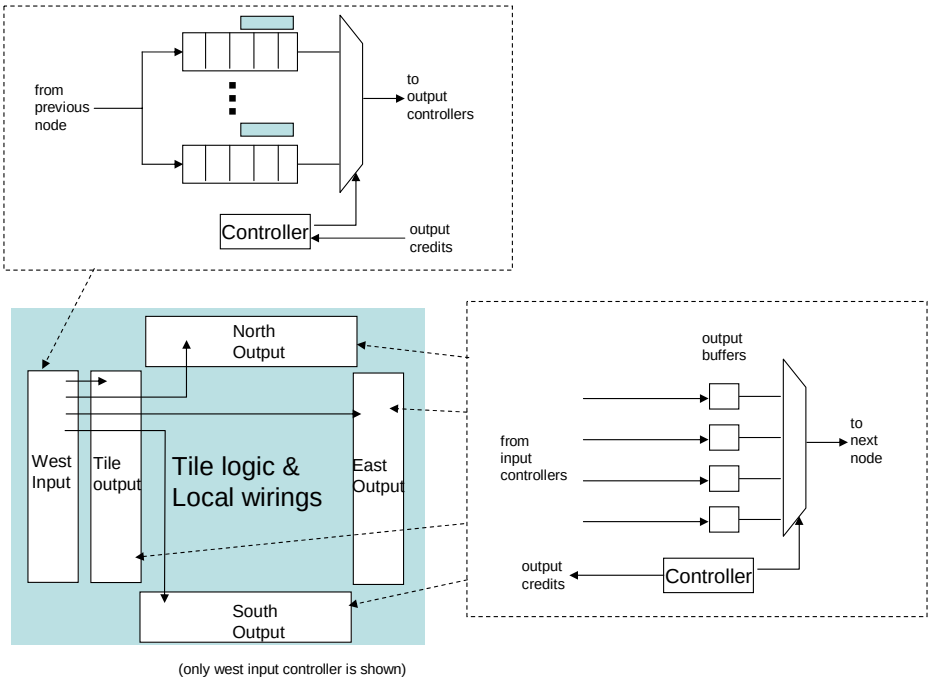


Figure 3-14 Router architecture of Dally's proposal

3.6.4. Matrix Interconnection Network in ACM

Adaptive Computing Machine (ACM) [Mas02] is a reconfigurable architecture proposed for processing mostly wireless applications. ACM has various types of computation nodes, which are prepared as libraries for ASIC development. Types of computation nodes and its combination are chosen by each user. The architecture framework using an NoC called Matrix Interconnection Network (MIN) to connect different kinds of nodes is prepared.

Figure 3-15 (a) shows the architecture overview. Each computation node is connected by an NoC of H-tree topology. Four of the computation nodes are connected by a network router, and the router is connected to routers of different nodes through another router which is a non-blocking root.

Figure 3-15 (b) is packet structure used in ACM. A single flit packet is used, and thus, a packet header is attached to data which is transferred in a single cycle. The packet header is 19 bits for each 32-bit data, which includes destination address, port number and task number as header fields.

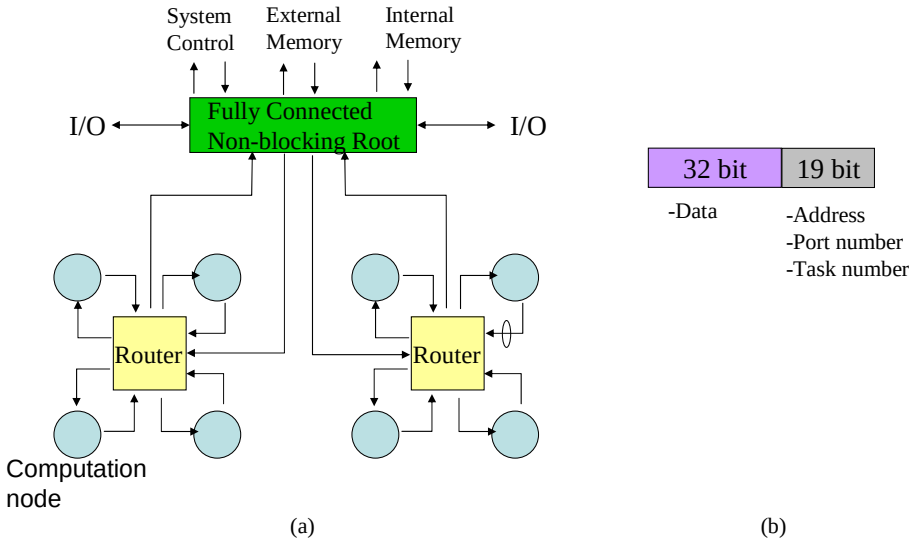


Figure 3-15 Network topology and packet structure of MIN in ACM

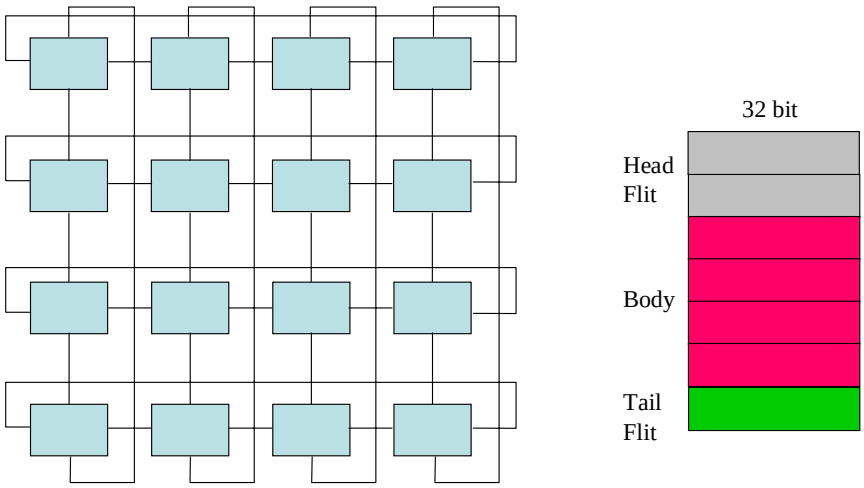
3.6.5. Marescaux's proposal

A reconfigurable SoC for multimedia applications which integrates an NoC in the center, is shown in [MBV02]. For prototyping on FPGA, a 2-D torus topology is chosen due to its 2-D array architecture. The logical 2-D torus topology is shown in Figure 3-16 (a), and its implementation applied folded torus structure which had been used in [DT01]. Figure 3-16 (b) shows a packet structure of this network, which has two header flits, fix-sized body and a tail flit.

The routing scheme used in this proposal is source routing [DYN02]. With the source routing, routers route data according to routing tags attached by a source node. These routing tags specify which port to forward data, to all the routers to pass. The source routing does not require routing tables in routers, but in nodes. Compared with a distributed routing which typically uses a global address as routing information to look up a routing table in routers, source nodes assign routing tags to each communication path, not to each destination node. Thus, it has flexibility to change routes according to types of communication paths.

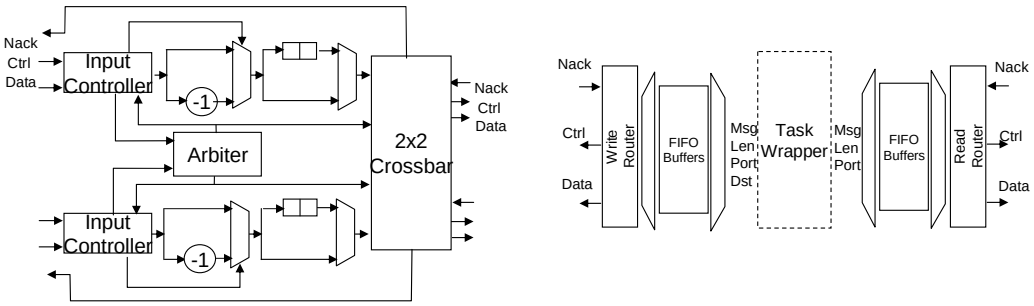
This work employed an e-cube routing [DS87] as routing algorithm, which calculates routing tags easily by only comparing a source node position and a destination node position in a source node. This e-cube routing is a sort of source routing only applicable to 2-D mesh or torus structures. The routing tags specify the number of hops to route along X- and Y-directions. A router transfers packets toward X-direction firstly until the number of X-hops becomes zero. Then it route toward Y-direction next.

The router architecture is shown in Figure 3-17 (a). It does not have routing table, but has circuits to check routing tags. The network interface used in this work is shown in Figure 3-17 (b). The proposed network interface which is located between a task in a core, and a router has packet generation and reception logic. This interface faces the task with signals of a destination logical address, a port number and a message length.



(a) (b)

Figure 3-16 Network topology and packet structure of Marescaux's proposal



(a) (b)

Figure 3-17 Router and network interface architecture of Marescaux's proposal

3.6. Positioning of this research

This section describes problem definitions of prior work to be solved, and clarifies the purposes and motivations of this research.

3.6.1. Problem definition of prior work

3.6.1.1. Wrapper-based buses

As described in the previous chapter, wrapper-based buses have been considered as a hopeful approach to encourage IP core reuses. However, VCI failed to be a major standard for wrapper-based buses, and some other counter proposals looked better solutions, like OCP. And finally OCP was chosen as a next version of VCI standard in 2003. The major difference between VCI and OCP is that the OCP interface is a single definition, where VCI classified three specifications according to performance ranges. Also the OCP interface specification is verified on silicon with SiliconBackplane-based implementation. So, interface based on the practical implementation is quite essential to acquire high credibility.

Several publications discussed implementation techniques to improve performance in wrapper-based buses. In [LV02], a technique to accelerate Read latency cycles by prefetching data into a slave wrapper is proposed, as described in Section 3.4.1. Also, the Advanced VCI specification includes the arbitration hiding mode, as described in Section 3.4.2. The combination of OCP and SiliconBackplane does not accelerate latency cycles or throughput, but guarantees inter-core throughputs, as shown in Section 3.3.1. In summary, conventional approaches have mostly focused on solving latency overhead and guaranteeing throughput.

Those prior work on wrapper-based bus implementations focused mostly only on performance improvement. However, improving performance without consuming further additional hardware is quite important in consumer market which is typically cost-sensitive. Thus, further considerations on wrapper-based

bus implementation from both aspects of cost and performance are quite essential.

3.6.1.2. Network-on-Chips

Table 3-1 shows a summarized table of conventional NoC proposals. To compare these NoCs, four criteria are listed, that are topology, packet length, size of header, and routing scheme.

- Prior work show that network topology depends on application types, such that H-tree topology is good for Wireless application [Mas02], a cost-effective application should take fat-tree [GG02], and 2-D mesh or torus are good for FPGA-based application [MBV02]. Thus, in NoC researches, consideration on multiple topologies is important to apply various applications.
- Conventional proposals use variable or fixed-length packets, and only ACM uses single-flit packet. So, most NoCs take the approaches like SANs, and the packet-based data-transfer is applied. Although ACM tried to explore another approach, that is a single-flit packet data-transfer, there is no obvious motivation revealed in their published papers.
- The required header size is 19-38 bits and relatively large when transferring 32- or 64-bit data. Size of the headers affects hardware amount of multiplexers, data buffers, and the routing table entries.
- All the routing schemes applied in these NoCs are the methods used in SANs. Routing function must be decided after considering communication patterns and cost requirements, since this has great impacts on them.

As summarized above, an NoC architecture must be discussed from the aspects of applications, hardware cost, and performance, all of which are quite important criteria in designing SoCs.

Table 3-1 Summarized table of conventional NoCs

	Topology	Packet Length	Header Size	Routing
Dally's proposal	2-D torus	Variable	38 bit	Source routing (16 bit)
SPIN	Fat-tree	Variable	32 bit	8 bit (Up/down routing)
ACM	H-tree	1 flit	19 bit	Distributed routing with node addresses
Marescaux's proposal	2-D torus	16 flit	32 bit	Source routing (e-cube routing:6 bit)

3.6.2. Purpose and motivation of this research

3.6.2.1. Purpose

The on-chip interconnection networks are in the generation of wrapper-based buses. And the next generation would be Network-on-Chips in sub-0.1μm CMOS process technologies. To both of these technical fields, this research focuses on how efficient the data transfer is, by considering both hardware amount and performance, differently from conventional approaches.

In the technical field of wrapper-based buses for IP-based SoCs, considerations on hardware amount and functionality as well as performance, are very important to achieve practical quality and support general-purpose use with enough functionalities. This research focuses on also showing functionalities and achieved performance as well as a proposal of a novel wrapper interface and a bus implementation. Also, a real chip design is included in this research as well as proposals of novel design techniques.

As for NoCs, conceptual proposals which simply applied conventional network architectures used in parallel computers or SANs, have been published already. The major purposes of this research on NoCs is to give an idea to transfer data

cost-efficiently and to show this scheme fits SoC environment which has wide range of IP cores, small or large, with practical applications.

3.6.2.2. Motivation of research on efficient design techniques for wrapper-based buses

As described in Section 3.6.1.1, implementations of wrapper-based buses are quite important for practical use. While conventional research and development of wrapper-based buses mostly focused only on performance improvement, this research tries to figure out its cost-efficiency by considering tradeoffs between performance and hardware cost. And as well as the performance and cost tradeoffs, connectivity of IP cores has been considered to solve potential obstacles in buses, such as livelock, retries and bit-width conversion. To show performance and cost-efficiency, impacts on these criteria of each proposed techniques are shown item by item. And finally, overall performance and hardware cost are shown, with a real chip implementation.

3.6.2.3. Motivation of research on efficient data-transfer scheme for Network-on-Chip

Although Network-on-Chips have been considered as a replacement of programmable switches, there are major three differences in networking environment, differently from SANs in PC clusters/parallel computers. The differences are described as bellows:

1) An NoC is used to connect computation nodes in coarse-grained programmable devices. Once a chip is fabricated, the numbers of IP cores and their functionalities are not changed in most cases. In SANs, the ability to change the number of nodes is important for scalability.
2) Although a SAN is based on chip-to-chip communication, an NoC is intra-chip communication. Thus, an NoC does not have pin count limitations between routers or nodes, and can take advantage of rich wires provided by semiconductor process scaling.

3) Cost of routers in SANs is important but not really too significant, because the node itself is typically made up of large amounts of hardware like microprocessors, north bridges, DRAMs, etc. In NoCs, the computation nodes vary from small to large. Thus, the router itself is expected to be small enough even with small IP cores.

Taking those environmental differences into account, this research on NoC proposes a novel technique to transfer data cost-efficiently. Major claims of this research are that applying a data-transfer scheme using separate routing information apart from the conventional packet data transfer, and on this scheme, a novel routing technique using local labels is proposed. The results include analysis of communication patterns in real applications, and show its hardware-cost efficiency and performance improvement.

Chapter 4 Efficient data-transfer schemes for wrapper-based buses

This chapter presents wrapper-based bus architecture for low-cost implementation and implementation techniques to improve performance. In Section 4.1, overview of this proposal is described, firstly. Next, conventional protocol charts with an existing wrapper interface are shown in Section 4.2. Section 4.3 describes the proposed wrapper interface and several design techniques which exploit this interface are described. The example SoC which uses the proposed bus is shown, and based on this architecture, evaluated results of performance and hardware amount are shown in Section 4.5.

4.1. Overview

This research has been done for the purpose of achieving cost and performance efficient implementations of wrapper-based buses, and solving some connectivity problems for better design TAT. In this research a general-purpose wrapper-based bus for better SoC performance that has less wrapper hardware [AOK02] is proposed. The goal is to develop wrapper-based bus architecture for a wide range of applications that require lower cost and better performance. To achieve this goal, unique wrapper interfaces including a flow-based slave interface are defined. Wrapper implementation techniques called Write buffer switching (WBS) and slave designated retry control (SDRC) with a livelock avoidance scheme are developed, under the proposed interfaces. Furthermore, to broaden the application range, a technique for

connecting cores that have different bit-widths by embedding converters in the physical bus have been developed.

4.2. Protocols with existing wrapper interface

In this section, previous work related to implementation techniques for wrapper-based buses is summarized.

4.2.1. Standard on-chip bus protocols

Using the on-chip bus protocols of AMBA [Arm99] or CoreConnect [Ibm99], we can connect an IP core including bus interface logic to a bus directly, and the core can communicate with another core also connected to the bus. However, IP cores complying with a standard bus protocol cannot be connected to another bus without changing their bus interface logic. For general purpose use, these protocols enable IP cores with a wide range of performance. Separate buses are defined for high- and low-performance and these buses are connected by a delivered bus bridge.

4.2.2. Wrapper interface definitions

To improve IP core reusability, wrapper interfaces have been defined to remove communication logic from the cores and put it into a bus wrapper. A standardized wrapper interface has been developed, and an interface called “Virtual Component Interface (VCI)” has been defined [VSI01]. The VCI has an interface protocol that can be bridged into any physical bus protocol with bus wrapper hardware. With this interface, IP cores can be retargeted into any physical bus protocol by replacing bus wrappers. The master and slave interface protocols are identical, so the master and slave cores can be viewed as directly connected. Another wrapper interface is called Open Core Protocol (OCP) [Son00], and it is similar to the advanced specification of VCI.

4.2.3. Wrapper-based bus implementations

The operation of the basic wrapper-based bus protocol using an existing wrapper-based interface [VSI01] is illustrated in Figure 4-1. The bus comprises a master IP core, a slave IP core, a master wrapper, a slave wrapper, and a physical bus connection. The master core issues requests, and the slave core receives them.

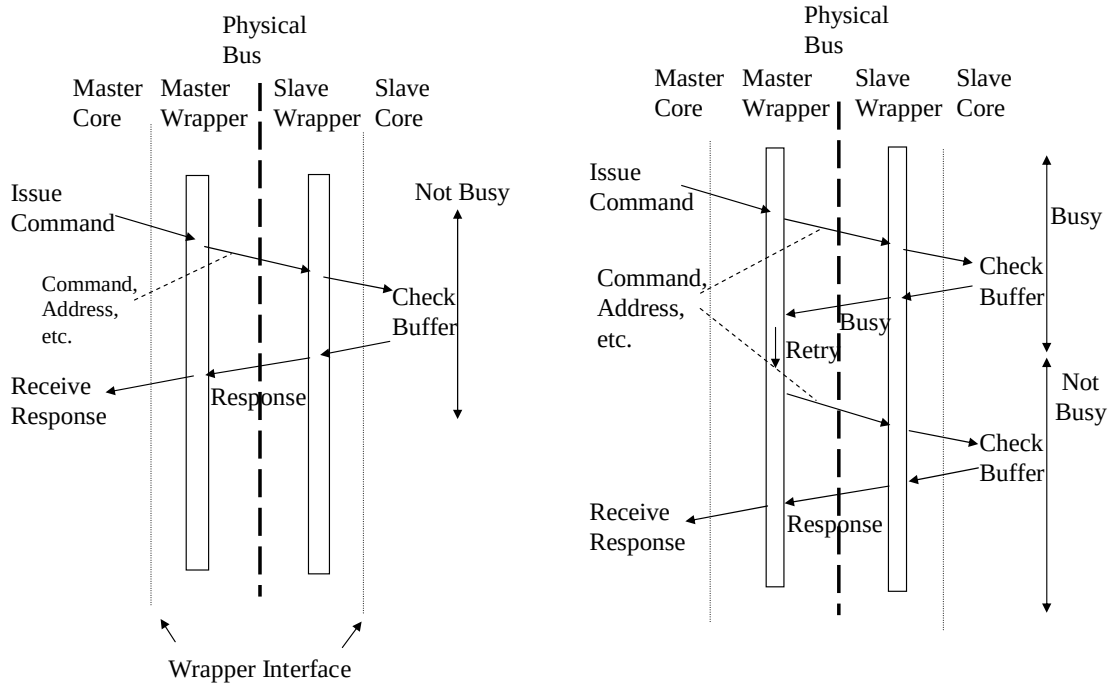
Figure 4-1 (a) shows the case when the slave core is ready to receive a request from the master core. The master and slave wrappers pass the request, and the slave core accepts it. Responses are sent back only for Read requests; Write transactions do not require responses. Write requests are sent with Write data as burst transfers.

Figure 4-1 (b) shows the case in which the slave core is not ready. Because it does not have buffer space available for receiving more requests, the major difference here is that the response is a busy response. The master wrapper continues to send the request until the slave core returns to a ready state and it accepts the request.

From the performance point of view, the slave-ready situation is the best case; the non-ready case degrades average performance. Some conventional wrapper-based bus implementations [Son02][YNL01] embed FIFOs to buffer request and Write data in the wrappers. This improves average performance because the retry transactions are accelerated. One proposed wrapper-based bus [Son02] improves performance by optimizing the size of the FIFO buffers in the master and slave wrappers, so that the retry accesses do not consume bus throughput and increase the number of access latency cycles. Another proposed wrapper-based bus [YNL01] can be implemented with 3K to 5K gates excluding those for FIFO buffers [LYB02].

A reported performance optimization technique [LV02] reduces the latency cycle overhead, which is one of the major drawbacks of wrapper-based buses. It specifically accelerates the Read latency cycles by placing accessible register copies inside the slave wrappers. Thus, interaction between a slave wrapper and the slave core for Read requests is unnecessary. Comparison of the

designed wrapper with a basic wrapper without embedded FIFO buffers and complying with the VCI protocol showed that the basic wrapper requires approximately 3K gates while the designed wrapper requires an additional number of gates, ranging from less than 1K to approximately 3K, depending on the size of the required register copies.



(a) Slave core when not busy (b) Slave core when busy, retry in master

Figure 4-1 Bus protocol with conventional wrapper interface

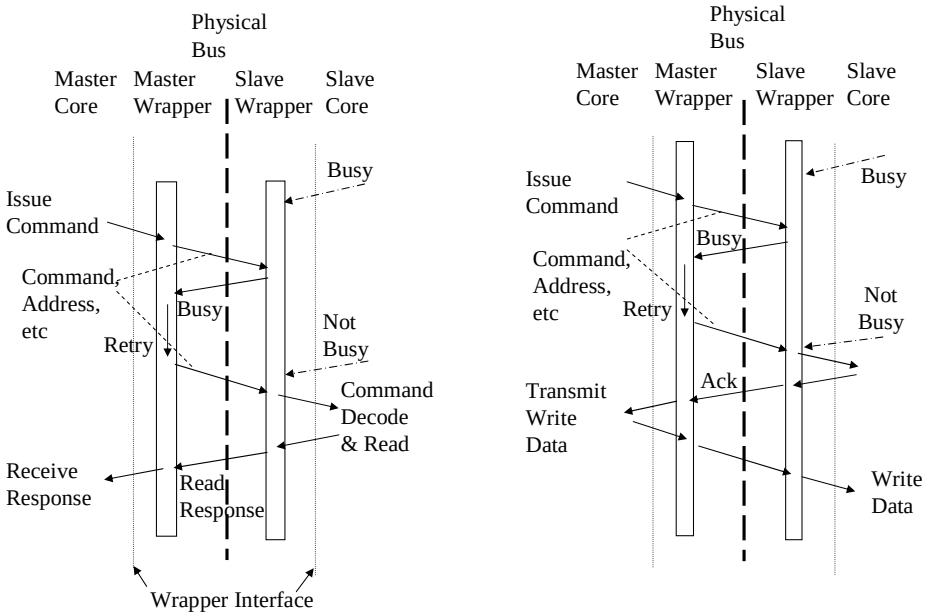
4.3. Developed bus architecture

4.3.1. Interface definition

The bus design strategy proposed in this research is focused on achieving better performance with minimum bus wrapper hardware so that it can be widely used in various applications. As discussed in Section 4.2, to achieve better performance with existing wrapper interfaces, buffer space for requests

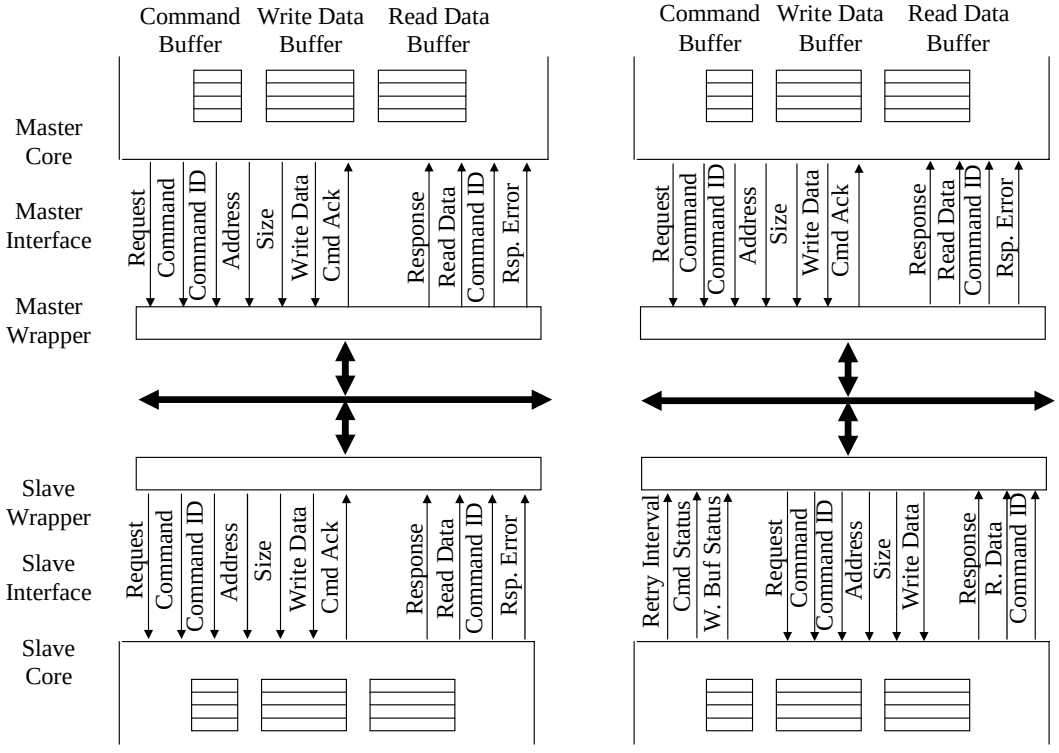
and Write data are typically embedded inside the wrapper. Since this approach increases the hardware, this proposal does not take it.

Figure 4-2 (a) and (b) show the operation of the proposed bus protocol for Read and Write transactions, with a no-buffer wrapper implementation. A flow-control protocol in the slave wrapper interface is used, rather than a handshaking interface with a request-response manner. The interface definition is shown in Figure 4-3 (a) and (b). It is basically the same as that in the conventional wrapper interface. Similar to the conventional one, the interface views advanced implementations using multiple-buses or crossbars as future possibilities. Thus, the protocol allows issuing simultaneous requests before waiting for completing corresponding data transfers to achieve non-blocking communications. The major difference is that the master and slave wrapper interfaces are not identical. For the slave interface, status signals are simply added to indicate busy or not busy for the request buffer and the Write data buffer in the slave core and retry interval signals for specifying the back-off interval for retrying. It is assumed that only a few latency cycles are required to transmit the status signals, and the slave wrapper can determine whether to accept or reject a request received from the master with these signals. Therefore, before passing a request to the slave core, the slave wrapper can determine whether the slave core has available buffer space, and thus there is no need for an interaction. This interface can potentially result in shorter latency cycles than those of the conventional wrapper-based interface. Furthermore, the proposed protocol decreases the hardware complexity of the slave core interface because it guarantees that all the transmitted requests are buffered and processed rather than being rejected due to a busy condition. The purpose of the retry interval signals is described in Section 4.4.2.



(a) Read access (b) Write access

Figure 4-2 Proposed wrapper-based bus protocol



(a) Conventional (b) Proposed

Figure 4-3 Bus interface definition

4.4. Bus architecture and protocol overview

In this research, a wrapper-based bus using the proposed interface is designed, to demonstrate that no-data-buffer wrappers with practical performance can be implemented at low cost.

Figure 4-4 shows the architecture of the developed bus. The physical layer consists of a command and Write data (CWD) bus and a Read data (RD) bus. The separated CWD and RD buses enable split transactions to be naturally supported, in which each Read transaction releases the CWD bus to another bus before the Read data is made available. Furthermore, non-blocking communication, in which multiple Read requests are issued from a master core before the corresponding Read data reaches the master core, is supported simply by attaching an ID to each command. The CWD bus conveys requests, addresses, commands, sizes, command IDs, master IDs, Write data, acknowledgements (Acks), negative-acknowledgements (Nacks), and retry information. The RD bus conveys responses, Read data, command IDs, and master IDs. The CWD and RD arbiters resolve any bus access exclusivity problems. The arbitration algorithms are independent of the wrapper architecture and should be optimized to meet the requirements of the SoC. An example application is described in Section 4.5.

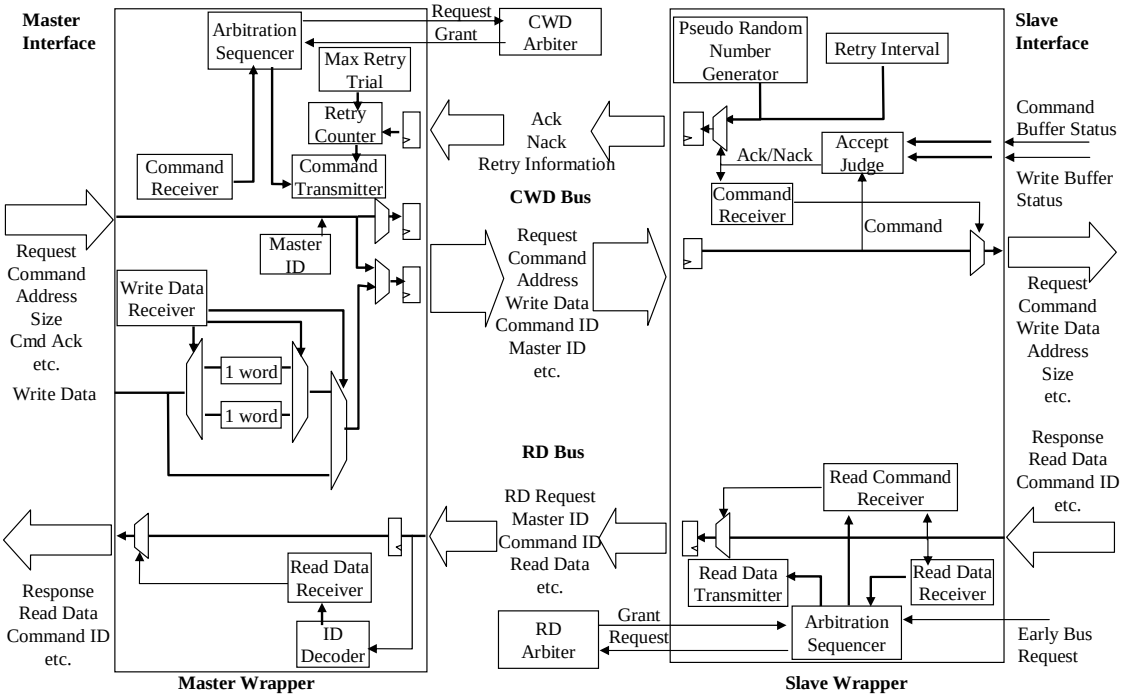


Figure 4-4 Developed wrapper architecture

The bus processing sequence is described below.

1) A master wrapper (MW) receives the command, address, command ID and size from the master core through a master interface. The command ID is used to issue multiple simultaneous transaction requests.
2) The MW requests arbitration from the arbiter and waits until it receives a grant signal.
3) The MW transmits the received information to the CWD bus along with a unique master ID, which is originally programmed into the MW.
4) A slave wrapper (SW) decodes the command to see if the request is a Read or Write one. For a Read request, the SW checks the command buffer status on the slave interface. If the status is not busy, the SW passes the request to the slave core. For a Write request, the SW checks the command buffer status and the Write data buffer status. If both are not busy, it accepts the request and returns an Ack to the MW. If either one of

the status is not an acceptable condition, it returns a Nack to the MW with retry information. No transaction occurs in the slave interface in this case, which is different from the conventional wrapper interface implementation. The retry sequence is described in Section 4.4.2.

5) If the MW receives an Ack, it transmits a command acknowledgement signal to the master core, indicating an available condition for the next command. The next command can be fetched during simultaneous processing of the current command or Write data so that the next command can be issued instantaneously. If it receives Nack, the MW starts the retry sequence and releases the bus to another master awaiting arbitration.
6) If the transmitted command is Write and the MW receives an Ack, it transmits the Write data. It receives the Write data from the interface and transmits it to the CWD bus. The SW also receives the Write data and passes it to the slave interface. If the Write data buffer is embedded in the MW, the master core can transmit the Write data before the MW receives an Ack.

The Read data response sequence is described here.

7) If Read data is prepared in the slave core, a response request and a master ID received with the request command are issued to the slave interface.
8) The SW sends an arbitration request to the RD arbiter. Arbitration hiding mode is used to reduce the number of Read latency cycles as it is in the conventional wrapper interface [VSI01]. The arbitration request is called an early bus request (EBR) and can be asserted several cycles before the Read response request is initiated.
9) If the SW receives a grant signal from the arbiter, it receives the Read data from the slave interface and passes it to the RD bus along with the master and command IDs.
10) The MW compares the received master ID with the unique ID for each master wrapper. Only the matching wrapper receives the command ID and the response data from the RD bus; it passes them to the master interface.

4.4.1. Write-buffer switching according to write data length

Deep FIFO buffers which are included in the conventional wrapper implementations [Son02][YNL01], may not be necessary since some IP cores may have redundant FIFOs or memories for buffering. To reduce cost for any case, the bus wrapper should include a minimized data buffer to achieve the required performance. Firstly, the tradeoff between sizes of the buffer in the master wrapper is evaluated.

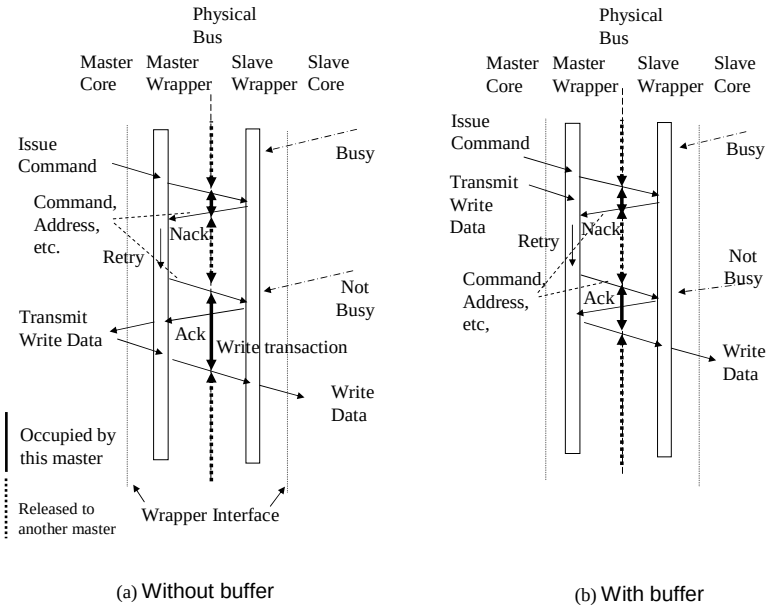
Considering the number of bits that needs to be stored, a request buffer requires 67 bits for the proposed interface, and the Write data buffer requires 128 bytes at the maximum burst size. Since the request buffer is considered as a negligible offset, and thus only the impact of the Write-data buffer is evaluated. Table 4-1 shows the amount of hardware required for a master wrapper for various buffer sizes. This wrapper handles 64-bit data, and the circuit design is optimized for 200-MHz operation with 0.15μm CMOS processes. The hardware amount is shown in NAND-equivalent gate counts, and the Write data buffer was designed as an array of flip-flops. Compared with a method that does not use a Write-data buffer, the hardware increase ranges from about 20 % to 326 % and depends on the size of the embedded buffer.

Next, the performance impact of using a Write data buffer is examined. As shown in Figure 4-5, the major difference is that latency and throughput of the CWD bus are consumed when the Write data buffer is not used, since the bus has been reserved for the upcoming Write data transfer after an Ack is detected. With a Write-data buffer, the master wrapper can output the buffered Write data onto the CWD bus as soon as it receives an Ack. Table 4-2 shows the performance impact of embedding the Write-data buffer as evaluated using RTL simulation for the case in which sustained Write transactions with the same burst size were issued. The master core could issue a command simultaneously with the transmission of the previous Write data. Since the bus assumed one master and one slave for simplicity, there were no arbitration latency. As shown in the table, the Write latency was lower and the throughputs were higher with

the embedded Write data buffer because of the instantaneous Write-data transmission following Ack reception. The Write latency was improved by about 10 % - 33 % depending on the size of the transmitted burst. The throughput was increased by about 9 % - 33 %.

Figure 4-6 shows the evaluated results of the latency and throughput increase per gate. As shown in the figure, the latency and throughput increase per gate degraded as the burst size was increased. In the implementation of a CPU-based SoC described in Section 4.5, a 16-byte buffer is used to keep the latency and throughput increase per gate fewer than 50 % degradation.

As described above, the master wrapper implementation impacts performance and has a cost tradeoff. Therefore, the Write buffer switching (WBS) technique is used in the master wrapper. As shown in Figure 4-4, the master wrapper has embedded Write-data buffers for handling two-beat bursts. With WBS, Write-data transactions shorter than or equal to the buffer size use this buffer, and longer transactions do not. The master wrapper determines whether to store the Write data in the buffer, by comparing the requested command size to the buffer size. To the best of our knowledge, conventional techniques simply embed a FIFO buffer in a wrapper or do not include any buffers.



(a) Without buffer (b) With buffer

Figure 4-5 Protocol comparison between wrappers with and without embedded Write-data buffer

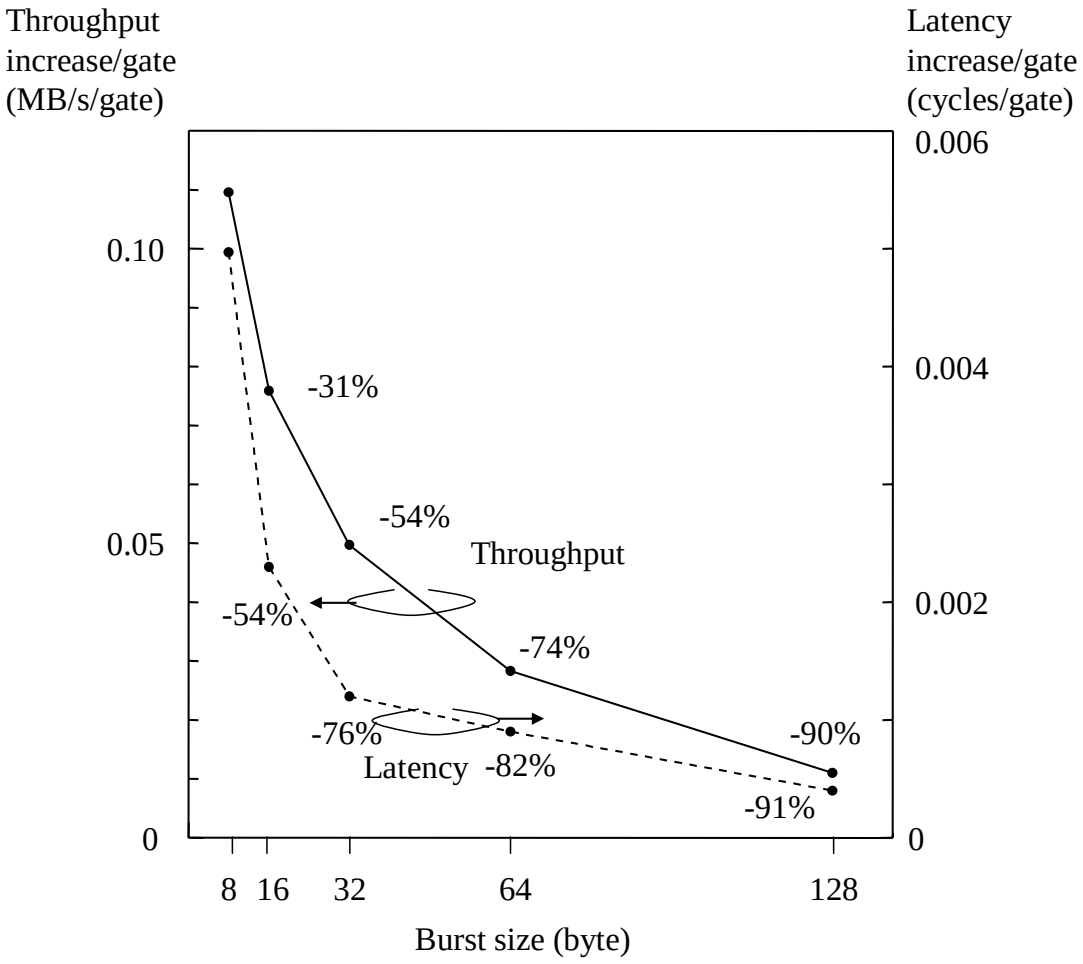


Figure 4-6 Cost and performance tradeoff of using Write-data buffer

Table 4-1 Master wrapper hardware required by Write-data buffer size

Buffer Size (byte)	*Amount of master wrapper hardware (gates)*	*Increase (%)*
0	*2976*	*–*
8	*3579*	*20.3*
16	*4275*	*43.7*
32	*5535*	*86.0*
64	*7622*	*156.1*
128	*12683*	*326.3*

Table 4-2 Performance impact of Write data-buffer in master wrapper

Burst Size (bytes)	Write Latency (cycles)			Bus Throughput (MB/s)		
	Without Buffer	With Buffer	Improvement (%)	Without Buffer	With Buffer	Improvement (%)
8	9	6	33.3	199	266	33.2
16	11	8	27.3	353	452	27.9
32	15	12	20.0	581	708	21.9
64	23	19	17.4	853	984	15.4
128	39	35	10.3	1112	1219	9.6

4.4.2. Slave designated retry control scheme

For the bus to be widely applicable, it must be able to connect to both fast and slow IP cores. Conventional standard buses can be categorized as high- and low-performance buses. Using the two types together requires complicated bus bridges, which can potentially suffer from deadlock [Pci01]. The wrapper-based approach eliminates this requirement, but communication between fast and slow cores is problematic, especially for retry sequences.

In conventional implementations [Son02][ZC02], a retry sequence is controlled by a master wrapper, and the intervals for retry requests are determined master by master, independently. The length of the interval after receiving a Nack from a slave wrapper affects performance [ZC02]. When fast and slow slaves are connected to the same bus, the interval before the slaves become available for another command differs. Therefore, using a constant interval for retry could degrade bus throughput when using a short interval or lengthen the access latency when using a long interval. To handle this problem with existing physical buses, the master wrappers must decode the addresses to determine the destination before transmitting requests to slaves and change the interval slave by slave. This requires dedicated wrappers that support different address maps in each SoC.

In this research, a retry control technique called slave designated retry control (SDRC) is proposed, which is illustrated in Figure 4-7. With this mechanism, when a retry occurs, the slave wrapper specifies the number of retry interval cycles which is used for the master wrapper to wait before re-issuing the request. The slave wrapper interface includes a retry interval signal that specifies the number of interval cycles which is based on the slave's speed. The physical bus layer is designed to convey the retry interval information over a retry information signal.

Consider two cases: a master wrapper communicates with fast slaves and with slow slaves. Shown in Figure 4-7 (a) is an example of when a slave wrapper can process one command within a few clock cycles. With this slave, an eight-cycle interval is specified as a core attribute of the slave. The slave wrapper returns a Nack with the retry information, including the specified retry interval of eight cycles. The master wrapper sets eight cycles for the retry interval and counts down to zero. It then re-sends the rejected request and address to the slave wrapper. By this time, the slave is not busy and accepts it. Figure 4-7 (b) shows an example of when the slave wrapper has a longer latency, 64 cycles, or a slower clock frequency. In this case, a longer retry interval is specified, which reduces the number of unnecessary retries. In this

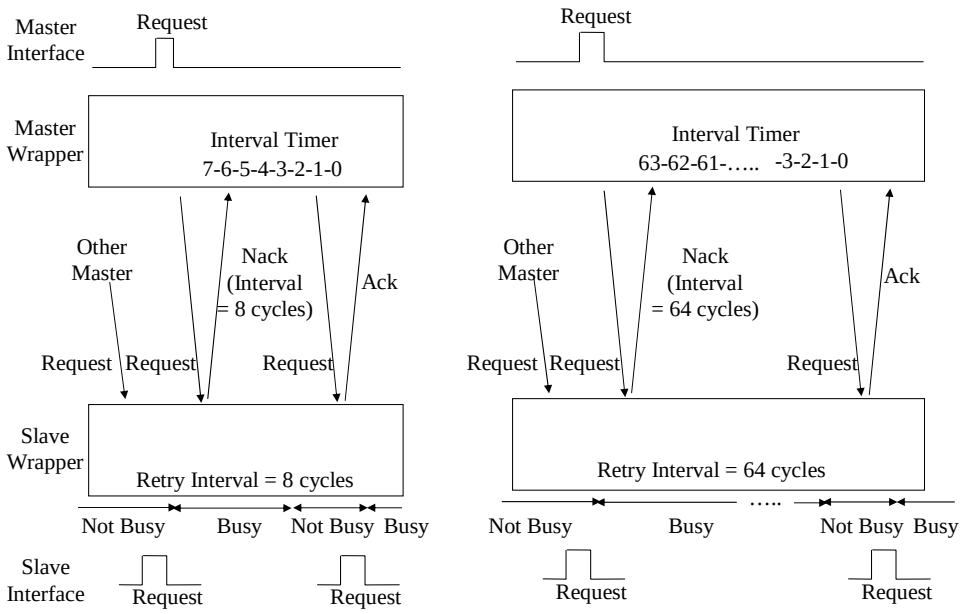
example, our scheme removes 7 unnecessary retries by using an interval of 64 cycles.

Another complicated problem regarding retry is livelock. Livelock is the case in which one master wrapper cannot access a slave while another can. Livelock is also referred to as the “starvation problem” [Son02]. In an Ethernet network, this is the well-known “Capture Effect” [RY94]. Ethernet does not arbitrate before transmitting, so a master may not be able to achieve good performance due to frequent collisions. A scheme similar to Ethernet has been applied to a bus [DG98]. Each master wrapper has a random number generator, and if a master receives a retry response from an accessed slave, it waits for the interval specified by the generated random number. As specified in the Ethernet standard, the master should generate the random number based on the number of rejected requests to avoid synchronizing the retry timing. Another possible solution to avoid livelock is having access queues in slave wrappers to remove back-off retry accesses [Arm99]. However, controlling the queue in the slave wrapper requires complicated logic and area overhead for the buffer. Our approach is to take the back-off retry sequence and extend this structure with small hardware to avoid livelock.

In the livelock example shown in Figure 4-8 (a), two master wrappers are trying to send Read requests to a slave core. The requests are serialized by the CWD bus. The request from Master 1 arrives first. It is accepted by the slave, and the slave enters a busy state. Therefore, when the request from Master 2 arrives, it is rejected. A rejection response is sent to Master 2, and a retry sequence is initiated. In the meantime, a Read response is sent to Master 1, and Master 1 sends the next command to the slave. Master 2 re-sends the retry Read request to the slave, and it is again rejected. This sequence continues until the retry trial counter overflows in Master 2, which may be reported as a bus error. In this example, only Master 1 can access the slave, and this is a livelock.

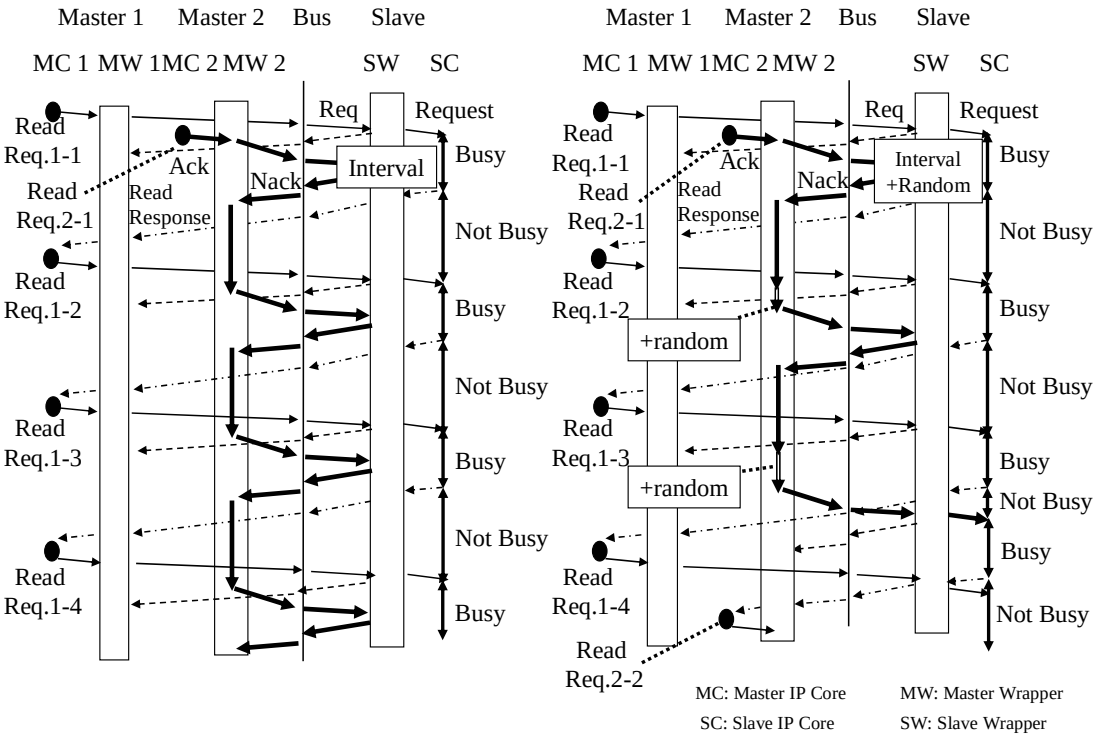
Our solution is to use a pseudo random number generated in the target slave. By generating the number in the slave, randomness for all accessing masters is ensured and synchronization of the retry timings is avoided. In addition, this

random number is used as an additional offset for the retry interval. Our SDRC can be easily implemented with only a small adaptation. As Figure 4-4 shows, the slave wrapper includes a random number generator. The number is sent to a master wrapper by using the retry information signal of the CWD bus. The master wrapper adds this number to the retry interval number specified by the slave. By using this technique, we can avoid the livelock problem, as shown in Figure 4-8 (b). The proposed livelock avoidance technique issues no throughput degradation, but small latency overhead due to the latency increase by a random offset for retry interval cycles. However, low-cost implementation is achieved by simply extending the back-off retry structure with a random number generator which requires only a few shift registers and exclusive-OR gates.



(a) Fast slave (b) Slow slave

Figure 4-7 Slave designated retry control (SDRC) technique



(a) Livelock condition (b) With SDRC

Figure 4-8 Livelock avoidance in SDRC with random interval

4.4.3. Converter-based multiple-bit-width core connection

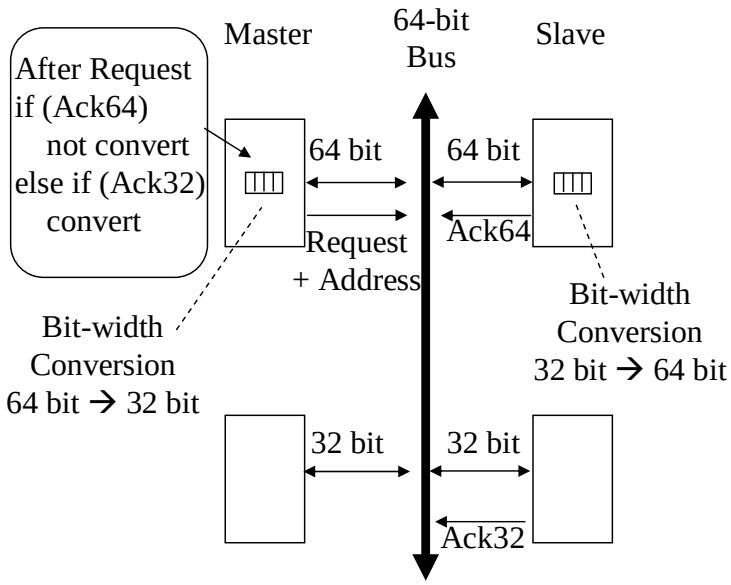
Data-width conversion should be supported for connecting various bit-width cores because changing the bus width of an IP core and re-verifying it are burdens for SoC designers. Therefore, the ability to enhance the bus bit-width is needed to be retained as well as a low-cost implementation.

Two conventional data-width conversion techniques are illustrated in Figure 4-9. Figure 4-9 (a) shows the one used for a PCI bus [Pci01]. This bus requires different acknowledgement signals that indicate the supported data width of a slave for reporting this to the master, and the bit-width is converted in a master. Therefore, for supporting further bit-widths, this technique requires modifying the physical bus specification and the master core logic for communicating with the slave. Figure 4-9 (b) shows the technique used in the conventional wrapper-based bus [Son02]. The strategy is for a wrapper to convert the bit-widths when necessary. When converting the bit-width, it must handle the bandwidth gap between the interface and the bus. Thus, it requires a FIFO buffer to absorb the gap, where a large amount of hardware is needed. To be able to enhance the supported bit-width, different wrappers must be developed.

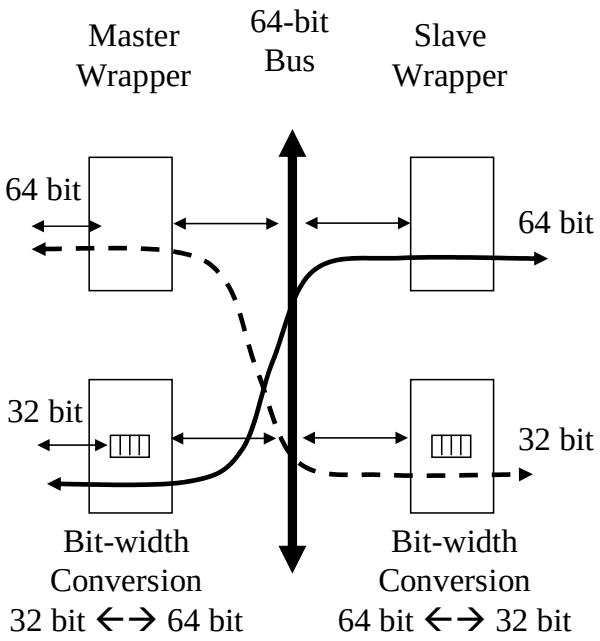
To convert the data-width while keeping the ability to expand it and to achieve a low-cost bus implementation, a data-width converter is embedded in the physical bus. As illustrated in Figure 4-10, two 64-bit master wrappers and two 64-bit slave wrappers are connected using a 64-bit CWD bus and a 64-bit RD bus, while two 32-bit master wrappers and two 32-bit slave wrappers are connected using a 32-bit CWD bus and a 32-bit RD bus. The 32-bit and 64-bit CWD buses are connected using a CWD bit-width converter, and the 32-bit and 64-bit RD buses are connected using a RD bit-width converter. The CWD and RD bit-width converters monitor the bus traffic, detect cases that require data-width conversion, and convert the data as required. With our approach, the master and slave wrappers do not need to recognize the bit-width capability of the destination. Thus, bit-width enhancement is easier than with the conventional techniques, and data buffers are not needed inside the wrappers

for bit-width conversion. Thus, the total bus system requires less hardware compared with existing approaches.

Figure 4-11 (a) shows the CWD converter architecture. The RD converter is not shown, as it is similar to that of the CWD. It receives request, command, size, Write data, ack, and nack signals from both the 32- and 64-bit CWD buses. The request, command, size, ack, and nack signals of each bus are passed to the other bus to share the same bus cycle in these buses. The required condition for data-width conversion is detected in the conversion condition detector (CCD). When the CCD detects a bus cycle in which data needs to be converted, the Write data is input into a FIFO buffer. This buffer supports data packing and unpacking, such as 32-bit input and 64-bit output, or 64-bit input and 32-bit output. The converted Write data is then transmitted to the other bus from the one where the original data came. Figure 4-11 (b) shows the CCD sequencer for the CWD converter. The detect condition is a Write command received from either the 32- or 64-bit bus that is acknowledged from the other bus. When this particular condition occurs, the converter translates the Write data from 32 to 64 bits, or from 64 to 32 bits. Additional bit-widths can be supported by enhancing the architecture.



(a) Conversion in 64-bit units



(b) Conversion in 32-bit wrappers

Figure 4-9 Conventional data-width conversion schemes

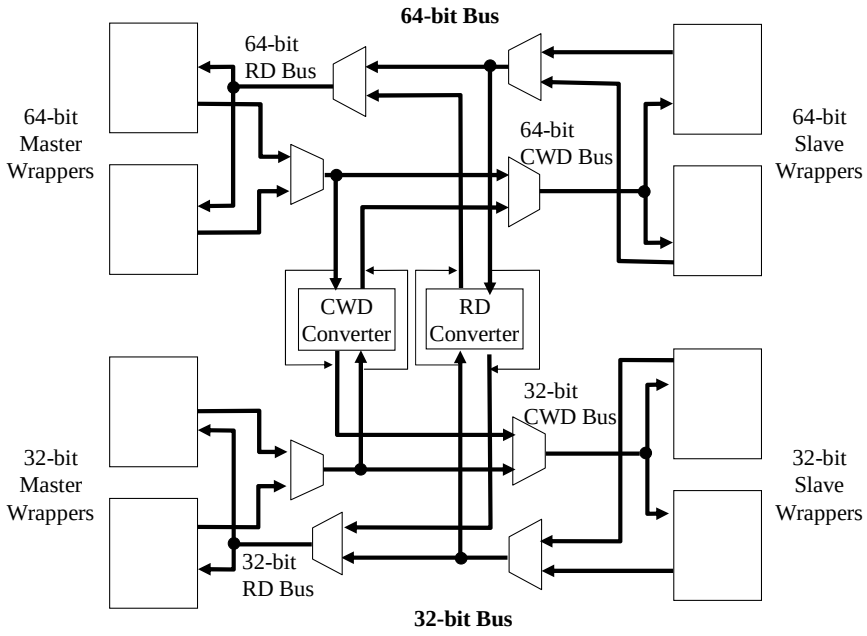


Figure 4-10 Bus circuit integrated with data-width converters

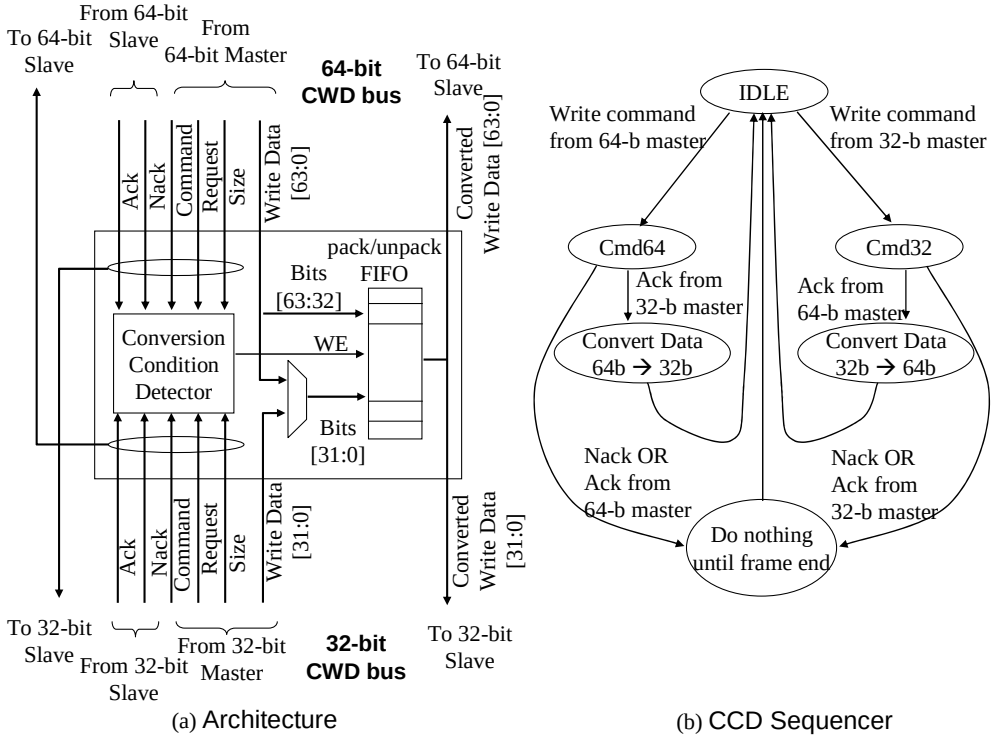


Figure 4-11 Bit-width converter architecture for CWD bus

4.5. SoC implementation and evaluation

4.5.1. SoC architecture

The developed wrapper-based bus is applied to a CPU-based SoC [Oka02]. As shown in Figure 4-12, this SoC has a 400-MHz 64-bit 2-issue out-of-order superscalar processor core, 256-KB level-2 cache, and a DDR-SDRAM interface. The SDRAM is accessible through the L2 cache from an on-chip bus. Along with the on-chip bus, a CPU core, a PCI-X interface, two 10/100-base Ethernet MACs, a local bus interface, and a performance monitor are connected as five masters and seven slaves. This SoC is targeted for use as a network packet controller in various systems. All the function blocks except the Ethernet ones have 64-bit data width; and the Ethernet blocks have 32-bit width. Thus, the on-chip bus requires an internal data-width converter. The bus operates at 200 MHz, and its targeted maximum throughput is 800 MB/s. It is fabricated using 0.15µm CMOS processes.

The unique features of the bus for this SoC are its arbitration algorithms and the Write-data buffer size. The arbitration algorithms are determined by taking the expected traffic patterns into account. For the CWD arbiter, a simple round-robin algorithm is used, and the parking master is chosen as the same master which used the bus last time. For the RD arbiter, the same algorithm is used, but the parking master is always an L2 cache because the outgoing and incoming packets of network routing applications are always from or to the L2 cache or SDRAM. The size of the Write-data buffer in a master wrapper is two-beat bursts, considering the cost-performance tradeoff, as described in Section 4.4.1. Also, in these applications, the CPU frequently issues two-beat burst Write transactions to the interfaces to maintain the registers for the purpose of DMA control, status monitoring, etc. Longer burst Write traffic is not latency sensitive, but is throughput sensitive.

Taking this SoC as an application example, its bus performance and the efficiency of the developed techniques are evaluated. Here, the evaluated impact of the SDRC and WBS techniques, the integrated bus performance, and the cost of the designed bus are presented. Using an evaluation method similar

to the one used previously [ZC02][LRL02], a simulation environment is established by using the RTL design of the developed bus. The environment consisted of the bus RTL design, a master model that could generate a maximum of eight simultaneous non-blocking Read requests, and a slave model that could receive one command at a time. In addition, the interval is set for receiving the next available command as a parameter in the slave model. The constructed platform was based on the five-master seven-slave bus system.

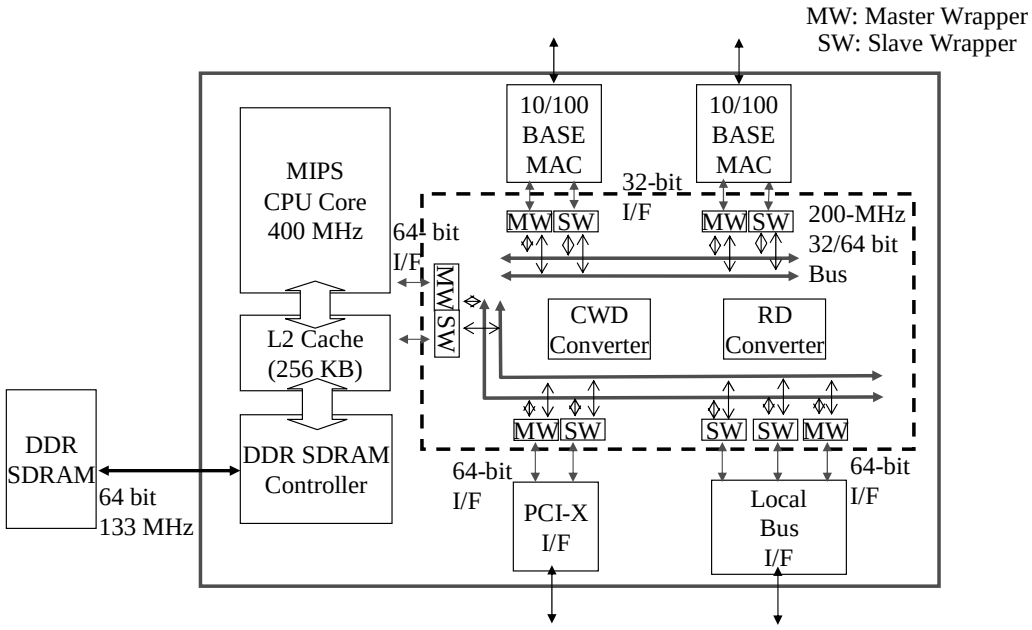
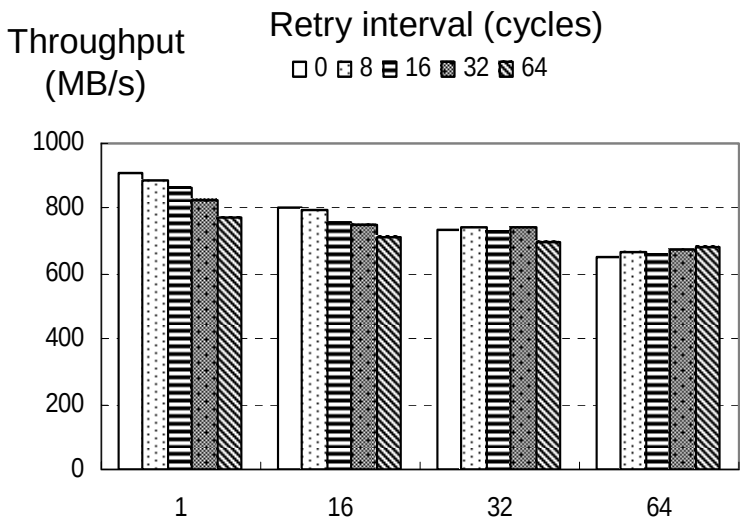


Figure 4-12 CPU-based SoC with developed wrapper-based bus

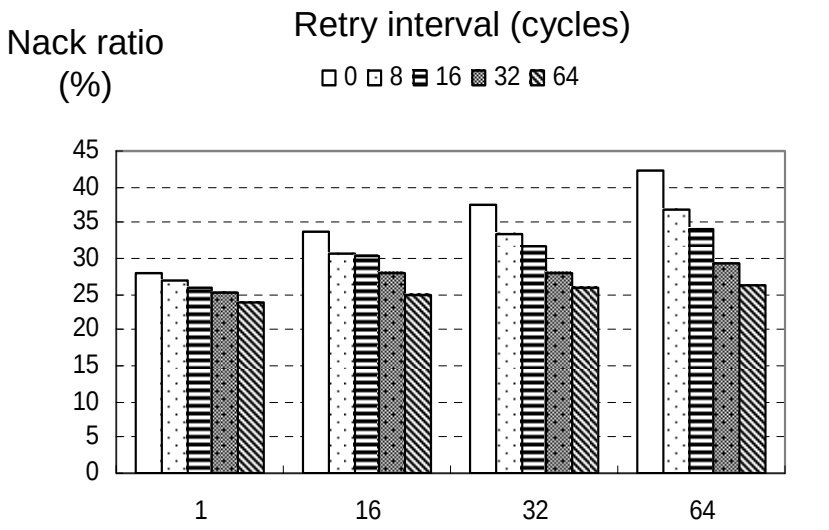
4.5.2. SDRC evaluation

Figure 4-13 shows the performance impact of the retry interval when using the SDRC mechanism. The modeled traffic is for a case when a master core accesses a fast and a slow slave, at the same time another master accesses the same fast slave core. This model is similar to that of a CPU accessing a slow I/O device while traffic flows into the SoC from the network interfaces. 75%

of the traffic from the master operating as a CPU was for the slow slave core and 25% was for the fast one. The traffic of the other master was randomly generated, and the ratio of Read and Write commands was even. Each slave core could handle one request at a time. All the master and slave cores were designed as 64-bit units for simplicity. Figure 4-13 (a) shows the throughput for four intervals to wait for the next available buffer, and five retry intervals. When slaves became available in 1 and 16 cycles, the throughput decreased as the retry interval was increased. When they became available in 32 and 64 cycles, the tendency changed. When slaves became available in 64 cycles, the bus with a 64-cycle retry interval had the highest throughput. Figure 4-13 (b) shows the corresponding Nack message ratio. The ratio difference increased with the slave available interval due to the increased number of Nack messages resulting from the too short retry interval. The differences between 0- and 64-cycle retry intervals were 3.9% and 15.9% in the 1-cycle and 64-cycle available intervals.



(a) Throughput



(b) Nack ratio

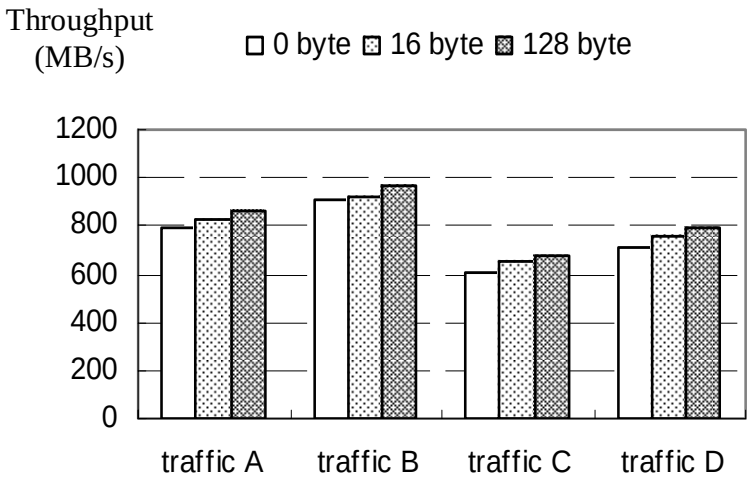
Figure 4-13 Performance impact of retry interval

4.5.3. WBS evaluation

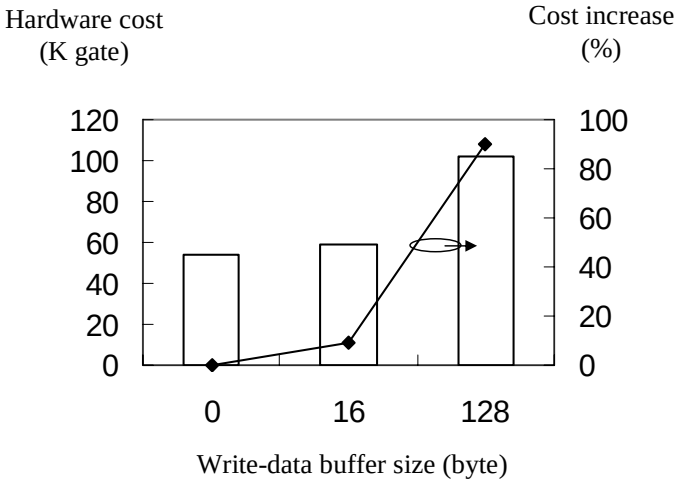
The throughput is evaluated for three sizes of the Write buffer in a master wrapper for four traffic patterns. Traffic pattern A was the average case in which all five masters accessed the slaves using bursts with random sizes from five supported sizes up to 128 bytes. In pattern B, one high-performance master required mostly longer burst transfers, while the other masters required shorter burst transfers. In pattern C, one master generated the same transactions which are used in the SDRC evaluation as a CPU model, while the other masters generated shorter burst transfers with request intervals up to 20 - 30 cycles. Pattern D was the modeled traffic of the targeted SoC. One master was modeled as a CPU, as in pattern C; another master was modeled as a PCI-X interface that required frequent DMA transactions from off-chip I/O devices. The other masters required shorter burst transfers corresponding to those of Ethernet interfaces.

All the master and slave cores were connected as 64-bit units to exclude the effect of converters and to focus on the WBS effects. With the developed WBS technique, using a 16-byte buffer improved the performance by about 1% to 9%, while a 128-byte buffer improved it by about 6% to 12%. The improvement was good for pattern C because the traffic was mostly short-burst transactions. There was little improvement for the pattern B due to the frequent long-burst transactions.

The hardware cost is estimated for a bus system comprising five 64-bit master wrappers, seven 64-bit slave wrappers, two decoders, multiplexers, and two arbiters. As shown in Figure 4-14 (b), the required hardware cost was approximately 60K gates for the master wrappers with 0- and 16-byte buffers, while the wrapper with a 128-byte buffer required 102K gates. With a 16-byte buffer, the WBS technique only imposed a hardware overhead of only 9%, while with a 128-byte buffer it imposed an 89% overhead. The WBS technique achieved better performance than one with no buffer wrapper, and it required a smaller hardware cost than a wrapper with a 128-byte buffer.



(a) Throughput



(b) Hardware cost

Figure 4-14 Performance and cost impact of Write-data buffer in master wrapper

4.5.4. Total bus evaluation

Table 4-3 shows the evaluated throughput of a bus system developed for the SoC. The system had two 32-bit masters, two 32-bit slaves, three 64-bit masters, three 64-bit slaves, a CWD data-width converter, and an RD data-width converter, with a conventional arbitration hiding technique, early bus request (EBR), using our WBS technique and a flow-controlled interface (I/F). The traffic patterns were the same as for the WBS-only evaluation (Section IV A). As Table 4-3 shows, applying all three techniques increased throughput by about 12 % to 16 %. With EBR alone, throughput was reduced in two cases. This was because advance arbitration requests unnecessarily occupied the RD bus, blocking the other transactions. In this application, most transactions were destined for the L2 cache, which could respond more quickly, so the impact of EBR was quite small. Table 4-4 and Table 4-5 show the Write and Read latency in cycles/word, similarly to a previous publication [LRL02]. The Write latency improved by about 14 % to 20 %, while the Read latency improved by about 8 % to 15 %.

The network application performance of the designed chip using the evaluation board is measured, where a gigabit Ethernet interface card is connected to a 133-MHz PCI-X bus of it. The photo of the evaluation system is shown in Figure 4-17. Linux runs on the CPU as routing software, and it achieved 500 MIPS when running a performance measurement tool of Linux. As routing operation, the CPU receives packets coming through the PCI-X bus and transmits them to the card through the PCI-X bus. For comparison, the same application has been evaluated on a personal computer which has 600-MIPS processor and a 33-MHz 32-bit PCI bus for connecting gigabit Ethernet interface card. As shown in Figure 4-16, the routing throughput result of the 500-MIPS processor linearly grows when the packet size increases. The 500-MIPS processor routes almost the same number of packets per seconds as the 600-MIPS processor does in small-sized packet cases. However, when the packet size increases, the PCI bus becomes a performance bottleneck. The designed on-chip bus used in the designed 500-MIPS CPU-based SoC is not performance critical in the gigabit

Ethernet routing application, due to its enough performance budget for this application range.

Table 4-3 Throughput of designed bus

Traffic Pattern	Base	Early Bus Request (EBR)	EBR + I/F	EBR + I/F + WBS	Improvement (%)
A	600	601	653	672	12.0
B	785	781	849	887	13.0
C	487	489	542	565	16.0
D	579	575	633	660	14.0

Table 4-4 Write Latency of designed bus

Traffic Pattern	Base	Early Bus Request (EBR)	EBR + I/F	EBR + I/F + WBS	Improvement (%)
A	9.0	8.9	8.0	7.5	16.7
B	9.9	9.5	8.8	8.5	14.1
C	9.8	10.0	8.5	7.9	19.4
D	9.2	9.3	7.9	7.7	16.3

Table 4-5 Read Latency of designed bus

Traffic Pattern	Base	Early Bus Request (EBR)	EBR + I/F	EBR + I/F + WBS	Improvement (%)
A	9.2	9.5	8.6	8.3	9.8
B	10.0	9.9	9.5	9.2	8.0
C	9.9	9.8	8.7	8.5	14.1
D	9.2	9.2	8.7	8.2	10.9

4.5.5. Cost evaluation

Figure 4-15 shows the hardware cost in gates of three data conversion methods for a total bus system comprising two 32-bit masters, two 32-bit slaves, three 64-bit masters, and five 64-bit slaves. The conventional technique requires data buffers in either the 32-bit or 64-bit wrappers. The 32-bit and 64-bit wrapper-based conversion methods required 85K gates and 122K gates. Our converter-based method required only 61K gates, so it decreased the hardware cost by 28 % and 50 % compared to two existing methods.

A die photograph of the designed SoC is shown in Figure 4-18, and the layout plot of the developed wrapper-based bus is shown in Figure 4-19. Our wrapper-based bus occupies a 3.3-mm^2 L-shaped area, and all the bus connections are contained in this area. The CWD and RD converters consume large part of this area due to their embedded buffer for data conversion. The wrappers require a smaller area due to our no-data-buffer implementation. This chip is fabricated using 0.15μm CMOS processes and has six metal layers. The chip package is a 500-pin Advanced BGA. The power voltage is 1.5 V for the core transistors and 2.5/3.3 V for the I/O transistors.

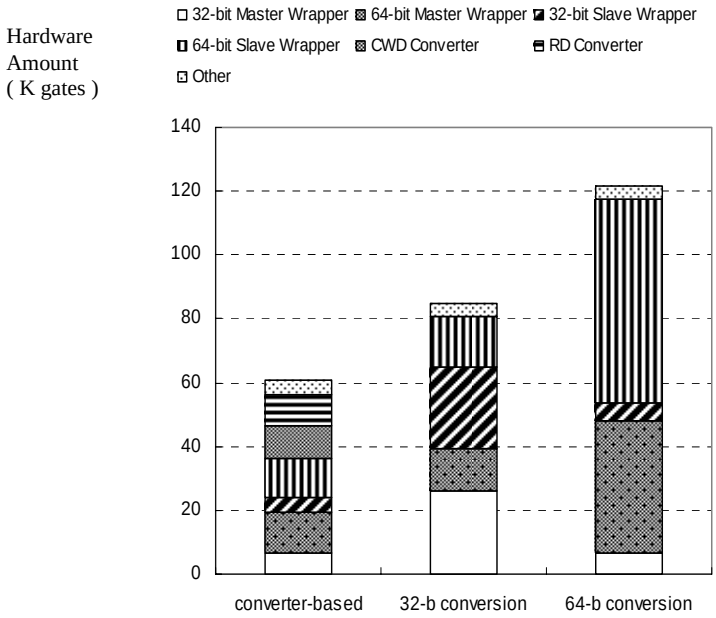


Figure 4-15 Hardware cost of three data-conversion methods

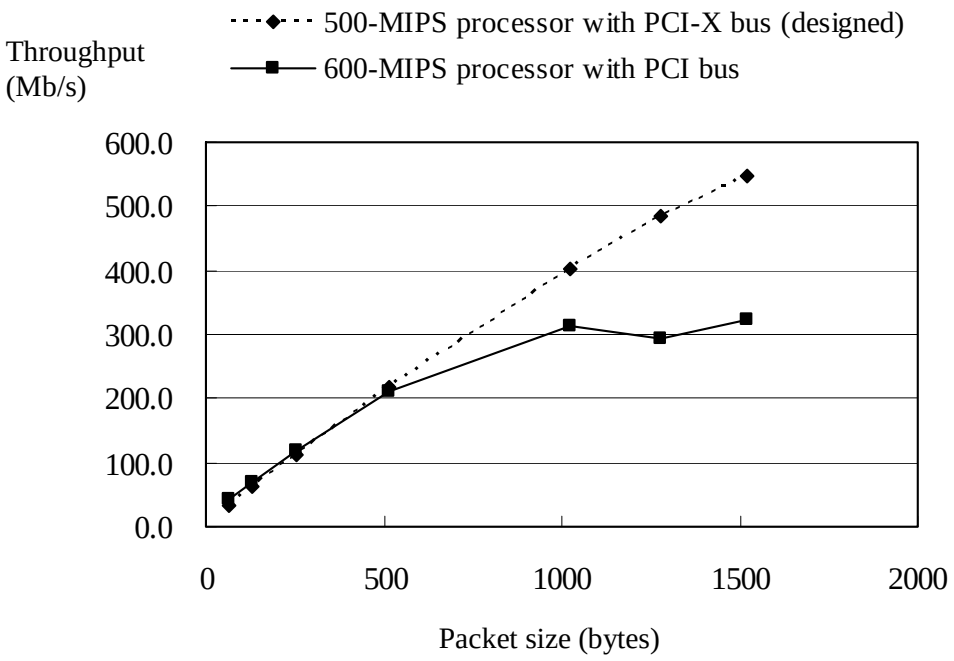


Figure 4-16 Measured throughput of 1Gb/s Ethernet routing function

Figure 4-17 Photo of routing evaluation system

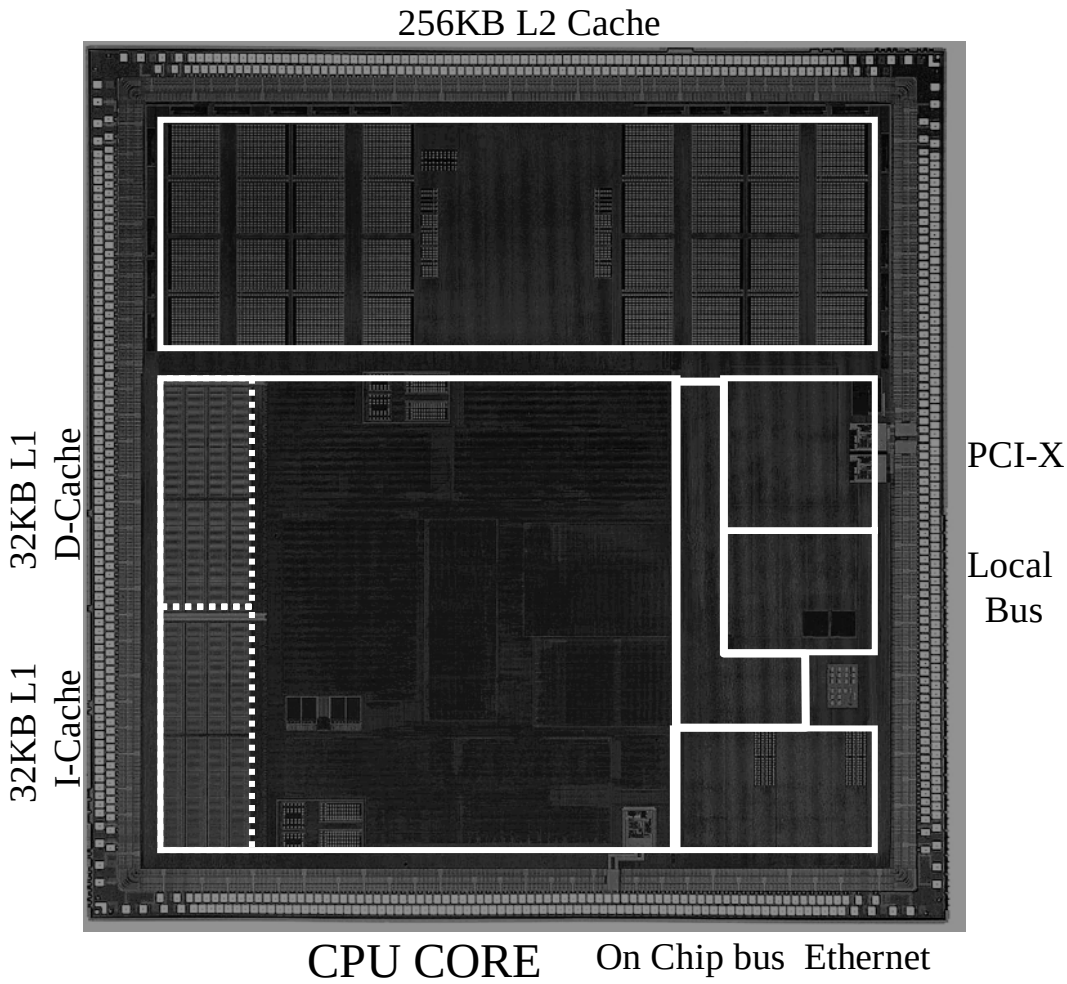


Figure 4-18 Die photograph of the designed SoC

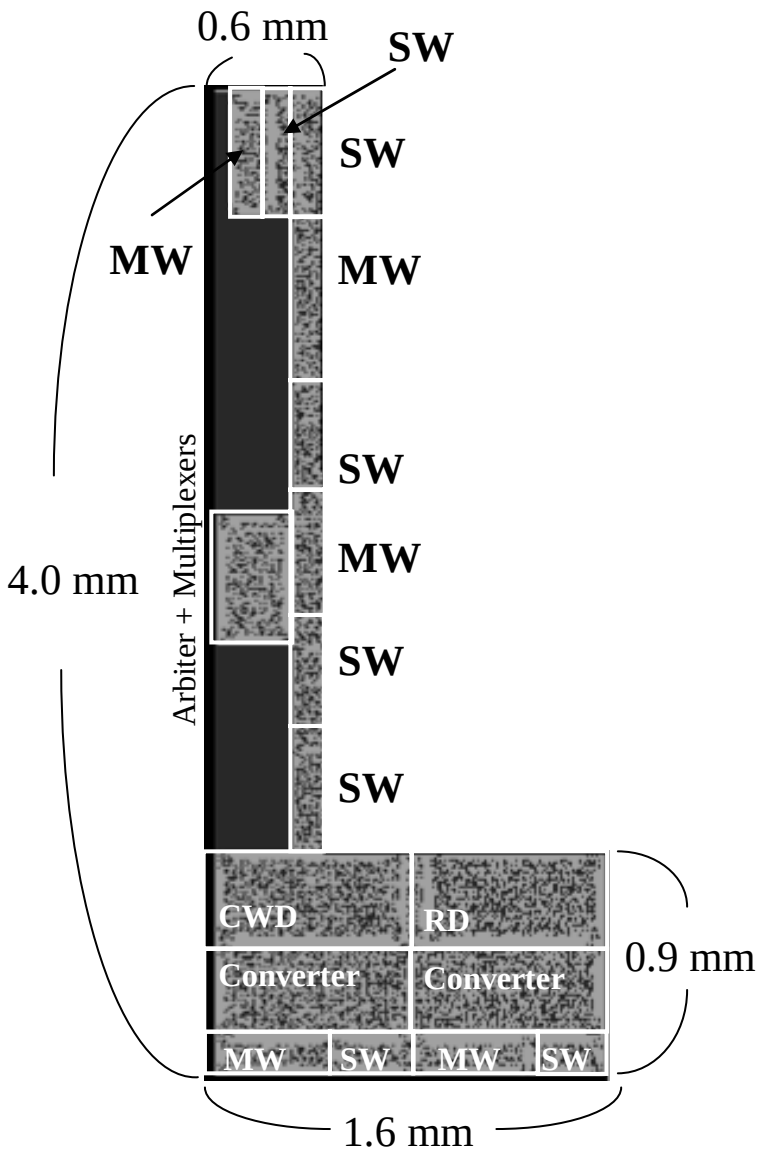


Figure 4-19 Layout plot of the on-chip bus in the SoC

4.6. Conclusion of this chapter

This chapter described a wrapper-based bus implementation that has practical performance with low hardware cost. The wrappers do not require a data buffer, and our wrapper interface supports the status signals of the request and Write data buffers, and the retry interval signals in each slave IP core.

Three novel wrapper-based bus implementation techniques are also described. The Write buffer switching technique increases throughput and reduces Write latency. There is a tradeoff between the master wrapper hardware cost, throughput, and Write latency for sustained Write transactions, so a guideline for determining the optimal buffer size by evaluating throughput and latency per gate, is developed. The second technique is called “slave designated retry control”. With this technique, the retry interval before a master wrapper re-issues a request is determined slave by slave according to the response speed. The number of retry intervals affects the overall throughput and negative-acknowledgement ratio. Furthermore, a livelock avoidance scheme that can be easily implemented by simply adding pseudo random number generator, has been developed. The third technique converts the data-width to enable IP cores with different bit-widths to be connected at a small hardware cost.

These techniques for a CPU-based SoC designed for networking applications are evaluated. For a bus system with two 32-bit masters, two 32-bit slaves, three 64-bit masters, and five 64-bit slaves, using our WBS technique and the proposed flow-based interface increased throughput by about 14 % compared to using a conventional wrapper-based implementation for the traffic pattern modeled for the targeted network application. It also reduced the Read and Write latency by about 16 % and 11 %. A hardware cost evaluation showed that our converter-based technique can reduce hardware costs in terms of gates by 28 % or 50 % compared with two conventional conversion techniques. A chip based on these techniques was implemented in 0.15μm CMOS process technologies; the area for the on-chip bus is 3.3 mm^2, and the operation frequency is 200 MHz.

Chapter 5 Efficient data-transfer schemes for Network-on-Chip

This chapter describes a novel data-transfer technique using local labels in NoCs for programmable devices. In Section 5.1, overview of this proposal is described. Next, in Section 5.2, a data transfer scheme using separate routing information is shown. Then, in Section 5.3, a novel routing scheme using local labels is proposed. In Section 5.4, evaluation results are shown, and Section 5.5 concludes this chapter.

5.1. Overview

In sub-0.1 μm CMOS generations, SoCs hit difficulties for design methodologies due to large number of transistors. Apart from the SoCs, programmable devices seem better solution instead of developing ASICs from the aspect of design and NRE costs. Thus, programmable devices in current generation such as FPGA and CPLD raise its market shares. Also, as future technologies, dynamically reconfigurable processors such as ACM, DRP, DAP/DNA, appear for better logic gate density than the current generation.

By those programmable devices and reconfigurable processors, design and fabrication costs are greatly reduced compared with the case when developing ASICs. However, hardware logic gate is emulated in those programmable approaches, while raw cell-based gates are used in ASICs. This difference of gate densities will affect the chip cost. So, in programmable devices, reducing chip area becomes a critical issue to achieve high credibility.

As described in Section 2.3, the key factor for reducing chip cost in programmable devices is an on-chip interconnection network, called

Network-on-Chip (NoC). The programmable devices in current generation employ the programmable switch as NoC. Its structure is basically a programmable crossbar, and it consumes large amount of hardware [DR04].

Another structure of NoC is using network routers [DT01][MBV02][GG02]. Although the programmable switch is an architecture which allows a single switch to be used by a single logical connection between a source and a destination. Here, by acquiring a scheme to transfer data used in network routers, which has been used in System Area Networks [SH96][BCF95][Ita01] in parallel computers or clusters, the required hardware amount is expected for reduction.

Although conventional NoCs simply applied SANs to SoCs as described in Section 3.6.2.3, environmental requirements of SoCs must be considered for less hardware amount. Three major environmental differences are that 1) environment is application oriented, 2) wire resources are rich and flexible, and 3) cost sensitivity. Taking these requirements into account, this research presents a novel data-transfer scheme for low-cost NoC implementation and improving performance as an interconnection network of programmable devices. First, a data-transfer scheme using separate routing information which takes advantage of rich wiring resources on chip, is presented for cost-efficient implementation. Then, a novel local labeling scheme for specifying destination in a fewer bits than the conventional global addressing scheme is proposed to reduce hardware amount of routers. Comparison between required hardware amount for the programmable switch and the router-based NoC is also shown.

5.2. Data-transfer using separate routing information

An example of a NoC structure in 2-D mesh topology is shown in Figure 5-1. It consists of network routers and IP cores. Each connection between neighboring routers and node/router is uni-directional, and it includes data and control lines. A set of data and control signals are called “channel” in this thesis. As an IP core (node), a microprocessor, a reconfigurable unit such as programmable

logic or processor arrays, a memory or hardware logic can be placed, and these nodes communicate each other.

Figure 5-2 shows a timing diagram of a data-transfer with packet structure used in conventional NoCs. On a channel, a packet header which includes fields at least for routing information and data length is transferred first, and then data itself and a packet tail follow. Packets are transferred in the unit of “flit” which corresponds to the data size equal to the bit-width of data signal in routers and nodes. This data-transfer scheme requires composing and decomposing packets in nodes and handling the packet structure in routers.

However, as described in the previous section, flexible wire resources can be used in NoCs, differently from SANs which have pin count limitation of chip packages. So, we can leave from the packet structure and this research utilizes a data-transfer scheme using separate routing information. As shown in Section 5.3, this scheme uses a dedicated signal for transmitting routing information in parallel to data. This apparently results in removing the cycle overhead for sending headers, and also simplifies router and node hardware from those for conventional packet data-transfer.

Figure 5-4 (a) and (b) show packet formats used in the packet data-transfer and the data-transfer using separate routing information. As shown in Figure 5-4 (a), a header is sent as a first flit of the packet and it consists of at least routing information and length field for specifying payload size. And, the tail flit is transmitted if necessary. As shown in Figure 5-4 (b), data which is transferred by a node is split into multiple of data to be transmitted in a cycle, and routing information which specifies destination is transmitted in parallel. In this scheme, length is not necessary in routers, and the upper layer logic in IP cores may or may not need it. Thus, this is meant to be the field in payload.

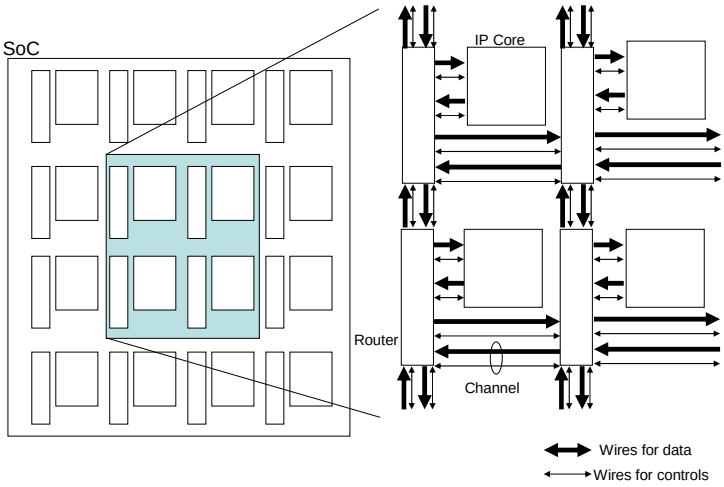


Figure 5-1 Structure of 2-D mesh NoC

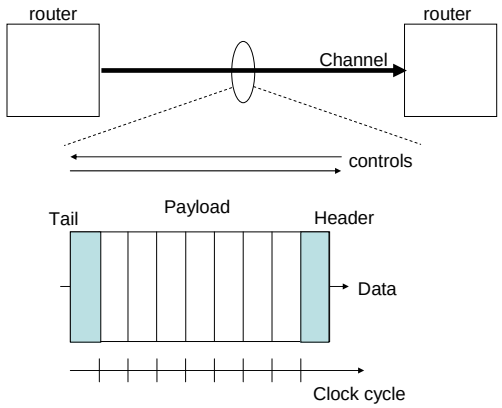


Figure 5-2 Data-transfer with packet structure

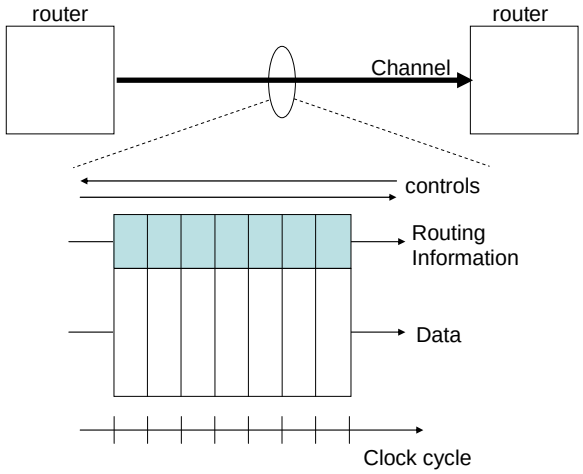


Figure 5-3 Data transfer scheme using separate routing information

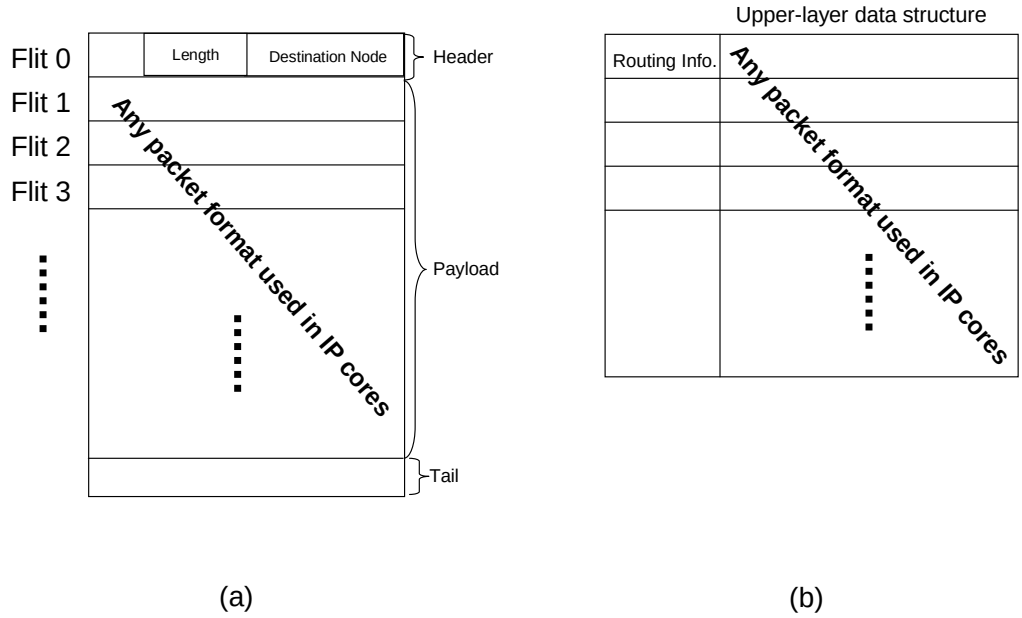


Figure 5-4 Packet structure comparison (a) packet data transfer (b) separate routing information transfer

5.3. Proposal of local labeling scheme

5.3.1. Conventional distributed routing using global addresses

Global addresses have been typically used to specify destination in conventional NoCs [PW04]. This conventional global addressing scheme simply attaches a global node number of a destination to transmitted data. In routers, each packet looks up routing tables and gets a port number to forward.

A routing function for the conventional distributed routing using node addresses is expressed with C, as a set of channels, and N as a set of nodes:

$$C \times N \rightarrow C$$

This function receives a set of input channel and a set of global node number as inputs, and outputs a set of cannel number as a result [DS87]. The bit amount required in this algorithm is $\lceil \log_2 N \rceil$, where N is a number of nodes.

This global addressing scheme is quite general purposed, since any of nodes can transmit packet to any other nodes. However, although this scheme is simple and flexible, it requires number of bits according to the number of nodes depending on a number of nodes in SoC. For example, the 4-bit routing information is required in 16 nodes, and 6 bits in 64 nodes.

5.3.2. Local labeling scheme

In many SoCs, applications are pre-determined and specified before chips are fabricated. So, we can take advantage of static analysis results of communication patterns in applications. This research proposes a method to utilize static analysis to reduce hardware cost by optimizing routing information field in the data-transfer scheme using separate routing information.

The proposed method for reducing routing information is applying a local labeling scheme. The local labeling scheme utilizes static analysis results of communication patterns to reduce a required number of bits for routing tags and routing table entries. As an identifier for specifying a destination, a local label which is only valid in a channel between a certain pair of neighboring routers or node/router, is used instead of global node addresses. Since the local label is a value which is only valid in a single channel, the same value can be used in a different channel for specifying a different destination. And thus, this local labeling scheme can reduce the required number of bits for specifying destination nodes. Compared with a conventional distributed routing, hardware cost becomes fewer when the maximum number of required local labels in channels is less than the number of nodes.

Here, algorithms to calculate the number of required local labels are shown in the following sections. Overall in an NoC, the maximum number of local labels required in all the channels is called “Crossing Path (CP)”, in this research. One algorithm shown is constant labeling scheme, and the other is renewable local labeling scheme [AYK04].

5.3.3. Static analysis of communication pattern

First of all, an algorithm to extract communication paths from an application traffic pattern is shown. A communication path is defined as a set of channels from a source node to a destination node. Before the analysis, an application must be split into multiple of tasks and each of them is mapped onto a certain node, by any mapping algorithm. Below is the static analysis algorithm to extract communication paths.

1) A logical counter is associated to each channel. All the counters are initialized 0.
2) A set of communication pairs of nodes is prepared. Choose one from the set of communication pairs which are not analyzed yet.
3) Any routing algorithm is applied to the chosen pair, and the communication path is established by being routed on a sequence of channels determined by the algorithm.
4) Increment all the counters which the communication path has.
5) Repeat from 2) to 4), before completing the analysis of all the communication pairs.

Each counter value after this analysis shows the number of communication paths routed on each channel, and thus, equals to the required number of local labels in each channel. So, the maximum number of the entire counters after the analysis becomes theoretical Crossing Path (CP).

An example of analyzed communication paths in a 3x3 2-D mesh network is shown in Figure 5-5. Each circle shows a router and unidirectional channels are connected between neighboring nodes. Nodes are not shown for simplicity. As shown in this figure, the theoretical CP here is 4, extracted by this static analysis.

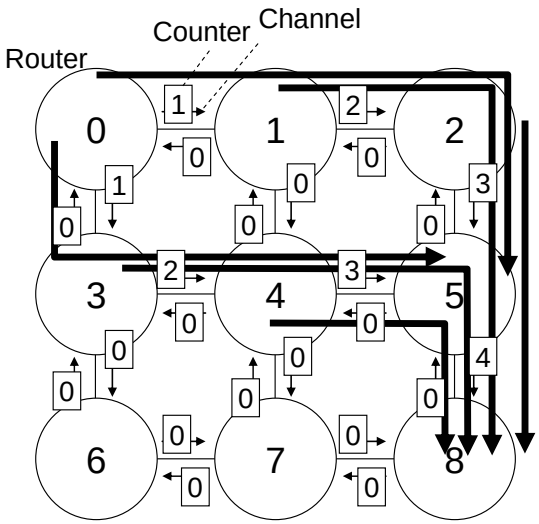


Figure 5-5 Static analysis result of an example pattern

5.3.4. Constant local labeling scheme

This section presents an algorithm to assign local labels to a set of communication paths derived by the static analysis algorithm described in the previous section. The constant labeling scheme proposed in this section assigns a single local label to a communication path. As constant local labeling schemes, the Low Port First algorithm is shown first, and the Crossing Paths Order algorithm, next.

5.3.4.1. Low Port First (LPF) algorithm

This subsection shows the Low Port First (LPF) algorithm as below. This is an algorithm which simply assigns constant local labels without any effort to reduce the number of required local labels.

1) First, a set of communication paths of applications is extracted by the static analysis algorithm. A unique node number is assigned to each node. And, a logical counter is assigned to an entire network as a label counter for local label values. This counter is initialized to 0.

2) A set of nodes which are not analyzed yet is derived. Choose a node which has the minimum node number in the set of nodes which is not analyzed yet, and remove the selected node from the set. When the set becomes empty, the algorithm terminates.
3) Extract all the communication paths which start from the selected node.
4) From the set of the extracted communication paths, pick all the independent communication paths any of which do not share any channel each other. Assign the current label counter value to the communication paths as their local labels, and increment the label counter. This procedure is repeated until the set of paths finishes.
5) Return to 2) for a next communication path.

Figure 5-6 is the result of applying the LPF algorithm to the static analysis result shown in Figure 5-5. This example pattern requires 5 local labels.

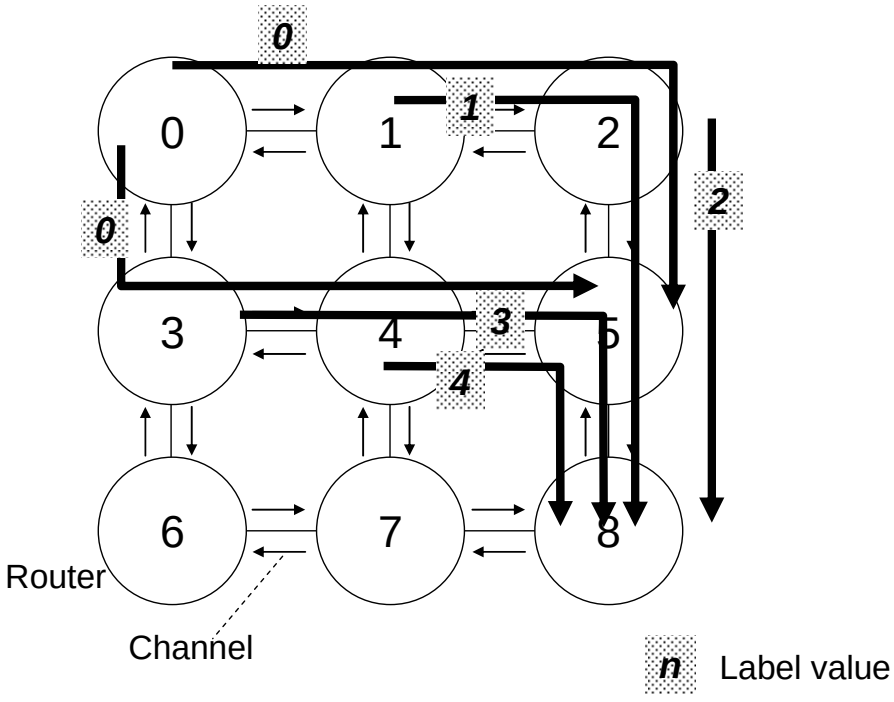


Figure 5-6 Label assignment result of Low Port First algorithm

5.3.4.2. Crossing Paths Order (CPO) algorithm

Although the LPF algorithm is simple, there is no effort for reducing required number of labels. The number of required local labels in the example shown in Figure 5-6 is 5, and it has a gap between its result and the theoretical Crossing Path, 4. This is because the LPF algorithm does not assign appropriate label values to the communication paths in the busiest channel C(5,8), where C(x, y) is a channel between node x and y, and x/y are global node numbers. Thus, assigning labels to the busiest channel should be done prior to the others for achieving a minimized number of local labels. Here, the Crossing Paths Order (CPO) algorithm is proposed as below.

1) First, a set of communication paths is derived by the static analysis algorithm.
2) Choose a channel which has a maximum label counter value as a result of the static analysis algorithm, from the set of channels which are not analyzed yet. If this set is empty, the algorithm terminates.
3) A set of communication paths which are routed on the selected channel in 2) is derived. And if any of these communication paths already have local labels, remove them from the set. Also, a set of local label values in the communication paths is prepared for the channel, so that the values of the communication paths which already have local labels are assigned as initial values for the set.
4) Prepare a logical label counter for the channel and initialized to a value which does not match any of the values in the set of local label values prepared in 3).
5) Pick a communication path from the set of communication paths prepared in 3). Assign the current label counter value as a local label to the selected communication path. Add the value to the set of local label values.
6) Set the counter value to the next larger value than the current which does not match any of the values in the set of local label values. If the set of

communication paths becomes empty after the removal of the communication path in 5), return to 2), otherwise go to 5).

Figure 5-7 shows the result of label assignment to the static analysis result presented in Figure 5-5, by the CPO algorithm. Crossing Path becomes 4, and it is fewer than the result of the LPF algorithm. This is because the CPO algorithm assigns a local label value to the communication path which has the largest number of local labels from the set of communication paths not analyzed yet. Thus, the CPO algorithm can reduce the number of required local labels.

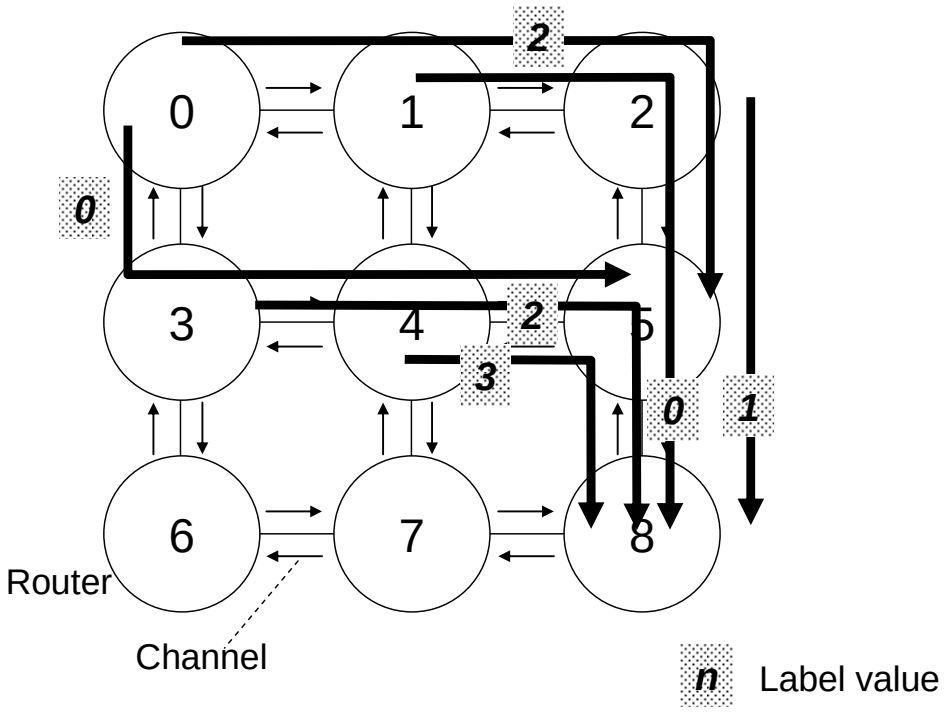


Figure 5-7 Label assignment result of Crossing Paths Order algorithm

5.3.5. Renewable Local Labels

The constant local labeling scheme assigns a single local label value to each communication path, and tries to minimize the required number of local labels. On the other hand, a renewable local labeling scheme presented in this section

is the algorithm which assigns a single local label to each channel, not to each communication path. This increases flexibility of label assignment and the required number of labels can be more reduced so that it becomes equal to the same value with the static analysis result. Below is the procedure of the renewable local labeling algorithm.

1) Assign a logical counter to each channel, and initialize them to 0.
2) Choose a single channel from the set of communication paths to which local labels are not assigned.
3) Assign logical counter values of the channels where the communication path is routed, to the communication path. Sequence of the local labels express the communication path itself.
4) Increment the counters used in 3).
5) Repeat 2) to 4)

The renewable local labeling algorithm assigns local labels so that the derived Crossing Path is equal to the Crossing Path of the static analysis result, thus this Crossing Path is theoretically minimized.

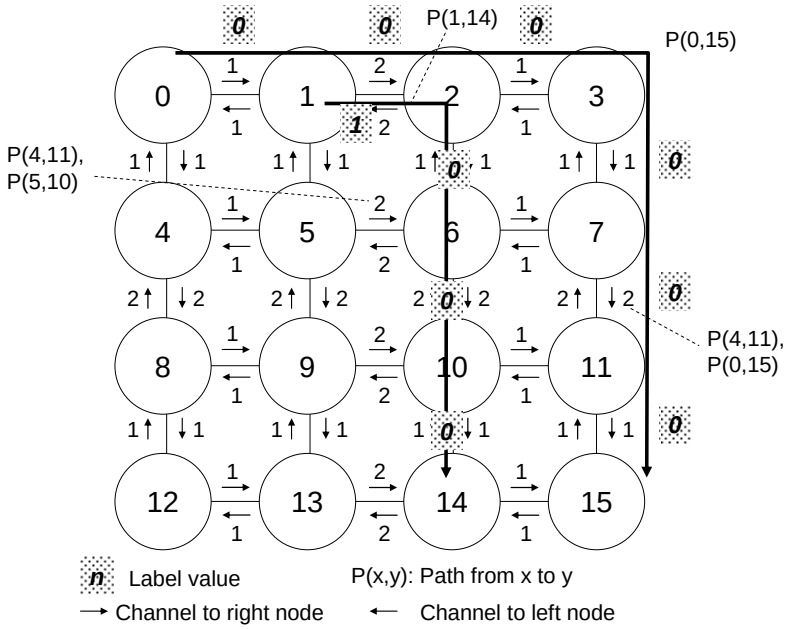


Figure 5-8 Required renewable local labels in complement pattern

Figure 5-8 shows the required number of local labels when using complement pattern [DYN02] and e-cube routing [DS87] on a 4x4 2-D mesh network. In this figure, the numbers assigned to all the channels present required local label numbers. This pattern consists of 16 communication paths that are P(0,15), P(1,14), P(2,13) … P(15,0), where P(x,y) is the communication path from a node x to a node y.

Figure 5-8 also shows examples of assigning labels. A local label sequence, “1000” is assigned to the communication path P(1,14) on C(1,2), C(2,6), C(6,10) and C(10,14). In the same manner, to the communication path P(0,15), a sequence “000000” is assigned to C(0,1), C(1,2), C(2,3), C(3,7), C(7,11) and C(11,15).

A key feature of the renewable local labeling scheme is that it allows different local labels to channels in a communication path. Thus, the derived number of Crossing Paths is minimized and equal to the theoretical CP derived by the static analysis algorithm.

However, the renewable local labeling scheme requires a function for updating local labels in routers, differently from the constant local labeling scheme. For example, in Figure 5-8, P(1,14) and P(0,15) have different local labels in the same channel C(7,11), where P(0,15) and P(4,11) have different labels in C(1,2), and P(1,14) and P(5,10) in C(6,10). And as another example, P(1,14) has different local labels in C(1,2) and C(2,6). This updating function is implemented in a router architecture described in Section 5.4.3.1.

Figure 5-9 shows the result of applying the renewable local labeling scheme to the static analysis result in Figure 5-5. The resulted Crossing Paths is 4, and this is the same result as the static analysis result.

The CPO algorithm assigns local labels so that the required number of labels in the busiest channel is minimized. However, the result of the CPO algorithm is not the minimum number for all possible cases, since it allocates a larger local label value in a case when a communication path not traversing the busiest path conflicts in other channels with all the communication paths in the busiest channel. On the other hand, the renewable local labeling can allocate the

minimized Crossing Path in the communication pattern which is equal to the result of the static analysis algorithm.

The routing function for the renewable local labeling scheme is described as follows, where C is channel and P is local label.

$$C \times P \rightarrow C \times P$$

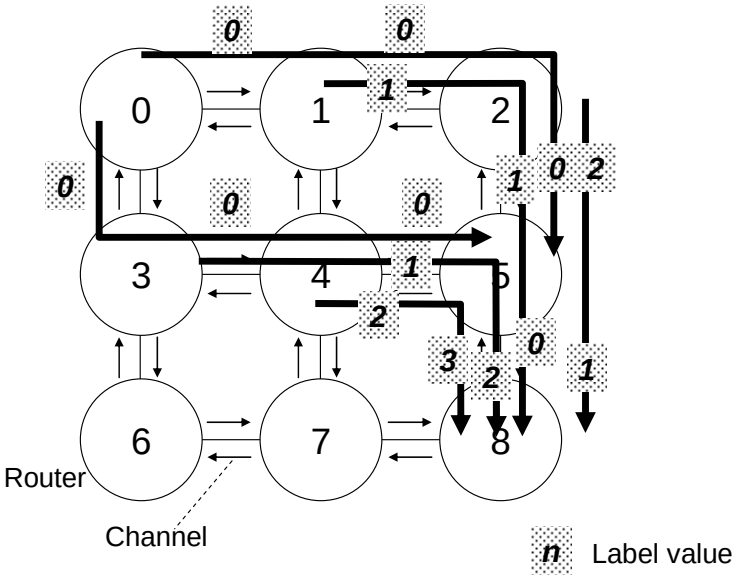


Figure 5-9 Renewable local labeling result of an example pattern

5.4. Evaluation

In this section, the performance results and the hardware cost of the proposed scheme are shown. First, the performance evaluation data when using the data-transfer with separate routing information is described. Then, evaluation data on the required number of local labels used in the proposed local labeling scheme is shown, to compare it with the conventional global addressing scheme.

5.4.1. Performance evaluation

First, this section shows the performance advantage in applying a data transfer scheme using separate routing information.

5.4.1.1. Environment

Performances for the cases using the data-transfer scheme with separate routing information and using the packet data-transfer are evaluated with flit-level simulator written in C++. The traffic patterns used for the evaluation are Uniform and Bit-reversal traffic patterns [DYN02]. The Uniform traffic is a pattern where a destination is chosen randomly data by data. And the Bit-reversal traffic is a pattern where each source node has its own fixed destination node.

Table 5-1 shows the simulation parameters. The simulated network topology is 4x4 2-D mesh, and a data-transfer scheme implemented in the router is wormhole routing without virtual channel. Size of the data buffer in the router is 1-flit length. And the header length is 1 flit for the packet transfer case, where the data transfer using separate routing information does not issue this overhead. Simulation was run for 50000 cycles, and initial 5000 cycles are ignored to remove the initialization phase. The data size of a flit is 32 bits.

Table 5-1 simulation parameters

Simulation time	50000 cycles
Topology	4x4 2-D mesh
Routing algorithm	e-cube routing
The number of VC	1
Data-transfer scheme	Wormhole

5.4.1.2. Performance result

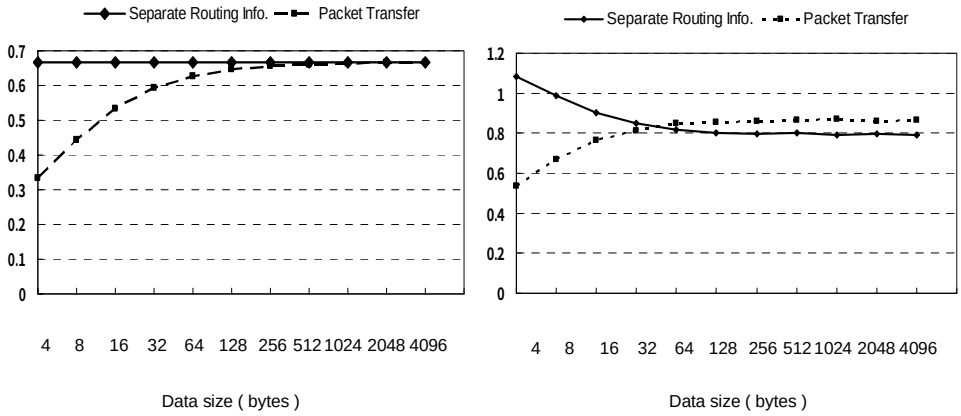
Figure 5-10 and Figure 5-11 show the simulated throughput and latency results for the Bit-reversal and the Uniform traffic patterns. The unit of throughput is bytes/cycle/node which shows throughput per node. The unit of latency is the

number of cycles and it shows average number of cycles for each flit to stay in the network per each cycle data.

As shown in Figure 5-10 (a), the throughput for the data-transfer using separate routing information in the Bit-reversal traffic is constant, independently of the data size. When data size is larger than the data size for a single cycle in using separate routing information, data is split into each cycle data and each of them has its own routing information. Thus, no cycle overhead for transferring routing information or headers is required. On the other hand, when using the packet data-transfer scheme, the header requires 1-cycle overhead. Thus, when the data size is small, the packet data-transfer cannot achieve that much performance as the data-size is large. In a case for the 4-byte data, the data-transfer using separate routing information doubled the throughput from that of the packet data-transfer.

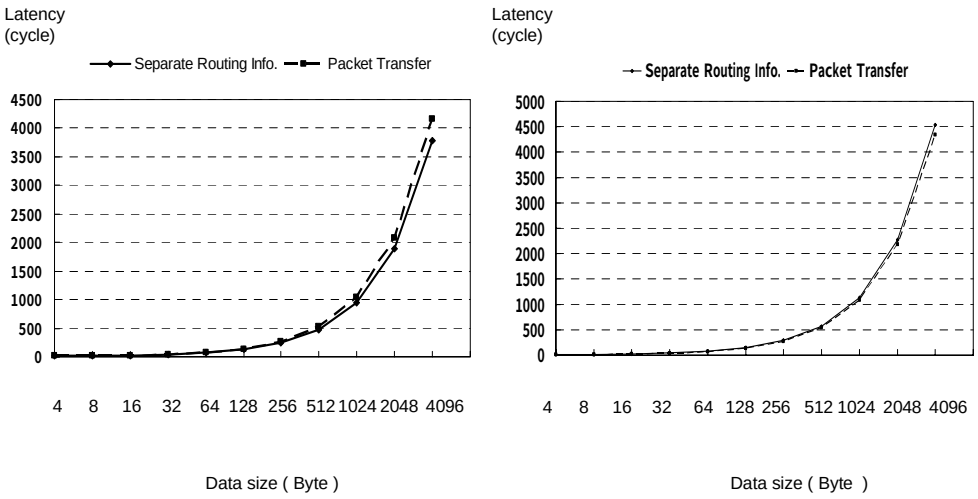
On the other hand, as shown in Figure 5-10 (b), when using the Uniform traffic pattern, the throughput for the data-transfer using separate routing information degrades when increasing the data size. This is because the destination is picked randomly, and the opportunity to conflict in a router becomes more frequent in using separate routing information.

Thus, using separate routing information results in good performance especially using the traffic pattern is fixed like the Uniform traffic pattern.



(a) Bit-reversal traffic (b) Uniform traffic

Figure 5-10 Simulated throughput results



(a) Bit-reversal traffic (b) Uniform traffic

Figure 5-11 Simulated latency results

5.4.2. Required number of local labels

5.4.2.1. Method and environment

As evaluation, communication patterns of typical applications are analyzed to evaluate required numbers of local labels in the constant and the renewable local labeling schemes. As applications, some of multimedia and communication applications are chosen as practical stream data processing. And as other applications, NAS parallel benchmarks (NPB) 2.3 [BHS95][SWW97] are analyzed for reference, where NPB cannot be realistic applications for SoCs.

First, an algorithm of each application is analyzed and each application is split into multiple functional tasks. A task-flow graph is drawn by the analysis result. The number of nodes is assumed to be less than 16, the same as [DT02]. Each application is mapped onto three topologies for 16 nodes, in 2-D mesh, 2-D torus and H-tree topologies. For the evaluation of a 64-node NoC, NPB is mapped onto a 64-node NoC. In this evaluation, a required number of local labels for each application is derived by applying the static analysis, the constant and the renewable local labeling algorithms to the mapping results of applications.

As network topologies to evaluate, three major topologies, 2-D mesh, 2-D torus and H-tree are evaluated, as shown in Figure 5-12. As routing algorithms, an e-cube (X/Y) routing is used in 2-D torus (mesh), and up/down routing in H-tree. Only in 2-D torus virtual channels are implemented in routers.

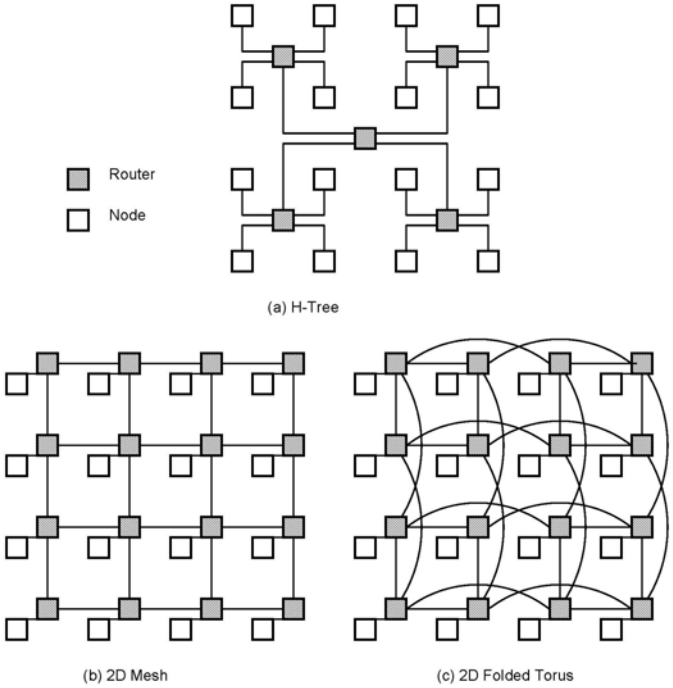


Figure 5-12 Topologies used in evaluation

5.4.2.2. Applications

Two types of applications are selected: stream data processing applications and NAS parallel benchmarks.

As stream processing applications, JPEG codec, Viterbi decoder, 4x4 network switch, OFDM (Orthogonal Frequency Division Multiplexing) and MPEG-2 encoder are analyzed. These applications are analyzed statically and task-flow graphs are created from the algorithms. The detailed descriptions of these applications are shown below.

JPEG Codec

JPEG codec is a set of JPEG decoder and encoder. The task flow graph is shown in Figure 5-13. This application compresses and decompresses static images. The decoder receives JPEG byte-stream as input and outputs raw image data in RGB (R: Red, G:Green and B:Blue) format. The encoder flow is opposite and outputs JPEG byte-stream data. Figure 5-13 (a) shows its task flow and Figure 5-13 (b) shows the task mapping result on 16-node 2-D mesh topology. Each square shows a node and the line between nodes correspond to channels. The arrow beside the channel is the data-flow of image data, and the dotted line shows the control data-flow, such as header information and table data. In this evaluation, different local labels are assigned to the communication paths between the same source node and the same destination node which convey different types of data.

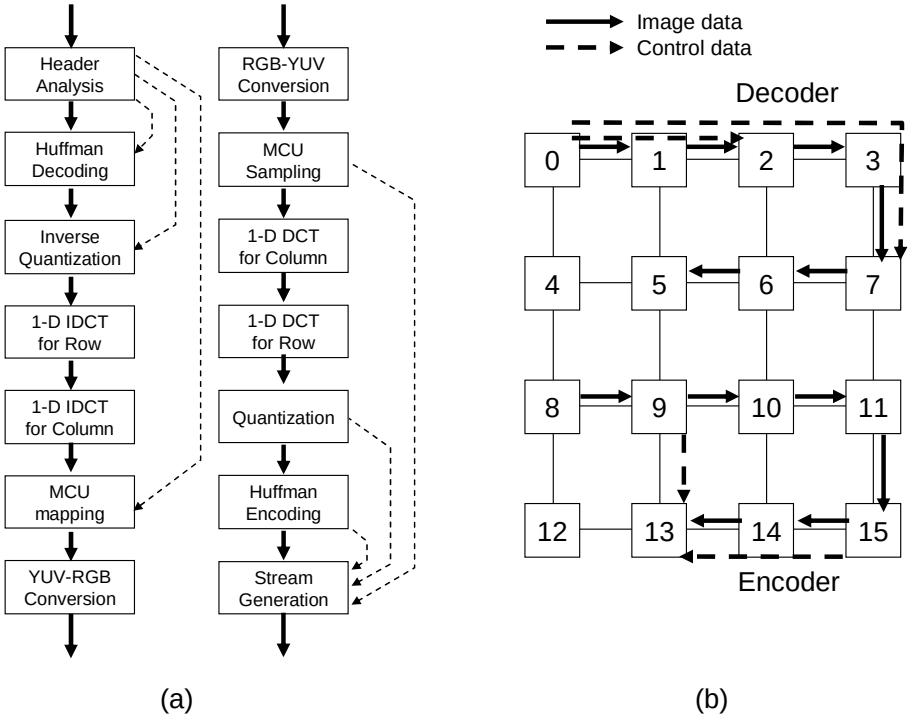


Figure 5-13 Task mapping result of JPEG codec for 2-D mesh topology

Viterbi Decoder

Figure 5-14 (a) shows a task-flow of the Viterbi decoder which is an error correction code widely used in many applications. This Viterbi decoder is a

Soft-In Soft-Out (SISO) Viterbi decoder [Kan03]. This firstly processes Add-Compare-Select (ACS) calculation, and then buffers the history of the calculated results in the FIFO buffer. In Delta, the difference between the histories is extracted, and Trace-Back (TB) phase processes the difference in a pipelined manner. The FIFO buffer consists of 64-bank memory macoros, but this task mapping tries to distribute and average the amount of hardware for each node, and use 4 nodes to process FIFO buffering history data.

4x4 Network Switch

Figure 5-14 (b) shows a task-flow of the 4-input 4-output network switch [AJA03]. In a Link module, an input header of data is analyzed, virtual channel operation is done, and arbitration cycles are issued. In 4x4 Crossbar (CB), it switches and transfers the output packet from the link modules.

MPEG-2 Encoder

Figure 5-14 (c) is a task flow of MPEG-2 encoder [LLS04]. A function block of a frame data is input to an Input Buffer module, and the reference data stored in Reference Buffer is used to search a motion vector for the input frame data in a Motion Estimation module.

OFDM

Figure 5-14 (d) shows a task-flow of OFDM which is a wireless communication protocol [LLS04]. Each processing core from Core 0 to Core 7 executes 2048-point Inverse-FFT (Fast Fourier Transform). The output complex numbers are calculated and normalized in MAC (Multiply-and Accumulate) units.

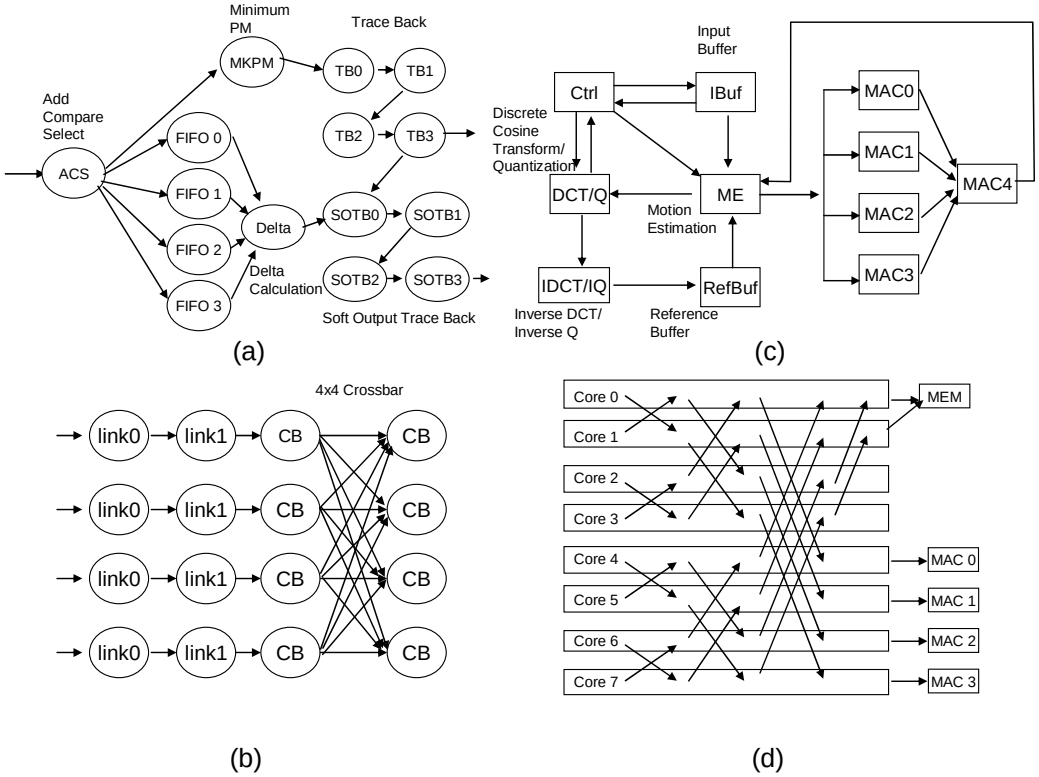


Figure 5-14 Task partition of applications (a) Viterbi decoder (b) 4x4 network switch (c) MPEG-2 encoder (d) OFDM

NAS parallel benchmarks 2.3

From NAS Parallel Benchmarks [BHS95][SWW97] which consist of typical numerical parallel application programs described with the MPI library [GLD96], six matrix computation programs are analyzed: BT (Block Tridiagonal solver), CG (Conjugate Gradient), LU (LU-decomposition), MG (Multi-Grid solver) and SP (Scalar Pentadiagonal solver). These are implemented and executed on the RHiNET-2 cluster with 64 nodes [WOT03], and the communication traces were obtained using MPI profiling libraries [GLD96]. Using obtained results, these communication paths are calculated under the X/Y and e-cube routing on the two-dimensional mesh and torus.

The task-flows of stream applications described above are simple, such that data is input to a node and output from it. In this research, however, for cases that other possible task mapping with complex communication pattern is

required, NPB 2.3 is also used for analysis as a prior research in [HP03]. NPB 2.3 is scientific calculation benchmarks, and thus it is not a realistic benchmark set for NoC, but typically used in PC clusters or parallel computers. It has characteristics to include broadcast communication and communication between neighboring nodes, differently from stream applications.

5.4.2.3. Comparison of local label numbers

The stream applications in the previous section are mapped onto a 16-node NoC. The mapping and static analysis results of the applications are shown in Table 5-2, and the average number of hops and the number of total communication paths are shown in Table 5-4. Table 5-3 is the table to show the Crossing Paths of NPB 2.3 and the average number of hops and the number of communication paths in Table 5-5. Some of the NPB applications are analyzed also on 64-node NoC. In Table 5-2 and Table 5-3, the number of bits required for local labels is calculated as $\lceil \log_2 m \rceil$ when the number of CP is m.

The total communication paths of the stream applications are approximately from 20 to 30, where 16-node applications of NPB result in more than 60 communication paths. This is because the analyzed stream applications simply receive input data and transmit output data, and every task operates as pipelined manner. In some applications, performance is improved by parallel processing like FIFO buffering after ACS calculation in the Viterbi decoder. However, no synchronization communication occurs in the system, and the total number of communication paths is not increased to that large number in NPB. In NPB, broadcast communication is required to synchronize timings between nodes, and IS requires 240 communication paths which are all-to-all combination of 16 nodes, for instance.

As shown in Table 5-2, the resulted CPs for the stream applications are less than 8 in 16-node NoC for all the topologies. Thus, the required number of bits for local labels in these applications is bit, where the conventional global addressing scheme requires 4 bits in 16-node NoC. Thus, the local labeling scheme reduced by 1 bit from the conventional global addressing scheme. The

source routing described in [DT02] requires 16 bits in 2-D mesh and torus topologies, and 8 bits for H-tree topology, and thus the local labeling scheme reduced by 13 bits and 5 bits from them.

In NPB, IS which includes the largest number of the communication paths results in the largest number of Crossing Paths, as shown in Table 5-3. Its CP is 16 in 2-D mesh topology and 15 in 2-D torus topology, and thus the 4-bit local labels are required. So, the local labeling scheme does not improve it from the global addressing scheme. However, in the other applications except for IS, they do not include all-to-all communication, and are less CPs than that of IS. In these applications, the renewable local labeling scheme results in 8 local labels which are 3 bits, at most, and the constant local labeling scheme requires 10 local labels, which are 4 bits. So, the renewable local labeling scheme reduced by 1 bit from the constant local labeling scheme in 2-D mesh and torus topologies. Also in the 64-node NoC, applications except for IS require 14 local labels which are 4 bits. So, it reduced by 2 bits compared with the global addressing scheme which requires 6 bits for 64-node NoC.

On the other hand in the H-tree topology, the required numbers of local labels for NPB are less than 48, which is 6 bits, including IS, and less than 20, which is 5 bits, excluding IS. They are larger numbers than that of the global addressing scheme. In the H-tree topology, all the communication paths traverse the vertex on top of the tree and those upward links require larger numbers of local labels. This implies that further research on appropriate topologies for local labels like in [YAK04] is necessary for achieving appropriate tree-based topology.

Next, discussion is given to the comparison between the renewable local labeling scheme and the CPO labeling scheme. As shown in Table 5-2, there is no difference between the numbers of CPs for stream applications for these algorithms. However, as shown in Table 5-3, applications except for LU result in smaller numbers of local labels using the renewable labeling algorithm. This is caused when any communication paths not in the busiest path conflicts in other channels with all the communication paths in the busiest channel. However, according to the analysis results of stream applications, there are no such cases in these stream applications.

Table 5-2 Crossing Path of stream applications in 16-node NoC

	2-D mesh			2-D torus			H-tree		
	Ren.	LPF	CPO	Ren.	LPF	CPO	Ren.	LPF	CPO
VITERBI	5	5	5	4	4	4	6	6	6
4x4 switch	4	5	5	4	4	4	4	4	4
JPEG	8	8	8	8	8	8	8	8	8
MPEG-2	5	5	5	3	3	3	6	6	6
OFDM	4	5	4	4	4	4	6	6	6

Ren.: Renewable local labeling

Table 5-3 Crossing Path in NPB 2.3 in 16- and 64-node NoC

	2-D mesh			2-D torus			H-tree		
	Ren.	LPF	CPO	Ren.	LPF	CPO	Ren.	LPF	CPO
BT.16	8	11	10	8	10	9	20	22	20
CG.16	5	8	6	5	7	6	11	14	12
IS.16	16	22	20	15	20	20	48	60	60
LU.16	6	6	6	6	6	6	12	14	12
MG.16	5	6	6	5	5	5	12	13	12
SP.16	8	11	10	8	11	10	20	22	20
CG.64	10	13	11	9	11	11	11	14	12
MG.64	9	14	14	9	13	13	48	58	48

Ren.: Renewable local labeling

Table 5-4 Number of communication paths and average hops in stream applications

	Number of hops			Number of Communication Paths
	2-D mesh	2-D torus	H-tree	
VITERBI	2.78	2.67	2.44	18
4x4 switch	2.83	2.67	2.00	24
JPEG	2.36	2.27	1.36	22
MPEG-2	2.84	2.53	2.37	19
OFDM	2.47	2.34	2.19	32

Table 5-5 Number of communication paths and average hops in NPB 2.3

	Number of hops			Number of Communication Paths
BT.16	3.00	2.50	2.25	128
CG.16	2.79	2.68	1.58	76
IS.16	3.67	3.13	2.60	240
LU.16	2.40	2.40	2.00	80
MG.16	2.60	2.40	2.20	80
SP.16	3.00	3.00	2.25	128
CG.64	3.80	3.58	2.16	440
MG.64	3.78	3.55	3.00	576

5.4.3. Evaluation of hardware amounts

This section describes the evaluation results using the required number of local labels derived in the previous section.

5.4.3.1. Designed router architecture

The purpose of the proposed local labeling scheme is to reduce the required hardware amount of a router by reducing required numbers of local labels. Thus, as well as the algorithms to reduce numbers of local labels, evaluation of router architectures to achieve small hardware amount is also important.

Figure 5-15 (a) and (b) show the designed router architecture using the renewable and the constant local labeling schemes for the 2-D mesh topology. These routers distribute routing tables to all the input ports for parallel lookups, and do not take centralized approach. And since communication patterns assumed to be statically analyzed by the proposed algorithms, each input port does not have virtual channels for the purpose of avoiding deadlocks. Inputs to the router are from the four directions and one node output, and the same for outputs. Thus, the crossbar is 5x5. And there is a simple I/O interface to initialize routing tables, which is not shown in the figure.

An input port controller of the renewable local labeling router shown in Figure 5-15 (a) looks up the routing table using an input local label as a table address. The table entry includes an output channel number and a local label at the output channel. So, the input port controller uses the output channel number as a select signal for the crossbar and replace the local label in the entry with the input local label. On the other hand in the constant labeling router architecture shown in Figure 5-15 (b), the input local label is simply transferred to the output port and there is no need for replacement. The entry for this router is simply an output channel number, and does not need the local label for replacement.

As described above, the routers for the renewable and constant local labeling schemes have almost the same structure, and the difference is the table format.

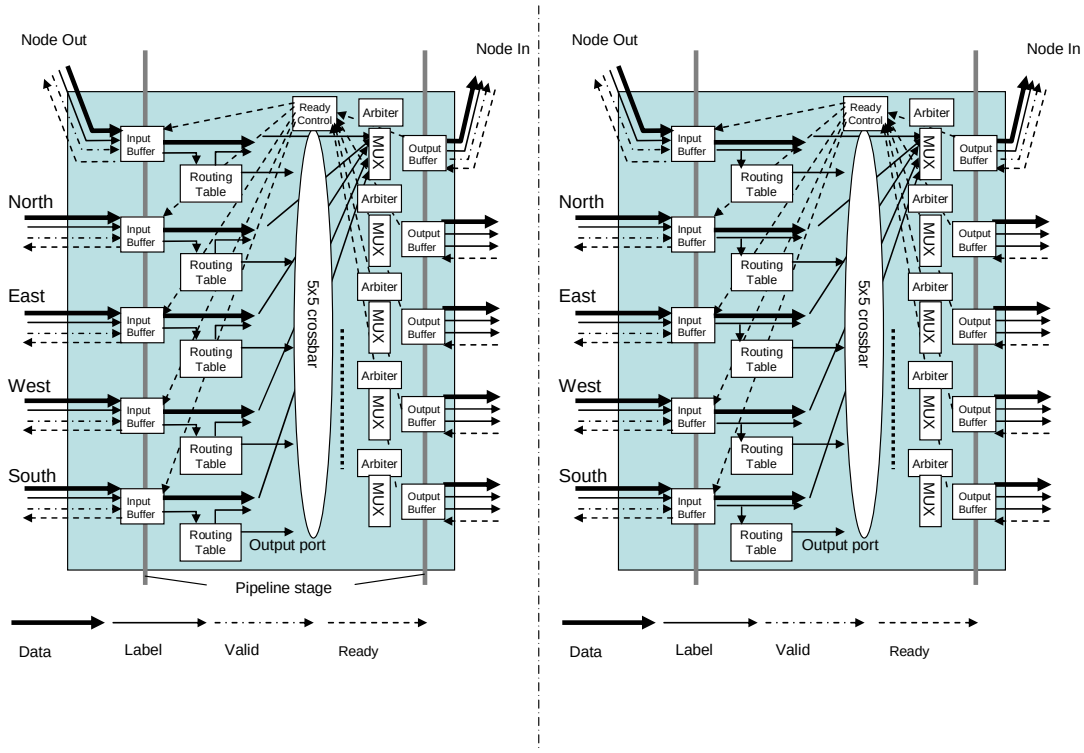


Figure 5-15 Router architectures for 2-D mesh topologies using (a) updated local labels and (b) non-updated local labels

5.4.3.2. Evaluation environment

To compare the required hardware amounts for the routers shown in Figure 5-15 (a) and (b), both these routers were designed in HDL and synthesized with Synopsys DesignCompiler using 0.15 μm CMOS cell libraries. The result is expressed in 2-input NAND equivalent gate.

5.4.3.3. Evaluated amount of router hardware

Table 5-6 shows required hardware amount for the renewable and constant local labeling routers for the 2-D mesh topology. The results are shown in 2-input NAND-equivalent gates. As shown in this table, the required hardware

amount of the router increases according to the number of local labels, due to the increase of routing table entries.

According to the result of required local labels for stream applications described in Section 5.4.2, 3-bit local labels are required in the 16-node NoC. Thus, the renewable and constant local labeling routers result in 5062 and 4049 gates, respectively. Since a router for the conventional global addressing scheme corresponds to the one with the constant local labeling router with 4 bit labels, it requires 5220 gates. Thus, using the constant local labeling router reduces 22 % gate counts than it, and the renewable local labeling router only reduces 3 %. So, in the case when the renewable local labeling algorithm does not reduce the required number of local labels from the constant local labeling algorithm, the constant labeling scheme resulted in a simpler router.

In the 64-node NoC, the required gate counts for the renewable and constant routers for NPB applications resulted in 4 bits for both cases, and 7863 and 5220 gates. When using the global addressing scheme, 6 bits are required for a routing tag and the router results in 11015 gates. Thus, the renewable labeling scheme reduced by 28 % and 46 % from the renewable and constant labeling routers. For the stream applications which require fewer communication paths than NPB, more hardware amount is expected to be reduced.

Table 5-6 Required number of gates for a 2-D mesh local labeling router in ASIC

The number of local labels	Constant (CPO) labeling (gates)	Renewable labeling (gates)
2 (1 bit)	3132	3189
4 (2 bit)	3450	3766
8 (3 bit)	4049	5062
16 (4 bit)	5220	7863
32 (5 bit)	7987	13455
64 (6 bit)	11015	26396

5.4.3.4. Ratio of evaluated hardware amount

Next, ratio between a router and each node of the Viterbi decoder SoC is evaluated to show the impact of the proposed scheme on the total SoC gate counts. Table 5-7 shows the ratio comparison with the conventional global addressing scheme. The Viterbi decoder consists of 15 nodes and each of them is listed in at left column. The ratios of the CPO labeling router and the global addressing router are shown.

As shown in Table 5-7, the tasks range from 1K gates to 62K gates, and the total number of gates of this SoC is approximately 320K gates. The router ratios from the sum of the router and each node range from 6.1 % to 78.3 % in using local labeling, where these of the global addressing scheme range from 7.7 % to 82.3 %. For the total gate counts, the ratio of local labeling routers is 15.8% where that of the global addressing scheme is 19.5%.

Table 5-7 Gate count ratio of routers in Viterbi decoder SoC

	Gate count of node*	Router ratio with local labels (%)	Router ratio with global address (%)
ACS	**20045**	**16.8**	**20.7**
MKPM	**5012**	**44.7**	**51.0**
FIFO0	**62560**	**6.1**	**7.7**
FIFO1	**62560**	**6.1**	**7.7**
FIFO2	**62560**	**6.1**	**7.7**
FIFO3	**62560**	**6.1**	**7.7**
Delta	**1125**	**78.3**	**82.3**
TB0	**6123**	**39.8**	**46.0**
TB1	**6123**	**39.8**	**46.0**
TB2	**6123**	**39.8**	**46.0**
TB3	**6123**	**39.8**	**46.0**
SOTB0	**5673**	**41.6**	**47.9**
SOTB1	**5673**	**41.6**	**47.9**
SOTB2	**5673**	**41.6**	**47.9**
SOTB3	**5673**	**41.6**	**47.9**
Total	**323606**	**15.8**	**19.4**

* without router

5.4.4. Comparison with programmable switch

In FPGAs, programmable switches [RB91] have been used as interconnection networks. The programmable switches require static analysis of all routing paths, and so the local labeling scheme, in a sense, can be viewed as a similar approach to these. Main difference between the local labeling scheme and the programmable switches is that wires connected with programmable switches carry only data from a single output, while wires in the local labeling scheme are used in a time multiplexing manner with several paths. Figure 5-16 outlines a typical structure for a programmable switch used in FPGAs. This switch has four connection links to neighboring switches, and the other link is connected to a calculation node.

In programmable switches, one channel in a link is only used by a single connection. Thus, to share a link with multiple connections, multiple channels are required. In this figure, M is the number of channels per link. FS in this figure, on the other hand, means switch flexibility. As described in [RB91], it has been introduced to reduce the size of the crossbar. The total potential input for the crossbar is 4M+1, and FS channels have been selected and multiplexed to each output.

Figure 5-17 compares the number of gates required for the 32-bit programmable switch and the 32-bit local labeling router. This graph plots the number of channels and labels versus the number of gates required for the programmable switch and the local labeling router. The proposed local labels only require 371 gates per label, while the programmable switch requires 431, 1060, 1670, and 3346 gates per channel. Thus, the proposed routing scheme using local labels reduced the number of gates per channel.

The number of gates for the local labeling router with a 4-bit label is almost the same as that of the switch when FS is four. The local labeling scheme router with 8-bit labels requires almost the same number of gates as the programmable switch whose FS is 2. However, according to [RB91], from the standpoint of routability, the case that FS is two is not practical. So, considering cases whose FS is more than or equal to four, the local labeling

scheme router with 3-bit labels is smaller than that of the programmable switch in most cases.

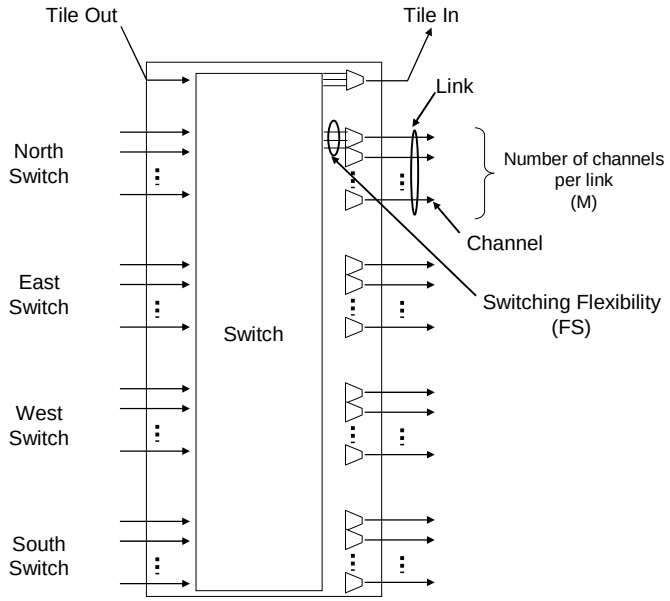


Figure 5-16 Programmable switch architecture

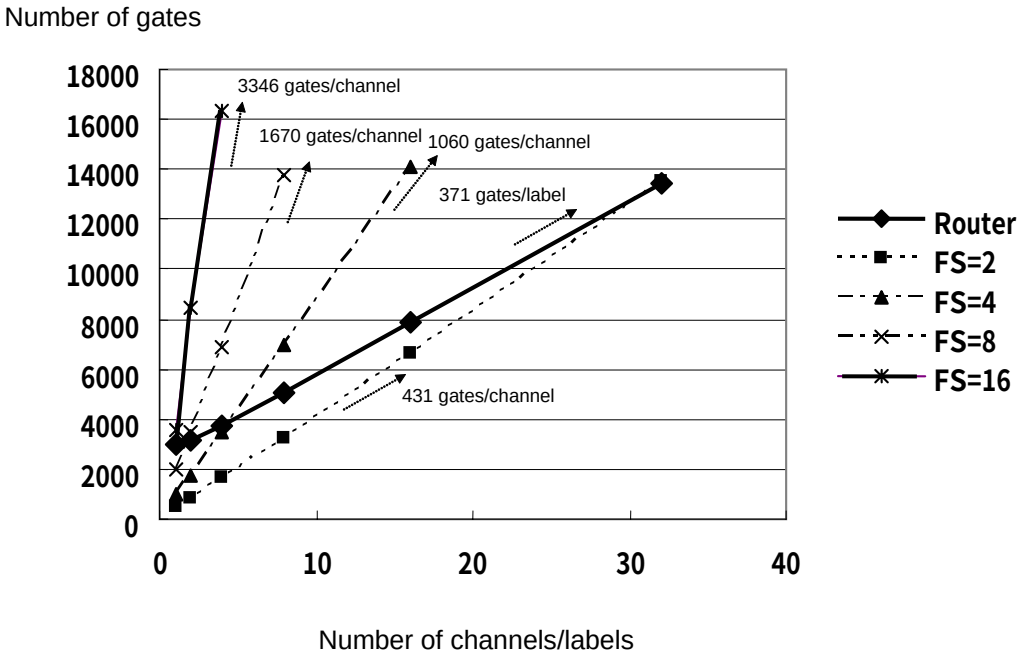


Figure 5-17 Comparison of required gate counts according to channels/labels

5.5. Conclusion of this chapter

In this chapter, a novel data-transfer method for an on-chip interconnection network in programmable devices is proposed. A local label is attached to each 1-cycle data as routing information. Unlike the traditional packet transfer, the local label is transferred on dedicated wires attached to data lines to remove complicated packet generation procedures in a node. Only a small-sized local label is required to specify routing tags to the destination, and intermediate routers change it to solve local label conflict between paths on a physical channel. Furthermore, it can simplify network interface structures in nodes, because it removes packet structure to assembly.

The results of flit-level simulation show that the data-transfer with separate routing information which transfers only 1-flit data, can increase throughput especially in cases with fixed communication patterns.

The evaluation results in 2-D mesh and torus topologies using streaming applications. The required number for labels is 8 in 16-node case which decreased 1 bit from the global addressing scheme. And for a 64-node case, it requires 4 bits for NAS parallel benchmarks which have more complicated communication pattern than typical streaming applications, and it reduced 2 bits from the global addressing scheme.

The hardware comparison results show that the proposed router is constructed in smaller hardware than the ones in distributed routing and source routing. It can reduce 4 % and 29 % in the 16-node case from them respectively, and 29 % and 74 % in the 64-node case.

In summary, the proposed local labeling scheme, which transfers 1-flit data and attaches a local label, is advantageous from various aspects, especially hardware cost. It can simplify the router structure and decrease the hardware amount of routers, with sustaining performance comparable to the conventional packet data transfers. All these results are shown under the practical application patterns for high credibility.

Chapter 6 Future work

This thesis focused on the research on implementation techniques of on-chip interconnection networks regarding cost and performance efficiency. However, there are some other possible approaches for these research topics

Layout and clocking

In this research, the designed bus has 5 masters and 7 slaves in a chip, and the designed layout is shown in Figure 4-18. The layout shown here assumes that the on-chip bus is laid out as an IP block. In this layout scheme, area consumed by an on-chip bus is roughly estimated in advance, and the corresponding area must be reserved. And other IP cores that are connected to the on-chip bus must have boundaries with the bus block.

However, when numbers of masters and slaves increase, there are several possible problems to solve.

1) It will be difficult to design an on-chip bus with small area and in simple shape, since all the IP blocks have share boundaries with it. The shape will be widely spread in a chip, and will not be a simple rectangular and more complex.
2) Because the bus structure is basically trees of multiplexers as shown in Figure 6-1, the multiplexers will be centralized into one spot. Thus, center location of a bus block will be congested due to multiplexers and signal wires. Distributed placing of multiplexers is important to reduce congestion of signal wires.
3) Bus operation frequency will be difficult to increase. Bus signals are distributed from the final stage of the multiplexer tree. Those signals are not only wired to slaves, but to masters in some cases when masters refer

to bus signals, such as a bus frame signal. So, bus signals travels from master core to the center of the bus block, and then to masters and slaves. In most buses, bus protocols are designed with the assumption of 1-cycle delay.

4) Netlist of the bus multiplexer tree must reflect the physical locations of masters and slaves. If each multiplexer multiplexes output signals from masters not neighboring each other, the location of the multiplexer will be centralized into a central spot of the bus block. Even with the physical compiler, the netlist should be changed to meet the physical locations of the masters and slaves to place and route effectively.

Another approach to layout the bus block other than handling it as an IP core is leaving it as top-level glue logic. To successfully layout with this approach, each IP core must be placed with sufficient spaces between them to place bus multiplexers and to route bus signals. The bus logic is typically placed and routed in the final stage of the layout flow, since these are handled as top-level glue logic. Thus, estimating the required space in the early stage floorplanning as precise as possible is more and more important. Another important issue is to layout the bus block with small latency penalty. With this layout scheme, the area for layouting bus logic is widely spread all over the chip. Thus, signaling delay is more likely to get long than the scheme which handles the bus as an IP core. Thus, physical synthesis other than simple logic synthesis will be much more important.

On the other hand, the router-based NoC has difficult shape to layout as a single IP core. The overall floorplan is based on the architecture shown in Figure 5-1. As it is shown in the figure, handling overall network as a single functional block is not a realistic approach, because the shape is not a simple rectangular. The easiest way is concatenating NoC router with each computation node. With this approach, layouting a computation node plus a router first, and then, dupulicating the prepared node and router.

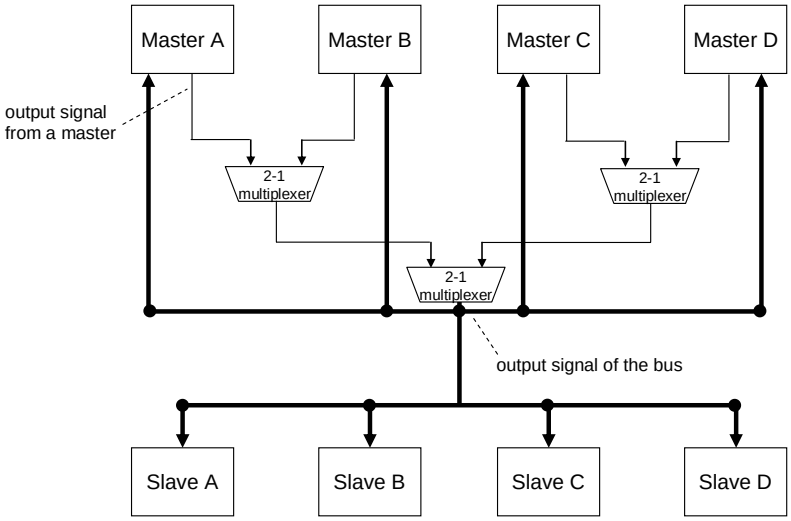


Figure 6-1 bus structure with multiplexer tree

Clock skews

The designed bus has a single operation clock frequency. In sub-0.1 μm CMOS devices, the frequency increases, and the allowable clock skew which is a difference of the clock arrival time within in the bus block, decreases. The clock distribution circuit has typically tree structure, and consists of clock drivers and signal wirings. For achieving less clock skew, designing a clock distribution tree with accurate delay estimation is a key issue. However, estimating clock delay accurately in sub-0.1 μm CMOS devices is not easy. It requires balanced structure of a clock distribution network, and accurate delay estimation of buffer and wiring delays.

The layout can be done in two styles as described in the layout suggestions.

- When the bus block is handled as an IP core, its clock distribution network is designed using hierarchical clock tree synthesis (CTS), in most cases. A clock distribution network of each IP core is designed

using CTS first, and then, the top-level clock distribution network is designed next. In this flow, designing a clock distribution network with small clock skew in the bus block which could be complex shape is important. To design the clock network, a clock input port for the bus block is defined. When the shape is complex, the lengths from the input port to flip-flops vary. In this case, CTS will insert clock buffers to the short paths and this increase the latency of the clock network. If the delay of the clock network is long, this will affect PLL's stability factor when it is feedbacked from the end point of the tree and will increase power consumption.

- When the bus is laid out as top-level glue logic, the top-level clock distribution network design will be important. All the IP cores have regions working with the same clock frequency, and they are widely spread inside the chip. With this scheme, all the flip-flops included inside the bus logic should be included in each connected IP cores, and only leave multiplexers as glue-logic.

In the router-based NoC, the number of connected nodes is assumed to be large and the layout is regular array structure of nodes and routers. So, the NoC region will be widely spread overall the chip. Although this research designed all the NoC region to be clocked with the same frequency, another design strategy that each router is operating in the same clock frequency with a neighboring node could be realistic.

When taking this approach, the clock synchronization circuit must be implemented in the boundaries of neighboring routers. In the current router design shown in Figure 5-15, there are input and output buffers which are interfaced with each neighboring router. The clock synchronization circuit could be implemented in those buffer designs. The possible side effect is the router operating with the slowest frequency will be a performance bottleneck. In that case, routing algorithm could be newly designed to select routers in fast frequencies prior to those in slow frequencies.

Clock synchronization

In this research, the clock frequency is assumed to be single, and if some other frequencies are required such as by I/O interfaces, the clock synchronization must be done in IP cores. The clock synchronization is considered as a critical issue since it could frequently be sources of logical bugs. Thus, clock synchronization circuits without bugs should be embedded inside bus wrappers or network routers. This will unburden IP core or computation node designers.

The basic implementation of the clock synchronization for a single-bit signal is simply clocked once with a source clock and more than twice with a destination clock, as it is well-known. Further implementation possibilities are embedding FIFOs as elastic buffer. FIFO consumes large number of transistors or requires large internal memories. However, dual port memory will ease the logic design with two clock domains. Besides FIFOs, when transferring multiple bits of data over the clock boundary, like memory pointer values, the value should be gray coded to ensure 1-bit transition.

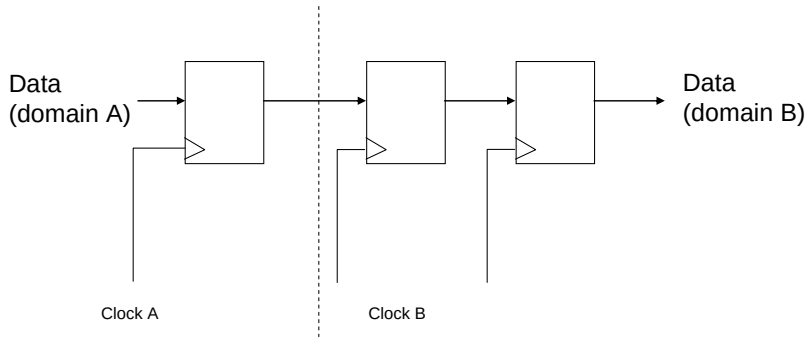


Figure 6-2 basic clock synchronization circuit

Automatic netlist generation and logic verification

On-chip buses and router-based NoCs are expected to support any kinds of SoCs or programmable devices. Thus, designing automatic netlist generation tool is a good idea. In this idea, netlist will be automatically generated according to the configured address map, and numbers of masters and slaves to be

connected. Although generating netlist itself is quite easy, how to ensure its logic quality by verification is an open issue.

Router-based NoCs and wrapper-based buses have the same feature that both use duplicated logic of bus wrappers and network routers. So, from the logic verification point of view, the unit-level verification is done easily, but verification for combination of routers is more difficult.

As for the logic verification, two approaches can be considered. One approach is static verification, using property checking for the interface protocols. This does not require any testbench generation for the RTL design and possible protocol rules are written for the property checking tool. The static approach will analyze the design statically. Thus, serious protocol errors, which are difficult to be found out in dynamic simulation approches, such as deadlock and livelock, are expected to check.

A dynamic simulation approach will require logic verification by generating testbenches. The logic simulation will require assertions and coverage to meet, which should be defined by logic designers and verification engineers. The assertions are the rules to check in any places in the design. The coverage to meet is the sufficient condition to complete the logic simulation and this must be defined as combination of timing conditions and function conditions. These verification schemes can be easily established using SPECMAN or VELA, released by Vericity and Synopsys.

Reduction of power consumption

Power reduction is an important aspect also in high-end products as well as in consumer products. In this research, power consumption is not critical in an on-chip bus or an NoC since its hardware amount is not really large compared with IP cores. However, saving power consumption is always important in any function blocks in a chip. Thus, listed below are the possible items for reducing power in on-chip buses and NoCs.

Dynamic power can be reduced by clock gating when it is not used. To implement gated clock in on-chip buses or NoCs, stopping operations of unused wrappers or NoCs is a possible idea. Clock gating is typically done by shutting a

clock for a certain IP core itself. However, in wrapper-based buses or router-based NoCs, each bus wrapper or router is connected to different IP cores or nodes.

As for wrapper-based buses, thus, only clocks for some of the wrappers are stopped, and the shutdowned wrappers are required to ensure not disturbing operations of other wrappers. Thus, an asynchronous status signal that indicates shutdown status should be generated. And in the boundary of the shutdowned wrapper, by using that status signal and connecting that to clear input port of flip-flops which hold the interface signals, such as arbitration request, enable signals for the physical bus.

As for router-based NoCs, those shutdowned routers will never pass packets through them. Thus, before shutting down routers, the management algorithm of the shutdown process is required. Also, as well as the case for the on-chip bus, the interfacing signals must be kept disabling when a router is shutdowned.

As for reducing static power consumption, it is not easy to achieve without changing process technology or controlling power voltage. A possible item is changing threshold voltage of transistors by controlling back-bias voltage. Another item is using high-K gate transistor for less leak current.

Error correction scheme

Since Network-on-Chip is a possible candidate for interconnection in future SoCs, or near-future programmable devices, the error correction scheme becomes much more important due to high risks of chip designs. On the other hand in on-chip buses, the error correction is not really a critical issue, but for achieving better soft-error-rate (SER), it is recommended to implement.

The possible risks include high NRE cost, long period required for reworking, functional errors due to soft-errors, and design bugs which are not found by chip testing.

- NRE cost is getting higher, generation by generation. Thus, avoiding bug risks is quite important.

- Once reworking process started, the re-design and re-verification are required as well as re-creating masks and re-fabricating. All these periods are getting longer, generation by generation.
- Soft-errors widely range according to their error sources, such as alpha-beam, etc. Soft-error will possibly occur and it will be recovered by resetting. Thus, idea of error correction will total operation stability since it enables recovering without resetting.
- Chip testing is increasingly getting difficult. Logic BIST (Built-In-Self-Test) is widely considered as a candidate for generating more chip test patterns, however integrating logic BIST into ASICs in generalized scheme is very important but difficult to establish.

Thus, according to these backgrounds, error correction schemes are required to be established. A possible implementation of error correction is to support ECC code in routers or bus wrappers. However, this will issue a certain cycle penalties to correct errors. A more system-level approach is using only parity data and detecting defects inside the chip. Then, retransmit it so as to avoid defects. Here, how to avoid defects is the key issue, and in this statement, just leaving it as an open issue.

Chapter 7 Conclusion

In this thesis, on-chip interconnection networks for current and near-future generation are researched, especially from the standpoint of performance and cost efficiency. As an interconnection network for current SoCs, novel wrapper-based bus architecture is proposed. And for programmable devices or processor-arrays as near-future SoC architectures, a router-based NoC with a novel routing scheme is proposed.

Firstly, a wrapper-based bus which has practical performance with low hardware cost is presented for current generation IP-based SoC. This thesis pointed out that:

- Conventional wrapper-based buses only focused on IP core reuse and sustaining performance. The wrapper-based bus proposed in this thesis takes an approach to reduce hardware amount with sustaining required performance and functionalities to reuse IP core. What is necessary in the bus wrapper design is not embedding a data buffer, and considering tradeoffs between cost and performance.

Furthermore, three novel wrapper-based bus implementation techniques for better IP core connectivity and better performance with small hardware cost are proposed.

- A proposed Write buffer switching technique increases throughput and reduces Write latency. The guideline for determining the optimal buffer size by evaluating throughput and latency per gate is shown.
- A slave designated retry control technique is proposed. This technique allows connecting fast and slow IP cores into the same wrapper-based bus without performance degradation.

- Novel data-width converter architecture is developed to allow connecting different bit-width IP cores to the same bus at a small hardware cost.

With the design of a CPU-based SoC, these techniques are evaluated with simulation and confirmed performance increase and cost effectiveness compared with conventional approaches. And for higher credibility, the real chip fabricated in 0.15μm CMOS process is confirmed for the stable operations.

Next, a novel data-transfer scheme in NoCs, which attaches a local label to each 1-cycle data, is presented. This local labeling scheme uses static analysis results of communication patterns in applications and suppresses the required number of bits for routing information compared with the conventional global addressing scheme.

To show the credibility of this proposed data-transfer technique, typical realistic applications for multi-media and communication are evaluated. And a result that shows the proposed labeling scheme decreased required amount of router hardware with sustaining performance. And further more, this thesis showed that local labeling router can achieve better hardware cost per channels compared with programmable switches used in current generation FPGAs.

In summary, this thesis showed two essential interconnection networks for current IP-based SoCs and near-future programmable devices, from the important aspects of cost and performance efficiency. The proposed two approaches have distinguished features that try to be simple and less-hardware with sustaining performance.

References

[Arm94] ARM Ltd., “The ARM6 Family Bus Interface,” ARM DAI 0018B, *available at http://www.arm.com*, December 1994.

[Arm99] ARM Ltd., “AMBA Specification Rev 2.0,” *available at http://www.arm.com*, 1999.

[Arm01] ARM Ltd., “Multilayer-AHB overview,” *available at http://www.arm.com*, 2001.

[AJA03] H. Amano, A. Jouraku, and K. Anjo, “A Dynamically Adaptive Switching Fabric on a Multicontext Reconfigurable Device,” *Proceedings of the Field-Programmable Logic and Applications*, pp.161-170, September 2003.

[AOK02] K. Anjo, A. Okamura, T. Kajiwara, N. Mizushima, M. Omori, and Y. Kuroda, “NECoBus: A high-end SoC bus with a portable and low-latency wrapper-bus interface mechanism,” *Proceedings of IEEE Custom Integrated Circuit Conference*, pp.315-318, May 2002.

[AYK04] K. Anjo, Y. Yamada, M. Koibuchi, A. Jouraku, and H. Amano, “BLACK-BUS: A New Data-Transfer Technique using Local Address on Networks-on-Chips,” *Proceedings of IEEE International Parallel and Distributed Processing Symposium*, pp.10a, April 2004.

[BCF95] N.J. Borden, C. Cohen, R.E. Felderman, A.E. Kulawik, C.L. Seitz, J.N. Seizovic and W.K. Su, “Myrinet: a gigabit-per-second local area network,” *IEEE Micro*, vol.15, no.1, pp.29-35, 1995.

[BHS95] D.Bailey, T.Harris, W.Saphir, R.Wijngaart, A.Woo and M.Yarrow, “The NAS Parallel Benchmarks 2.0," *NAS Technical Report*, NAS-95-020, Dec, 1995.

[BM02] L. Benini and G.D. Micheli, “Network On-chips,” *IEEE Computer*, vol. 35, No. 1, pp.70-78, January 2002.

[CBQ00] R. Casado, A. Bermudez, F.J. Quiles, J.L. Sanchez and J. Duato, “Performance Evaluation of Dynamic Reconfiguration in High-Speed Local Area Networks,” *Proceedings of The 6th International Symposium on High-Performance Computer Architecture*, pp.85-96, January 2000.

[CV96] J. Carbonaro and F. Verhoorn, “Cavallino: the teraflops router and NIC,” *Proceedings of Hot Interconnect IV*, pp.157-160, 1996.

[DA93] W.J. Dally and H. Aoki, “Deadlock-free Adaptive Routing in Multicomputer Networks Using Virtual Channels,” *IEEE Transactions on Parallel and Distributed Systems*, vol.4, No.4, pp.466-475, 1993.

[Deh04] A. DeHon, “Unifying Mesh- and Tree-Based Programmable Interconnect,” *IEEE Transactions on Very Large Scale Integration Systems*, Vol.12, No.10, pp.1051-1065, October 2004.

[DG98] G. D.Donley and M. Gujral, “Livelock Avoidance,” *U.S. Patent 5761446*, June.02, 1998.

[DR04] A. DeHon and R. Rubin, “Design FPGA Interconnect of Multilevel Metalization,” *IEEE Transactions on Very Large Scale Integration Systems*, vol.12, No.10, pp.1038-1050, Oct. 2004.

[DS87] W.J. Dally and C.L. Seitz, “Deadlock-free message routing in multiprocessor interconnection networks,” *IEEE Transactions on Computers*, vol.36, no.5, pp.547-553, 1987.

[DT01] W. J. Dally and B. Towles, “Route Packets, Not Wires: On-Chip Interconnection Networks,” *Proceedings of Design Automation Conference*, pp.684-689, June 2001.

[DVK01] J.A. Davis, R. Venkatesan, A. Kaloyeros, M. Beylansky, S.J. Souri, K. Banerjee, K.C. Saraswat, A. Rahman, R. Reif and J.D. Meindi, “Interconnect limits on gigascale integration in the 21st century,” *Proceedings of IEEE*, vol.89, no.3, pp.305-324, 2001.

[DYN02] J. Duato, S. Yalamanchili and L. Ni, “Interconnection networks: an engineering approach,” *Morgan Kaufmann*, 2002.

[Fly97] D. Flynn, “AMBA: Enabling reusable on-chip designs,” *IEEE Micro*, vol.17, no.4, pp.20-27, July 1997.

[GG02] P. Guerrier and A. Greiner, “A generic architecture for on-chip packet-switched interconnections,” *Proceedings of Design Automation and Test in Europe*, pp.250-256, March 2002.

[GLD96] W. Gropp, E. Lusk, N. Doss, A. Skjellum, "A high-performance, portable implementation of the MPI message passing interface standard", *Parallel Computing*, vol. 22, no. 6, pp. 789-828, September 1996.

[HP03] W.H. Ho and and T.M. Pinkston, “A Methodology for Designing Efficient On-Chip Interconnects on Well-Behaved Communication Patterns,” *Proceedings of the Ninth International Symposium on High-Performance Computer Architecture*, pp.377-388, February 2003.

[Ibm99] IBM Corporation, “The CoreConnect™ Bus Architecture,“ *available at http://www.chips.ibm.com/ products/coreconnect/index.html*, 1999.

[Ita01] I.T. Association, “Infiniband architecture specification vol.1 release 1.0.a,” *available at http://www.infinibandta.com*, 2001.

[Itr04] International Roadmap Technology for Semiconductor 2004 Update, *available at http://www.itrs.net/Common/2004Update/2004Update.htm*, 2004.

[Kan03] Naoto Kaneko, “Design and Power Analysis of Soft-Input Soft-Output Viterbi Decoder on Multicontext Device DRP,” *Master Thesis, Department of Open and Environmental System, Faculty of Science and Technology, Keio University*, 2003.

[KFJ01] M. Koibuchi, A. Funahashi, A. Jouraku and H. Amano, “L-turn Routing: An adaptive routing in irregular networks,” *Proceedings of the International Conference on Parallel Processing*, pp.374-383, September 2001.

[KRD03] U. J. Kapasi, S. Rixner, W. J. Dally, B. Khailany, J. H. Ahn, P. Mattson and J. D. Owens, “Programmable Stream Processors,” *IEEE Computers*, pp.54-62, August 2003.

[Lai85] C. Leiserson, “Fat-trees: Universal Networks for Hardware-Efficient Supercomputing,” *IEEE Transactions on Computers*, vol.34, No.10, pp.892-901, October 1985.

[LLS04] J. Liang, A. Laffely, S. Srinivasan and R. Tessier, “An Architecture and Compiler for Scalable On-Chip Communication,” *IEEE Transactions on Very Large Scale Integration Systems*, vol.12, no.7, pp.711-726, July 2004.

[LRL02] K. Lahiri, A. Raghunathan and G. Lakshminarayana, “LOTTERYBUS: A New High-Performance Communication Architecture for System-on-Chip Designs,” *Proceedings of Design Automation Conference*, pp.15-20, June 2002.

[LV02] R. Lysecky and F. Vahid, “Prefetching for Improved Bus Wrapper Performance in Cores,” *ACM Transactions on Design Automation of Electronic Systems*, Vol. 7, No. 1, pp.58-90, January 2002.

[LYB02] D. Lyonnard, S. Yoo, A. Baghdadi, and A.A. Jerraya, “Automatic Generation of Application-Specific Architectures for Heterogeneous MPSoC through Combination of Processors,” *Colloque CAO*, pp.15-18, May 2002.

[Mas02] P. Master, “The age of adaptive computing is here,” *Proceedings of the International Conference on Field Programmable Logic and Applications*, pp.1, 2002.

[MBV02] T. Marescaux, A. Bartic, D. Verkest, S. Vernalde and R. Lauwereins, “Interconnection networks enable fine-grain dynamic multi-tasking on FPGAs,” *Proceedings of 12th International Conference on Field-Programmable Logic and Applications*, pp.795-805, 2002.

[Mot02] M. Motomura, “A dynamically reconfigurable processor architecture,” *presented in Microprocessor Forum*, San Jose, CA, 2002.

[NIN00] N. Nishi, T. Inoue, M. Nomura, S. Matsushita, S. Torii, A. Shibayama, J. Sakai, T. Ohsawa, Y. Nakamura, S. Shimada, Y. Ito, M. Edahiro, M. Mizuno, K. Minami, O. Matsuo, H. Inoue, T. Manabe, T. Yamazaki, Y. Nakazawa, Y. Hirota, Y. Yamada, N. Onoda, H. Kobinata, M. Ikeda, K. Kazama, A. Ono, T. Horiuchi, M. Motomura, M. Yamashina and M. Fukuma, “A 1GIPS 1W single-chip tightly-coupled four-way multiprocessor with architecture support for multiple control flow execution.” *Proceedings of the IEEE International Solid-State Circuits Conference*, pp.418-419, February 2000.

[Oka02] A. Okamura, “VR7701: A High Performance Superscalar Processor with Integrated L2 Cache and Peripherals,” *presented in Embedded Processor Forum*, San Jose, CA, 2002.

[Pci01] PCI Special Interest Group, “PCI Local Bus Specification Rev. 2.3,” October 2001.

[PW04] B. Plunkett and J. Watson, “Adapt2004 ACM Architecture Overview,” *available at http://www.quicksilvertech.com*, 2004..

[RB91] J. Rose and S. Brown, “Flexibility of interconnection structures for field-programmable gate arrays,” *IEEE Journal of Solid-State Circuits*, vol.26, no.3, pp.277-282, March 1991.

[RD03] R. Rubin and A. DeHon, “Design of FPGA Interconnect for Multilevel Metalization,” *Proceedings of the International Symposium on Field Programmable Gate Arrays*, pp.154-163, February, 2003.

[RS97] J.A. Rowson and A. Sangiovanni-Vincentelli, “Interface-Based Design,” *Proceedings of 34th Design Automation Conference*, pp.178-183, 1997.

[RY94] K.K. Ramakrishnan and H. Yang, “The Ethernet Capture Effect: Analysis and Solution,” *Proceedings of IEEE 19th Conference on Local Computer Networks*, pp.228-240, October 1994.

[SB77] H. Sullivan and T.R. Bashkow, “A large scale, homogeneous, fully distributed parallel machine,” *Proceedings of the 4th International Conference on Computer Architecture*, March 1977.

[Sin00] H. Singh, L. Ming-Hau, L. Guangming, F.J. Kurdahi, N. Bagherzadeh, and E.M. Chaves Filho, “Morphosys: An Integrated Reconfigurable System for

Data-parallel and Computation Intensive Applications, “ *IEEE Transactions on Computers*, vol.49, no.5, pp.465-481, 2000.

[SH96] S.L. Scott and G.T. Horson, “The Cray T3E network: adaptive routing in a high performance 3D torus,” *Proceedings of Hot Interconnects IV*, pp.147-156, 1996.

[SR00] J.C. Sancho and A. Robles, “Improving the up*/down* routing scheme for networks of workstations,” *Proceedings of the European Conference on Parallel Computing*, pp.882-889, 2000.

[SRD01] J.C. Sancho, A. Robles and J. Duato, “Effective strategy to compute forwarding tables for Infiniband networks,” *Proceedings of the International Conference on Parallel Processing*, pp.48-57, 2001.

[Son02] Sonics Inc., “SONICS uNetwork Technical Overview,” *available at http://www.sonicsinc.com*, January 2002.

[Son00] Sonics Inc., “Open Core Protocol Specification 1.0,” *available at http://www.socworks.com*, 2000.

[SWW97] W. Saphir, R. Van Der Wijngaart, A. Woo and M. Yarrow, “New Implementations and Results for the NAS Parallel Benchmarks 2,” *Proceedings of 8th SIAM Conference on Parallel Processing for Scientific Computing*, Mar. 1997.

[VSI01] Virtual Socket Interface Association, “Virtual Component Interface Specification Version 2 (OCB 2 2.0),” *available at http://www.vsi.org*, April 2001.

[Wol03] W. Wolf, “Application-Specific Networks-on-Chip,” *Proceedings of SASIMI*, pp.111-118, 2003.

[WOT03] K. Watanabe, T. Otsuka, J. Tsuchiya, H. Harada, J. Yamamoto, H. Nishi, T. Kudoh and H. Amano, “Performance evaluation of RHiNET-2/NI: a network interface for distributed parallel computing systems,” *Proceedings of the International Symposium on Cluster Computing and the Grid*, pp.318-325, 2003.

[Xil04] Xilinx, Inc., “Virtex II Platform FPGAs: Complete Datasheet v3.3,” *available at http://www.xilinx.com*, June 2004.

[YAK04] Yutaka Yamada, Hideharu Amano, Michihiro Koibuchi, Akiya Jouraku, Kenichiro Anjo and Katsunobu Nishimura, “Folded Fat-H-Tree: an interconnection network topology for Dynamic Reconfigurable Processor Array,” *Proceedings of International Conference on Embedded and Ubiquitous Computing*, pp.301-311, August 2004.

[YNL01] S. Yoo, G. Nicolescu, D. Lyonnard, A. Baghdadi, and A.A. Jerraya, “A Generic Wrapper Architecture for Multi-Processor SoC Cosimulation and Design,” *Proceedings of the ninth International Symposium on Hardware/software Co-design*, pp.195-200, April 2001.

[ZC02] L. Zhang and V. Chaudhary, “On the Performance of Bus Interconnection for SoCs,” *Proceedings of 4th Workshop on Media and Stream Processors*, pp.1-10, November 2002.

A List of Related Papers

Journal Papers

Kenichiro Anjo, Michihiro Koibuchi, Yutaka Yamada, Akiya Jouraku and Hideharu Amano, “Evaluation of Local Labeling Scheme on Network-on-Chip,” IEICE Transactions on Information and Systems (In Japanese, to appear).

Kenichiro Anjo, Atsushi Okamura and Masato Motomura, “Wrapper-based Bus Implementation Techniques for Performance Improvement and Cost Reduction,” IEEE Journal of Solid State Circuits. Vol.36, No.5, pp.804-817, May 2004.

Kenichiro Anjo, Hiroaki Inoue, Mitsuru Sato, Tomohiro Kudoh, Hideharu Amano, and Kei Hiraki, “MBP-light: DSM Management Processor on Massively Parallel Processor JUMP-1,” ISPJ Journal, vol.39, no.6, pp.1632-1643, June 1998. (In Japanese)

Hideharu Amano, Akiya Jouraku, and Kenichiro Anjo, “A dynamically reconfigurable hardware on Dynamically Reconfigurable Processor,” IEICE Transactions on Communications, Vol.E86-B, No.12, pp.3385-3391, December 2003.

Hiroaki Nishi, Ken-ichiro Anjo, Tomoiro Kudoh, Hideharu Amano, “The RDT Router Chip: A versatile router for supporting a distributed shared memory,” IEICE Transactions on Information and Systems, Vol.E80-D, No.9, pp.854-862, September 1997.

International Conferences

Kenichiro Anjo, Yutaka Yamada, Michihiro Koibuchi, Akiya Jouraku, and Hideharu Amano, “BLACK-BUS: A New Data-Transfer Technique using Local Address on Networks-on-Chips,” Proceedings of IEEE International Parallel and Distributed Processing Symposium, pp.10-17, April 2004.

Kenichiro Anjo, Atsushi Okamura, Tomoharu Kajiwara, Noriko Mizushima, Masafumi Omori, and Yasuaki Kuroda, “NECoBus: A high-end SoC bus with portable & low-latency wrapper-based interface mechanism,” Proceedings of IEEE Custom Integrated Circuit Conference, pp.315-318, May 2002.

Yutaka Yamada, Hideharu Amano, Michihiro Koibuchi, Akiya Jouraku, Kenichiro Anjo and Katsunobu Nishimura, “Folded Fat-H-Tree: an interconnection network topology for Dynamic Reconfigurable Processor Array,” Proceedings of International Conference on Embedded and Ubiquitous Computing, pp.301-311, Aug 2004.

Masayasu Suzuki, Yohei Hasegawa, Yutaka Yamada, Naoto Kaneko, Katsuaki Deguchi, Hideharu Amano, Kenichiro Anjo, Masato Motomura, Kazutoshi Wakabayashi, Takeo Toi, and Toru Awashima, "Stream Applications on the Dynamically Reconfigurable Processor," Proceedings of International Conference on Field Programmable Technology, pp.137-144, December 2004.

Masayasu Suzuki, Yohei Hasegawa, Yutaka Yamada, Katsuaki Deguchi, Kenichiro Anjo, Toru Awashima, and Hideharu Amano, "Stream Application Evaluation on the DRP-1,” Proceedings of International Conference on COOL Chips VII, pp.33-47, April 2004.

Noriaki Suzuki, Syunsuke Kurotaki, Masayasu Suzuki, Naoto Kaneko, Yutaka Yamada, Katsuaki Deguchi, Yohei Hasegawa, Hideharu Amano, Kenichiro Anjo, Masato Motomura, Kazutoshi Wakabayashi, Takeo Toi, and Toru Awashima, "Implementing and Evaluating Stream Applications on the Dynamically Reconfigurable Processor,” Proceedings of International Conference on Field-Programmable Custom Computing Machine, April 2004.

Hideharu Amano, Akiya Jouraku, and Kenichiro Anjo, “A dynamically adaptive switching fabric on a multicontext reconfigurable device,” Proceedings of International Conference on Field Programmable Logic and Application, pp.161-170, September 2003.

Toshiro Kitaoka, Hideharu Amano, and Kenichiro Anjo, “Reducing the Configuration Loading Time of a Coarse Grain Multicontext Reconfigurable Device,” Proceedings of International Conference on Field Programmable Logic and Application, pp.171-180, September 2003.

Masayuki Mizuno, Kenichiro Anjo, Yoshikazu Sumi, Hitoshi Wakabayashi, Toru Mogami, Tadahiko Horiuchi, and Masakazu Yamashina, “On-Chip Multi-GHz Clocking with Transmission Lines,” Proceedings of IEEE International Solid-State Circuits Conference, pp.366-367, February 2000.

Masayuki Mizuno, Kenichiro Anjo, Yoshikazu Sumi, Muneo Fukaishi, Hitoshi Wakabayashi, Toru Mogami, Tadahiko Horiuchi, and Masakazu Yamashina, “Clock Distribution Networks with On-Chip Transmisson Lines,” Proceedings of International Interconnect Technology Conference, pp.3-5, June 2000.

Taro Fujii, Koichiro Furuta, Masato Motomura, Masahiro Nomura, Masayuki Mizuno, Kenichiro Anjo, Kazutoshi Wakabayashi, Yoshinori Hirota, Yoetsu Nakazawa, Hiroshi Ito, and Masakazu Yamashina “A Dynamically Reconfigurable Logic Engine with a Multi-Context/Multi-Mode Universal-Cell Architecture and Distributed Reconfiguration Control Mechanism,” Proceedings of IEEE International Solid-State Circuits Conference, pp.364-365, February 1999.

Taro Fujii, Koichiro Furuta, Masato Motomura, Masahiro Nomura, Masayuki Mizuno, Kenichiro Anjo, Kazutoshi Wakabayashi, Yoshinori Hirota, Yoetsu Nakazawa, Hiroshi Ito, and Masakazu Yamashina, “A 0.25μm CMOS, 5.1M-Transistor, Dynamically Reconfigurable Logic Engine(DRLE) LSI,” Proceedings of IEICE COOL Chips, pp.51-63, April 1999.

Hiroaki Inoue, Kenichiro Anjo, Junji Yamamoto, Jun Tanabe, Masaki Wakabayashi, Mitsuru Sato, Hideharu Amano, and Kei Hiraki “The preliminary evaluation of MBP-light

with two protocol policies for a massively parallel processor-JUMP-1," Proceedings of IEEE Frontiers of Massively Parallel Computation, pp.268-275, February 1999.

Hiroaki Nishi, Hideharu Amano, Katsunobu Nishimura, Ken-ichiro Anjo, and Tomohiro Kudoh, "The RDT network router chip", Proceedings of IEEE Asia and South Pacific Design Automation Conference, pp.675-676, January 1997.

Hiroaki Nishi, Katsunobu Nishimura, Kenichiro Anjo, Tomohiro Kudo, and Hideharu Amano, "The JUMP-1 router chip: a versatile router for supporting a distributed shared memory, Proceedings of IEEE International Phoenix Conference on Computers and Communications, pp.158-164, March 1996.

Hiroaki Inoue, Ken-ichiro Anjo, Jun Tanabe, Katsunobu Nishimura, Mitsuru Satoh, Kei Hiraki, Hideharu Amano, "MBP-light: A Processor for Management of Distributed Shared Memory on JUMP-1," Proceedings of IEICE COOL chips, pp.169-182, April 1999.

Hiroaki Inoue, Ken-ichiro Anjo, Jun Tanabe, Katsunobu Nishimura, Mitsuru Satoh, Kei Hiraki, Hideharu Amano, "MBP-light: A Processor for Management of Distributed Shared Memory," Proceedings of the 3rd International Conference on ASIC, pp.199-202, October 1998.

Magazine

Kenichiro Anjo, "On-chip buses," IPSJ Magazine vol.44, no.1, pp.73-77, January 2003 (In Japanese).