An SDRAM-Aware Router for Networks-on-Chip

Wooyoung Jang, *Student Member, IEEE,* and David Z. Pan, *Senior Member, IEEE*

***Abstract*—Networks-on-chip (NoCs) may interface with lots of synchronous dynamic random access memories (SDRAM) to provide enough memory bandwidth and guaranteed quality-of-service for future systems-on-chip (SoCs). SDRAM is commonly controlled by a memory subsystem that schedules memory requests to improve memory efficiency and latency. However, a memory subsystem is still a performance bottleneck in the entire NoC. Therefore, memory-aware NoC optimization has attracted considerable attention. This paper presents a NoC router with an explicit SDRAM-aware flow control. Based on priority-based arbitration, our SDRAM-aware flow controller schedules memory requests to prevent bank conflict, data contention, and short turn-around bank interleaving. Moreover, our multi-stage scheduling scheme further improves memory performance and saves NoC hardware costs. Experimental results show that our cost-efficient SDRAM-aware NoC design significantly improves memory latency and utilization compared to the conventional NoC design with no SDRAM-aware router.**

***Index Terms*—Flow control, memory, networks-on-chip, router, scheduler.**

I. Introduction

As SILICON technology enters the nanometer-scale era, thousands of core may be integrated into a single system-on-chip (SoC) for enhanced performance and functionality by 2015 [2]. Global interconnection is a key potential bottleneck to the advancing performance of the nanometer SoCs. Networks-on-chip (NoCs) has been proposed as a promising alternative to the conventional point-to-point interconnection and shared bus architecture [3], [4].

Memory bandwidth to feed a number of cores is another key issue in the modern and future SoC designs. A synchronous dynamic random access memory (SDRAM) is commonly used as an off-chip shared memory since it provides high memory capacity and infinite endurance for modern SoCs. However, current SoCs mostly interface with a single or dual SDRAM since the number of pin assigned to interface with SDRAMs is limited. Consequently, there would be insufficient memory bandwidth to keep up with a lot of high speed cores. For example, Intel Teraflop which is the state-of-the-art NoC and composed of 80 cores is supported by a dual shared memory [6]. If cores will have access to the single or dual memory at the same time, memory latency will be too long to provide real-time computing.

Manuscript received September 21, 2009; revised February 2, 2010; accepted May 7, 2010. Date of current version September 22, 2010. This paper was recommended by Associate Editor L. Benini.

The authors are with the Department of Electrical and Computer Engineering, University of Texas, Austin, TX 78712 USA (e-mail: wyjang@cerc.utexa.edu; dpan@ece.utexas.edu).

Digital Object Identifier 10.1109/TCAD.2010.2061251

As an effective solution of interconnection and memory bandwidth, 3-D NoC based on through-silicon-via technology [5] is gaining momentum and industry adoption. 3-D NoC can be embedded with many SDRAMs on top of processing elements at different layers [7]. It achieves higher system performance and more reliable electrical features than the conventional 2-D NoC that interfaces with a single or dual external SDRAM. Furthermore, it provides low power consumption, low electromagnetic interference, small die and printed circuit board area, and low pin density [7].

Most existing 2-D/3-D NoCs with many cores require a dedicated memory subsystem to control SDRAMs. A memory subsystem schedules SDRAM requests and generates SDRAM interface signals. A memory subsystem is one of the most important components in most SoCs since the performance of the entire system depends on its performance. However, the conventional memory subsystem still underperforms due to special operation flows of SDRAM [8] and dynamic accesses by various processing elements. For example, double data rate (DDR) II SDRAM utilization gets deteriorated up to 55% in a digital television (DTV) application [9], where memory utilization is defined as the number of clock cycles used for data transfer divided by the number of total clock cycles. Moreover, the corresponding number of memory subsystems should also be equipped to control a number of SDRAMs. Our experimental results show that the gate count of a single memory subsystem may occupy over 35% of the entire 3×3 NoC platform, thus the NoC design cost will rapidly increase by more memory subsystems. Therefore, considerable attention has been shifted toward memory-aware NoC exploration to improve memory utilization and latency with the economical design cost of NoC platform [25].

This paper presents an SDRAM-aware NoC router to improve SDRAM utilization and latency. It also decouples the NoC design cost from the number of SDRAM. Our key ideas are twofold. First, if a NoC router schedules SDRAM access packets, the packets arrive at a memory subsystem in the order that is friendly to SDRAM operations. Since our SDRAM-aware router uses existing resources to schedule the packets, e.g., input buffers for storing blocked packets or other flow-control mechanisms, additional circuitry is tiny. On the contrary, a heavy reordering buffer and a complex scheduler are removed in a memory subsystem. Second, a scheduling scheme performed by multiple SDRAM-aware routers outperforms a scheduling scheme performed by a single memory subsystem. The reason is that the performance of single-stage scheduling mainly depends on the number of port/buffer in a single memory subsystem. However, the multi-stage scheduling uses all of the buffers in multiple routers

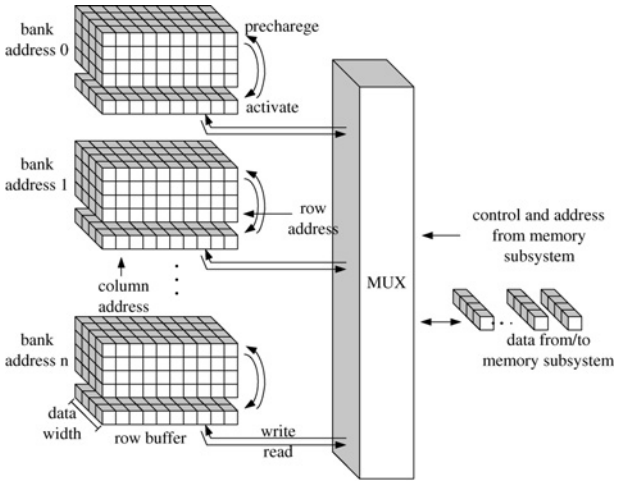


Fig. 1. SDRAM architecture and operation.

to schedule SDRAM access packets/requests. Based on these ideas, the major novelty and contribution of this paper include the following.

1) We propose a novel NoC router architecture with explicit SDRAM-aware flow control to schedule SDRAM access requests instead of using the conventional memory subsystem.
2) We propose SDRAM-aware flow control algorithms to resolve problems of bank conflict, data contention, and short turn-around bank interleaving, which employs priority-based arbitration and multi-stage scheduling.
3) We show that a NoC design embedding our SDRAM-aware router achieves higher memory utilization, shorter memory latency, and cheaper design cost than the conventional NoC design with an SDRAM-unaware router.
4) We show that performance of our SDRAM-aware router gets better for complex NoC architectures and high-performance SDRAM.

To the best of our knowledge, this is the first work that addresses an SDRAM-aware router for NoC. The rest of this paper is organized as follows. In the next section, we survey related works. In Section III, we review basic SDRAM operation principles and SDRAM request scheduling. In Section IV, a problem of the conventional SDRAM-unaware NoC router is presented and our basic solution is proposed. Section V presents detailed description of our SDRAM-aware router. Experimental results are shown in Section VI. Finally, Section VII concludes this paper.

II. Related Works

3-D NoC can be embedded with more SDRAMs than the conventional 2-D NoC commonly interfacing with single or dual external SDRAMs since it has no limitation of the number of pins. As a result, 3-D NoC provides more memory bandwidth, shorter memory latency, and less number of timing-critical paths. However, the corresponding number of a memory subsystem is also required to manage a lot of SDRAMs. A memory subsystem usually consists of three parts, i.e., a buffer, a SDRAM scheduler, and a SDRAM interface signal generator, where a depth of buffer and an SDRAM scheduler for reordering dynamic SDRAM access requests are key components for higher memory utilization and shorter memory latency. A memory scheduler proposed in [10] supports preemption and reordering to optimize offered net bandwidth and average latency. Schedulers discussed in [11] and [12] support preemption for high-priority requests to decouple latency and rate. In [13], PREDATOR is proposed with two step approaches: grouping memory requests and predictable arbitration for the group. A memory scheduler proposed in [14] adopts an adaptive history-based scheduler that uses a history of recently scheduled operations to improve memory efficiency.

Flow control in NoC is on how network resources, e.g., channel bandwidth, buffer capacity, and control state, are allocated to packets traversing a network. In previous works, congestion control is well studied for macro-networks. For example, decentralized control and predictive explicit-rate control are developed in [15], where sources adjust their traffic generation rates based on feedbacks received from bottleneck links. In [16], a predictive flow controller managing a packet injection rate to regulate the number of packet is proposed, based on traffic sources and router models. To minimize overall execution time and link utilization of applications, optimal link scheduling and shared buffer router architecture are proposed in [17]. An open-loop flow control scheme is proposed in [18] to reduce conflicts of data transfers from multiple memory modules to the same masters.

Our SDRAM-aware flow-control mechanism included in multiple NoC routers schedules SDRAM access packets based on priority-based arbitration. It lets the order of packets be friendly with SDRAM operations and works together with other flow control mechanisms mentioned above and with various network routing schemes. Therefore, our SDRAM-aware NoC design achieves higher memory utilization, shorter latency, and cheaper design cost.

III. Preliminaries

A. Basic SDRAM Operation

SDRAM has a 3-D structure, i.e., a bank, a row, and a column as shown in Fig. 1. Basic commands to access SDRAM are activation (ACT), read/write (R/W), and precharge (PRE), where the ACT command is executed with a bank address (BA) and a row address (RA), the R/W command is executed with a BA and a column address (CA), and the PRE command is executed only with a BA. A bank becomes active by an ACT command and idle by a PRE command. An R/W command can be executed only after a bank is activated. In Fig. 1, when a bank is activated, one row data of the bank move to a row buffer of the bank. It takes t_{RCD} to complete an ACT command. Timing parameters of DDR I, II, and III SDRAM used in this paper is shown in Table I [8]. The faster the clock rate is used in DDR SDRAM, the more clock cycles are required to complete SDRAM operations (Table I). For example, DDR I SDRAM working at 133 MHz clock frequency spends only two clock cycles activating a bank, while DDR III SDRAM working at 800 MHz clock frequency spends 11 clock cycles activating a bank. Then, an R/W command is executed on

TABLE I
TIMING PARAMETER OF DDR I, II[a], AND III[a] SDRAM [8]

Timing Parameter	DDR I SDRAM			DDR II SDRAM				DDR III SDRAM			
	133 MHz	167 MHz	200 MHz	200 MHz	267 MHz	333 MHz	400 MHz	400 MHz	533 MHz	667 MHz	800 MHz
CL[b]	2	2.5	3	3	4	4	6	6	8	10	11
WL[c]	1	1	1	2	3	3	5	5	6	7	8
t_{RCD} [d]	2	3	3	3	4	4	6	6	8	10	11
t_{CCD} [e]	1	1	1	2	2	2	2	4	4	4	4
t_{RP} [f]	2	3	3	3	4	4	6	6	8	9	11
t_{WR} [g]	2	3	3	3	4	5	6	6	8	10	12
t_{WTR} [h]	1	1	2	2	2	3	3	4	4	5	6
t_{RTW} [i]	–	–	–	–	–	–	–	7	8	9	9

[a] We assume that posted CAS additive latency is 0 in DDR II/III SDRAM.
[b] CAS latency or read latency.
[c] WL.
[d] Row access strobe (RAS) to CAS delay time.
[e] CAS-to-CAS command delay.
[f] Row PRE time.
[g] Write recovery (WR) time.
[h] Internal write-to-read (WTR) command delay time.
[i] Internal read-to-write (RTW) command delay time for DDR III SDRAM. If burst length (BL) is 8, it is $CL + t_{CCD} + 2 - WL$, and if BL is 4, it is $(= CL + t_{CCD}/2 + 2 - WL)$. In this table, BL is 8.
Row precharge time (RP), row access strobe to column access strobe delay time (RCD)

the active row buffer. After either read latency called column access strobe (CAS) latency (CL) or write latency (WL), successive data go from or to SDRAM. Finally, a PRE command is executed to deactivate the active row buffer in the bank, i.e., data in the row buffer move to the bank of the row buffer. It takes the bank state t_{RP} to become an idle state.

B. SDRAM Scheduling

SDRAM consists of independent multiple banks whereas address and data pin/wire resources serialize accesses to different banks, as shown in Fig. 1. The benefit of this architecture is that pin/wire resources between SDRAM and SoC can be saved and commands to different banks can be pipelined, i.e., while data are transferred to or from any bank, the rest of the bank becomes idle and active for the latter request. Based on this principle, memory subsystems schedule SDRAM access requests. However, improvement of memory performance is still limited due to special operation flows of SDRAMs and clock cycles wasted by timing constraints in Table I. Moreover, it is much worse in a high performance SDRAM. Main factors which deteriorate memory performance are bank conflict, data contention, and short turn-around bank interleaving explained in the next three sections.

1) *Bank Conflict:* Continuously accessing one bank with different RAs is called *bank conflict*, which is the most critical to SDRAM performance. Since a bank activated by the former request should get idle and then active for the latter request again, a lot of clock cycles are required to complete these operations. For example, in Fig. 2, there are two SDRAM schedulers reordering four read requests, i.e., read 1 (RA 0, BA 0, CA 0), read 2 (RA 1, BA 0, CA 0), read 3 (RA 0, BA 1, CA 0), and read 4 (RA 1, BA 1, CA 0). We assume that all schedulers work for DDR II SDRAM at 333 MHz clock frequency. In Fig. 2(a), let them be scheduled in the order, read 1, read 2, read 3, and read 4 by scheduler 1. After performing read 1, read 2 cannot be immediately executed since a row buffer of bank 0 is already occupied by data of RA 0. Hence, a PRE command should release the open row buffer of bank 0 and then an ACT command should be executed to fill the row buffer of bank 0 with data of RA 1. On the contrary, read 3 can be pipelined, called *bank interleaving*, since it has different BA with read 2. As shown in Fig. 2(a), while the bank 0 is activated and accessed for read 2, bank 1 gets activated for read 3. As a result, data 3 accessed by read 3 are generated with no loss of clock cycle. The last read 4 conflicts with read 3 since they have the same BAs, but different RAs.

On the contrary, scheduler 2 changes the execution order of four read requests read 1, read 3, read 2, and read 4 as shown in Fig. 2(b). Since this order does not cause any bank conflict, all read requests are pipelined. That means the second SDRAM scheduler lets all requests to complete faster and the latency of data 3 and 4 to be shorter than the first SDRAM scheduler. In this example, the first scheduler achieves 9.5% (=4 data/42 clock cycles) memory utilization and the second scheduler achieves 13.3% (=4 data/30 clock cycles) memory utilization. Therefore, the second one is more desirable.

2) *Data Contention:* A case of a write request followed by a read request or a read request followed by a write request is called *data contention*. Data pins/wires are bidirectional in most SDRAMs while control and address pins/wires are unidirectional. As a result, input data may be collided with output data. To transfer data to SDRAM after receiving data from SDRAM, there should be at least one clock cycle interval between writing data and reading data in DDR I/II SDRAM. Since internal RTW command delay time (t_{RTW}) is required in DDR III SDRAM, as shown in Table I, an interval between read data and write data happens up to two clock cycles. t_{RTW} is $CL+t_{CCD}+2-WL$ if BL is 8 or t_{RTW} is $CL+t_{CCD}/2+2-WL$ if BL is 4. Hence, data contention is naturally hidden behind this delay time in DDR III SDRAM.

On the contrary, a read command following a write command needs internal WTR command delay time (t_{WTR}) to be executed. Then, after read latency or CL, reading data can be received from SDRAM. WTR data contention is naturally hidden behind t_{WTR} and CL, but they cause memory utilization and memory latency degraded critically. Therefore, continu-

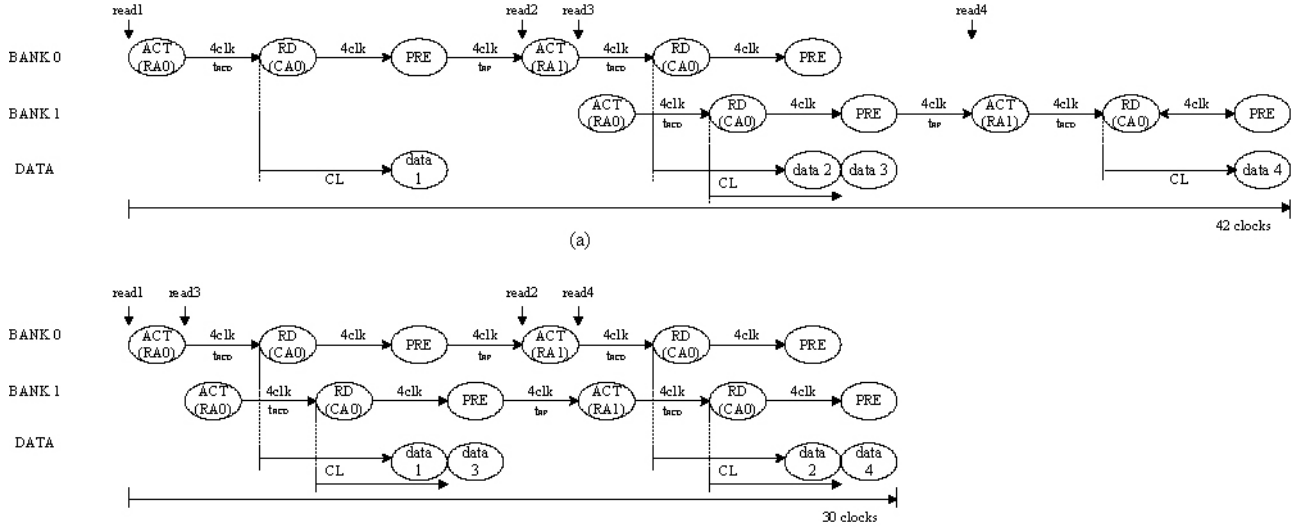


Fig. 2. Examples of bank conflict and interleaving for DDR II SDRAM at 333 MHz. (a) Scheduler 1: read 1 (RA 0, BA 0, CA 0), read 2 (RA 1, BA 0, CA 0), read 3 (RA 0, BA 1, CA 0), and read 4 (RA 1, BA 1, CA 0). (b) Scheduler 2: read 1 (RA 0, BA 0, CA 0), read 3 (RA 0, BA 1, CA 0), read 2 (RA 1, BA 0, CA 0), and read 4 (RA 1, BA 1, CA 0).

ous read or write requests are preferred to access SDRAM efficiently.

For example, in Fig. 3, there are two SDRAM schedulers reordering two write requests and two read requests, i.e. write 1 (RA 0, BA 0, CA 1), read 2 (RA 0, BA 0, CA 2), write 3 (RA 0, BA 0, CA 3), and read 4 (RA 0, BA 0, CA 4). All schedulers interface with DDR II SDRAM working at 266 MHz clock frequency. As shown in Fig. 3(a), let them scheduled in the order, write 1, read 2, write 3, and read 4 by scheduler 1. In this figure, read 2 cannot be immediately performed after writing all data 1 since t_{WTR} is required to accept the next read command. Furthermore, since data 2 are received from SDRAM after read latency or CL, a read request following a write request wastes total t_{WTR} and CL cycles even if bank conflict does not happen between two requests. If both bank conflict and data contention happen simultaneously, bank conflict is commonly prioritized. Since bank conflict wastes more clock cycles than data contention, data contention is hidden behind bank conflict. On the other hand, a write request following a read request has no internal command delays in DDR I/II SDRAM. Instead, a write command performing write 3 should be given to DDR SDRAM when it does not cause any collision with data 2. Most DDR I/II SDRAM schedulers get at least one clock cycle interval between read data and write data. If DDR III SDRAM is used, data contention is hidden naturally behind t_{RTW}. The last read 4 requires both t_{WTR} and CL before transferring data 4.

On the contrary, scheduler 2 changes the order of two write requests and two read requests read 2, read 4, write 1, and write 3 as shown in Fig. 3(b). Since this order causes one data contention wasting just one clock cycle, all R/W requests are performed faster than scheduler 1. In common SDRAM operations, after writing data 3, WR time (t_{WR}) is required to accept a PRE command. Scheduler 1 and scheduler 2 take 28 and 20 clock cycles, respectively, until bank 0 becomes idle after performing all requests. As a result, scheduler 1 achieves 14.3% (=4 data/28 clock cycles) memory utilization and scheduler 2 achieves 20% (=4 data/20 clock cycles) memory utilization. Therefore, continuous read or write requests are encouraged to access SDRAM efficiently.

3) *Short Turn-Around Bank Interleaving:* A bank interleaving approach as a solution of bank conflict is the efficient technique. Hence, high memory utilization and short memory latency can be achieved as explained in Section III-B1. However, bank interleaving may achieve little improvement, in particular, in high performance SDRAM even if bank interleaving is performed completely. In Table I, as an operating clock of SDRAM is faster and faster, ACT delay time (t_{RCD}), deactivation delay time (t_{RP}), and R/W latency (CL/WL) are also longer and longer. The long delay times let the benefit of bank interleaving critically degraded since a bank interleaved may not get sufficient time to be deactivated or reactivated after the bank is accessed by the previous request with different RA.

For example, in Fig. 4, there are two SDRAM schedulers reordering four read requests, i.e., read 1 (RA 0, BA 0, CA 0), read 2 (RA 0, BA 1, CA 0), read 3 (RA 1, BA 0, CA 0), and read 4 (RA 0, BA 2, CA 0). We assume that all schedulers work for DDR III SDRAM at 800 MHz clock frequency. In Fig. 4(a), let them be scheduled in the order, read 1, read 2, read 3, and read 4 by scheduler 1 such that all read requests are performed without bank conflict. After performing read 1, bank 0 is deactivated and read 2 starts to receive data 2. Then, read 3 waits until all data 2 are received. However, read 3 accessing bank 0 cannot be performed even if read 2 is done and the relation between read 2 and read 3 is bank interleaving. The reason is that bank 0 accessed by read 1 is not deactivated due to too long t_{RP}, i.e., operations for read 3 such as deactivation, reactivation, and R/W cannot be hidden behind the process of read 2. Hence, while bank 0 is deactivated, reactivated with data of RA 1 and ready to transfer data 3, any data cannot be transferred or received from other banks, which makes memory utilization and latency degraded.

On the contrary, scheduler 2 changes the execution order of read 3 and read 4 as shown in Fig. 4(b). As a result,

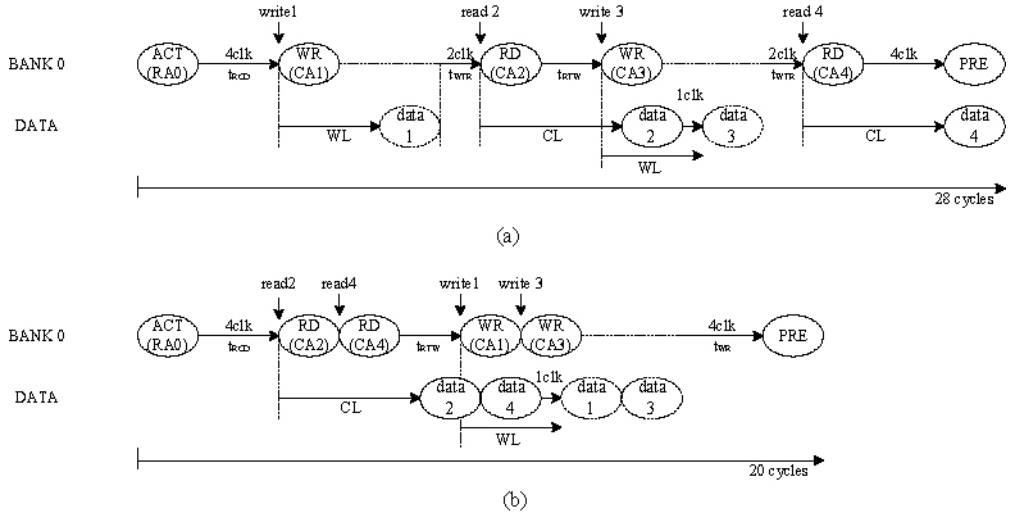


Fig. 3. Examples of data contention for DDR II SDRAM at 266 MHz. (a) Scheduler 1: write 1 (RA 0, BA 0, CA 1), read 2 (RA 0, BA 0, CA 2), write 3 (RA 0, BA 0, CA 3), read 4 (RA 0, BA 0, CA 4). (b) Scheduler 2: read 2 (RA 0, BA 0, CA 2), read 4 (RA 0, BA 0, CA 4), write 1 (RA 0, BA 0, CA 1), write 3 (RA 0, BA 0, CA 3).

read 4 accessing bank 2 can be hidden behind the process of executing read 2 and even read 3 accessing bank 0 can be hidden behind the process of executing read 4. If there is another read 5 accessing bank 3 and it is performed between read 4 and read 3, data may be transferred more continuously with no loss of clock cycle. Consequently, memory utilizations by scheduler 1 and scheduler 2 are 6.7% (=4 data/60 clock cycles) and 7.4% (=4 data/54 clock cycles), respectively. Since this problem is more serious in high performance DDR SDRAM, a memory subsystem should check when banks get active again even if bank interleaving is performed completely.

IV. NoC Design with SDRAM

A. Problem Description

Bank conflict and data contention frequently happen in the conventional NoC design due to limited resources such as an input buffer in a memory subsystem. Moreover, short turn-around bank interleaving also happens in high performance DDR SDRAM. Fig. 5 shows a simple example of bank conflict in a 2×3 NoC design under the limited resources. This NoC includes a single memory subsystem that consists of an input buffer, a memory scheduler, and an SDRAM interface signal generator. The memory scheduler reorders packets stored in the input buffer to avoid bank conflict, data contention, and short turn-around bank interleaving. In this figure, *RxBy* means that a RA and a BA of packet are *x* and *y*, respectively. An arrow indicates that a packet will move to the direction at the next clock cycle. We assume that a length of all packets is 1, the memory subsystem includes a two-depth input buffer to store two packets and the scheduler makes one of two stored packets executed every cycle (although execution time is actually longer than one cycle). In Fig. 5, round-robin arbitration [19] is adopted as a flow control mechanism of NoC routers to assign a channel and an input buffer of the next node to one packet among several competing packets. At cycle 0, three packets, R2B0, R2B1, and R3B0 get a competition for an advance to the router interconnected to the memory subsystem and we assume that R2B0 wins. R0B1 is executed in the memory subsystem. At cycle 1, R2B0 advances to the router interconnected to the memory subsystem and then R3B1 also advances to the empty router by the advance of R2B0. Then three packets, R2B1, R3B0, and R3B1 also get the competition such that R3B0 wins by round-robin arbitration. In the memory subsystem, R0B0 but not R1B1 is executed for avoiding bank conflict since R0B1 accessing bank 0 is performed at cycle 0. At cycle 2, R3B0 advances in the router interconnected to the memory subsystem and R1B1 is executed in the memory subsystem. Then, two packets, R3B1 and R2B1 get the competition such that R2B1 wins by round-robin arbitration. At cycle 3, bank conflict happens in the memory subsystem since current execution is a bank 0 request and two buffers are also stored with bank 0 requests, where all RAs are different. Although the efficient memory subsystem is included in the NoC design, it is difficult to avoid bank conflict completely under the limited depth of a buffer and the dynamic SDRAM accesses of processing elements. Data contention and short turn-around bank interleaving can happen in the conventional NoC design by similar mechanism to this example.

B. Basic Idea of Our Approach

In our NoC design, scheduling SDRAM request packets is performed by multiple SDRAM-aware routers. This architecture makes the possibility of bank conflict lower since packets arrive at a memory subsystem in the order that is friendly to SDRAM operations. Fig. 6 shows how NoC with our SDRAM-aware router works well without bank conflict. At the first competition (cycle 0) for an advance to the router interconnected to the memory subsystem, the winner is R2B1 accessing bank 1 since the former packet (R1B0) passed in this router accesses bank 0. The rest of packet causes bank conflict since they R/W data in the same bank but different RAs from the former packet. At cycle 1, R2B1 advances

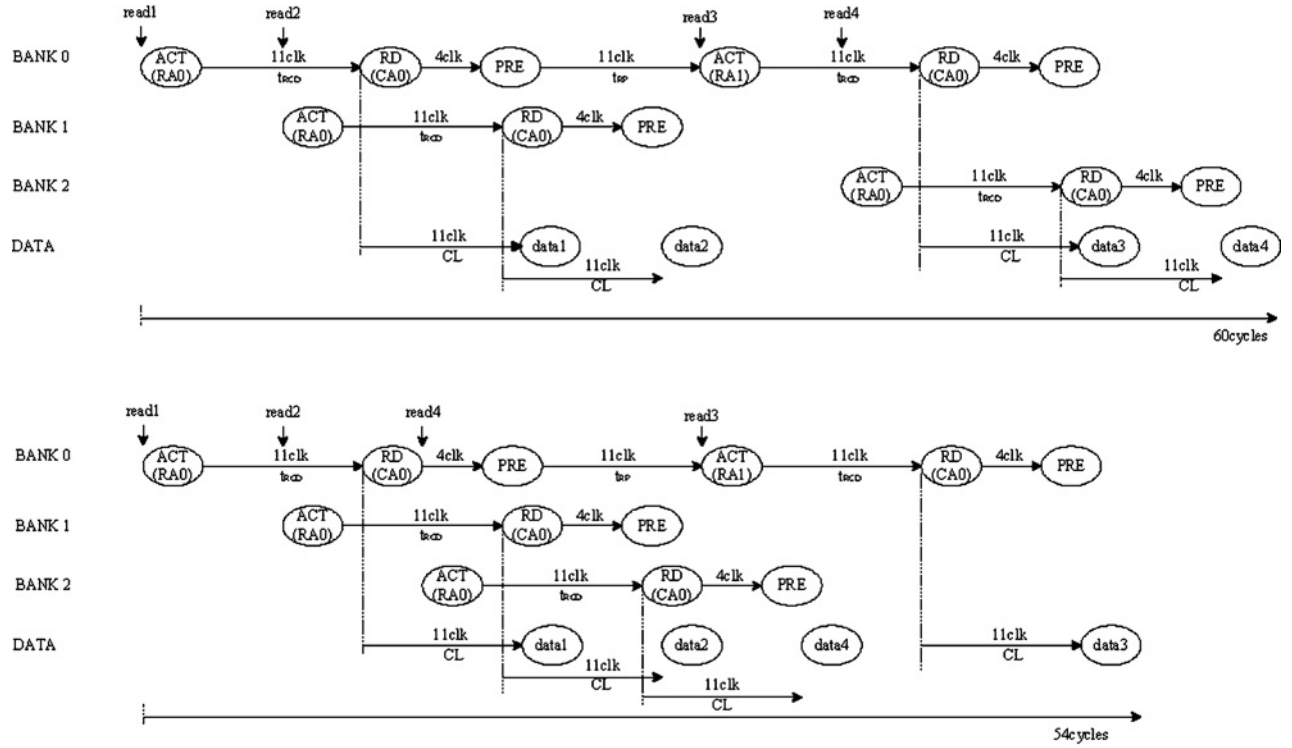


Fig. 4. Examples of short turn-around bank interleaving for DDR III SDRAM at 800 MHz. (a) Scheduler 1: read 1 (RA 0, BA 0, CA 0), read 2 (RA 0, BA 1, CA 0), read 3 (RA 1, BA 0, CA 0), and read 4 (RA 0, BA 2, CA 0). (b) Scheduler 2: read 1 (RA 0, BA 0, CA 0), read 2 (RA 0, BA 1, CA 0), read 4 (RA 0, BA 2, CA 0), and read 3 (RA 1, BA 0, CA 0).

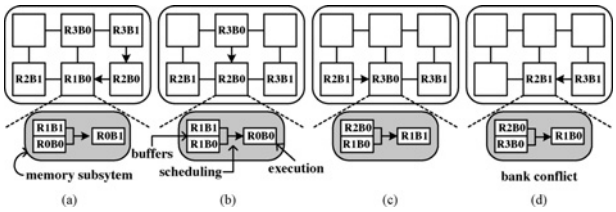


Fig. 5. NoC router with round-robin flow control. (a) Cycle 0. (b) Cycle 1. (c) Cycle 2. (d) Cycle 3.

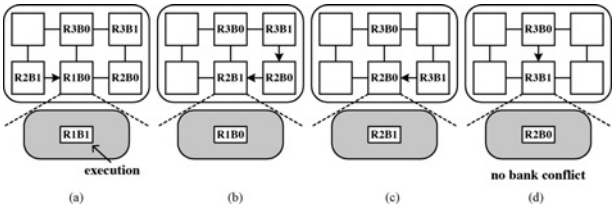


Fig. 6. NoC router with SDRAM-aware flow control. (a) Cycle 0. (b) Cycle 1. (c) Cycle 2. (d) Cycle 3.

to the router interconnected to the memory subsystem and then R2B0 and R3B0 get the competition. Both can be a winner for the next advance since they access bank 0. In this example, R2B0 is chosen by our SDRAM-aware router. At cycle 2, R2B0 advances to the router interconnected to the memory subsystem and R3B1 avoiding bank conflict wins against R3B0 for the next advance. Finally, R3B1 advances to the router interconnected to the memory subsystem and R3B0 follows R3B1 at cycle 3. As a result, a NoC design with our SDRAM-aware router avoids bank conflict better than a NoC design with the conventional memory subsystem and router.

A single memory subsystem usually controls one channel of SDRAM in the conventional NoC, which means the same number of memory subsystem as the number of SDRAM channel is required. Whereas it is allowable to use multiple SDRAMs for high performance, it is not desirable to use a corresponding number of memory subsystems. The reason is that the memory subsystem as shown in Fig. 5 is too high in terms of hardware cost due to the heavy input buffer and the complex scheduler. Furthermore, a depth of input buffer rapidly increases as a length of packet is longer and longer in a high definition graphics/video system. On the contrary, the proposed architecture saves the NoC design cost since any input buffer and any scheduler are not required in the memory subsystem as shown in Fig. 6. Instead, a simple flow controller is included in multiple routers, which has a very low hardware cost compared to an input buffer and a scheduler in a memory subsystem. In the next section, we present a novel SDRAM-aware NoC router in detail.

V. SDRAM-Aware Router

For a wide range of applications, the proposed NoC router is about a novel paradigm for SDRAM-aware-NoC exploration, which has a flow-control mechanism that improves memory utilization and memory latency with a cost-effective NoC platform. Indeed, based on our idea present in Section IV-B, any deterministic and adaptive routing scheme can be combined to implement our SDRAM-aware router. Another

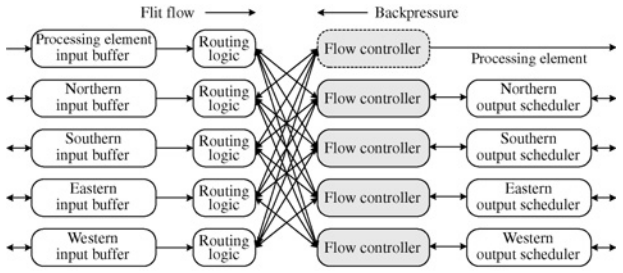


Fig. 7. Proposed NoC router for mesh architecture.

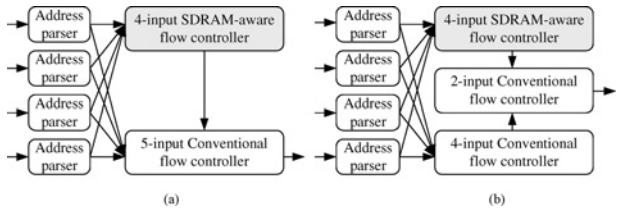


Fig. 8. SDRAM-aware flow controller combined with conventional flow controller in mesh network. (a) Serial implementation. (b) Parallel implementation.

flow-control mechanism mentioned in Section II can be also combined to avoid deadlock and livelock [19], to make traffic load balanced on a network [15]–[17] and to manage buffers and channel bandwidth [18].

A. Router Description

Our NoC router consists of an input buffer, a routing logic, a flow controller, and an output scheduler as shown in Fig. 7. A packet is split into so-called flits (flow control digits), which are then routed and stored in a pipelined fashion. The input buffers are managed by a wormhole flow control mechanism or a virtual-channel flow control mechanism and backpressure is used to inform upstream nodes when they must stop transmitting flits because all of the downstream input buffers are full. For our experiment, the wormhole flow control mechanism is implemented due to its simplicity and wide popularity [19] and an on/off flow control mechanism for the backpressure is employed to avoid a loss of flits.

Our SDRAM-aware router can be implemented to either deterministic or adaptive routers according to a routing logic that guarantees deadlock and livelock freeness. Virtual channels and deterministic dimension-ordered routings (e.g., XY routing, odd-even routing) are commonly used to prevent deadlock [19]. We implement XY routing that is a deterministic and minimal path routing algorithm such that it guarantees deadlock-free and livelock-free routing. In addition, we consider an ordering issue when a master core sends a read request to another slave core before the master core receives a read data from one slave core or when a master core requests another read data to a slave core in NoC employing an adaptive router before the master core receives one read data from the slave core. This ordering issue can be solved by [20] or under the following constraint: a master core can send a read request to a slave core only after the master core receives all data requested. The latter solution is employed in our implementation. In addition, since our SDRAM-aware flow control algorithm is performed with in-order buffers, the ordering problem does not happen in each SDRAM-flow control.

In this router, more than two flits arriving on different input buffers at the same time may both desire the same channel toward a memory subsystem. In this situation, our flow-control mechanism resolves this contention, allocating the channel to one packet and dealing with the others, blocked packets. Fig. 8 shows our SDRAM-aware flow controller combined with the conventional flow controller. In Fig. 8, a address parser sends an incoming memory request packet to our SDRAM-aware flow controller and an incoming normal packet to the conventional flow controller. Our SDRAM-aware flow controller schedules the memory packets in order to prevent bank conflict, data contention, and short turn-around bank interleaving. In the next section, the SDRAM-aware flow-control algorithm using a priority-based arbitration is described minutely. Then, the resulting memory request packet competes with normal packets by the conventional flow control mechanism. Hence, normal packets can reach their destination with no additional communication delay.

Fig. 8(a) shows its serial implementation. This architecture causes a timing path to be much longer since a 5-input conventional flow control algorithm is performed after performing our 4-input SDRAM-aware flow control algorithm. On the contrary, in Fig. 8(b), our 4-input SDRAM-aware flow controller for memory packets and a 4-input conventional flow control algorithm for normal packets are parallelly performed. Finally, two resulting packets are scheduled by a 2-input conventional flow controller. This parallel implementation can minimize an increase of timing path whereas its design cost is more expensive than the design cost of the serial implementation. We adopt this parallel implementation in our experiment. In addition, our flow controllers adopt winner-take-all bandwidth allocation that allocates all of the bandwidth to just one packet until it is finished or blocked before serving the other packets [19].

An output scheduler either detects if an input buffer of the next router is available or expects when the input buffer is available. When an input buffer of the next router is full and a deterministic routing logic is implemented, an output scheduler lets the corresponding SDRAM-aware flow controller stop scheduling packets. On the contrary, packets given multiple routing paths performed by an adaptive routing logic can be scheduled to other flow controllers less busy.

B. SDRAM-Aware Flow Control for Avoiding Bank Conflict and Data Contention

Our flow control acts to allocate a channel to one of competing flits of which destination is a memory subsystem interfacing with SDRAM. Therefore, our flow-control mechanism performs arbitration to determine which flit gets the channel it has requested. After the arbitration, a winning flit advances over this channel. Our arbitration algorithm also decides how to dispose of any flits that do not get their requested channel.

In Algorithm 1 called scheduling packet to avoid bank conflict and data contention (SP), our arbitration is a priority-based algorithm, where a priority is determined by SDRAM

Algorithm 1 Scheduling Packet to Avoid Bank Conflict and Data Contention

Input: $h(n)$, $h_i(n+1)$ and Table II

1: **for** each $h_i(n+1)$, $i \epsilon I$ **do**
2: **if** $h_i(n+1)$ is a new packet entering to the router **then**
3: $w_i = 0$;
4: **else**
5: $w_i = w_i$ + waiting cycles from previous arbitration(n);
6: **end if**
7: d_i = delay cycle between $h(n)$ and $h_i(n+1)$ from Table II;
8: $p_i = w_i - d_i$;
9: **end for**
10: $h_i(n+1)$ with maximum(p_i) is allocated to a channel;

Output: $h(n+1)$

awareness. The priority is assigned to all head flits of which destination is a memory subsystem. Let $h(n)$ be a head flit of a packet, which already has been allocated a channel by the SDRAM-aware flow control at the nth arbitration. Body and tail flits are assigned the same channel as their head flit. Let $h_i(n+1)$ be one of all competing head flits (I) which should be allocated to the same channel as $h(n)$ by the SDRAM-aware flow control at the $(n+1)$th arbitration, where $i \epsilon I$. The head flits, $h(n)$ and $h_i(n+1)$ contain address and command information to access SDRAM, denoted by (RA_n, BA_n, R/W_n) and ($RA_{n+1,i}$, $BA_{n+1,i}$, $R/W_{n+1,i}$), respectively. At the $(n+1)$th arbitration, all $h_i(n+1)$ are compared to $h(n)$ and then are given a delay penalty from Table II (line 7) that is composed from DDR I, II, and III SDRAM working at 133–800 MHz clock frequency (with 266 MHz to 1.6 GHz data rate) [8].

Table II shows how many clock cycles are wasted by bank conflict and data contention or a combination thereof when $h_i(n+1)$ accesses SDRAM after $h(n)$. If bank conflict and data contention happen simultaneously, bank conflict is commonly prioritized since bank conflict wastes more clock cycles than data contention. According to a R/W command, a BA and a RA, there are 12 cases as shown in Table II. Twelve cases are also classified into eight delay types that are described in the next sections.

1) *Delay a:* Case 1 and case 10 have no clock cycle loss since $h_i(n+1)$ is the same R/W command, BA and RA as $h(n)$. These cases indicate that the same row data of the same bank are again accessed by the same command. Thus, the bank does not need to be deactivated and reactivated, which causes the clock cycle loss. In addition, R/W latency of $h_i(n+1)$ can be hidden while $h(n)$ is accessed. In Fig. 3(b), the relation between read 2 and read 4 is case 1 and the relation between write 1 and write 3 is case 10.

2) *Delay b:* Case 2 is the read-to-read bank conflict explained in Section III-B1. Before executing the latter read accessing the same bank but a different row, the bank must be deactivated, i.e., data in the row buffer move to the corresponding row of the bank. Then, the bank must be activated again, which indicates that the row buffer should be again filled with new data for the latter read. Thus, it takes t_{RP}+t_{RCD}+CL to receive data of the latter read after receiving data of the former read. This case is shown in the relation between read 1 and read 2 and in the relation between read 3 and read 4 in Fig. 2(a).

3) *Delay c:* Case 3 and case 12 have no clock cycle loss since bank interleaving is completely performed as mentioned in Section III-B1. Case 3 is the read-to-read bank interleaving, as shown in Fig. 2(b), and case 12 is the write-to-write bank interleaving. Since a BA of the latter request is different from that of the former request, the bank accessed by the latter request can be activated while data of the former request are transferred to or from SDRAM. Then, when data of the former request complete to transfer, data of the latter request can be accessed with no loss of clock cycle.

4) *Delay d:* Case 4 and case 6 have at least one clock cycle interval between the former read data and the latter write data to avoid data contention in DDR I/II SDRAM as shown in Section III-B2. In DDR III SDRAM, the latter write command can be executed internal RTW command delay time (t_{RTW}) after the former read command. Then, write data can be transferred to SDRAM after WL. Thus, actual write data are transferred to DDR III SDRAM, two clock cycles after receiving the last read data. Thus, t_{RTW} lets data contention hidden naturally. In Fig. 3(a), the relation between read 2 and write 3 is case 4. Case 6 is data contention with bank interleaving.

5) *Delay e:* In case 5, data contention and bank conflict happen at the same time since it is a RTW access and BAs of the read request and the write request are same but their RAs are different. As mentioned before, bank conflict should be considered preferentially since it wastes more clock cycles than data contention. Before writing data to the different row in the same bank, the row buffer should be idle after reading data and then active. It takes t_{RP} to be idle and t_{RCD} to be active again. Then, data can be written after WL. Data contention hides naturally behind this bank conflict.

6) *Delay f:* Case 7 is the WTR data contention when the latter read request accesses data placed in the same bank and row as the former write. Case 9 is also the WTR data contention with bank interleaving since the latter read has a different BA. To read data after writing data placed in the same bank and row or in a different bank, a read command is accepted internal WTR command delay time (t_{WTR}) after writing the last data to SDRAM. Then read data are transferred from the SDRAM after read latency.

7) *Delay g:* Case 8 causes the longest delay time due to the WTR bank conflict. Data contention is ignored since the bank conflict is more critical. The latter read request accesses data placed in the same bank but a different row. Thus, after the former write request, the bank should be idle. t_{WR} is required to accept a PRE command for deactivation after writing the last data and it takes t_{RP} to complete the PRE command. Furthermore, t_{RCD} is required to activate the row buffer for the latter read request. Then, data can be received read latency after accepting a read command. Thus, total delay time is t_{WR}+t_{RP}+t_{RCD}+CL.

TABLE II
DATA DELAY BETWEEN $h(n)$ AND $h_i(n+1)$

Relation of $h(n)$ with RW_n, BA_n, and RA_n and $h_i(n+1)$ with $RW_{n+1,i}$, $BA_{n+1,i}$, and $RA_{n+1,i}$				DDR I SDRAM (MHz)			DDR II SDRAM (MHz)				DDR III SDRAM (MHz)				Case
				133	167	200	200	267	333	400	400	533	667	800	
$RW_n = R$	$RW_{n+1,i} = R$	$BA_n = BA_{n+1,i}$	$RA_n = RA_{n+1,i}$	0	0	0	0	0	0	0	0	0	0	0	1[a]
			$RA_n \neq RA_{n+1,i}$	6	8.5	9	9	12	12	18	18	24	30	33	2[b]
		$BA_n \neq BA_{n+1,i}$		0	0	0	0	0	0	0	0	0	0	0	3[c]
	$RW_{n+1,i} = W$	$BA_n = BA_{n+1,i}$	$RA_n = RA_{n+1,i}$	1	1	1	1	1	1	1	2	2	2	2	4[d]
			$RA_n \neq RA_{n+1,i}$	5	7	7	8	11	11	17	17	22	27	30	5[e]
		$BA_n \neq BA_{n+1,i}$		1	1	1	1	1	1	1	2	2	2	2	6[d]
$RW_n = W$	$RW_{n+1,i} = R$	$BA_n = BA_{n+1,i}$	$RA_n = RA_{n+1,i}$	3	3.5	5	5	6	7	9	10	12	15	17	7[f]
			$RA_n \neq RA_{n+1,i}$	8	11.5	12	12	16	20	24	24	32	40	45	8[g]
		$BA_n \neq BA_{n+1,i}$		3	3.5	5	5	6	7	9	10	12	15	17	9[f]
	$RW_{n+1,i} = W$	$BA_n = BA_{n+1,i}$	$RA_n = RA_{n+1,i}$	0	0	0	0	0	0	0	0	0	0	0	10[a]
			$RA_n \neq RA_{n+1,i}$	7	10	10	11	15	15	23	23	30	37	42	11[h]
		$BA_n \neq BA_{n+1,i}$		0	0	0	0	0	0	0	0	0	0	0	12[c]

[a] Accessing the same row data one more time.
[b] $t_{RP} + t_{RCD} + CL$.
[c] Bank interleaving.
[d] One clock cycle interval between input and output in DDR I/II SDRAM. Two clock cycle intervals between input and output in DDR III SDRAM.
[e] $t_{RP} + t_{RCD} + WL$.
[f] $t_{WTR} + CL$.
[g] $t_{WR} + t_{RP} + t_{RCD} + CL$.
[h] $t_{WR} + t_{RP} + t_{RCD} + WL$.

8) *Delay h:* Case 11 is the write-to-write bank conflict since the latter write request accesses the same bank but a different row from the former write request. As presented in the previous case, *delay g*, it takes a bank t_{WR} and t_{RP} to be idle after writing the last data. Then, its row buffer gets active with row data for the latter write request. It takes t_{RCD} to be active and write data are finally transferred to SDRAM after WL.

Our priority-based arbitration guarantees the upper bound latency even if a high delay penalty of packet given from Table II lasts for a long time. For example, let a packet with case 11 lose a competition against a packet with case 10. If it meets another packet with case 10 at the next competition, the defeated packet keeps losing the competition since the delay penalty is not changed. Thus, the defeated packet is required to escape from this competition after several defeats. To solve this starvation problem, our flow control counts the number of clock cycle passed from the first competition to the current competition (line 5) for each defeated packet. Then, this waiting clock cycle is subtracted by the delay clock cycle obtained in Table II (line 8). By this operation, any packet delayed for the amount of the worst delay (case 8) does not have a lower priority than a new packet entered in the router. For example, in DDR III SDRAM working at 533 MHz and 800 MHz clock frequency, the packets waiting for 32 and 45 clock cycles get higher priority than any new packet entered in the router, respectively.

Finally, the packet with the maximum p_i is allocated to a channel (line 10). In our SDRAM-aware flow control, the packet with longer waiting cycle and shorter delay cycle gets a higher priority. Then, the rest of packet that is blocked waits for the next competition or get another competition at a different SDRAM-aware flow controller if multiple routing paths are allocated by a routing logic. Thus, if an adaptive router instead of a deterministic router is employed in a routing logic, the performance would be better.

C. SDRAM-Aware Flow Control for Avoiding Short Turn-Around Bank Interleaving

As mentioned in Section III-B3, a short turn-around bank interleaving problem is not critical for low-performance SDRAM such as DDR I SDRAM since a bank has sufficient time to be deactivated or reactivated until the bank is accessed again. The reason is that short deactivation ($t_{WR}+t_{RP}$ for writing and t_{RP} for reading) or reactivation time (t_{RCD}) is hidden behind the process of accessing a different bank. On the contrary, deactivation, reactivation, and R/W latency time are so long in high-performance SDRAM that it is difficult for them to hide behind the process of accessing a different bank. For example, in DDR III SDRAM working at 800 MHz clock frequency, it takes a bank 23, 11, and 11 clock cycles to deactivate, reactivate, and output data, respectively, after writing data. Thus, before the written bank is again read with a different RA, a scheduler should let different banks accessed for at least 23 clock cycles to improve memory utilization.

The proposed SP algorithm just schedules memory request packets to prevent bank conflict and data contention. Hence, it should check whether a bank accessed by $h_i(n+1)$ is given sufficient deactivation time before the bank is activated. It is well explained together with hardware/architecture of an SDRAM interface signal generator. Fig. 9 is an SDRAM interface signal generator commonly used, where an input packet passes three buffers to generate SDRAM commands such as a PRE command, an ACT command, and an R/W command. First, an input packet arriving at an SDRAM interface signal generator is stored in a deactivation buffer but not an ACT buffer as shown in Fig. 9. That means a bank keeps activating after accessing data, called open page mode. It can be useful for high memory utilization since most of the cores access data placed in continuous memory addresses, i.e., the same bank and RAs but different CAs. Then, if the input packet accesses the bank previously activated with a different RA, a PRE command is output to an SDRAM interface

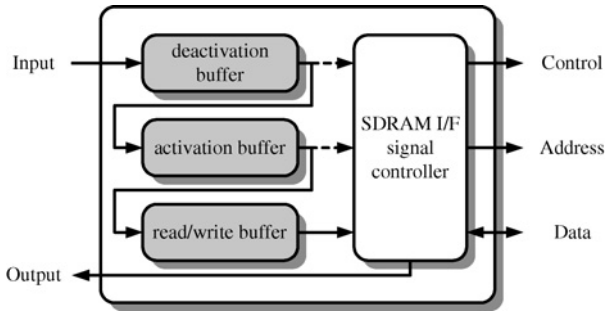


Fig. 9. SDRAM interface signal generator.

signal controller to deactivate the bank and then the packet moves to an ACT buffer. On the contrary, if the input packet accesses the bank previously activated with the same RA or if the input packet accesses the bank already deactivated, the packet just passes a deactivation buffer with no PRE command. If a packet stored in an ACT buffer accesses the bank deactivated, the packet lets an ACT command generated to an SDRAM interface signal controller and then moves to a R/W buffer. Finally, a packet stored in a R/W buffer always lets an R/W command generated to an SDRAM interface signal controller. An SDRAM interface signal controller receives a PRE command, an ACT command, and an R/W command from those buffers and then generates the final interface signals to SDRAM.

To solve the short turn-around bank interleaving problem happening in this SDRAM interface signal generator, a packet that is output from a deactivation buffer should pass $(t_{\mathrm{WR}})+t_{\mathrm{RP}}$ until the packet is output from an ACT buffer. In addition, a packet that is output from an ACT buffer should pass t_{RCD} until the packet is output from a R/W buffer. Since the deactivation time is longer than the ACT time and R/W latency, the interval between packets accessing the same bank and a different row should be at least t_{RP}or $t_{\mathrm{WR}} + t_{\mathrm{RP}}$ depending on a read request or a write request that the previous packet accessing each bank is.

Algorithm 2, called assigning priority to avoid short turn-around bank interleaving (AP), is executed instead of line 8 in our SP algorithm to solve the short turn-around bank interleaving problem. In the AP algorithm, a clock cycle (d_{is}) required to deactivate each bank is recorded after a R/W operation is completed. Thus, the same number of counter as the number of bank is required to save and count d_{is}. If the packet *h(n)* performed is a write request, d_{is} of bank that *h(n)* accesses is set to $t_{\mathrm{WR}} + t_{\mathrm{RP}}$ in line 4. If the packet *h(n)* performed is a read request, d_{is} of bank that *h(n)* accesses is set to t_{RP} in line 6. Then, all d_{is} are reduced by 1 every clock cycle in line 9. If any packet, $h_i(n + 1)$ is in cases 3, 6, 9, and 12 of Table II with $h(n)$, our AP algorithm checks if the bank accessed by the $h_i(n+1)$ has sufficient deactivation time (lines 11–15). For this operation, d_{id} captures d_{is} in line 12 if the relation between $h(n)$ and $h_i(n + 1)$ is cases 3, 6, 9, and 12. Otherwise, d_{id} is 0 in line 14. Then, d_{id} is compared to d_i obtained from Table II. Finally, larger delay time is chosen as effective delay time and then subtracts a waiting clock cycle as shown in line 16. Our solution makes banks accessed as uniformly as possible such that the banks get the sufficient time to be deactivated for the next request.

Algorithm 2 Assigning Priority to Avoid Short Turn-Around Bank Interleaving

Input: w_i, d_i, $h(n)$ and $h_i(n + 1)$
1: **for** every clock cycle **do**
2: **if** *h(n)* is done **then**
3: **if** *h(n)* is write request **then**
4: d_{is} *of bank(h(n))*= t_{WR}+ t_{RP};
5: **else**
6: d_{is} *of bank(h(n))*= t_{RP};
7: **end if**
8: **end if**
9: d_{is}= d_{is}– 1 for all d_{is};
10: **end for**
11: **if** relation of *h(n)* and $h_i(n + 1)$ is cases 3, 6, 9, and 12 **then**
12: d_{id} = d_{is} of *bank*($h_i(n+1)$);
13: **else**
14: d_{id} = 0;
15: **end if**
16: p_i = w_i– max(d_i, d_{id});
Output: p_i

D. Hardware Complexity

Memory scheduling is performed by our SDRAM-aware flow controller included in multiple NoC routers instead of a single memory subsystem. Thus, simple logics are added for our SP and AP algorithm to compute SDRAM access delays (d_i and d_{id}) and waiting time (w_i) whereas a buffer and a scheduler of memory subsystem are removed as shown in Fig. 6. A buffer in a memory subsystem is used to store several packets and then to reorder the packets for successive delivery of SDRAM data. However, as the massive size of packet is recently generated in graphics processing units (GPU) and a high-definition video system, the size of buffer gets larger. The proposed NoC design does not require any buffers in a memory subsystem since memory scheduling is performed in multiple NoC routers and the maximum four input buffers per router in a regular mesh network substitute for a buffer in a memory subsystem. In addition, the size of input buffer in the router does not increase according to the size of packet since the input buffer is managed by wormhole flow control. Consequently, a distinguished hardware decrease by a buffer in a memory subsystem exceeds a hardware increase by the SDRAM-aware flow controller in multiple routers such that total gate count is reduced.

VI. Experimental Results

Our SDRAM-aware NoC router is implemented with Verilog hardware description language (HDL). We implement a memory subsystem operating for DDR I SDRAM working at 133 MHz and 200 MHz clock frequency, DDR II SDRAM working at 266 MHz, 333 MHz, and 400 MHz clock frequency,

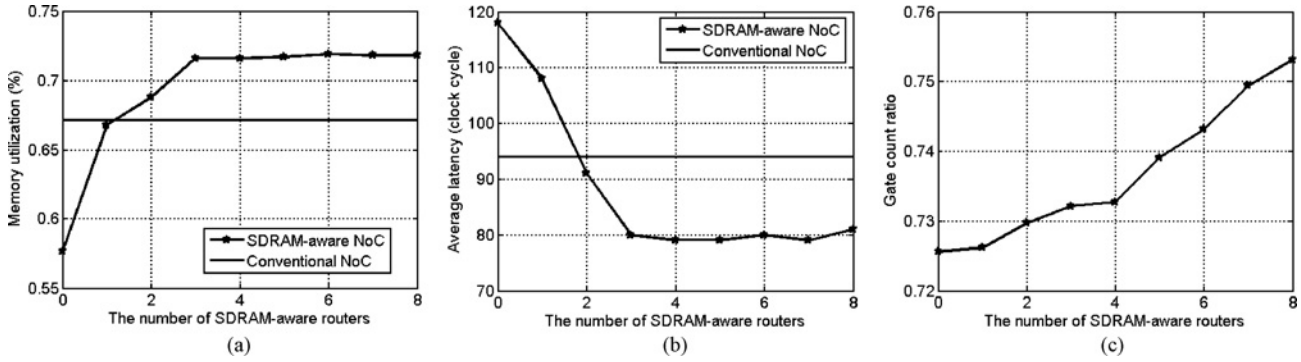


Fig. 10. Comparison in DTV application according to the number of SDRAM-aware routers. (a) Memory utilization. (b) Average latency. (c) Gate count ratio.

and DDR III SDRAM working at 533 MHz and 800 MHz clock frequency [8], all of which consist of four banks. The memory subsystem is implemented with a design concept of Sonics MemMax [21] and Denali Databahn [22]. MemMax schedules memory requests to prevent bank conflict and data contention similar to our SP algorithm. In addition, users can choose a depth of buffers, modes of operation and quality-of-service settings that best suit various applications. Since MemMax supports an open core protocol where request signals and data signals are separated, MemMax requires both a request buffer and a data buffer per thread. We use 4-thread MemMax where each thread requires a 32-flit request buffer and a 32-filt data buffer. Databahn is an SDRAM interface signal generator which schedules RAS, CAS, PRE, and refresh operation. Both are included in the conventional NoC design with a round-robin flow control based router. This is compared to our NoC design including multiple SDRAM-aware routers and an SDRAM interface signal generator instead of a full memory subsystem. Processing elements are mapped to mesh grid by A3MAP [23] and each simulation runs for one million clock cycles.

A. Digital Television Application

The conventional NoC design and our SDRAM-aware NoC design are applied to a Samsung DTV system that consists of nine subsystems, i.e., a central processing unit that consists of ARM9 and several peripherals, a moving picture experts group (MPEG) decoder, a digital natural image engine, GPU, an audio decoder, a transport stream decoder, an audio/video format converter, a channel decoder, and a memory subsystem that interfaces with DDR II SDRAM working at 333 MHz clock frequency. In the conventional NoC design, a router using a round-robin flow control algorithm is gradually replaced with our SDRAM-aware router using the proposed SP algorithm in the order where the router that is the closest to a memory subsystem is replaced first and where the router that is the farthest away from a memory subsystem is replaced last. Fig. 10 shows the results depending on the number of SDRAM-aware router placed in the order.

In Fig. 10(a) and (b), memory utilization and memory latency achieved by the conventional NoC design are 67.2% and 94 cycles, respectively. Memory utilization and memory latency performed by our SDRAM-aware NoC design is just 57% and 119 cycles, respectively, in case that there are no input buffer and no memory scheduler in a memory subsystem and no SDRAM-aware router. However, whenever our SDRAM-aware router is substituted for the conventional router in the DTV system, memory utilization and memory latency improves rapidly. As a result, when three SDRAM-aware routers are substituted for three conventional routers, memory utilization increases up to 72% (i.e., 7.1% better than the conventional NoC design). However, more than four SDRAM-aware routers do not improve memory utilization any more since the solvable bank conflict and data contention are almost prevented by three SDRAM-aware routers. Similarly, in Fig. 10(b), memory latency is also shortened by three SDRAM-aware routers up to 79 cycles (i.e., 16% shorter than the conventional NoC design) since high memory utilization makes a packet performed as fast as possible and our SDRAM-aware flow controller manages the upper bound latency.

Our SDRAM-aware NoC design and the conventional NoC design are synthesized by Synopsys Design Vision with a TSMC130LV library. The gate count of our SDRAM-aware NoC design is 26.8% smaller when three round-robin routers are replaced with our SDRAM-aware routers in Fig. 10(c). In addition, its gate count is 24.8% smaller even if all round-robin routers are replaced with our SDRAM-aware routers. The reason is that a large buffer and a complex scheduler in a memory system are removed whereas an additional hardware increased by our SDRAM-aware flow controller is minimal.

We also implement the SDRAM-aware NoC based a DTV system interfacing with a variety of DDR SDRAMs working at 133–800 MHz clock frequency. Our DTV system works for real-time computing when it interfaces with DDR II SDRAM working at 333 MHz clock frequency. However, to show the benefit of our SDRAM-aware NoC design in various DDR SDRAMs, we let a packet injection rate of each IP changed similar to a change of the SDRAM clock speed. Table III shows memory utilization and latency in our NoC design including three SDRAM-aware routers compared to the conventional NoC design. Our SDRAM-aware NoC design proves more merits on high-performance DDR SDRAM in Table III. For example, our SDRAM-aware NoC improves more 3.7% memory utilization and 14.3% memory latency than the conventional NoC when they all interface with DDR I SDRAM working at 133 MHz clock frequency. On the contrary, our SDRAM-aware improves more 26% memory utilization and 30.8% memory latency than the conventional

TABLE III
COMPARISON IN DTV APPLICATION ACCORDING TO VARIOUS DDR SDRAMS

Measure	Router	DDR I SDRAM		DDR II SDRAM		DDR III SDRAM	
		133 MHz	200 MHz	266 MHz	400 MHz	533 MHz	800 MHz
Memory utilization	Conventional NoC	73.1%	72.2%	70%	61.4%	52.6%	43.5%
	SP	75.8%	74.5%	74.8%	68%	62.3%	54.8%
	Difference (improvement)	**2.7% (3.7%)**	**2.3% (3.2%)**	**4.8% (6.9%)**	**6.6% (10.7%)**	**9.7% (18.4%)**	**11.3% (26%)**
Memory latency	Conventional NoC	84 cycles	85 cycles	96 cycles	110 cycles	126 cycles	146 cycles
	SP	72 cycles	76 cycles	76 cycles	83 cycles	90 cycles	101 cycles
	Improvement	**14.3%**	**10.6%**	**20.8%**	**24.5%**	**28.6%**	**30.8%**

TABLE IV
COMPARISON IN SYNTHETIC BENCHMARKS ACCORDING TO NETWORK SIZE

Measure	Router	3 × 3 NoC	4 × 4 NoC	5 × 5 NoC	6 × 6 NoC	Average
Memory utilization	Conventional NoC	59.4%	58.7%	52.9%	53.2%	56.1%
	SP	63.7%	65.5%	60.3%	61.2%	62.7%
	Difference (improvement)	**4.3% (5.6%)**	**6.8% (11.6%)**	**7.1% (13.4%)**	**8.0% (15%)**	**6.6% (11.8%)**
Memory latency	Conventional NoC	65 cycles	79 cycles	94 cycles	99 cycles	84 cycles
	SP	59 cycles	66 cycles	80 cycles	71 cycles	69 cycles
	Improvement	**9.2%**	**16.4%**	**14.9%**	**28.3%**	**18%**

TABLE V
COMPARISON OF SP AND SP + AP

Measure	Router	DDR I SDRAM		DDR II SDRAM		DDR III SDRAM	
		133 MHz	200 MHz	266 MHz	400 MHz	533 MHz	800 MHz
Memory utilization	Conventional NoC	67%	69.3%	61.8%	51.9%	38.8%	30.9%
	SP	68.2%	69.9%	64.3%	54.9%	47.8%	39.2%
	SP+AP	69.1%	70.5%	66.1%	58.3%	51.4%	42.8%
	Difference (improvement)*	**0.9% (1.3%)**	**0.6% (0.9%)**	**1.8% (2.8%)**	**3.4% (6.2%)**	**3.6% (7.5%)**	**3.6% (9.2%)**
Memory latency	Conventional NoC	85 cycles	87 cycles	96 cycles	115 cycles	151 cycles	186 cycles
	SP	83 cycles	84 cycles	92 cycles	106 cycles	126 cycles	152 cycles
	SP+AP	83 cycles	84 cycles	90 cycles	100 cycles	114 cycles	138 cycles
	Improvement*	**0%**	**0%**	**2.2%**	**5.7%**	**9.5%**	**9.2%**

*It is the difference (improvement) between SP and SP+AP.

NoC when they all interface with DDR III SDRAM working at 800 MHz clock frequency. Since timing constraints caused by bank conflict is about six times longer in DDR III SDRAM working at 800 MHz clock frequency than in DDR I SDRAM working at 133 MHz clock frequency, our SDRAM-aware NoC design achieves better improvement of memory utilization and latency in DDR SDRAM operating at a fast clock frequency.

The SDRAM-aware NoC design implemented by our SP algorithm is also applied into dual DTV model [24] containing dual MPEG decoders and dual memory subsystems. Consequently, the improvement of memory utilization and latency is similar to a single memory subsystem. However, it saves more than 42% gate count compared to dual DTV model implemented by the conventional NoC design since our SDRAM-aware NoC design does not need eight 32-flit request and data buffers and two complex memory schedulers in a dual memory subsystem.

B. Synthetic Benchmarks

We evaluate the improvement of memory utilization and memory latency obtained from several randomly generated applications on industrial intellectual properties (IP) with DDR II SDRAM working at 333 MHz clock frequency. The SDRAM-aware router adopts the SP algorithm and all of the conventional routers are replaced with our SDRAM-aware routers. The IPs are mapped into 3×3 to 6×6 mesh network by A3MAP [23] and generate 4–32 flits per packet at dynamic intervals. Table IV shows our SDRAM-aware NoC improves 11.8% memory utilization and 18% memory latency on average compared to the conventional NoC. In particular, the improvement of memory utilization and memory latency is higher in 6×6 NoC than in 3×3 NoC since packets passing through more SDRAM-aware routers have more opportunities to be scheduled well for SDRAM operations. Therefore, we can expect that the improvement of memory utilization and memory latency would be greater in larger or complex NoC.

C. Comparison of SP and SP + AP

We evaluate the improvement of memory utilization and latency of SP+AP algorithm that considers the short turn-around bank interleaving problem. For this experiment, we use a 4×4 mesh network including three SDRAM-aware routers and execute several randomly generated applications on industrial IPs. Table V shows the SP+AP algorithm achieves better memory utilization and memory latency than the SP algorithm, in particular, in high-performance DDR SDRAM. As shown in Table V, the short turn-around bank interleaving problem is not critical in low-performance DDR SDRAM since the improvement of memory utilization and latency achieved by the SP+AP algorithm is just around 1% compared to the SP algorithm. On the contrary, it causes memory performance critically degraded when high-performance DDR SDRAM is adopted in a NoC design. For example, in DDR III SDRAM working at 800 MHz clock frequency, the SP+AP algorithm achieves 9.2% higher memory utilization and 9.2% shorter memory latency than the SP algorithm. The proposed SP+AP algorithm requires an additional hardware such as four counters, one comparator, and some control logics to

check each bank state. However, the additional circuitry is tiny.

VII. Conclusion

This paper presented an SDRAM-aware NoC design where multiple NoC routers adopting our SDRAM-aware flow control algorithm allocate an SDRAM access packet to a channel for the efficient SDRAM operation. Our SDRAM-aware flow control algorithms solved three memory scheduling problems, such as bank conflict, data contention, and short turn-around bank interleaving to improve memory utilization and latency. The proposed SP algorithm solved the bank conflict problem and the data contention problem and the proposed SP+AP algorithm solved the short turn-around bank interleaving problem. Experimental results showed that our SDRAM-aware flow controller adopting the SP algorithm delivered superior memory utilization and latency with the small design cost compared to the conventional NoC design. In addition, our SP+AP algorithm achieved higher memory performance than the SP algorithm, in particular, in high-performance DDR SDRAM. Our SDRAM-aware router can achieve better performance improvement when it is employed in complex NoC or its routing scheme is adaptive. In conclusion, the proposed SDRAM-aware router provides more opportunities to support bandwidth-hungry NoC designs with the small hardware cost.

References

[1] W. Jang and D. Z. Pan, "An SDRAM-aware router for networks-on-chip," in *Proc. Design Automat. Conf.*, 2009, pp. 800–805.

[2] S. Borkar, "Thousand core chips: A technology perspective," in *Proc. Design Automat. Conf.*, 2007, pp. 746–749.

[3] L. Benini and G. D. Micheli, "Network on chips: A new SoC paradigm," *Computer*, vol. 35, no. 1, pp. 70–78, Jan. 2002.

[4] W. J. Dally and B. Towles, "Route packets, not wires: On-chip interconnection networks," in *Proc. Design Automat. Conf.*, 2001, pp. 684–689.

[5] P. Morrow, M. J. Kobrinsky, S. Ramanathan, C.-M. Park, M. Harmes, V. Ramachandrarao, H. M. Park, G. Kloster, S. List, and S. Kim, "Wafer-level 3-D interconnects via Cu bonding," in *Proc. Adv. Metalizat. Conf.*, 2004, p. 19.

[6] S. R. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, A. Singh, T. Jacob, S. Jain, V. Erraguntla, C. Roberts, Y. Hoskote, N. Borkar, and S. Borkar, "An 80-tile sub-100-w teraflops processor in 65-nm CMOS," *IEEE J. Solid-State Circuits*, vol. 43, no. 1, pp. 29–41, Jan. 2008.

[7] F. Li, C. Nicopoulos, T. Richardson, Y. Xie, V. Narayanan, and M. Kandemir, "Design and management of 3-D chip multiprocessors using network-in-memory," in *Proc. Int. Symp. Comput. Architecture*, 2006, pp. 130–141.

[8] DDR I, II, and III. *Device Operations and Timing Diagram*. Samsung Electronics [Online]. Available: http://www.samsung.com/global/business/semiconductor

[9] L. A. Vervoort, P. Yeung, and A. Reddy, *Addressing Memory Bandwidth Needs for Next-Generation Digital TVs with Cost-Effective, High-Performance Consumer DRAM Solutions*. Rambus Inc., Los Altos, CA, Jul. 2007.

[10] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens, "Memory access scheduling," in *Proc. Int. Symp. Comput. Architecture*, 2000, pp. 128–138.

[11] S. Heithecker and R. Ernst, "Traffic shaping for an FPGA-based SDRAM controller with complex QoS requirements," in *Proc. Design Automat. Conf.*, 2005, pp. 575–578.

[12] W. D. Weber, *Efficient Shared DRAM Subsystem for SoCs*. Sonics, Inc., Milpitas, CA, 2001.

[13] B. Akesson, K. Goossens, and M. Ringhofer, "PREDATOR: A predictable SDRAM memory controller," in *Proc. Int. Conf. Hardware/Software Codesign Syst. Synthesis*, 2007, pp. 251–256.

[14] I. Hur and C. Lin, "Memory scheduling for modern microprocessors," *ACM Trans. Comput. Syst.*, vol. 25, no. 10, pp. 10.1–10.36, Dec. 2007.

[15] F. Paganini, J. Doyle, and S. Low, "Scalable laws for stable network congestion control," in *Proc. Int. Conf. Decision Control*, 2001, pp. 185–190.

[16] D. Qiu and N. B. Shroff, "A predictive flow control scheme for efficient network utilization and QoS," *IEEE Trans. Netw.*, vol. 12, no. 1, pp. 161–172, Feb. 2004.

[17] U. Y. Ogras and R. Marculescu, "Prediction-based flow control for network-on-chip traffic," in *Proc. Design Automat. Conf.*, 2006, pp. 839–844.

[18] W.-C. Kwon, S.-M. Hong, S. Yoo, B. Min, K.-M. Choi, and S.-K. Eo, "An open-loop flow control scheme based on the accurate global information of on-chip communication," in *Proc. Design, Automat. Test Europe*, 2008, pp. 1244–1249.

[19] W. J. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. San Francisco, CA: Morgan Kaufmann, 2004.

[20] W.-C. Kwon, S. Yoo, S.-M. Hong, B. Min, K.-M. Choi, and S.-K. Eo, "A practical approach of memory access parallelization to exploit multiple off-chip DDR memories," in *Proc. Design Automat. Conf.*, 2008, pp. 447–452.

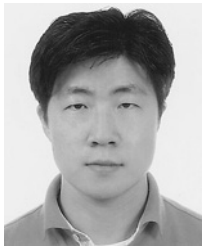
[21] *MemMax Scheduler*. Sonics, Inc. [Online]. Available: http://www.sonicsinc.com

[22] *Databahn DRAM Memory Controller IP*. Denali Software, Inc. [Online]. Available: http://www.denali.com

[23] W. Jang and D. Z. Pan, "A3MAP: Architecture-aware analytic mapping for networks-on-chip," in *Proc. Asian South Pacific Design Automat. Conf.*, 2010, pp. 523–528.

[24] "Samsung unveils HDTV system-on-chip." *EE Times*. (2004, Jul.) [Online]. Available: http://www.eetimes.com/electronics-news/4049612/Samsung-unveils-HDTV-system-on-chip

[25] N. Dutt, "Memory-aware NoC exploration and design," in *Proc. Design Automat. Test Eur.*, 2008, pp. 1128–1129.



Wooyoung Jang (S'08) received the B.E. degree in radio science and technology from Kyung Hee University, Suwon, South Korea, in 1998, and the M.S. degree in electrical and computer engineering from Yonsei University, Seoul, South Korea, in 2000. He is currently pursuing the Ph.D. degree in electrical and computer engineering from the University of Texas, Austin.

From 2000 to 2006, he was with the Digital Television Development Team, System Large Scale Integration Division, Samsung Electronics, Suwon, as a Senior Engineer. His current research interests include computer architecture and nanometer physical design for on-chip communication.

Mr. Jang was the recipient of the SK Telecom Scholarship for 1994–1997, the Samsung Outstanding Achievement Award in 2005, and the Samsung Scholarship for 2006–2010.

David Z. Pan (S'97–M'00–SM'06) received the Ph.D. degree in computer science from the University of California, Los Angeles (UCLA), in 2000.

From 2000 to 2003, he was a Research Staff Member with the IBM T. J. Watson Research Center, Yorktown Heights, NY. He is currently an Associate Professor and the Director of the Design Automation Laboratory, Department of Electrical and Computer Engineering, University of Texas, Austin. His current research interests include nanometer VLSI physical design, design for manufacturing, vertical integration of technology, design and architecture, and design/CAD for emerging technologies. He has published over 120 refereed papers in international conferences and journals, and is the holder of seven U.S. patents.

Dr. Pan has served as an Associate Editor for the IEEE TRANSACTIONS ON COMPUTER AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS-PART I, IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS-PART II, IEEE CAS Society Newsletter, and the *Journal of Computer Science and Technology*. He was a Guest Editor of the TCAD special section on "International Symposium on Physical Design" in 2007 and 2008. He is serving as the Chair of the IEEE Computer-Aided Network Design

Committee and the Association for Computing Machinery (ACM)/Special Interest Group on Design Automation Physical Design Technical Committee (SIGDA). He is a member of the Design Technology Working Group of the International Technology Roadmap for Semiconductor. He has served the technical program committees of major VLSI/CAD conferences, including ASPDAC (Topic Chair), DAC, DATE, ICCAD, ISPD (Program Chair), ISLPED (Exhibits Chair), ISCAS (CAD Track Chair), ISQED (Topic Chair), GLSVLSI (Publicity Chair), SLIP (Publication Chair), ACISC (Program Co-Chair), ICICDT (Award Chair), and VLSI-DAT (EDA Track Chair). He is the General Chair of the ISPD 2008 and the ACISC 2009. He is a member of the Technical Advisory Board of Pyxis Technology, Inc. He has received a number of awards for his research contributions and professional services, including the ACM/SIGDA Outstanding New Faculty Award in 2005, the NSF Career Award in 2007, the SRC Inventor Recognition Award thrice from 2000 to 2008, the IBM Faculty Award thrice from 2004 to 2006, the UCLA Engineering Distinguished Young Alumnus Award in 2009, the Best Paper Award from ASPDAC in 2010, the Best Interactive Presentation Award from DATE in 2010, the Best Student Paper Award from ICICDT in 2009, the IBM Research Bravo Award in 2003, the SRC Techcon Best Paper in Session Award in 1998 and 2007, the Dimitris Chorafas Foundation Research Award in 2000, the ISPD Routing Contest Award in 2007, the eASIC Placement Contest Grand Prize in 2009, five Best Paper Award Nominations from ASPDAC, DAC, ICCAD, and ISPD, and the ACM Recognition of Service Award in 2007 and 2008. He was a Distinguished Lecturer of the IEEE CAS Society from 2008 to 2009.