Efficient Parallel Buffer Structure and Its Management Scheme for a Robust Network-on-Chip (NoC) Architecture

Jun Ho Bahn and Nader Bagherzadeh

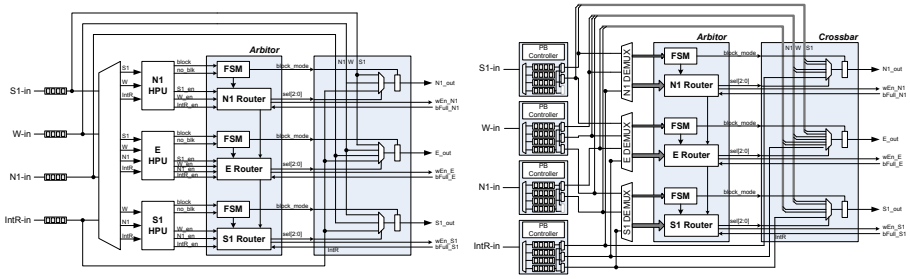
Department of Electrical Engineering and Computer Science,
University of California Irvine - Irvine, CA 92697–2625
{jbahn,nader}@uci.edu

Abstract. In this paper, we present an enhanced Network-on-Chip (NoC) architecture with efficient parallel buffer structure and its management scheme. In order to enhance the performance of the baseline router to achieve maximum throughput, a new parallel buffer architecture and its management scheme are introduced. By adopting an adjustable architecture that integrates a parallel buffer with each incoming port, the design complexity and its utilization can be optimized. By utilizing simulation-based performance evaluation and comparison with previous NoC architectures, its efficiency and superiority are proven. Major contributions of this paper are the design of the enhanced structure of a parallel buffer which is independent of routing algorithms, and its efficient management scheme for the Network-on-Chip (NoC) architecture adopting a minimal adaptive routing algorithm. As a result, the total amount of required buffers can be reduced for obtaining the maximum performance. Additionally a simple and efficient architecture of overall NoC implementation is provided by balancing the workload between parallel buffers and router logics.

Key words: Network-on-Chip, On-chip network, virtual channel, parallel buffer, router

1 Introduction

In designing Network-on-Chip (NoC) systems, there are several issues to be considered, such as topology, routing algorithm, performance, latency, and complexity. All these factors are taken into account when the design of an NoC architecture is considered. Regarding routing algorithms, many researchers have developed better performance routing algorithm using oblivious/deterministic or adaptive routing algorithms [1],[2],[3],[4],[5],[6]. In addition, the adoption of virtual channel (abbreviated to VC) has been prevailing because of its versatility. By adding virtual channels and proper utilization of their channels, deadlock-freedom can be easily accomplished. Network throughput can be increased by dividing the buffer storage associated with each network channel into several virtual channels [4]. By proper control of virtual channels, network flow control



(a) base router (b) PB enhanced router

Fig. 1. Micro-architecture of base and PB enhanced router

can be easily implemented [7]. Also to increase the fault tolerance in a network, the concept of virtual channel has been utilized [8],[9]. However, in order to maximize its utilization, allocation of virtual channels is a critical issue in designing routing algorithms [10],[11].

We proposed a base NoC architecture adopting a minimal adaptive routing algortihm with near-optimal performance and feasible design complexity [6]. Based on this NoC architecture, a new routing-independent parallel buffer structure and its management scheme are proposed instead of VC. As a result, the channel utilization and maximum throughput in performance are improved.

The orginazation of this paper is as following. In the next section, a brief introduction of base NoC architecture adopting a minimal adaptive routing algorithm will be provided. While explaining the proposed parallel buffer (PB) structure and its management scheme, the enhanced NoC architecture including these parallel buffers will be introduced. In order to prove its benefit, several simulation-based evaluation results and comparison with the base NoC architecture will be provided. Finally some conclusions will be drawn.

2 Base Network-on-Chip (NoC) Architecture

We proposed an adaptive routing algorithm and the baseline architecture for a flexible on-chip interconnection [6]. It adopts a wormhole switching technique and its routing algorithm is livelock-/deadloc- free in 2-D mesh. Specifically to eleminate the deadlock situation and simplify the routing algorithm, two disjoint vertical channels are provided instead of using virtual channels. The use of a vertical channel is limited by the direction of delivered data. That is, each vertical channel is denoted by *N1/S1* for east-bounded and *N2/S2* for west-bounded packets, respectively. Also, the data from the internal processing element (PE) or execution unit (EU) connected with router uses separate injection ports, *IntL-in* and *IntR-in*, depending on its direction of target node. As a result, available routing ports are grouped as {*W-in, N1, E-out, S1, IntR-in*} and {*E-in, N2, W-out, S2, IntL-in*} where *N1/N2* or *N2/S2* represent incoming/outgoing ports

Algorithm 1 Pseudo routing algorithm for base router

```
1: local variable: MappediPortFlag[ ] contains whether iPort[ ] is routed or not
2: initialize MappediPortFlag[ ] to 0
3: for oIndex ← N1-out to Int-out do
4:   if oPort[oIndex].Status is active then
5:     traverse data from iPort[oPort[oIndex].RoutediPort] to oPort[oIndex]
6:     decrease oPort[oIndex].RestFlits by 1
7:     MappediPortFlag[oPort[oIndex].RoutediPort] set to 1
8:     if oPort[oIndex].RestFlits is zero then
9:       oPort[oIndex].Status set to inactive
10:    end if
11:  else
12:    for iIndex ← value from left to right at the row of the corresponding outgoing port in
       Table 1 do
13:      if MappediPortFlag[iIndex] is 0 then
14:        if iPort[iIndex].HeaderReady is true then
15:          traverse data from iPort[iIndex] to oPort[oIndex]
16:          extract the length of flits and set oPort[oIndex].RestFlits
17:          oPort[oIndex].RoutediPort ← iIndex
18:          MappediPortFlag[iIndex] ← 1
19:        end if
20:      end if
21:    end for
22:  end if
23: end for
```

simultaneously(*-in* an incoming port, and *-out* an outgoing port for the given channel, respectively).

For each set of ports, the routing decision is independently performed. For instance, in the set of east-bounded ports, i.e. {*W-in, N1, E-out, S1, IntR-in*}, incoming ports are routed to each output port depending on port priority shown in Table 1. There are two different levels of priority on incoming ports and outgoing ports, respectively. The priority on outgoing ports is given in the order of *N1-out*, *E-out*, *S1-out*, and *Int-out*. Thus, it is organized by starting from north in clockwise direction to the port to EU, *Int-out*, having the lowest priority. The other priority on incoming ports is differently assigned depending on the given outgoing port. For the given outgoing port, the priority on incoming ports is given in the order of clockwise direction starting from the incoming port next to the given outgoing port if the incoming port has a deliverable flit to the given outgoing port. If any incoming port is routed to the outgoing port with higher priority, that port is not considered in routing decision for the outgoing ports with lower priority. Algorithm 1 summarizes a detailed procedure of the routing decision. Based on these operations, the micro-architecture of either *Right* or *Left* router is designed as Figure 1(a).

3 Enhanced Network-on-Chip (NoC) Architecture with Parallel Buffers

In order to enhance the performance of base NoC architecture, an approach similar to parallel buffer technique of virtual channels is selected as shown in Figure 1(b). Instead of using dedicated buffers for each port, parallel buffers with

Table 1. Priority assignment on incoming/outgoing ports

outgoing ports	incoming ports
N1-out	*S1-in, W-in, IntR-in*
E-out	*S1-in, W-in, N1-in, IntR-in*
S1-out	*W-in, N1-in, IntR-in*
N2-out	*E-in, S2-in, IntL-in*
S2-out	*N2-in, E-in, IntL-in*
W-out	*N2-in, E-in, S2-in, IntL-in*
Int-out	*N1-in, N2-in, E-in, S1-in, S2-in, W-in*

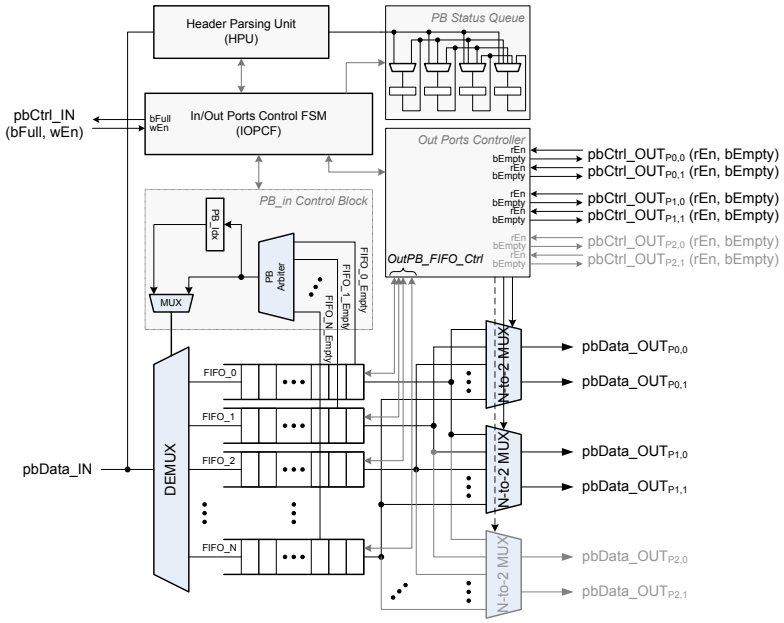


Fig. 2. Proposed parallel buffer structure

small depth queue or FIFO are added in front of each incoming port. The difference from previous approaches with virtual channels is a routing-independent parallel buffer structure, and its efficient management scheme which will be described in detail.

Figure 2 shows a detailed parallel buffer structure applied in the enhanced NoC architecture. To maximize the utilization of channels, multiple outputs from a parallel buffer for each forwarded direction are provided to the routers. By virtue of 2-D mesh topology, the maximum number of forwarded directions is 3. For each forwarded direction, maximum 2 outputs from a parallel buffer are provided. The following example explains how to extract the maximum number of outputs from parallel buffer for each output port.

Let's assume that a parallel buffer with 8 FIFOs at each incoming port is allocated and all FIFOs only in the parallel buffer at the incoming port *W-in*

Algorithm 2 Pseudo routing algorithm for enhanced PB router

```
1: local variable: MappediPortFlag[ ][ ] contains whether iPort[ ][ ] is routed or not
2: initialize MappediPortFlag[ ] to 0
3: for oIndex ← N1-out to Int-out do
4:   if oPort[oIndex].Status is active then
5:     traverse data from iPort[oPort[oIndex].RoutediPort][oPort[oIndex].RoutediPortPB] to
       oPort[oIndex]
6:     decrease oPort[oIndex].RestFlits by 1
7:     MappediPortFlag[oPort[oIndex].RoutediPort][oPort[oIndex].RoutediPortPB] set to 1
8:     if oPort[oIndex].RestFlits is zero then
9:       oPort[oIndex].Status set to inactive
10:    end if
11:  else
12:    for iIndex ← value from left to right at the row of the corresponding outgoing port in
       Table 1 do
13:      for iPBIndex ← value from the order of PB queue do
14:        if MappediPortFlag[iIndex][iPBIndex] is 0 then
15:          if iPort[iIndex][iPBIndex].HeaderReady is true then
16:            traverse data from iPort[iIndex][iPBIndex] to oPort[oIndex]
17:            extract the length of flits and set oPort[oIndex].RestFlits
18:            oPort[oIndex].RoutediPort ← iIndex
19:            oPort[oIndex].RoutediPortPB ← iPBIndex
20:            MappediPortFlag[iIndex][iPBIndex] ← 1
21:          end if
22:        end if
23:      end for
24:    end for
25:  end if
26: end for
```

contain packets. Also the packets occupying FIFOs in the parallel buffer at *W-in* port, arrived at different times. The destination of each packets occupying each FIFO in the parallel buffer is *E*, *NE*, *N*, *SE*, *SE*, *S*, *NE*, and *S* in the order of $\{PB_0, PB_1, PB_2, PB_3, PB_4, PB_5, PB_6, PB_7\}$ where PB_i represents the i-th FIFO in the given parallel buffer. Also the order of arrival time for each packet is $\{PB_1, PB_3, PB_0, PB_2, PB_4, PB_6, PB_5, PB_7\}$. If each FIFO in the parallel buffer is grouped in the order of arrival time and its available routing direction, the resultant groups of FIFOs are $\{PB_1, PB_2, PB_6\}$ for *N-out*, $\{PB_1, PB_3, PB_0, PB_4, PB_6\}$ for *E-out*, and $\{PB_3, PB_4, PB_5, PB_7\}$ for *S-out*. Because no other incoming ports than *W-in* have deliverable data, the routing decision is performed only on the port *W-in*, According to the described priority in Table 1, the outgoing port *N-out* is the first one to be considered. For *N-out* outgoing port, the packet stored in PB_1 will be selected. For *E-out* outgoing port, the packet stored in PB_3 will be chosen because PB_1 is already occupied by the outgoing port *N-out* with higher priority. Finally to *S-out* outgoing port, the packet stored in PB_4 will be forwarded because the earlier packet in PB_3 is already served for *E-out*. Therefore, instead of searching all the entries in each group, the first 2 entries are sufficient for checking the available incoming packet for the routing decision. Algorithm 2 summarizes a detail routing procedure for the enhanced PB router.

Different from the conventional VC approaches, the operation of the proposed parallel buffer is no longer dependent on neighboring routers. By autonomous management of a parallel buffer depending on outgoing port read requests, the

parallel buffer can be simply assumed as single FIFO. That is, from the previous neighboring router, the parallel buffer is recognized as a single FIFO with ordinary interfaces such as `bFull` (buffer fullness) or `wEn` (write enable). Therefore, the only task of this parallel buffer is to store the incoming packets and manage their occupancy with respect to their destination and read operations depending on routing decision. In order to manage the empty FIFOs in the given parallel buffer, as illustrated in Figure 2, simple logic circuits are added in *PB_in Control Block*. With given empty signals from all input FIFOs, one of empty FIFO indices is chosen which controls the path of storing incoming *flit* into the corresponding FIFO. Simultaneously to control the incoming packet in *flit*, the header parsing unit (HPU) and associated control unit (IOPCF) are needed.

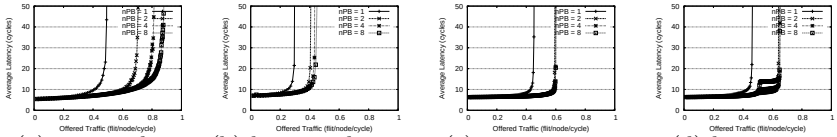
In the proposed parallel buffer structure, every two outputs among the allocated FIFOs in the parallel buffer are chosen and forwarded to the inputs of routing decision logic for the corresponding outgoing port. The parallel buffer controller manages the history of arrival packets and their residence in FIFOs at the parallel buffer, and groups of in-use FIFOs based on its outgoing direction. For this purpose, in the parallel buffer controller, header parsing unit (HPU) for incoming packets is required. By moving the location of header parsing unit which is originally placed in the router logic as Figure 1(a), the critical path in the enhanced router can be reduced. Because the performance of FIFOs is much faster than the one for the router logic with HPU [6], it results in balancing the workload of each blocks with respect to the timing. Therefore, for the enhanced NoC architecture, the better timing performance in real implementation can be expected.

4 Evaluation of the Performance in the Enhanced NoC Architecture

4.1 Evaluation Environment

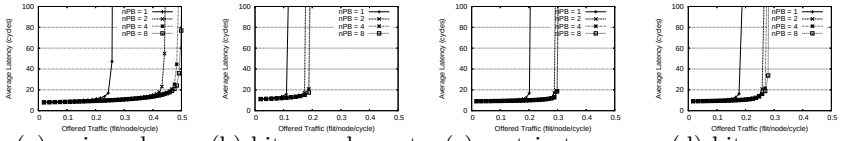
In order to evaluate the performance of the base NoC architecture, a time-accurate simulation model was implemented in SystemC. By comparing with different routing algorithms, its competitive performance has been evaluated. In this paper, by adding parallel buffers with efficient management scheme, overall performance increment is expected. Therefore, the parallel buffer with proposed management scheme is modeled similarly in SystemC. And the previous FIFO module is swapped with this parallel buffer module.

All the network simulations were performend using 100,000 cycles with 4 commonly used traffic patterns such as uniform random, bit-complement, matrix-transpose traffic, and bit-reverse traffic. Two different sizes of 2-D mesh topologies based on 4×4 and 8×8 were studied. Also the number of FIFOs in the parallel buffer per incoming port is varied. However, the depth of FIFO in the parallel buffer is fixed as 4 and 4-flit long packets are used. For the measurement of throughput and adjusting incoming traffic, the standard interconnection network measurement technique [1] was adopted.



(a) uni-random (b) bit-complement (c) matrix-trans (d) bit-reverse

Fig. 3. Performance in 4×4 mesh



(a) uni-random (b) bit-complement (c) matrix-trans (d) bit-reverse

Fig. 4. Performance in 8×8 mesh

4.2 Simulation results and their analysis

Throughout the SystemC simulations with various traffic patterns and two different network topologies, many experimental results of PB enhanced NoC architecture are well collected. With these collected data, the plots of average latency vs. offered traffic are drawn in Figures 3 and 4 for 4×4 and 8×8 mesh, respectively. In 4×4 mesh network, both uniform random and bit-reverse traffic patterns show the notable increase of maximum throughput, approximately 25% and 19%, respectively. In 8×8 mesh network, uniform random, matrix-transpose, and bit-reverse traffic patterns show the noticeable improvement, around 28%, 28%, and 18%, respectively. However, for bit-complement traffic pattern in 4×4 and 8×8, the improvement of performance seems to be minor. The reason of minor improvement for bit-complement traffic pattern is because it has relatively lower flexibility in choosing routing paths from source to destination and most of the traffic patterns concentrate on the central region for a given mesh, resulting in severe routing contention and blocking similar with the analysis in [11]. Also as the size of 2-D mesh topology increases, the effect of parallel buffer in improving the performance is growing because the increased size provides much higher degree of flexibility in routing paths.

As shown in [6], the previous base NoC architecture reaches up to 0.4 offered traffic at uniform random traffic pattern in 8x8 mesh network even when infinite buffers are allocated between links. However, in the new parallel buffer adopted NoC architecture, the performance already outperforms even when two-FIFO parallel buffer per incoming port are applied as shown in Figure 4(a). Furthermore, by applying four-FIFO parallel buffer per incoming port, the maximum throughput in 8x8 mesh reaches up to 0.45 (about 13% improvement). With comparison to the base NoC architecture, four-FIFO parallel buffer per incoming port achieves an optimal performance benefit. Also comparing with general virtual channel application [3] where at least 8 virtual channels per physical

channel are required to get the nominal performance and resolve deadlock problem, the proposed NoC architecture with parallel buffers has its own benefit.

5 Conclusion

We proposed a new parallel buffer structure and its management scheme, as well as its optimal micro-architecture. By applying this proposed parallel buffer to the previous base NoC architecture, noticeable performance improvement was observed using simulation of various traffic patterns. Even though the deadlock-freedom is realized by providing disjoint vertical channels instead of using virtual channels which is a general approach for this purpose, notable performance benefit can be extracted by adding parallel buffers with smaller number of FIFOs. Also by moving the header parseing unit into the parallel buffer controller, the timing balance between parallel buffer and router logic can be obtained at micro-architecture level.

References

1. Dally, W.J., Towles, B.: Principles and Practices of Interconnection Networks, Morgan Kaufmann, San Francisco (2004)
2. Sullivan, H., Bashkow, T.R., Klappholz, D.: A Large Scale, Homogeneous, Fully Distributed Parallel Machine. In: ISCA '77, pp. 105–117, ACM Press, New York (1977)
3. Seo, D., Ali, A., Lim, W., Rafique, N. Thottethodi, M.: Near-Optimal Worst-Case Throughput Routing for Two-Dimensional Mesh Networks. In: ISCA '05, pp. 432–443, ACM Press, New York (2005)
4. Dally, W.J., Seitz, C.L.: Deadlock-Free Message Routing in Multiprocessor Interconnection Networks. IEEE Trans. Computer, vol. C-36, no. 5, pp. 547–553, IEEE Computer Society, Washington (1987)
5. Glass, C.J., Ni, L.M.: The Turn Model for Adaptive Routing. J. ACM, vol. 31, no. 5, pp. 874–902, ACM Press, New York (1994)
6. Bahn, J.H, Lee, S.E., Bagherzadeh, N.: On Design and Analysis of a Feasible Network-on-Chip (NoC) Architecture. In: ITNG '07, pp. 1033–1038, IEEE Computer Society, Washington (2007)
7. Dally, W.J.: Virtual-Channel Flow Control. IEEE Trans. Parallel and Distributed Systems, vol. 3, no. 2, pp. 194–205, IEEE Press, Piscataway (1992)
8. Boppana, R.V., Chalasani, S.: Fault-Tolerant Wormhole Routing Algorithms for Mesh Networks. IEEE Trans. Computers, vol. 44, no. 7, pp. 846–864, IEEE Computer Society, Washington (1995)
9. Zhou, J., Lau, F.C.M.: Adaptive Fault-Tolerant Wormhole Routing with Two Virtual Channels in 2D Meshes. In: ISPAN '04, pp. 142–148, IEEE Computer Society, Los Alamitos (2004)
10. Vaidya, A.S., Sivasubramaniam, A., Das, C.R.: Impact of Virtual Channels and Adaptive Routing on Application Performance. IEEE Trans. Parallel Distributed Systems, vol. 12, no. 2, pp. 223–237, IEEE Press, Piscataway (2001)
11. Rezazad, M., Sarbazi-azad, H.: The Effect of Virtual Channel Organization on the Performance of Interconnection Networks. In: IPDPS '05, pp. 264.1, IEEE Computer Society, Washington (2005)