LETTER

A Low-Cost Standard Mode MPI Hardware Unit for Embedded MPSoC

Won-young CHUNG[†a)], ***Student Member***, **Ha-young JEONG**[†], **Won Woo RO**[†b)], ***and*** **Yong-surk LEE**[†c)], ***Nonmembers***

SUMMARY In this paper, we propose a novel low-cost Message Passing Interface (MPI) unit between processor nodes, which supports message passing in multiprocessor systems using distributed memory architecture. Our MPI unit operates in the standard mode – using the buffered mode for small amounts of data transaction and the synchronous mode for large amounts of data transaction. This results in increased performance by reducing the control message transmission time for the small amount of data. We verified the performance with a simulator designed based on SystemC. Additionally, we designed the MPI unit using VerilogHDL, and we synthesized it with a synopsys design compiler. The proposed standard mode MPI unit shows a high performance even though the size of the MPI unit occupies less than 1 % of the whole chip. Thus, with respect to low-cost design and scalability, this MPI hardware unit is useful to increase overall performance of the embedded Multiprocessor System on a Chip (MPSoC).

key words: *MPI, distributed memory, MPSoC, embedded systems, low-cost*

1. Introduction

With the recent increase in the use of various applications in embedded systems, it is difficult to guarantee the timing constraint for QoS (Quality of Service). For this reason, there is great demand for high-performance processing. However, there is a limit to the pace of development of an application to increase single processor frequency. Thus, there have been more studies on the MPSoC (Multiprocessor System on a Chip) recently, which can reduce the processing time of the whole system. With this trend, many commercial MPSoC emerged such as ARM's Cortext-A9 and TI's OMAP. However, these MPSoC use additional hardware support for memory coherency because their design relies on shared memory architecture. Consequently, there is a limitation on scalability. By using distributed memory architecture on MPSoC, however, we are able to reduce the scalability problem on conventional shared memory MPSoC [1].

The most recent research on distributed memory architecture systems focuses on the use of a hardware MPI Message Passing Interface (MPI) unit to improve communication performance [2]–[4]. In MPI, a message may be sent in one of four communication modes – ready mode, buffered mode, synchronous mode and standard mode – which approximately corresponds to the most common protocols used for the point-to-point communication. In our previous study, we proposed the synchronous mode MPI hardware unit for a distributed memory architecture system, which can manage synchronization through the unit itself, thereby reducing the burden of implementing a software MPI library [5]. However, when the synchronous mode transmits small amounts of data, synchronization before the data transmission accounts for most of the execution delay. The synchronous mode is especially inefficient because the size of transmit data is generally small in embedded systems.

Therefore, we proposed a standard mode MPI hardware unit, supporting not only synchronous mode but also buffered mode. When the size of data is smaller than the threshold value, our MPI hardware unit can improve the whole system performance due to the reduced synchronous time.

2. Conventional Hardware MPI Unit Architecture

In our previous MPI unit, there were two kinds of message: control message for synchronization, and data message. Figure 1 shows processor node 1 sends data messages to processor node 0 in the previous architecture. After synchronization through the control message in steps 1-7, the sender node sends the data message to the receiver node. In the previous work, there is safe communication due to saving and managing the message in the queue. However, if there is a small amount of data, the synchronization takes comparatively longer than the data transmission time, so the synchronization is overloaded in embedded MPSoC with a lot of small data transmissions. Therefore, this paper proposes the standard mode MPI unit adding a buffered mode which processes small data transmission effectively to the synchronous mode MPI unit from the previous work.

Figure 2 is a graph showing control message transmission time (synchronization time) and data message transmission time, according to the size of the data. When the size of the data is 4 bytes, it takes the same amount of time to send data messages and to send control messages. If this is transmitted through the buffered mode, the synchronization time is close to 0 in the ideal case, and then the performance improves approximately twofold. Moreover, if there is no matching send instruction in the ready queue of the sender node, the process iterates that the sender node

Manuscript received November 9, 2010.
Manuscript revised March 15, 2011.

[†]The authors are with the Department of Electrical and Electronic Engineering, Yonsei University, Seoul, Korea.
a) E-mail: wychung@mpu.yonsei.ac.kr
b) E-mail: wro@yonsei.ac.kr
c) E-mail: yonglee@yonsei.ac.kr

DOI: 10.1587/transinf.E94.D.1497

transmits a busy control message to the receiver node, and the receiver node transmits a request control message to the sender node [5]. If there is an increased exchange of the control message before the data message transmission, there is a longer synchronization time. In this case, when we use the buffered mode, the performance is significantly improved. Thus, this study suggests the standard mode hardware MPI unit with a small data transmission applies to the buffered mode, and a relatively large data transmission applies to the synchronous mode.

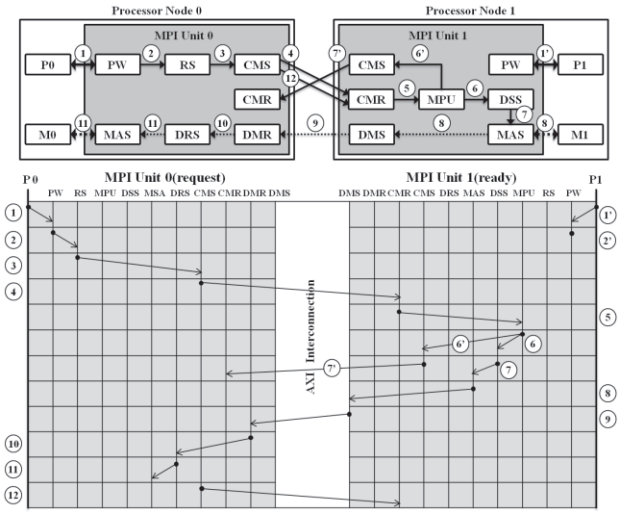

Fig. 1 Data transfer and synchronization mechanism (Sync. mode).

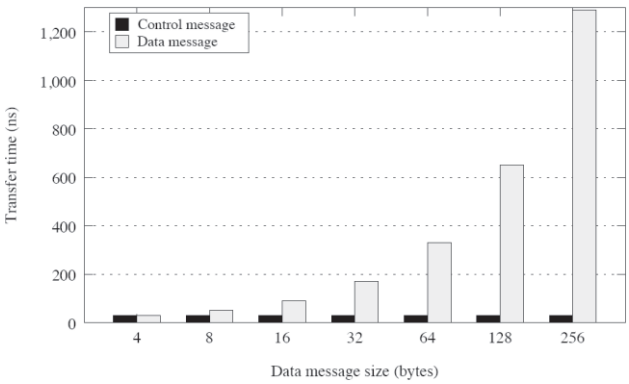

Fig. 2 Comparing transfer time of data message and control message.

3. Proposed Hardware MPI Unit Architecture

Figure 3 shows the entire system of the proposed standard mode hardware MPI unit. We added SReqQ and SRdyQ, which are the local message queues that manage the small size data transmission. The function to schedule the memory access between small data and large data is done by adopting Round-robin scheduling. We also added SMS and SMR, which are exclusive ports for small size data transmission. These SMS and SMR prevent bottlenecks between large data and small data transmission.

Figure 4 shows the procedure in the buffered mode. The P0 sends the receive instruction (request control message) to MPI unit 0, and the P1 sends the send instruction (ready control message) to the MPI unit 1 (①, ①'). At the PW of the MPI unit 0, if the data size is smaller than the threshold value, the receive instruction is stored in the SReqQ. At the PW of the MPI unit 1, the send instruction is stored in the SRdyQ, and DSS selects the transmit data from the SrdyQ (②, ③). The transmit data is read from memory through the MAS (④). SMS sends 'm_data_on' (the control message) with the data message, and the SMR of the receiver node stores the control message in the relevant channel buffer (⑤).

The SMR compares the control message with the match bits in the SReqQ of the MPI unit 0, and if there is entry which matches each other, then the data message is stored in the memory through both the DRS and the MAS (⑥, ⑦). If there is no entry which matches each other, then the data message is stored in the channel buffer of the SMR

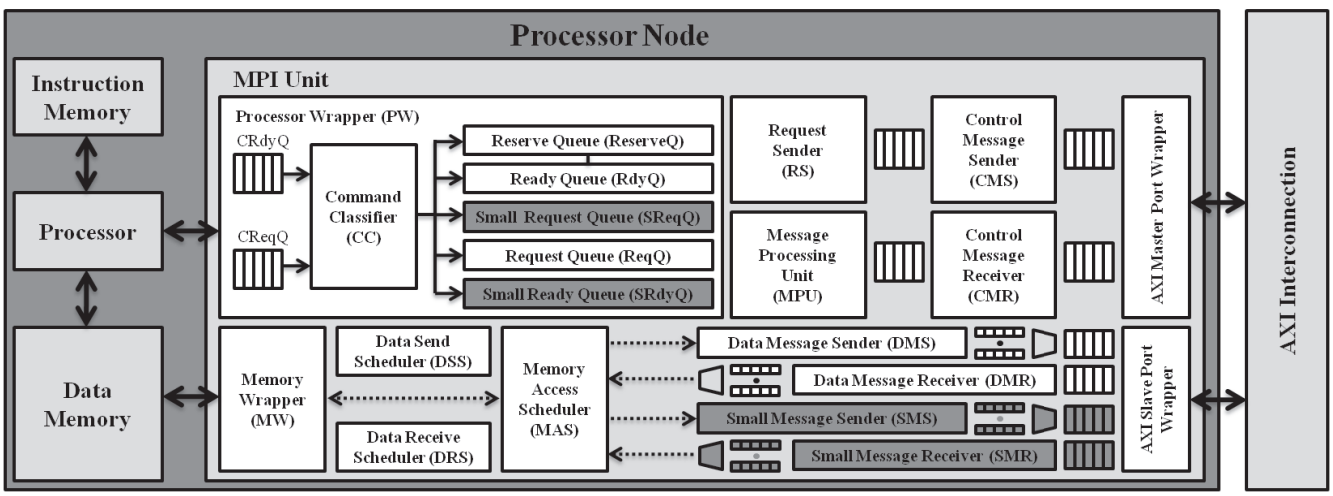

Fig. 3 Proposed standard mode hardware MPI unit architecture.

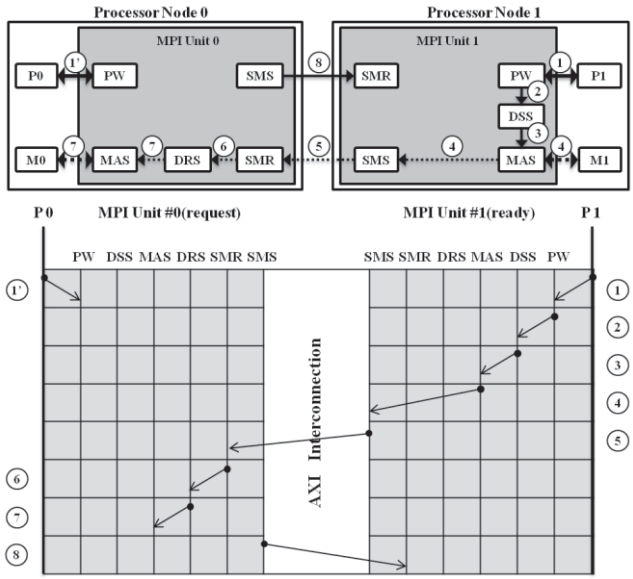

Fig. 4 Data transfer and synchronization mechanism (Buffered mode).

as well as the control message. The size of the channel buffer depends on the threshold value. The SMR channel buffer does not need be large in size because only small data messages are stored there. When the data message transmission has completed, 'm_complete' (the control message) is sent to MPI unit 1, and then communication between two nodes are complete (⑧).

In conclusion, when we compare the standard mode MPI unit with the synchronous mode MPI unit, we can see that total area of the proposed standard mode MPI unit is increased by adding hardware logic. However, if small amounts of data transfer frequently occur according to application, this can highly improve performance.

4. Simulation Results

To measure the communication performance of the proposed standard mode MPI unit, we compare it with the previous synchronous mode MPI unit [5]. Each unit is designed as a Bus Functional Model (BFM) based on SystemC [6]. We designed the BFM considering the delay time of each of the blocks, and it generates communication traffics in a specific simulation environments.

4.1 Performance Evaluation of Transmission Time

We measure the control message transmission time (synchronization time) and data message transmission time according to the size of the data in the synchronous mode model to determine the threshold value, which decides if it is the synchronous mode or the buffered mode. Table 1 shows the speed-up ratio when the synchronous mode is changed to the buffered mode. Ideally, the buffered mode has no synchronization time before transmitting the data message, because the sender node directly transmits the data message to the receiver node. As the buffered mode directly transmits the data message from the sender node to the receiver node, in an ideal case, there is no control message transmit time before transmitting the data message. Therefore, we assumed that the control message transmission time is 0 in the buffered mode.

Table 1 Speed-up ratio of changed from sync mode to buffered mode.

BYTE	4	8	16	32	64	128	256	512	1024
Speed-up ratio	2.00	1.60	1.33	1.18	1.09	1.05	1.02	1.01	1.00

4.2 Performance Evaluation of Proposed Model

In this section, we compared the proposed standard mode MPI unit with the synchronous mode MPI unit from the previous study, in terms of the results of the simulation in one-to-one, one-to-many, and many-to-many communications. Figures 5, 6, and 7 show the speed-up ratio of the proposed MPI unit when the average of transmission time for each size of data is 1 in the synchronous mode MPI unit. The simulation test vector is transmitted according to its size, from small data to large data, and it iterates 1000 times. Also, we simulate the proposed MPI unit using the higher threshold value with the basis of 32 bytes, whose approximate speed increase rate is more than 20%.

Figure 5 shows the speed-up ratio of 1-versus-1 nodes communication. When the threshold is 8 bytes, there are 2.57 times performance improvement with the 4 bytes data, and 2.4 times improvement in performance with the 8 bytes data. Still, the speed decreases to 0.9 times when data size is 16 bytes. Though the transmit bus between each MPI unit has two transmit routes with the basis of the threshold value, the improvement still decreases in the small interval with excessing threshold because the MPI unit in the processor node access the memory with one bus, which causes conflict between the data from small data transmit route and large data transmit route. Also, in the case of a threshold of 8 bytes, if the size of the message becomes larger than 16 bytes, the performance shows approximately 1. This is because the data transmission time takes far longer than the synchronization time. If the threshold increases, though the width of speed-up ratio gets smaller, the speed-up rate increases in a wide range of data.

Figures 6 and 7 show the simulation results of the communication through 1-to-3 nodes and 3-to-1 nodes, respectively. In both cases, the overall transmission delay time is longer than in the 1-to-1 node communication, so the speed-up ratio has increased by just a small amount. This is because the message increased, the fixed buffer capacity exceeded (which generated full), and the control message retransmitted. Here is a summary of the simulation results of communication through one-to-one, one-to-many, and many-to-one:

1. If the threshold is set low, then we can expect a large

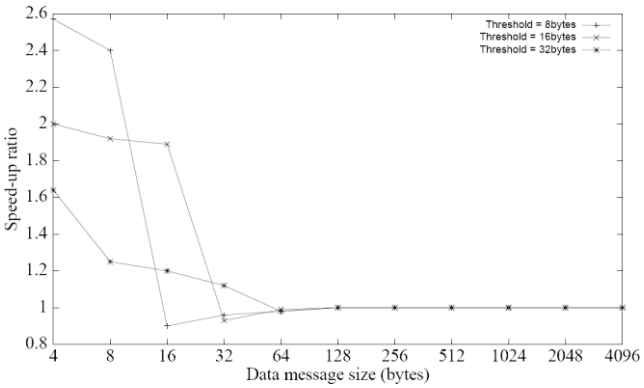

Fig. 5 Speed-up ratio of 1 versus 1 communication.

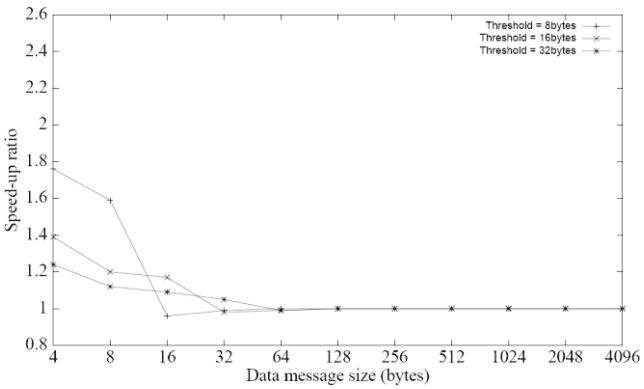

Fig. 6 Speed-up ratio of 1 versus 3 communication.

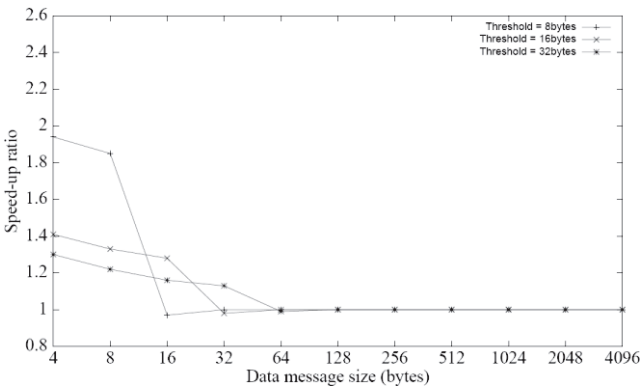

Fig. 7 Speed-up ratio of 3 versus 1 communication.

improvement of speed-up in a small range. But if threshold is set high, then we can expect a small improvement of speed-up even in wide range.

2. According to the scheduler for memory access between the MPI unit and memory, there can be a difference in transmission time for both small data (lower than the threshold) and large data (higher than the threshold). If we set a high priority for small data transmission (lower than the threshold), then we can expect more speed-up in the buffered mode.
3. If the speed-up ratio is increased in the buffered mode, then the speed-down ratio increases by a small amount in the synchronous mode. This is because the message transmission time is much longer than the synchronization time, if the size of the data is large.

Table 2 Synthesized results.

HARDWARE RESOURCE	RELATIVE AREA
Total Chip	100.00
Memory	68.80
RISC core	10.94
Standard mode MPI unit	1.02
etc.	19.24

4.3 Implementation and Verification

After verifying the performance through SystemC simulator, we designed the MPI unit using VerilogHDL. We synthesized a synopsys design compiler, and we used MagnaChip 0.18 μm for the synthesis library. To verify the proposed MPI unit, we designed the multiprocessor based on RISC MIPS DLX architecture. There are 2 processor nodes and a AXI bus on a chip. Each processor nodes is consisted of one RISC core, one instruction memory, one data memory and one MPI unit. The processor nodes communicate through an AXI interconnection [7]. While Table 2 shows that the memory occupies most of the areas, the MPI unit as well as the processor core occupy fewer areas. The Number of equivalent NAND gates of the proposed standard mode MPI unit is 24724.82, and it occupies less than 1.02 % of the whole chip – this is a negligible size based on the entire chip size. It is approximately 13.2 % larger than synchronous mode MPI unit. Thus, with a negligible area increase effort, the total system performance can increase by the proposed standard mode MPI unit.

5. Conclusion

To reduce bottlenecks from communication overhead in multiprocessor systems, we propose the MPI unit which optimizes MPI performance in distributed memory architecture. In particular, we increase performance by adding hardware logic that supports MPI buffered mode for a small-sized data message, which is frequently used in embedded MPSoC. The proposed MPI hardware unit combines 5 queues that save and manage the synchronized messages, and it performs the multiple outstanding issue and out of order completion. Additionally, the proposed standard mode hardware MPI unit can be used for adjusting the threshold value according to its application, so it is useful for the scalability aspect. Furthermore, if the processor supports multithread, the processor issues an order to the MPI unit, and then the processor can process another thread, thereby increasing overall system performance. If the proposed MPI unit is used, the message delay time from the small hardware overhead is minimized, and the bandwidth of transaction is

maximized. It is therefore very efficient in multiprocessor systems that use distributed memory architecture.

References

[1] L. Benini and G.de Micheli, "Networks on chip: A new SoC paradigm," Computer, vol.35, no.1, pp.70–78, Jan. 2002.
[2] F. Poletti, A. Poggiali, D. Bertozzi, L. Benini, P. Marchal, M. Loghi, and M. Poncino, "Energy-efficient multiprocessor systems-on-chip for embedded computing: Exploring programming models and their architectural support," IEEE Trans. Comput., vol.56, no.5, pp.606–621, May 2007.
[3] F. Dumitrascu, I. Bacivarov, L. Pieralisi, M. Bonaciu, and A. Jerraya, "Flexible MPSoC platform with fast interconnect exploration for optimal system performance for a specific application," Design, automation and test in Europe: Designers' Forum, pp.166–171, 2006.
[4] S. Han, A. Baghdadi, M. Bonaciu, S. Chae, and A.A. Jerraya, "An efficient scalable and flexible data transfer architecture for multiprocessor SoC with massive distributed memory," Proc. 41st annual Design Automation Conference, pp.250–255, San Diego, CA, USA, June 2004.
[5] H.-Y. Jeong, W. Hur, and Y.-S. Lee, "Scalable distributed memory embedded system with a low-cost hardware message passing interface," IEICE Electronics Express, vol.6, no.12, pp.837–843, 2009.
[6] S. Mahadevan, F. Angiolini, M. Storgaard, R.G. Olsen, J. Sparso, and J. Madsen, "A network traffic generator model for fast network-on-chip simulation," Proc. Conf. Design, Automation and Test in Europe, vol.2, pp.780–785, Munich, Germany, 2005.
[7] AMBA AXI Specification, ARM Limited 2003.