JOURNAL OF SEMICONDUCTOR TECHNOLOGY AND SCIENCE, VOL.17, NO.4, AUGUST, 2017
https://doi.org/10.5573/JSTS.2017.17.4.505
ISSN(Print) 1598-1657
ISSN(Online) 2233-4866

# Converting Interfaces on Application-specific Network-on-chip

**Kyuseung Han[1], Jae-Jin Lee[1], and Woojoo Lee[2]**

***Abstract*—As mobile systems are performing various functionality in the IoT (Internet of Things) era, network-on-chip (NoC) plays a pivotal role to support communication between the tens and in the future potentially hundreds of interacting modules in system-on-chips (SoCs). Owing to intensive research efforts more than a decade, NoCs are now widely adopted in various SoC designs. Especially, studies on application-specific NoCs (ASNoCs) that consider the heterogeneous nature of modern SoCs contribute a significant share to use of NoCs in actual SoCs, i.e., ASNoC connects non-uniform processing units, memory, and other intellectual properties (IPs) using flexible router positions and communication paths. Although it is not difficult to find the prior works on ASNoC synthesis and optimization, little research has addressed the issues how to convert different protocols and data widths to make a NoC compatible with various IPs. Thus, in this paper, we address important issues on ASNoC implementation to support and convert multiple interfaces. Based on the in-depth discussions, we finally introduce our FPGA-proven full-custom ASNoC.**

***Index Terms*—Network-on-chip, NoC, application-specific NoC, SoC, processor, computer architecture**

## I. INTRODUCTION

Drastic technological advances in the semiconductor device fabrication have offered an opportunity to implement mobile embedded systems on a single chip. It has led to the advent of large application-specific multiprocessor system-on-chips (MPSoCs) and general-purpose chip-multiprocessors (CMPs), which are advantageous to traditional single processors in terms of performance, power consumption and footprint. In MPSoCs and CMPs, on-chip communication plays a vital role that cannot be achievable by traditional non-scalable bus architecture. Instead, networks-on-chips (NoCs) have been widely adopted as a \textit{de facto} architecture in SoCs [1-3].

Intensive research has been performed on NoC designs more than a decade to improve their performance and energy efficiency or reduce area overhead. Many NoC topologies such as mesh, torus, etc. have been presented (to list a few, [4, 5]), various design parameters such as operating frequency, bandwidth, etc. have been studied as well [6, 7]. Significant research efforts also have been put into developing routing strategy that aims to find the optimal routing paths while the dead-lock free operation should be guaranteed [8, 9].

While the most NoC works have focused on regular NoC designs that consider uniformly distributed nodes composed of homogenous processing cores and memories, relatively fewer works have studied on application-specific NoC (ASNoC) designs that contain non-uniform processing units and flexible router positions and communication paths [10-16]. A design automation flow of ASNoC was introduced in [10, 11], and a design method that considers wiring complexity of

Manuscript received Jan. 17, 2017; accepted Jul. 5, 2017
[1] SoC Design Research Group, Electronics and Telecommunications Research Institute (ETRI), Daejeon, Korea
[2] Corresponding author, Department of Electronic Engineering, Myongji University, Korea
E-mail : spacelee@mju.ac.kr

the ASNoC during the topology synthesis process was proposed in [1]. More recent works have focused on optimal floorplanning of cores and placement of routers in ASNoC [12-14] and heterogeneous networks [15, 16]. Due to the fact that intellectual properties (IPs) including cores in a SoC are highly non-uniform, those works on ASNoC designs have rapidly reached the industrial practice. As a result, several commercial ASNoC architectures are available in the market, and are popularly adopted in modern SoCs. FlexNoC from Arteries [17], NIC-301 from ARM [18] and SonicsGN from SONICS [19] are the representative commercial ASNoC architectures.

However, unfortunately none has addressed the issues to convert protocols in ASNoC or network interface (NI) development. IPs uses different protocols such as advanced extensible interface (AXI), advanced high-performance bus (AHB), advanced peripheral bus (APB), etc. They also have different data widths. For example, processors use 32 bit data while memories provide 128 bit access. A lot of papers including [20-23] present novel NI implementations, but only some work [24, 25] discusses the details when supporting the existing protocols of AXI and OCP. Even in two papers, they only pay attention to convert the protocols into their own internal ones, but not to convert two existing protocols. The conversions between the existing protocols seem trivial and straightforward, but it is not actually. Thus in this paper, we will reveal important conversion issues on ASNoC implementation. After in-depth discussions, we implemented one of possible candidates in verilog RTL code and verified functionality by running H.265 CODEC on Xilinx FPGA.

## II. BACKGROUND

This section offers a brief review of important terminology and concepts relevant to ASNoC. For a more comprehensive introduction, please take a look at the ARM advanced microcontroller bus architecture (AMBA) protocol manuals available online [26].

***On-chip protocol.*** Not only which components or blocks it houses in a SoC, but also how they interconnect has become a overriding concern for SoC industry. As a solution for the blocks to interface with each other, the AMBA protocol has become the most widely used on-chip interconnect specification in SoC designs. It includes AXI, AHB and APB.

***Transaction and transfer.*** *Transaction* and *transfer* are the terms sometimes used as synonyms in fields. We define one from another clearly in this paper as follows: Transaction means a complete operation for a request, while transfer stands for a data transmission in a certain cycle. In other words, a transaction is a set of transfers.

***Burst transaction and length.*** AXI and AHB protocol support a number of consecutive data transfers in a transaction, which is called *burst*. *Length* is the number of data in a transaction, namely the number of transfers in a transaction is determined by the length. Meanwhile, a transaction with single data is called a single transaction.

***Data width, data size, and narrow transfer.*** *Data width* means a maximum allowable data size, which equals the number of physical wires for data transfer, whereas *data size* is a size of actual transferring data. Note that the data size is irrelevant to the length, and the data size cannot be bigger than the data width. AXI and AHB provide options to specify the data size, which is called *narrow transfer* when the data size is smaller than the data width.

***Transaction ID (TID) and ordering.*** AXI protocol has a TID field, which expresses dependencies among other transactions. Although it is named as `ID', it is not a unique value but it is a kind of a group ID. Transactions assigned the same TID by the same master belong to the same group, and they should be processed in the order as generated. If transactions are issued by different master IPs or get different TIDs from a master IP, they should be processed exclusively, and thus can be processed in parallel. In other words, a slave can switch the completion order of them, which implies that a NoC should take account for the ordering requirement of transactions with same TID to guarantee the correctness of the system.

## III. ISSUES ON PROTOCOL CONVERSIONS

Given that IPs in a SoC adopt different on-chip protocols, an ASNoC should support communication between different protocols, namely protocol conversion should be implemented. Indeed, it is not difficult to implement such protocol conversion if the different

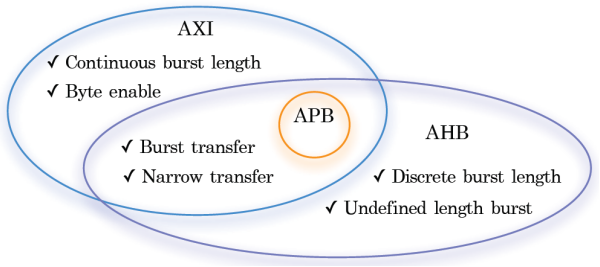



**Fig. 1.** Supportive features in different protocols.

protocols support the similar features (e.g., single transaction or incremental burst of length 4), but unfortunately they do not in reality. For example, AXI provides continuous burst length, but AHB and APB do not have such feature. Ditto for byte-enable feature. Fig. 1 shows the exclusive features belonging to the specific protocols while common ones are omitted. Now then, how to implement the protocol conversion? Before getting into the details, we would like give a brief answer that implementing a perfect protocol conversion mechanism in a NoC is almost impossible (i.e., theoretically it may be possible, but impractical), but cooperation with NoC designers and IP designers can lead practicable solutions.

### 1. AXI → AHB

As seen in Fig. 1, the burst length of AXI can be an arbitrary number (thus, continuous length), whereas AHB supports only 4, 8, 16 lengths (thus, discrete length). Let us now suppose that a master IP using AXI transmits a burst with length 6 to a slave IP using AHB. Then, a conversion problem happens. To resolve this problem, there are two possible solutions. The first is that the slave NI changes the burst to just one transaction by exploiting an INCR burst of AHB (incrementing address, unspecified number of transactions). The next is that the slave NI splits the burst into multiple transactions of length 1, 4, 8 or 16, and merges the responses after the slave IP processes. Implementing the first is simple, but the performance would not be guaranteed if a slave IP like a memory controller is optimized for a certain fixed length transactions (i.e., generally it is). In that case, the later may result the better performance although the additional hardware implementation is required. We select the first solution in our ASNoC design. Table 1 wraps up the discussion.

**Table 1.** Converting incompatible *Burst Length* between AXI and AHB

| | One transaction of undefined length | Multiple transactions of length 1,4,8,16 |
|---|---|---|
| Add. logics | - | - |
| Comparison | Maybe unoptimized for salve IPs | Complex hardware requirement |

Another prominent difference between AXI and AHB is supporting *byte-enable* feature; AXI does, but AHB does not. Supposed that a master IP using AXI sends a burst transaction with a byte-enable signal, and the byte-enable signal has more than one bit to be zero (a zero byte-enable bit means the corresponding data must be ignored), then a slave IP using AHB cannot process the burst transactions directly. Therefore, we have to find a solution of this problem.

Here is a conservative way to resolve the problem: enforce a slave NI to deliver a transaction to a slave IP only when the slave NI receives all transfers completely.

Only when all bytes are valid (i.e., there is no zero bit inside the byte-enable signal), the transaction can be treated as a burst. Otherwise, the transaction should be divided into byte-level write ones. This way is similar to the well-known telecommunications technique, *store-and-forward*. Like what the store-and-forward technique has a disadvantage of performance penalty, this conservative way does the same. More precisely, let us compare the communications of AXI to AXI and AXI to AHB. The AXI to AXI exploits the pipelined transmissions like the *cut-through* technique, which can achieve the performance enhancement. However, the AXI to AHB cannot use the pipelining method, resulting in performance degradation.

The design complexity due to handling the zero bits inside the byte-enable signal is another problem. As aforementioned, a burst should be split into byte-level transactions if there is any zero bit inside the byte-enable signal. However, if the split transactions have all valid byte-enable bits and they are consecutive, they can be merged to a burst for the fast operation. For example, let us suppose that there is a burst of length 8, and only the $4^{th}$ transfer has two zero bits in the byte-enable. Then the first three byte-level transactions can be transmitted as a burst, the $5^{th}$ to $8^{th}$ transactions ditto. The $4^{th}$ transfer has to be transmitted as two byte-level transactions. Due to

**Table 2.** Converting *byte-enable* from AXI to AHB

| | Conservative | Speculative |
|---|---|---|
| Byte Enable | Considered | Ignored |
| Similarity | Store-and-forward | Cut-through |
| Comparison | Slow & complex | Fast & simple |
| Correctness | Guaranteed by NoC | Considered by master IPs |

**Table 3.** Ways to realize the AHB to AXI communication: converting to multiple single transactions or multiple burst transactions

| | Multiple single | Multiple burst |
|---|---|---|
| Add. logics | - | Collecting logics |
| Drawback | Transaction increase | Collecting overhead |

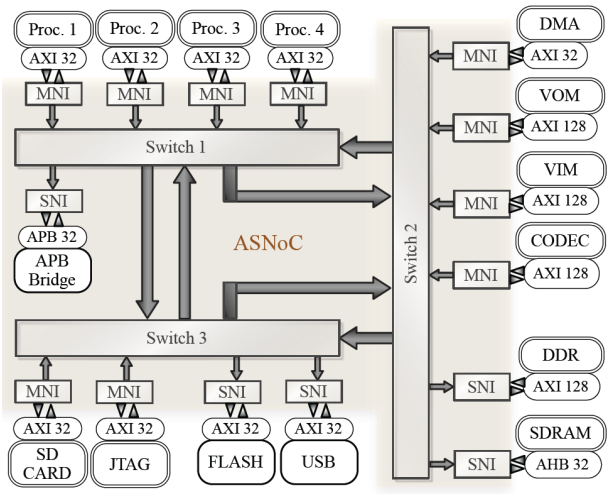
the fact that the zero bits in the byte-enable signal are random, thus there are too many cases that a burst can be split and re-merged, the above procedure may substantially increase the design complexity.

Finally, we would like to suggest that SoC designers deactivate the byte-enable feature for the AXI to AHB communication. Then the NoC speculates there is no byte-enable signals in all transactions, thus can exploit the cut-through technique for all AXI to AHB communications. Table 2 concludes discussion about the byte-enable issue, and we offer both options.

### 2. AHB → AXI

The problem here is from the INCR burst feature that is supportive in AHB but is not in AXI (i.e., again, AXI cannot deal with the undefined burst length as seen in Fig. 1. To resolve this problem, the easiest solution with a simple hardware support is to dismantle a burst transaction and re-generate multiple single transactions. However, this way may cause low performance of AXI slave IPs that are optimized to the burst transactions, and busy network traffic due to the number of transactions increase.

On the other hands, we may hold benefit of using burst transactions by packing a part of transfers as new transactions of the fixed length. A master NI or a slave NI has a certain number of buffers and generates a transaction when the buffers are full or the burst ends. This method extends the latency due to gathering while the network traffic can be reduced. Depending on the number of buffers, there are a trade-off on these effects. Table 3 summarizes the answers of the question in this




* indicates a master IP, while is for a slave IP. Proc. stands for processor core. DMA refers direct memory access. VOM and VIM are video input and output modules, respectively.

**Fig. 2.** An example SoC architecture that our ASNoC targets.

subsection. In our ASNoC, because few IPs exploit the INCR feature, we choose the first method, the simple one.

### 3. (AXI or AHB) → APB

As shown in Fig. 1, APB does not have any special feature that AXI or AHB has. In other words, a master IP has strict rules to use APB. Therefore, conversion issues on AXI or AHB to APB is straightforward: a master IP that transmit any transaction to an APB slave IP should transmit transactions fit to APB requirement. A master IP must not use the byte-enable. Because APB does not support the narrow transfer, data widths of transactions from a master IP should be same to the APB data width.

### 4. Other Protocols

We have discussed AMBA protocols since they are most common in commercial uses, but there exist other protocols such as wishbone [27] and TileLink [28]. It seems inefficient to convert one by one but it may require a common intermediate format that covers all protocols, which is another challenge.

## IV. Issues on Data Width Conversions

Fig. 2. is an example of SoC architectures that our ASNoC targets. This architecture is composed of ten master IPs and five slave IPs. Later in Section V, we will

describe the details of this architecture, but now we would like focus on the data widths of the IPs. As seen in the figure, some IPs use 32 bit operation, whereas others use 128 bit. To make the IPs using different data widths communicate with each other, the data widths should be converted. And, here comes the questions: how to perform the data width conversion, and who takes charge of it?

**1. Narrow Slave Transaction**

We call the case when a master IP's data width is greater than a slave IP's the *narrow slave transaction*, i.e., this is different with a term, narrow transfer. In this case, any data from a master IP must be split according to the data with of a target slave IP. Splitting data causes the *length* (the number of data in a transaction) increase in proportion to data width of a master IP divided by that of a slave IP. For instance, if a 128 bit master IP sends 8 data in a transaction to a 32 bit slave IP, the converted length becomes 32 (= 8·128/32). This conversion process may look simple. However, unfortunately, there is a limitation of the maximum allowable length in AMBA protocol (e.g., 16 in AXI protocol). So the 32 lengths transaction resulting from the example should be divided into two transactions, each of which has 16 lengths. Of course, the split transactions should be merged to one after they are processed in the slave IP.

If implementing master NIs to take charge of the splitting & merging process, we have only concern to make master NIs process their own transactions well. In other words, a master NI splits a transaction from its master IP and sends them to a target slave IPs. When the split transactions are processed and returned to the master NI, the master NI merges the results and sends it back to the master IP. A problem that we are facing here is to let the master NI correctly merge the results of the split transactions to the results of the original transactions. Let us suppose that there are many transactions given to the master NI, each of which has own TID assigned by the master IP. The master NI splits and transfers the transactions in a way that each split transaction holds the TID same to that of its parent transaction. Then, because a slave IP can process the transactions with different TID in any order, the returned results of split transactions may not follow the original order that the transactions were sent. For example, A and B transactions are divided into A0~A3 and B0~B3, respectively. A master NI sends transactions in the order A0-A1-A2-A3-B0-B1-B2-B3, but they can be returned in the order A0-B0-B1-A1-B2-B3-A2-A3. Therefore, the master NI has to reorder the returned (out of order) results of the split transactions to merge them.

To avoid overhead due to the reordering process and accompanying hardware, we can design a master NI to assign a new TID to the split transactions instead of using the parent TID, i.e., the parent TID may be recorded in the master NI for the future merging process. By assigning the same TID to all the split transactions (thereby a slave IP sequentially process the transactions), the master NI expects the results will be returned in order. This way, however, can cause the performance degradation due to the low degree of parallelism.

Meanwhile, placing master NIs in charge of the narrow slave transaction may inflict a loss of network utilization. A network that supports a communication between 128 bit master and slave IPs has a bandwidth of 128 bit. Converting a 128 bit transaction to four 32 bit transactions results in only partial utilization of the 128 bit bandwidth, and multiple cycle transmission causes the low network latency. The resulting multiple packets also induce the network overhead. To overcome these issues, we can design the slave NIs to perform the data width conversion. Because the transactions are split and merged in slave NIs after transmitting the network, the network can be fully utilized.

However, merging process may become more complicated than the previous master NI-centered method. A slave NI should take care of (the number of master) × (the number of TIDs in one master) TIDs at the same time, which may require the more complex reordering process. Similar to what the master NIs assign their own TIDs to the transactions to skip the reordering process, we can adopt the way that a slave NI gives a (same) new TID to all transactions. In our ASNoC implementation, we choose this way. The discussion about the *narrow slave transaction* is summarized in Table 4.

**2. Narrow Slave Transactions of a Wrapping Burst**

A wrapping burst is similar to an incrementing burst,

**Table 4.** Splitting & Merging for Narrow Slave Transactions

| | By MNI | | By SNI | |
|---|---|---|---|---|
| | TIDs from Master IP | A TID from a MNI | Concatenated TIDs | A TID from a SNI |
| MNI | Reordering is required | - | - | |
| Switch | Overhead due to more transactions, Partial utilization | | Full utilization | |
| SNI | - | Low DoP | Reordering is required | - |

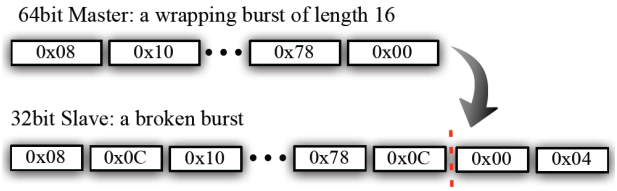



**Fig. 3.** An example of the broken wrapping burst in the narrow slave transactions.

except that the address wraps around to a lower address if the address reaches the predefined limit [26]. Because the data width conversion may change the sequence of addresses, the resulting split transactions may become no longer the wrapping bursts like Fig. 3. In the figure, a 64 bit master sends a wrapping burst of length 16, which will be transformed to 32 data in a 32 bit slave. Supposed that a slave NI divides it into two 16 data, 0x08$\sim$0x44 and 0x48$\sim$0x04, since the 32 data exceeds the possible *length*. The former part can be expressed as an incrementing burst of 16. However, unfortunately the later part cannot be any burst since the address of the wrapping burst of length 16 starting from 0x48 is wrapped to 0x40 after 0x78.

The easiest way maybe is converting each burst to multiple single transactions. It makes a design quite simple, but the number of transactions are skyrocketing to the *length*, which may cause significant overhead to network and/or a slave. One way to increase the efficiency is that first we split a transaction at the wrapping point, which corresponds to the red line in Fig. 3 and then adopt the approaches in Section IV-A as the second step. This solution may reduce the performance overhead due to the increase of transactions, but the required hardware may become complicated. Another possible solution is that a master NI or a slave NI issues a wrapping burst as an incrementing burst and then corrects a sequence by reordering the returned data autonomously. It is the most efficient way in terms of network traffic, but it may cause latency increase and design complexity. Table 5 sums up this subsection.

**Table 5.** Handling the Narrow Slave Transactions of Wrapping Burst

| | Multiple single | Multiple burst | |
|---|---|---|---|
| | | Sequential | Reordered |
| Number of Transactions | Many (= *Length*) | Low | Lowest |
| Comparison | Slow & complex | Fast & complex | Fastest & Most complex |

**Table 6.** The ASNoC resource consumption in the FPGA. (value) is the total resource consumption

| Logic LUTs 35,450 (665,814) | | FFs 50,248 (457,910) | LUTRAMS 990 (4,936) |
|---|---|---|---|
| SRLs 48 (1,439) | RAMB36 0 (485) | RAMB18 0 (367) | DSP48 0 (1,129) |

## 3. Wide Slave Transactions

There exists an opposite case to the narrow slave transaction whereby data width of a slave IP is greater than that of a master IP. We call this *wide slave transaction*. If a target slave IP uses AXI or AHB that supports the narrow transfer, slave NIs just send their small data width transactions to the slave IP (no problem occurs). However, a slave IP using APB that does not support the narrow transfer does not know what to do with the partial data. For example, if a master IP requests a 16 bit write operation to a slave IP using 32 bit APB, the slave IP cannot perform the write operation directly. Instead, it may have to go through several steps that read all the 32 bit data for the target address first, change the last 16 bit of the data to the requested ones, and write the 32 bit data to the address. This solution looks plausible, but the fact that APB slave IPs carry out various operations (e.g., specific register controls) as well as the read/write operations, the presented solution may incur unexpected malfunctions. Consequently, there may not exist such a perfect solution of the wide slave transactions for ABP.

In our ASNoC, we implement APB slave IPs to perform read/write regardless of the original data widths of transactions.


**Fig. 4.** An example of our ASNoC demonstrations: it is implemented on a Virtex-7 FPGA board to run the H.265 codec..

## V. IMPLEMENTATION

We have implemented our ASNoC in synthesizable verilog RTL codes. To verify the functional correctness of our ASNoC, we have designed a platform architecture shown in Fig. 2 and applied our ASNoC to it. The architecture contains ten master IPs including a quad-core processor, a H.265 codec, video input and output modules, etc. and five slave IPs including memories, JTAG and USB, etc. The IPs use AXI, AHB and APB protocols. The data width of the IPs also varies. The network consists of three switches in order to link the IPs. Since each IP has its own frequency, master NIs and slave NIs use asynchronous FIFOs for clock domain crossing. The FIFOs are also inserted between switches if they runs at different clocks.

The processors can use DDR or SDRAM as a main memory, and controls other masters through APB bridges where slave ports of masters are connected. The video input module periodically writes raw images from a camera into memories. On the contrary, the video output module reads images from memories and displays them on a monitor through a HDMI cable. The H.265 codec operates by collaborating with these two video modules persistently.

We have developed several testbenches to check the system performance as well as functionality of each module. Finally, we have mapped the architecture onto a FPGA board, Xilinx Virtex-7 FPGA XC7V2000T [29]. The consumed resources are listed in Table 6. In the FPGA, the ASNoC runs at 100 MHz while the clock frequency of other modules varies from 50 to 100 MHz. Fig. 4 is an example that captures one of testbenches running in the board, displaying a H.265 video.

## ACKNOWLEDGMENTS

This research is partially supported by the IT R&D program of MSIP/IITP (2016-0-00088, Development of Intelligent Semiconductor Common Platform Technology for Smart Devices), and Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2017R1D1A1 B03027911).

## REFERENCES

[1] S. Murali et al., “Designing Application-specific Networks on Chips with Floorplan Information,” in Proc. of the Int’l Conf. on Computer-Aided Design, pp. 355–362, 2006.

[2] M. Dall’Osso et al., “Xpipes: A latency insensitive parameter- ized network-on-chip architecture for multi-processor SoCs,” in Proc. of the IEEE Int’l Conf. on Computer Design, pp. 45–48, 2012.

[3] D. Zhu et al., “TAPP: Temperature-aware pplication mapping for NoC-based many-core processors,” in Proc. of the Int’l Conf.. on Design, Automation & Test in Europe, pp. 1241–1244, 2015.

[4] W. Dally and B. Towles, Principles and Practices of Interconnection Networks. Morgan Kaufmann Publishers Inc., 2003.

[5] P. P. Pande et al., “Performance evaluation and design trade-offs for network-on-chip interconnect architectures,” IEEE Trans. Comput., vol. 54, pp. 1025–1040, Aug 2005.

[6] U. Y. Ogras, J. Hu and R. Marculescu, “Key research problems in NoC design: A holistic perspective,” in Proc. of the Int’l Conf. on Hardware/Software Codesign and System Synthesis, pp. 69–74, 2005.

[7] A. B. Kahng et al., “ORION 2.0: A fast and accurate noc power and area model for early-stage design space exploration,” in Proceedings of the Int’l Conference on Design, Automation & Test in Europe, pp. 423–428, 2009.

[8] J. Hu and R. Marculescu, “Energy- and

performance-aware mapping for regular noc architectures," IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, vol. 24, no. 4, pp. 551–562, 2005.

[9] M. Palesi et al., "A methodology for design of application specific deadlock-free routing algorithms for noc systems," in Pro- ceedings of the Int'l Conf. on Hardware/Software Codesign and System Synthesis, pp. 142–147, 2006.

[10] K. Han, J. J. Lee, J. Lee, W. Lee, and M. Pedram, "TEI-NoC: Optimizing Ultra-Low Power NoCs Exploiting the Temperature Effect Inversion," in IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, doi: 10.1109/ TCAD.2017.2693269

[11] K. Goossens et al., "A Design Flow for Application-Specific Networks on Chip with Guaranteed Performance to Accelerate SoC Design and Verification," in Proc. of the Interna- tional Conference on Design, Automation & Test in Europe, pp. 1182–1187, 2005.

[12] L. Benini, "Application Specific NoC Design," in Proc. of the Int'l Conf. on Design, Automation & Test in Europe, pp. 1–5, 2006.

[13] S. Kwon, S. Pasricha and J. Cho, "POSEIDON: A framework for application-specific network-on-chip synthesis for heterogeneous chip multi-processors," in Proc. of the Int'l Symp. on Quality Electronic Design, pp. 1–7, 2011.

[14] K. S. M. Li, "CusNoC: Fast full-chip custom NoC generation," IEEE Transactions on Very Large Scale Integr. Sys., vol. 21, pp. 692–705, April 2013.

[15] J. Soumya and S. Chattopadhyay, "Application-specific network-on-chip synthesis with flexible router placement," J. Syst. Archit., vol. 59, pp. 361–371, Aug 2013.

[16] B. Grot et al., "Kilo-NOC: A heterogeneous network-on-chip architecture for scalability and service guarantees," in Proceed- ings of the Int'l Symp. on Comp. Arch., pp. 401–412, 2011.

[17] A. K. Mishra, O. Mutlu and C. R. Das, "A heterogeneous mul- tiple network-on-chip design: An application-aware approach," in Proc. of the Design Automation Conf., pp. 36:1–36:10, 2013.

[18] Arteris, "http://www.arteris.com/flexnoc."

[19] ARM, "http://www.arm.com/products/system-ip/ interconnect/corelink-nic-family.php."

[20] Sonics, "http://sonicsinc.com."

[21] A. Radulescu et al., "An efficient on-chip network interface offering guaranteed services, shared-memory abstraction, and flexible network configuration," in Proc. of the Int'l Conf. on Design, Auto. & Test, vol. 2, pp. 878–883, 2004.

[22] K. Goossens, J. Dielissen and A. Radulescu, "Aethereal network on chip: concepts, architectures, and implementations," IEEE Design & Test of Computers, vol. 22, pp. 414–421, Sept 2005.

[23] J. Sparsø, E. Kasapaki and M. Schoeberl, "An area-efficient net- work interface for a tdm-based network-on-chip," in Proceed- ings of the Int'l Conf. on Design, Auto. & Test in Europe, pp. 1044–1047, 2013.

[24] A. Radulescu et al., "An efficient on-chip ni offering guaran- teed services, shared-memory abstraction, and flexible network configuration," IEEE Trans. on Computer-Aided Design of Integr. Circuits and Systems, vol. 24, pp. 4–17, Jan 2005.

[25] X. Yang et al., "Nisar: An axi compliant on-chip ni architecture offering transaction reordering processing," in Proc. of the Int'l Conf. on ASIC, pp. 890–893, 2007.

[26] T. Bjerregaard et al., "An ocp compliant network adapter for gals-based soc design using the mango network-on-chip," in Proceedings of the Int'l Symp. on System-on- Chip, pp. 171–174, 2005.

[27] ARM,"https://www.arm.com/products/system-ip/ amba-specifications."

[28] Wishbone, "https://opencores.org/opencores,wishbone."

[29] TileLink,"http://bar.eecs.berkeley.edu/projects/201 4-tilelink.html."

[30] XILINX, "https://www.xilinx.com/products/boards-and-kits/ek-v7-vc707-g.html."

**Kyuseung Han** received the B.S. and Ph.D. degrees in Electrical Engineering and Computer Science from Seoul National University, Seoul, Korea, in 2008 and 2013. From 2014, he has been working at Electronics and Telecommunications Research Institute, Daejeon, Korea. His research interests include reconfigurable architecture, network-on-chip, aand low power embedded systems.

**Jae-Jin Lee** received the B.S., M.S., and Ph.D. degrees in Computer Engineering from Chungbuk National University in 2000, 2003, and 2007, respectively. He is a group leader of the SoC Design Research Group at Electronics and Telecommunications Research Institute. His research interests include processor and compiler designs in ultra-low power embedded systems..

**Woojoo Lee** received the B.S. degrees from the Department of Electrical Engineering, Seoul National University, Seoul, Korea in 2007, and the M.S. and Ph.D. degrees in Electrical Engineering from University of Southern California, Los Angeles, CA, in 2010 and 2015. He worked at Electronics and Telecommunications Research Institute as a senior researcher in SoC Design Research group, and is currently an assistant professor with the Department of Electronic Engineering, Myongji University. His research interest includes ultra-low power VLSI designs, SoC designs, embedded system designs, and system-level power and thermal management.