Received May 24, 2019, accepted June 6, 2019, date of publication June 14, 2019, date of current version July 1, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2923219

MMNoC: Embedding Memory Management Units into Network-on-Chip for Lightweight Embedded Systems

HYEONGUK JANG[1,2], KYUSEUNG HAN[1], (Member, IEEE), SUKHO LEE[1], JAE-JIN LEE[1,2], AND WOOJOO LEE[3], (Member, IEEE)

[1]Electronics and Telecommunications Research Institute, Daejeon, South Korea
[2]Department of ICT, University of Science and Technology, Daejeon, South Korea
[3]School of Electrical and Electronics Engineering, Chung-Ang University, Seoul, South Korea

Corresponding author: Woojoo Lee (space@cau.ac.kr)

This work was supported in part by the ICT R&D program of MSIT/IITP under Grant 2018-0-00197, development of ultra-low power intelligent edge SoC technology based on lightweight RISC-V processor, and in part by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education under Grant 2017R1D1A1B03027911.

ABSTRACT With the advent of the Internet-of-Things (IoT) era, the demand for lightweight embedded systems is rapidly increasing. So far, ultra-low power (ULP) processors have been leading the development of lightweight embedded systems. However, as the IoT era gets more sophisticated, existing ULP processors are expected to reach a critical limit in the absence of a memory management unit (MMU) in that multiple programs cannot be run in the MMU-less embedded systems. To tackle this issue, we propose an architecture in which the MMU is embedded in a network-on-chip (NoC). Through the proposed approach, NoC offers MMU functionality without modifying the processor design, allowing developers to easily leverage the existing ULP lightweight processors and build embedded systems that support multiprocessing. In this paper, along with the details of the proposed MMU-embedded NoC (MMNoC) design, a prototype platform including the MMNoC and dual RISC-V processors is provided. The prototype platform is synthesized with FPGA and Samsung 28 nm FD-SOI technology to verify the functional accuracy and small performance, area, and power overhead of the MMNoC.

INDEX TERMS Network-on-chip, NoC, memory management unit, MMU, embedded system.

I. INTRODUCTION

As the age of Internet-of-Things (IoT) begun, many changes are taking place in the world of embedded systems. One of the major shifts is the explosion in demand for small embedded systems with an emphasis on energy efficiency, easy development and affordability of the systems [1]–[3]. In these small embedded systems, simple, narrow-issue and low-power processors, referred to as *lightweight* processors, are desirable in that system developers can dynamically integrate multiple lightweight processors to achieve greater energy efficiency with less costly design efforts. And even if performance degradation is severe, an ultra-low-power (ULP) processor that consumes significantly less power than traditional low-power processors is more desirable. In line with this trend, the ULP and lightweight processors have been intensively researched and developed from both academia [4]–[8] and industry [9]–[12]. In this paper, we call such small embedded systems using the ULP lightweight processors the *lightweight embedded systems*, and research on their design methodologies.

Meanwhile, another big change in the embedded systems is that the required functionality of the embedded system are being expanded. Apart from the migration of existing large embedded systems to smaller ones, the recent embedded systems are desired to be able to run multiple programs alternately or concurrently. In other words, as the embedded systems are used to collect and analyze various types of data in the IoT era, the multiprocessing capabilities of embedded systems are becoming more important. For example, embedded systems for IoT end nodes with various sensors tend to perform varied types of lightweight data processing [13]–[15], and therefore multiprocessing is necessary to these types of embedded systems. Especially for the embedded

The associate editor coordinating the review of this manuscript and approving it for publication was Adnan M. Abu-Mahfouz.

systems targeting biomedical applications whereby multi-lead biological signals requires to be be processed in parallel and energy efficiently, multiple ULP processors are used in such systems to support multiprocessing [16], [17]. In this regard, it is not surprising that the well-known IoT platform, PULP (parallel and ultra low power platform) [7], [18], [19], supports multiprocessing.

Between the two trends we have discussed above, we can find a clear niche in what to do if we develop the lightweight embedded system that support multiprocessing. To support the multiprocessing, memory management unit (MMU) is essential [3], [20], and must be implemented in the embedded system. But unfortunately, due to the area, power, and cost overhead of integrating the MMU into the processor [21], most commercial ULP lightweight processors are MMU-less (i.e., there are few ULP processors with own MMUs in academia [17], [18]). Consequently, embedded systems using such ULP lightweight processors cannot support multiprocessing by default.

To solve this problem, we explored an idea of detaching MMUs from processors and placing them in the other hardware intellectual properties (IPs), which has been researched in the multiprocessor system-on-chip (MPSoC) field [22]–[26]. While most of the previous research based on the MMU isolation idea focused primarily on improving the performance of MPSoC for high-performance platforms, we found this idea to be a solution to our attention. In other words, embedding MMU to another IP in a platform may allow the embedded system developers to build the MMU-integrated platform by using existing ULP lightweight processors.

Especially, we paid attention to the network-on-chip (NoC) that is a common IP for system interconnect in modern embedded system platforms [8], [27], [28]. Then, attention has been paid to previous researches on the placement of MMUs in network-on-chip (NoC) [23]–[25]. Starting with a distributed MMU that uses multiple MMUs as the resources of NoC to perform operations to handle memory access requests, we propose a new NoC design that includes a lightweight MMU architecture suitable for lightweight embedded systems. This approach eliminates the need to modify the processor design, therefore any existing ULP lightweight processors can be used in the lightweight embedded systems that support multiprocessing. In addition, the use of the same MMU design in the proposed NoC simplifies the development of embedded system software, otherwise different types of MMUs should be taken carefully into the consideration in the software development. In this paper, the details of the proposed MMU design are presented, and application-specific NoC is introduced along with the proposed MMU (i.e., we call it MMNoC). Finally, an entire system prototype, including MMNoC and RISC-V processors, is implemented in register transfer level (RTL) Verilog HDL. For the verification and evaluation of the MMNoC, the prototype is synthesized in both Xilinx FPGA and Samsung 28nm FD-SOI technology. The simulation results with the synthesized prototype show the functional accuracy of the proposed MMNoC and lightness suitable for lightweight embedded system.

The remainder of this paper is organized as follows. Section II is dedicated to a preliminary of the memory management systems for multiprocessing in embedded systems. Section III elucidates the details of the proposed architecture, including an embedded MMU design, a network interface, and MMNoC architecture. Related work is also provided in Section III. Section IV is to present the experimental setups, executions and simulation results, while Section V concludes the paper.

II. MEMORY MANAGEMENT SYSTEM: A PRELIMINARY

Memory management system (MMS) plays a pivotal role to manage the primary memory (e.g., SRAM, DRAM) by controlling and tracking the status of each memory allocation. Traditionally, system softwares such as operating systems (OS's) have MMS and take a responsibility of the memory management [3], [20]. However, the lightweight embedded systems that generally do not have an OS, the embedded system software developers (i.e., hardware users) should take care of the memory management by themselves.

Meanwhile, physical (memory) address space for a program (or process) depends on the embedded hardware platforms. Moreover, even on the same hardware platform, physical address spaces change on the fly, so they change over time. For example, various embedded hardware platforms can have different sizes, types, and numbers of RAM (random access memory). And, if a hardware module inside the platform uses memory-mapped IO, the address space assigned to the hardware module must not be assigned to the other modules or programs. In addition, if some range of address space is already assigned to a running program, a newly running program only can access the remaining parts of the address space, therefore the physical addresses assigned for the new program running on the same type of the hardware platforms may be different at time. For the reasons stated above, a compiler assumes that the address space is ideal when compiling the code. This means that the address space is supposed to be large enough, and the addresses are assumed to be consecutive and start at address 0. Compared to the physical address space, this ideal space is called *virtual* address space.

To run a program (process), it is necessary to map the virtual address to a physical address at runtime. One of the best known technique to manage the mapping is *demand paging*. Managing the arbitrary size of data can induce significant increase of system complexity. Therefore, it is preferred to use a fixed size of data, which is called *page*. And the memory management system based on pages is called paging. FIGURE 1 shows the paging based memory management. Since paging is a kind of caching techniques between main memory and storage, the paging has similar behaviors with the cache allocation. When a process references an address,

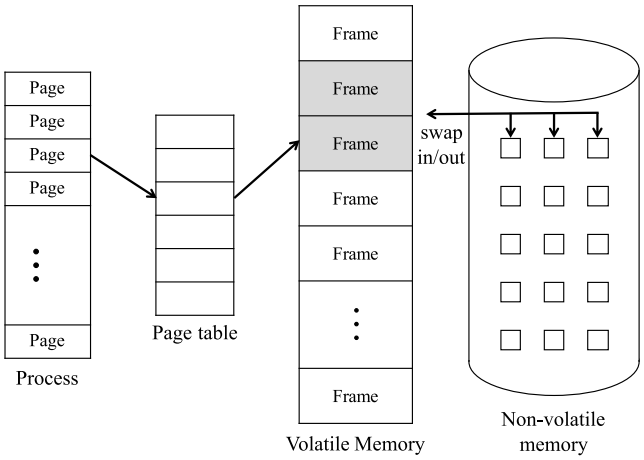

FIGURE 1. A structure of the paged memory management.

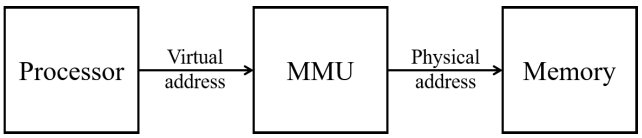

FIGURE 2. Conversion of virtual address into physical address in a hardware perspective.

the MMS checks whether the page has already been allocated to physical memory, by referring to the *page table* where the page table contains all the information of the mapping. If it exists, the MMS will service the request. Otherwise, which is called *page fault*, the MMS finds an empty *frame*, which is a segment of physical memory, in order to store a new page reading from the storage. If there is no room, some pages should be swapped out to the storage, similar to the cache eviction. After securing a frame, the mapping between the page and the frame is updated to the page table and then the requested reference is serviced.

MMU is a hardware for MMS that converts a virtual address to a physical address using a page table as shown in FIGURE 2. MMUs are usually integrated within the middle and high end processors, namely each processor has its own MMU. Typically, the entire page table is always in RAM, and the MMU caches the most recently used entries in the page table.

III. MMU EMBEDDED NoC

A. PROPOSED APPROACH

Modern embedded systems are required to perform a variety of functions that multiprocessing must support. The emerging lightweight embedded systems are no exception to this trend. In other words, these lightweight embedded systems require multiprocessing capabilities, making MMUs indispensable for such systems. The direct way to do this is to develop a new ULP lightweight processor with an MMU, because there is no MMU in the existing ULP lightweight processors. But from the embedded system platform engineers' point of view, developing a new processor is not practical because it requires a lot of design effort and development cost. Instead, if there

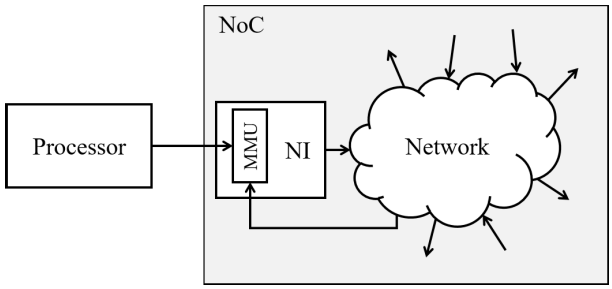

FIGURE 3. Implementing an MMU on an NI that is dedicated to a processor.

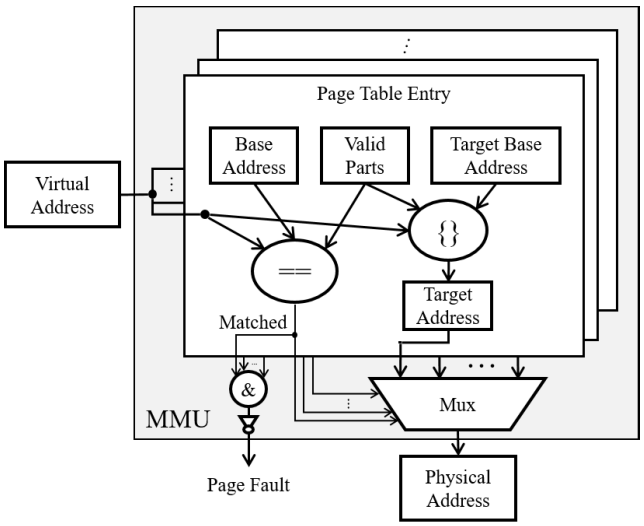

FIGURE 4. The proposed MMU architecture.

is a way to integrate MMU functionality into an embedded system without modifying the existing processor design (and thus the platform engineer can choose any existing processor based on the design specification), that would be the most practical approach.

To realize this approach, we focused on NoC, an IP commonly found in embedded system platforms, and propose to embed an MMU into NoC. NoC plays an important role in supporting concurrent communication on embedded system platforms [8], [27], [28]. In state-of-the-art embedded systems, NoC has become one of the most popular system interconnect IPs owing to its ability to overcome the limitations of the conventional bus-based system interconnects (e.g., unbearable increasing density and complexity induced by the system interconnect) [29]–[31]. Motivated by the fact that a processor in the platform with NoC communicates with other IPs only through the dedicated network interface (NI) in the NoC, we come up with an idea that we implement an MMU in an NI, as shown in FIGURE 3, so that we can easily provide the MMU functionality to the platform regardless of the processor types in the platform.

B. RELATED WORK

Compared to the common processor-individual MMU, some previous research have tried to place the MMU

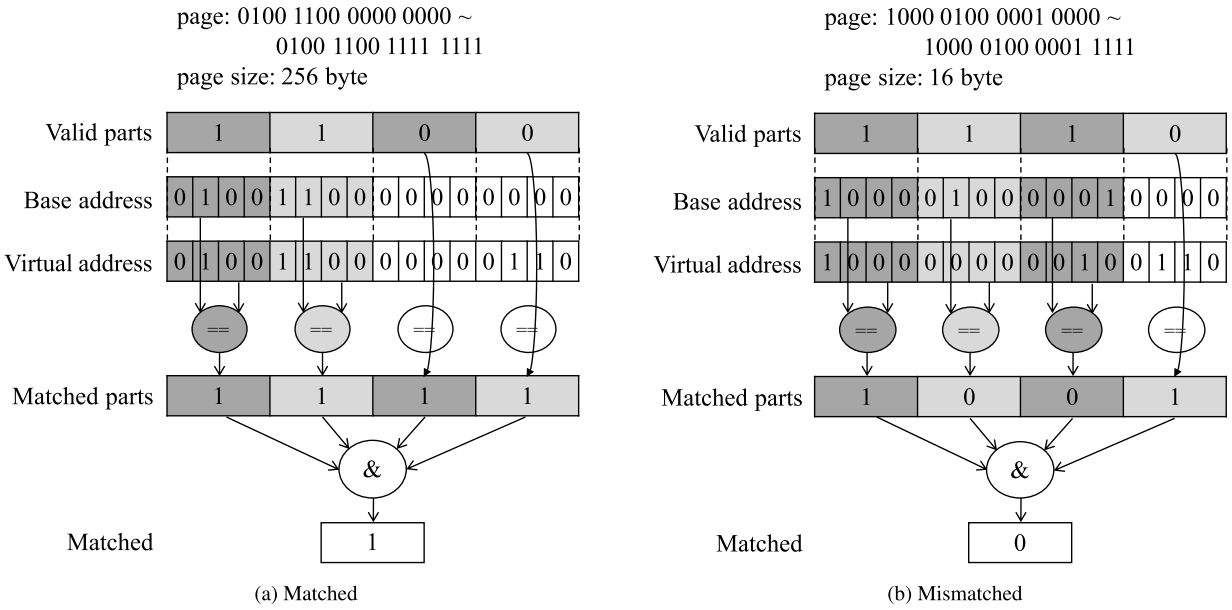

FIGURE 5. Examples of the PTE operation, when the page size is (a) 256kB and (b)16kB.

outside the processors [22]–[26]. For example, an on-chip centralized hardware MMU module was presented for data allocation on the distributed shared memory space in the NoC-based MPSoC [22]. And, a distributed MMU architecture exploiting the NoC architecture was introduced to reduce the memory access bottleneck in contrast of traditional MMU [23]. This work targets the mesh-based NoC and provides detailed memory access mechanism that effectively improves network throughput. Meanwhile, in order to reduce the design complexity and increase the flexibility of memory management in MPSoC, a programmable microcoded controllers including mini-processors was proposed [24]. The microcoded controllers are processor-independent, each of which locates in the node of the NoC. In [25], a concept of memory protection unit (MPU) based on NoC was introduced. The proposed concept of the MPU covers most of the complex functions of the MMU in the high-end processors, in that it provides data protection in the shared memory as well as the address translation. More recently, a lightweight MMU for many-core accelerators was proposed [26]. The proposed lightweight MMU provides virtual memory support for the cluster-based many cores, and there is a host processor that manages the lightweight MMU.

Our approach in this paper is similar to the previous works that separate MMUs from the processors and deploy them in NoC [23]–[25]. However, compared to the previous works aimed at MPSoC for high-performance platforms and to improve the performance of MPSoC, this approach aims at a lightweight platform in which the MMUs embedded in the NoC should be light and to simply support the address translation. In this regard, the previous studies except [24] that only perform simulations based on conceptual MMU design methods without implementing real MMUs in NoC may not be practical for the lightweight embedded systems. In this paper, we devise a lightweight MMU on NoC (MMNoC) and implement the lightweight embedded system prototype equipped with the MMNoC. Based on the prototype, we provide detailed experimental results in Section IV. Note that the MMNoC prototype demonstrates the much smaller number of gates are required for the MMNoC than the microcoded controller presented in [24].

C. MMNoC ARCHITECTURE

On the basis that the MMNoC targets lightweight embedded system platforms whereby the low power and small area are desirable, the MMU is designed to have small footprint, and therefore supports the minimal functionality of common MMU and NoC. The details of the proposed MMU architecture in the MMNoC is described in FIGURE 4. As seen in the figure, the MMU has several page table entries (PTEs), each of which contains conversion information of a single page. Each PTE generates intermediate results, *matched* and *target address*. The combination of the intermediate results produces the final results, *page fault* and *converted physical address*.

FIGURE 5 shows the examples of how the PTEs operate in the MMU. We divide an address into several parts and configure the number of parts as four in these examples. A *base address* and *valid parts* represent a virtual address page to be converted by one PTE. The valid parts determine the size of the page by indicating the parts to be compared between a virtual address and a base address. For example, the valid parts "1100" in FIGURE 5 (a) and "1110" and FIGURE 5 (b) mean that each page size is 256kB and 16kB, respectively. The comparison result of the virtual address and the base address is processed to the matched parts as seen in the figure.

Algorithm 1 PTE Operations

```
1:  function CALCULATE_A_MATCH(va,ba,vp)
2:      n_p : the number of parts
3:      pa_v : an array of parted virtual addresses
4:      pa_b : an array of parted base addresses
5:      vp : an array of validity of parts
6:
7:      if all vp = 0 then
8:          matched ← 0
9:      else
10:         for i = 0 to n_p − 1 do
11:             if vp[i] = 1 then
12:                 if pa_v[i] = pa_b[i] then
13:                     matched_parts[i] ← 1
14:                 else
15:                     matched_parts[i] ← 0
16:                 end if
17:             else
18:                 matched_parts[i] ← 0
19:             end if
20:         end for
21:         if all matched_parts = 1 then
22:             matched ← 1
23:         else
24:             matched ← 0
25:         end if
26:     end if
27: end function
```

The comparison procedure is designed to be performed only when the corresponding valid parts is 1. For instance in FIGURE 5 (a), since only the first and second valid parts are 1, no comparison is performed for the remaining parts, while a comparison between the base address and the virtual address is performed for the first and second parts. As the comparison results, both first and second parts report 1 to the matched parts, because the virtual and base addresses in each part fall into line. Without comparison, the third and fourth parts just write 1 to the corresponding matched part. Meanwhile, as seen in FIGURE 5 (b), because the first, second and third valid parts are 1, the comparisons are performed to these parts. The base and virtual addresses in the first part are same, which writes 1 to the matched part, while the second and third pare not, thereby writing 0 to the matched parts.

Finally, we conduct an 'AND' operation on the matched part to confirm if the virtual address matches the target range of the PTE. From the matched bit in FIGURE 5 (a) and (b), each of which is 1 and 0, we conclude that it is matched and mismatched, respectively.

The detailed procedure of the PTE operation is introduced in Algorithm 1. Compared to the description of the above example, we add a new process to cover the invalid PTE in the algorithm, which is described in the line 7∼8. In a case when all valid parts are 0, it semantically means that the PTE is not

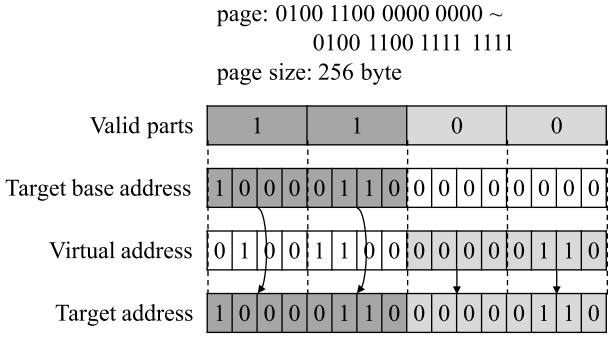

FIGURE 6. An example of the address translation.

configured. In this case, however, the match bit can be set to 1, which is undesirable. Therefore, we add the process that forces to set the matched bit to 0, when all the valid parts are 0.

By collecting the matched bits of all PTEs, the page fault can be determined. In other words, if there is no matched bit that holds 1 (i.e., all the matched bits are 0), a page fault will be generated. Otherwise, the address translation should be performed for each case where the matched bit is 1. On the hardware side, there may be many cases where the match bit is 1, so the system software must manage that the addresses should not be overlapped. Meanwhile, as described in FIGURE 4, we use a multiplexer to select a target address that will be converted to a physical address. The target address is generated by concatenating the valid parts of the target base address and the invalid parts of the virtual address. An example of generating a target address is provided in FIGURE 6.

The proposed MMU has an interface for configuring three kinds of registers for the base address, the valid part and the target base address. We design the interface to have memory-mapped I/O implemented as an advanced peripheral bus (APB) protocol for configuration. However, this implementation may cause a critical problem, which must be handled and addressed carefully. Since the addresses for the configuration also pass the MMU, they can cause infinite page faults. This means that if a processor tries to write to the MMU for the first time when the PTE is not configured, a page fault will occur. Because of the page fault, the processor repeatedly attempts to access the MMU via memory-mapped I/O, resulting in the infinite loop. In order to prevent this problem, Therefore, to prevent this problem, we add a special PTE that is responsible for the configuration address. When the MMU receives a configuration address, then this special PTE always set the matched bit to be 1, so that the address passes the MMU without a page fault. The special PTE is fixed at the design time and not changeable during the execution time.

The entire architecture of the proposed MMU is designed to be configurable and customizable for different platforms. Depending on how platform developers/designers set different values for the valid parts, the MMU can have multiple page sizes. This allows the developer to maximize memory usage. Especially, the developers can choose the page sizes

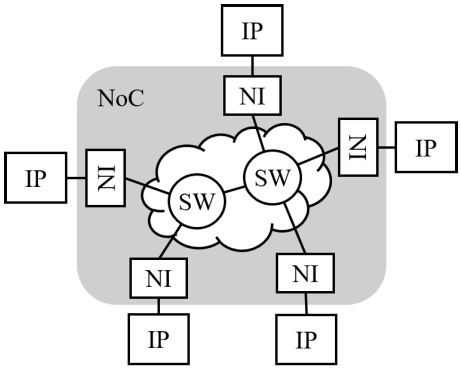

FIGURE 7. Application specific NoC architecture.

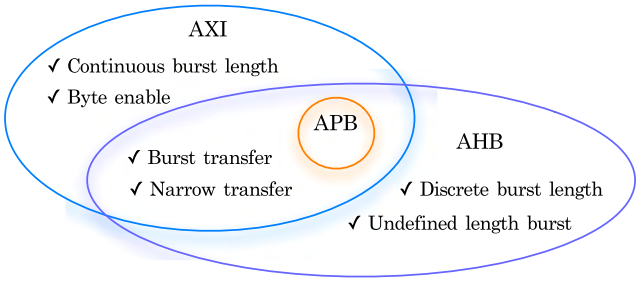

FIGURE 8. Comparison of three AMBA protocols.

by adjusting the number of address parts in the design time, and the selected page sizes can be used during the run time. In addition, developers can minimize the occupying area of the MMU, by inserting the minimum numbers of PTEs based on the expecting system requirement.

D. MMNoC DESIGN

The proposed MMNoC exploits the application-specific NoC (ASNoC). AsNoC [32] is the most practical and widely used NoC type in modern embedded system platforms. ASNoC tools such as FlexNoC [33], NIC-301 [34], and SonicsGN provide development environments for customizing NoC depending on a target system. FIGURE 7 shows a representative architecture of the ASNoC, which consists of NIs and switches (SWs). AsNoC is designed by arranging NI to connect with IP and then configuring the SW according to the target platform. Converting IP interfaces is one of the key issues in ASNoC design, unlike regular structured NoCs. This is because the interfaces vary in protocol and data width [31]. For example, the three AMBA protocols [35], advanced extensible interface (AXI), advanced high-performance bus (AHB), and APB, have similar but different characteristics to each other, which can be depicted as Venn diagram in FIGURE 8. In the figure, the relative complement sets should be managed by the ASNoC. We developed our own ASNoC based on the presented architecture in [31], that supports various types of IP interface conversions and designed very compact for low power and low cost embedded systems. The proposed MMU is then implemented in this ASNoC.

To design the MMNoC, the NI for the AXI master is first designed, which is shown in FIGURE 9. The AXI protocol has five channels including two address channels. Because

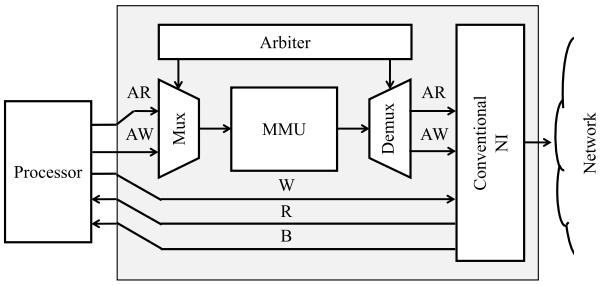

FIGURE 9. The proposed NI architecture that extends beyond the traditional NI designs.

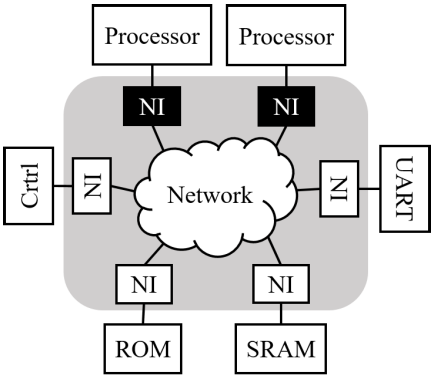

FIGURE 10. Hardware platform for verification. The proposed MMU is integrated into two black colored NI in the figure.

duplicating an MMU for each address channel causes large overhead, the NI is design to support the two channel arbitration so as to share a single MMU. For this purpose, we implement a logic with an arbiter, a mux, and a demux as described in the figure. The virtual address is translated to the physical address in the two channels through the MMU, and then the channels enters to the AXI inputted NI. Then the NIs for the other protocols (e.g., APB, AHP) are easily designed based on the presented AXI NI, because they may be implemented with only on channel. In addition, the proposed MMNoC design is a general solution for embedding MMU into NoC that is not limited to the presented ASNoC, in that the new NI design does not need to modify the original internal structure of NI.

The proposed MMU has the APB interface described in Section III-C, and it is connected to NoC as shown in FIGURE 3. Consequently, the processor can control the MMU using simple read/write memory operations that can be written in a high-level language instead of an assembly language. Namely, the simple read/write operations to the MMU control are independent of the processor architecture and can be made into application program interfaces (APIs) written in a high-level language. For this reason, software engineers can easily develop the system softwares.

IV. EXPERIMENTAL WORK

In order to verify the proposed MMNoC, we implemented a full system including a verification hardware platform as described in FIGURE 10, where the two black colored NI

TABLE 1. Resource consumption on Xilinx Artix-7 FPGA.

	LUTs		FFs	
MMU	85	(23%)	105	(16%)
Arbiter	5	(1%)	32	(5%)
Conv. NI	371	(100%)	669	(100%)
Entire NI	461	(124%)	806	(121%)
Conv. NoC	4460	(100%)	6484	(100%)
Proposed NoC	4640	(104%)	6758	(104%)

includes the proposed MMU. Two ORCA processors [36] based on RISC-V instruction set architecture were implemented for the dual processors in the platform. Note that the ORCA processor has no MMU. The platform was designed to have a boot ROM and 64kB SRAM. The Ctrl In FIGURE 10 has a responsibility to control the platform based on external JTAG signals, and the UART is one of the standard I/Os for the external interface. Since only the processors utilize the MMU, we setup the processor dedicated NIs to have the MMU. Each MMU has four PTEs, and the number of the address parts is set to eight.

To test the functional accuracy of the proposed MMNoC, we first synthesized the platform by using Xilinx Vivado [37] targeting Artix-7 FPGA, and the target operating frequency is 50Mhz. Note that conventional ULP processors have clock frequency ranges from tens of kHz to tens of MHz [4-12], so 50 MHz clock frequency is one of the fastest clock frequencies in the ULP processor. The resource consumption of the MMU and NI are reported in TABLE 1. The designed MMU consumes only 85 loop-up tables (LUTs) and 105 flip-flops (FFs), which cause 23% and 16% overhead in LUTs and FFs compared to the conventional NI (i.e., the conventional NI is the NI in the NoC presented in [31]). As a result, the resource consumption of the proposed MMNoC including two MMUs is increased by 4% in both LUTs and FFs.

The synthesized FPGA prototyping platform is shown in FIGURE 11. The USB connection in the left side is in charge of configuring Xilinx FPGA chip and linking the UART to the screen of a host machine. When the textitprintf function used in the C library send characters to the UART, the output from the FPGA is printed in the screen. The wires in the top-right corner is for the JTAG signals, which is connected to Ctrl in the platform. An OpenOCD program [38] running on the host controls the platform through the JTAG and sets up the multiprocessing.

At the same time, we programmed two testbenches to test multiprocessing capability of the platform. The two testbenches perform sorting 16 numbers (*sort*) and generating 96 prime numbers (*prime*). Binary codes for the two testbenches were loaded into 0x10000000 and 0x10004000 in SRAM. Then the first processor is set to execute the *sort*, while its MMU converts the 1kB address starting with 0x0 into 0x10000000. The second processor runs the *prime*, while its MMU converts the 1kB address from 0x0 into 0x10004000. FIGURE 12 shows the execution results of the two testbenches. For better readability, we use a *lock* so that

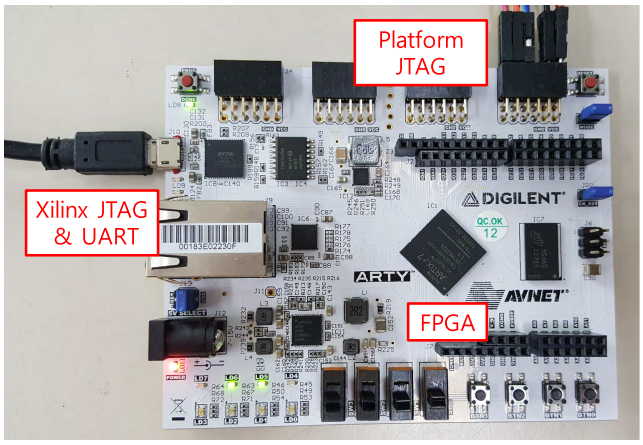

FIGURE 11. The prototyping platform on Xilinx Artix-7 FPGA.

```
003 005 007 00b 00d 011 013 017 01d 01f 025 029 02b 02f 035 03b
03d 043 047 049 04f 053 059 061 065 067 06b 06d 071 07f 083 089
08b 095 097 09d 0a3 0a7 0ad 0b3 0b5 0bf 0c1 0c5 0c7 0d3 0df 0e3
0e5 0e9 0ef 0f1 0fb 101 107 10d 10f 115 119 11b 125 133 137 139
13d 14b 151 15b 15d 161 167 16f 175 17b 17f 185 18d 191 199 1a3
1a5 1af 1b1 1b7 1bb 1c1 1c9 1cd 1cf 1d3 1df 1e7 1eb 1f3 1f7 1fd
** Finished computing 96 prime numbers.
* Original list :
  -> 22 5 67 98 45 32 101 99 73 10 7 120 1 55 21 16
* Sorting Algorithm : Selection(X), Bubble(O)
* Sorted list, in ascending order, is :
  -> 1 5 7 10 16 21 22 32 45 55 67 73 98 99 101 120
* 66 moves were made to sort this list
```

FIGURE 12. Multiprocessing screen output with two running testbenches.

the testbenches are executed sequentially. The results demonstrate the functional accuracy of the proposed MMNoC by confirming that the two programs are running correctly.

After verifying the functional correctness of the MMNoC, we performed evaluations of the MMNoC in terms of performance, area and power overheads induced by the MMNoC. Regarding the performance overhead of MMNoC, we investigated how long it would take to run a single non-multi-program on a platform with MMNoC compared to when running on a platform without MMNoC. To do that, we implemented two prototype platforms in the Artix-7 FPGAs. Both platforms have one RISC-V core, but one has MMNoC and the other has no MMU. Then, selected five benchmark programs, *Coremark*, *Sieve*, *Bubble sort*, *Fibonacci*, and *Sobel filter* were run in the two platforms. TABLE 2 reports the resulting execution times of the benchmarks. The execution time is increased by only about 7.5% on all benchmarks, leading to the conclusion that the multiprocessing capability of MMNoC can outweigh the small performance degradation.

In order to investigate the area and power overheads, we synthesized the prototyping platform in FIGURE 10 by using a state-of-the-art commercial CMOS technology, Samsung 28nm FD-SOI technology. TABLE 3 provides the resulting area and power overhead of the MMNoC, that are calculated based on the comparison between the two MMUs equipped MMNoC and the MMU-less NoC. The proposed MMNoC increases gate count only by 5% and power 3.5%

TABLE 2. Investigation results of the performance overhead of the MMNoC. $\text{Exec}_{\text{no\_MMU}}$, $\text{Exec}_{\text{MMNoC}}$, and $\text{Overhead}_{\text{perf.}}$ imply the benchmark execution time (μs) of the platform without MMU, with MMNoC, and the performance overhead of the MMNoC (%), respectively.

Benchmark	$\text{Exec}_{\text{no\_MMU}}$	$\text{Exec}_{\text{MMNoC}}$	$\text{Overhead}_{\text{perf.}}$
Coremark	91493736	98421447	7.57
Sieve	1355	1459	7.67
Bubble sort	474	510	7.59
Fibonacci	16474513	17705038	7.46
Sobel filter	12845787	13823595	7.61

TABLE 3. Area and power overhead of the MMNoC. The gate count (gate equivalent, GE) and power (mW) values are extracted from the synthesized platform in FIGURE 10. The overheads are calculated based on the platform with the two MMU equipped MMNoC.

	NoC	MMU (two MMUs)	Overhead
Gate count (GE)	22740	1182 (2364)	10.39%
Power (mW)	2.28	0.079 (0.16)	6.92%

compared to the MMU-less NoC. In other words, for the case of the target platform that has dual cores, the resulting area and power overheads of the MMNoC are about 10% and 7%. Compared to the synthesized microcoded controller proposed in [24] whereby two mini processors, a NI and a core interface are required and take 44k GE, the lightweight MMU in this paper only requires 1K (i.e., the whole NoC with six NIs and two MMUs even takes only 25k GE, still almost half of the microcoded controller in [24]).

V. CONCLUSION

This paper have introduced a new approach to MMU functionality in the lightweight embedded systems targeting low power and low cost. A novel architecture to embed MMU into NoC has been proposed, enabling multiprocessing in the lightweight embedded system platforms. The details of the MMU design and how it is integrated into NoC have been presented with specific AsNoC designs. We called the proposed MMU embedded NoC, MMNoC.

Since the MMNoC does not need to modify the original architecture of the NoC, the proposed method is easily applied to the various types of NoC. The MMNoC allows embedded system hardware engineers to leverage the existing lightweight processors (that normally do not have MMU) to build a target platform that supports multiprocessing. Furthermore, owing to the simple way to program a code for the MMU control in the MMNoC, embedded system software engineers can easily develop the system software.

A entire prototype platform with MMNoC has been implemented in synthesizable verilog RTL codes and synthesized with FPGA and Samsung 28nm FD-SOI technology. The FPGA synthesized results have verified the functional accuracy of the MMNoC and 16∼23% more resources requirements than a NI, which is only 4% of the entire NoC. The 28nm FD-SOI synthesized results have demonstrated that a MMU in the MMNoC takes only 1.1K GE and consumes 79μW, which leads us to conclude that the multiprocessing capability can outweigh the associated overhead.

REFERENCES

[1] M. O. Ojo, S. Giordano, G. Procissi, and I. N. Seitanidis, "A review of low-end, middle-end, and high-end IoT devices," *IEEE Access*, vol. 6, pp. 70528–70554, 2018.
[2] G. A. Akpakwu, B. J. Silva, G. P. Hancke, and A. M. Abu-Mahfouz, "A survey on 5G networks for the Internet of Things: Communication technologies and challenges," *IEEE Access*, vol. 6, pp. 3619–3647, 2018.
[3] A. Musaddiq, Y. B. Zikria, O. Hahm, H. Yu, A. K. Bashir, and S. W. Kim, "A survey on resource management in IoT operating systems," *IEEE Access*, vol. 6, pp. 8459–8482, 2018.
[4] K. Craig, Y. Shakhsheer, S. Arrabi, S. Khanna, J. Lach, and B. H. Calhoun, "A 32 b 90 nm processor implementing panoptic DVS achieving energy efficient operation from sub-threshold to high performance," *IEEE J. Solid-State Circuits*, vol. 49, no. 2, pp. 545–552, Feb. 2014.
[5] C. Wang, J. Zhou, L. Liao, J. Lan, J. Luo, X. Liu, and M. Je, "Near-threshold energy- and area-efficient reconfigurable DWPT/DWT processor for healthcare-monitoring applications," *IEEE Trans. Circuits Syst. II, Express Briefs*, vol. 62, no. 1, pp. 70–74, Jan. 2015.
[6] A. Roy, P. J. Grossmann, S. A. Vitale, and B. H. Calhoun, "A 1.3 μW, 5pJ/cycle sub-threshold MSP430 processor in 90nm xLP FDSOI for energy-efficient IoT applications," in *Proc. 17th Int. Symp. Qual. Electron. Design (ISQED)*, pp. 158–162, Mar. 2016.
[7] M. Gautschi, P. D. Schiavone, A. Traber, I. Loi, A. Pullini, D. Rossi, E. Flamand, F. K. Gürkaynak, and L. Benini, "Near-threshold RISC-V core with DSP extensions for scalable IoT endpoint devices," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 10, pp. 2700–2713, Oct. 2017.
[8] Y. Pu, C. Shi, G. Samson, D. Park, K. Easton, R. Beraha, A. Newham, M. Lin, V. Rangan, K. Chatha, D. Butterfield, and R. Attar, "A 9-mm^2 ultra-low-power highly integrated 28-nm CMOS SoC for Internet of things," *IEEE J. Solid-State Circuits*, vol. 53, no. 3, pp. 936–948, Mar. 2018.
[9] STMicroelectronics. *STM32L151C6: Ultra-Low-Power ARM Cortex-M3 MCU With 32 Kbytes Flash, 32 MHz CPU, USB*. Accessed: Jun. 17, 2019. [Online]. Available: https://www.st.com/en/microcontrollers/stm32l151c6.html
[10] M. Integrated. *MAX32626: Ultra-Low Power, High-Performance ARM Cortex-M4 with FPU-Based Microcontroller for Wearables*. Accessed: Jun. 17, 2019. [Online]. Available: https://www.maximintegrated.com/en/products/microcontrollers/MAX32626.html
[11] NXP. *K32W0x MCUs for Wireless IoT Applications*. Accessed: Jun. 17, 2019. [Online]. Available: https://www.nxp.com/docs/en/fact-sheet/K32W0XFS.pdf
[12] Samsung. *Bio-Processor*. Accessed: Jun. 17, 2019. [Online]. Available: https://www.samsung.com/semiconductor/products/bio-processor
[13] R. Kumar, E. Kohler, and M. Srivastava, "Harbor: Software-based memory protection for sensor nodes," in *Proc. 6th Int. Conf. Inf. Process. Sensor Netw.*, Apr. 2007, pp. 340–349.
[14] L. S. Bai, L. Yang, and R. P. Dick, "MEMMU: Memory expansion for MMU-less embedded systems," *ACM Trans. Embedded Comput. Syst.*, vol. 8, no. 3, p. 23, Apr. 2009.
[15] H.-P. Chang, Y.-T. Liu, and S.-S. Yang, "Surviving sensor node failures by MMU-less incremental checkpointing," *J. Syst. Softw.*, vol. 87, pp. 74–86, Jan. 2014.
[16] A. Y. Dogan, D. Atienza, A. Burg, I. Loi, and L. Benini, "Power/performance exploration of single-core and multi-core processor approaches for biomedical signal processing," in *Integrated Circuit and System Design. Power and Timing Modeling, Optimization, and Simulation*. Berlin, Germany: Springer, 2011, pp. 102–111.
[17] A. Y. Dogan, J. Constantin, M. Ruggiero, A. Burg, and D. Atienza, "Multi-core architecture design for ultra-low-power wearable health monitoring systems," in *Proc. Design, Automat. Test Eur. Conf. Exhib. (DATE)*, Mar. 2012, pp. 988–993.
[18] D. Rossi, F. Conti, A. Marongiu, A. Pullini, I. Loi, M. Gautschi, G. Tagliavini, A. Capotondi, P. Flatresse, and L. Benini, "PULP: A parallel ultra low power platform for next generation IoT applications," in *Proc. IEEE Hot Chips 27 Symp. (HCS)*, Aug. 2015, pp. 1–39.
[19] F. Montagna, A. Rahimi, S. Benatti, D. Rossi, and L. Benini, "PULP-HD: Accelerating brain-inspired high-dimensional computing on a parallel ultra-low power platform," in *Proc. 55th Annu. Design Automat. Conf.*, Jun. 2018, p. 111.

[20] B. Shi, B. Li, L. Cui, and L. Ouyang, "Vanguard: A cache-level sensitive file integrity monitoring system in virtual machine environment," *IEEE Access*, vol. 6, pp. 38567–38577, 2018.
[21] L. Zuolo, G. Miorandi, C. Zambelli, P. Olivo, and D. Bertozzi, "System interconnect extensions for fully transparent demand paging in low-cost MMU-less embedded systems," in *Proc. Int. Symp. Syst. Chip (SoC)*, Oct. 2013, pp. 1–6.
[22] M. Monchiero, G. Palermo, C. Silvano, and O. Villa, "Exploration of distributed shared memory architectures for NoC-based multiprocessors," in *Proc. Int. Conf. Embedded Comput. Syst., Archit., Modeling Simulation*, Jul. 2006, pp. 144–151.
[23] C. Man, X. Bin, Q. Fuming, S. Qingsong, C. Tianzhou, and Y. Like, "Distributed memory management units architecture for NoC-based CMPs," in *Proc. 10th IEEE Int. Conf. Comput. Inf. Technol.*, Jun./Jul. 2010, pp. 54–61.
[24] X. Chen, Z. Lu, A. Jantsch, and S. Chen, "Supporting distributed shared memory on multi-core Network-on-Chips using a dual microcoded controller," in *Proc. Design, Autom. Test Eur. Conf. Exhibit. (DATE)*, Mar. 2010, pp. 39–44.
[25] J. Porquet, A. Greiner, and C. Schwarz, "NoC-MPU: A secure architecture for flexible co-hosting on shared memory MPSoCs," in *Proc. Design, Autom. Test Eur.*, Mar. 2011, pp. 1–4.
[26] P. Vogel, A. Marongiu, and L. Benini, "Lightweight virtual memory support for many-core accelerators in heterogeneous embedded SoCs," in *Proc. Int. Conf. Hardw. Softw. Codesign Syst. Synth. (CODES+ISSS)*, Oct. 2015, pp. 45–54.
[27] S. Khan, S. Anjum, U. A. Gulzari, T. Umer, and B.-S. Kim, "Bandwidth-constrained multi-objective segmented brute-force algorithm for efficient mapping of embedded applications on NoC architecture," *IEEE Access*, vol. 6, pp. 11242–11254, 2018.
[28] H. Ali, U. U. Tariq, Y. Zheng, X. Zhai, and L. Liu, "Contention & energy-aware real-time task mapping on NoC based heterogeneous MPSoCs," *IEEE Access*, vol. 6, pp. 75110–75123, 2018.
[29] L. Chen, D. Zhu, M. Pedram, and T. M. Pinkston, "Power punch: Towards non-blocking power-gating of NoC routers," in *Proc. HPCA*, Feb. 2015, pp. 378–389.
[30] K. Han, J.-J. Lee, J. Lee, W. Lee, and M. Pedram, "TEI-NoC: Optimizing ultralow power NoCs exploiting the temperature effect inversion," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 37, no. 2, pp. 458–471, Feb. 2018.
[31] K. Han, J.-J. Lee, and W. Lee, "Converting interfaces on application-specific network-on-chips," *J. Semicond. Technol. Sci.*, vol. 17, no. 4, pp. 505–513, Aug. 2017.
[32] L. Benini, "Application specific NoC design," in *Proc. Conf. Design, Automat. Test Eur.*, Mar. 2006, pp. 491–495.
[33] Arteris. *FlexNoC*. Accessed: Jun. 17, 2019. [Online]. Available: http://www.arteris.com/flexnoc
[34] ARM. *NIC*. Accessed: Jun. 17, 2019. [Online]. Available: http://www.arm.com/products/system-ip/interconnect/corelink-nic-family.php
[35] ARM. *AMBA*. Accessed: Jun. 17, 2019. [Online]. Available: https://www.arm.com/products/silicon-ip-system/embedded-system-design/amba-specifications
[36] *RISC-V*. Accessed: Jun. 17, 2019. [Online]. Available: https://riscv.org
[37] Xilinx. *Vivado 2016.4*. Accessed: Jun. 17, 2019. [Online]. Available: https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vivado-design-tools/2016-4.html
[38] *OpenOCD*. Accessed: Jun. 17, 2019. [Online]. Available: https://openocd.org

HYEONGUK JANG received the B.S. and M.S. degrees in electrical engineering from Gyeongsang National University, Jinju, South Korea in 2013 and 2015. Since 2015, he has been pursuing the Ph.D. degree with the University of Science and Technology. He has also been with the SoC Design Research Group, Electronics and Telecommunications Research Institute. His research interests include network-on-chip and system software in embedded systems.

KYUSEUNG HAN (M'18) received the B.S. and Ph.D. degrees in electrical engineering and computer science from Seoul National University (SNU), Seoul, South Korea, in 2008 and 2013. At SNU, he researched on computer architecture and design automation. Since 2014, he has been with the Electronics and Telecommunications Research Institute (ETRI), Daejeon, South Korea, and he currently belongs to the SoC Design Research Group as a Senior Researcher. His current research interests include reconfigurable architecture, network-on-chip, and ultra-low power techniques in embedded systems.

SUKHO LEE received the Ph.D. degree in information communications engineering from Chungnam National University, Daejeon, South Korea, in 2010. He is currently a Principal Researcher with the SoC Design Research Group, Electronics and Telecommunications Research Institute, Daejeon, South Korea. His current research interests include ultra-low power system-on-chip design, embedded system design, video codec design, and video image processing.

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, Electronics and Telecommunications Research Institute, and a Professor with the Department of ICT, University of Science and Technology. His research interests include processor and compiler designs in ultra-low power embedded systems.

WOOJOO LEE (M'15) received the B.S. degree in electrical engineering from Seoul National University, Seoul, South Korea, in 2007, and the M.S. and Ph.D. degrees in electrical engineering from the University of Southern California, Los Angeles, CA, in 2010 and 2015, respectively. From 2015 to 2016, he was with SoC Design Research Group, Electronics and Telecommunications Research Institute, as a Senior Researcher, and from 2017 to 2018 with the Department of Electrical Engineering, Myongji University, as an Assistant Professor. He is currently an Assistant Professor with the School of Electrical and Electronics Engineering, Chung-Ang University, Seoul, South Korea. His research interest includes ultra-low power VLSI and SoC designs, embedded system designs, and system-level power and thermal management.