Nanotechnology Perceptions
ISSN 1660-6795
www.nano-ntp.com

An Enhanced Design Of Mmu In Network On Chip Based Multiprocessors For Performance Improvement

Debasis Behera[1*], Suvendu Narayan Mishra[2]

[1]Department of Electrical and Electronics Engineering, C.V.Raman Global University, Bhubaneswar, Odisha, India , 752054*
Department of Electronics and Telecommunication Engineering,Veer Surendra Sai University of technology, Burla, Odisha, India, 768018
[2]Department of Electronics and Telecommunication Engineering,Veer Surendra Sai University of technology, Burla, Odisha, India, 768018

The demand for the development of lightweight embedded systems has been growing extensively with the increase in the development of the IoT era. Past decades have seen the development of ultra-low power (ULP) processors in the design of lightweight embedded systems. However, these systems lacked the ability to perform multiple operations without affecting the performance speed of the processors. The performance of the processor was deteriorated due to a large number of memory access requests. In order to overcome this issue, this research proposes a robust architecture which involves the embedding of MMUs in a Network-On-Chip (NoC). The proposed NoC architecture supports the multiprocessing capacity of the embedded systems without changing the design of the multi-core processors. The paper provides a detailed description of the processor design along with router logic and network interface components. The design was synthesized with FPGA platform and was simulated using a Vivado 2017.2 tool. Results show that the proposed approach achieved a lower overhead and reduces the power consumption significantly in the FPGA based MMU embedded NoC.

Keywords*:* Lightweight embedded systems, Memory Management Units, Network-On-Chip, Multiprocessors, Power consumption

1. Introduction

With the recent developments in the area of Internet-of-Things (IoT), there has been a significant transformation in the field of embedded systems and one of the important aspects related to this is the increase in the demand for lightweight and small embedded systems. These systems have seen a sharp rise in their demand because of their simple design, energy efficiency, and reliability (Ojo et al., 2018) [1] (Musaddiq et al., 2018) [2]. The design of lightweight embedded systems incorporates a low-power, simple and narrow processor known as lightweight processors. These processors are most desirable for the development of small embedded systems wherein multiple lightweight processors are integrated to increase the energy efficiency
with minimized power utilization and reduced design cost (Raza & Azeemuddin, 2014) [3]. In line with this trend, lightweight processors are being widely researched by various researchers

and embedded system developers [4-7]. One of the prominent requirements of these processors is to expand the functionality of the lightweight embedded systems to work effectively with multiple processors. The objective is to make the lightweight embedded systems capable of running multiple programs alternatively or simultaneously i.e, the embedded systems are programmed to collect and evaluate different types of data from various sources. The multiprocessing functionality of the embedded systems is gaining more prominence since it enables the system to carry out multiple lightweight data processing for different data types (Bai et al., 2009) [8] (Chang et al., 2014) [9]. However, lightweight embedded systems require a potential support to perform multiprocessing. In this context, memory management units (MMUs) are deployed in the embedded system to support multiprocessing in a lightweight embedded system. The MMUs are mainly responsible for storing and directly accessing the memory of the transactions carried out by the embedded systems. However, due to high power consumption, increased area and cost of the design it is quite challenging to implement MMUs directly into the embedded systems (Shalan & Mooney, 2002) [10]. In order to solve this problem, this research explored an effective solution of detaching the MMUs from the processors and implementing them on other platforms of the embedded systems such as network-on-chip (NoC) (Monchiero et al., 2007) [11]. In this research, the MMUs are placed on the NoC which is a common hardware intellectual property (IP) for the embedded systems to connect with advanced embedded system platforms (Pu et al., 2018) [12] (Khan et al., 2017) [13] (Ali et al., 2018) [14]. The MMUs are embedded into NoCs for improving the performance of the lightweight embedded systems with respect to low power consumption and low design cost. The architecture of the MMU is illustrated in figure 1.1 (Behera & Jena, 2020) [15]

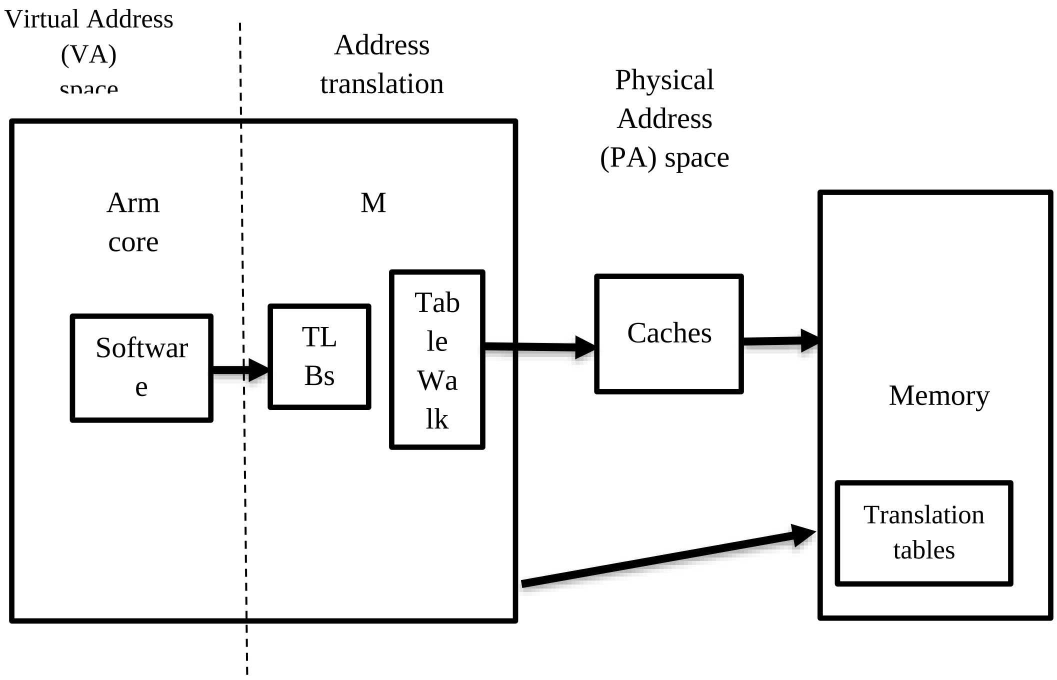

Figure 1.1 Fundamental architecture of the memory management unit

1.1 MMUs in NoC-based multiprocessors

Network on Chip (NoC) has become one of the prominent design requirements for communication-based System-On-Chip (SOC) applications. The reliability of NoC has made it a desirable contender in the development of future Chip Multiprocessor (CMP) (Abad et al., 2007) [16]. The basic architecture of the NoC consists of a m x n mesh composed of multiple design system components and switches wherein the system components communicate with each other using data packets (Kumar et al., 2002) [17]. A system component can either be a memory unit, processing element (PE) or any other custom-built on-chip hardware. The architecture of NoC allows the integration of a large-scale heterogeneous devices on a single chip microprocessor and the number of hardware components can vary up to tens or hundreds of processing elements meshed through NoC for forming a robust central processing unit (CPU). In general, the memory system in processors is central and bus-based. With the increase in the number of processing elements on a single chip it becomes difficult for the memory system to achieve desired performance. Every on-chip processing element requires a specific memory reference and multiple PEs require a higher number of memory references. The increased number of memory references affects the synchronization of the controller and deteriorates the overall efficiency of the processor (Nesbit et al., 2007) [18]. The distributed architecture of the NoC based multiprocessor demands a potential memory controller to cope with larger numbers of PEs on a single chip. In this context, implementation of MMU on-chip gains a major advantage. Since MMU is considered as one of the important components for the memory references, implementation of a MMU in the distributed NoC architecture boosts the performance of the embedded systems in terms of multiprocessing capability.

MMU can access the memory directly and hence it is common phenomenon that an optimized MMU architecture can be used to overcome the memory reference problem. In this research, a distributed architecture of MMUs for NoC-based multiprocessors is considered which incorporates the advantages of efficient memory access, synchronization, efficient on-chip memory organization and fair shared memory access. The distributed architecture requires a greater number of MMUs on the NoC platform for managing memory access requests. Unlike a single on-chip memory unit, it prevents excess and redundant communication by limiting the amount of memory access requests sent to each MMU on the NoC. With a suitable number of MMUs placed appropriately on the chip, helps to reduce the memory bandwidth requirement and communication traffic in the NoC platform.
The contributions of this research are as follows: First, this paper provides an efficient memory management architecture for NoC-based multiprocessors for reducing power consumption and design cost. Second, it is shown that the proposed design overcomes the problem of memory reference problem and exhibits desired performance for memory access. Third, this paper implemented a distributed memory organization for NoC-based processors for optimizing the performance of application-specific memory access. Finally, it is shown that the proposed architecture improves the memory handling capacity of the multiprocessors and decreases the amount and distance of barrier communication.

The rest of the paper is structured as follows: Section 2 provides a comprehensive analysis of the existing literary works related to the design of MMUs. Section 3 discusses the details of the proposed system architecture which includes the design of an embedded MMU, memory subsystem, a network interface and design of NoC-based processors. Section 4 presents the results of the experimental setups and results of the simulation analysis. Section 5 provides the conclusion of this research work.

2. Related Works

This research mainly focussed on the network-on-chip (NoC) which is fundamental intellectual property in embedded systems. Furthermore, the attention was given to existing literary works carried out with respect to the placement of MMUs in network-on-chip (NoC) (Man et al., 2010) [19] (Chen et al., 2010) [20] (Chen et al., 2015) [21] (Tatas et al., 2014) [22]. An application-specific memory management unit for FPGA-SoCs was proposed by (Goebel et al., 2018) [23]. In this work, the proposed MMU enabled effective memory utilization by the FPGA. The proposed system architecture was integrated into a FPGA-SoC which simplified the implementation of hardware and software-based code design on SoC. Performance evaluation showed that the implementation of MMU to analyse the patterns of memory access boosted the performance of memory management of the FPGA-SoC. Additionally, the system configuration reduced the overhead without affecting the bandwidth requirements. (Siast et al., 2019) [24] identified the fundamental requirements for NoCs and the basic functionalities of FPGAs. This work proposed an FPGA-based NoC known as RingNet. The proposed RingNet incorporates a unique property of communicating through a centralized memory unit which aims to prevent network traffic and to reduce the network buffer. One of the preliminary objectives of the proposed RingNet design is to make an effective use of FPGA resources. Especially, network buffers are deployed in distributed RAM placed inside FPGA platforms and the virtual cut-through technique was employed for performing effective switching in FPGA. It can be inferred from the simulation results that the proposed RingNet achieved improved throughput, fair network access and predictable latency. The adaptability of the RingNet approach was validated using the results for different FPGAs from different platforms namely Lattice, Intel and Xilinx. It was also shown that the NoC implementation using RingNet required a lesser number of resources and it worked effectively for higher clock frequencies. A unique approach for memory management unit was discussed by (Gordon-Ross et al., 2019) [25]. This work presented a One-Cycle first in first out (FIFO) buffer for MMUs in manycore systems. The study also discussed the inter-core data transfer in maycore systems. The proposed work was designed to minimize the overhead due to various hardware components and to prevent the latency delays. This objective was achieved by employing both the rising and falling clock edges for reading and writing operation. This feature made this design appropriate for handling increased processing element (PE) usage by expanding the memory bandwidth in SoC networks. Results showed that the proposed design has the ability to work 5 times faster than existing techniques utilizing the same supply voltage and the operational speed can be 44 times faster compared to other approaches with an increase in the supply voltage by 2.5 times. The overall power utilized by the proposed design was 7.8 mW with a total transistor count of 34,470. (Raparti & Pasricha, 2019) [26] discussed the implementation of an approximate NoC and memory controller architectures for GPGPU (General Purpose Graphics Processing Units) accelerators. Existing works showed that high interconnect bandwidth is important to achieve desired efficiency in various core GPGPU architectures which are used to perform data parallel operations. The parallel threads that run on the shader cores produce a huge volume of reading requests to the main memory unit because of the restricted data size at the shader cores. This results in the increase of volume of the reply data from the DRAM which introduces a bottleneck at memory controllers (MCs) which in turn sends reply data to the controllers through the network-on-chip (NoC). In order to cope with such a huge volume of data, it is essential to deploy an intelligent memory scheduling and advanced NoC system architecture. In order to overcome the problem of memory bottleneck in GPGPUs, this study implemented an advanced AMC which minimizes the latency of the DRAM by statistically leveraging the buffer locality and memory request scheduling. This significantly reduces the quantity of reply data packets that are sent through NoC thereby preventing the network congestion. In addition to this, the study also aimed to achieve high throughput and to reduce the energy consumption due to excessive communication in GPGPUs, this study proposed a low power, approximate NoC architecture (Dapper) which elevates the effective utilization of the available network bandwidth. Results from the experimental analysis showed that the combined architecture of Dapper and AMC together improves the throughput of NoCs up to 21% and decrease the latency of NoC by up to 45.5% and the power utilized by the NoC and the memory controller was reduced by up to 38.3% with a very minimum effect on the accuracy of the output compared to other existing approximate NoC/MC architectures. (Kumar & Reddy, 2020) [27] proposed a novel approach with advanced memory management unit architecture for optimizing the performance of NoC. The research proposed a random-access memory (RAM) which connects the crossbar switch and the input port. The proposed design was simulated using the Xilinx 14.7 ISE platform and was implemented on a Vertex-6 FPGA device. The design utilized the benefit of empty/full flags and the distributed MMU architecture was more compatible for handling large scale memory access. The performance of the proposed work was compared with other works and it was proven that this work was more effective and reliable compared to other existing approaches. The experimental framework proved that the advanced distributed MMU based 3-D NoC improved the delay and throughput of the processor compared to existing techniques. (Jang et al., 2019) [28] proposed a novel architecture wherein the MMU was embedded into a network-on-chip (NoC). The proposed architecture allows the NoC to optimize the performance of the MMU in a multiprocessor without changing its design. This allows the embedded system developers to exploit the advantages of existing ultra-light processors and design the embedded systems which support multiprocessing. The proposed research provided a brief overview of the design of MMU-embedded NoC (MMNoC) which includes a dual RISC-V processor along with the MMNoC. The proposed system design was implemented on a FPGA platform for verifying the functional accuracy, efficiency, power overhead and area of the proposed MMNoC.

3. Proposed design methodology:

The research aims to achieve a low power and low-cost lightweight embedded systems by embedding memory management units into Network-on-Chip (NoC). The design process is divided into three stages namely: design of NoC-based processors, a memory management unit and design of MMNoC. The research mainly focuses on achieving efficient memory access and synchronization like, efficient on-chip memory organization, fair shared memory access, and efficient many-core synchronization. The barrier synchronization is used for synchronizing the execution of parallel processor cores. The proposed design implements a distributed architecture for memory organization for NoC-based processors. Furthermore, a special-purpose memory organization was designed in order to optimize the performance of application-specific memory access. As a part of the target application, a Fast Fourier Transform (FFT) and proposed multi-bank data memory specialized for FFT computation. The detailed implementation process is discussed in further sections.

3.1 Memory Management system

Memory management system (MMS) is an important module which effectively manages the primary memory (both static and dynamic RAM) by effectively monitoring the memory allocation process. In general, the design of the operating system in the embedded systems incorporates a memory management system which is responsible for managing the memory access and requests (Musaddiq et al., 2018) [29] (Shi et al., 2018) [30]. However, lightweight embedded systems operate without an operating system and the memory management is usually performed by the developers themselves. The address space reserved for the program or a process depends on the size and capacity of the embedded hardware platforms. For executing a program, it is essential to provide a virtual address with a physical address during execution. This is performed using demand paging technique. It is challenging to manage the inconsistent and random size of data since it increases the complexity of the system design. Hence the size of the data is fixed, and a fixed size is referred to as a page. The conversion of virtual address into a physical address is shown in figure 3.1.

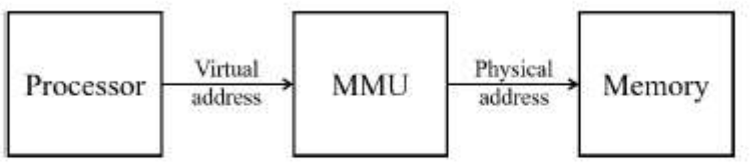

Figure 3.1 Conversion of virtual address into physical address in a hardware module

For converting the virtual address, MMUs employ a page table as shown in figure 3.1. MMUs are deployed in between the processor and the memory unit. The page table is always stored in RAM and the MMU stores the recent transactions in the page table.

3.2 MMU Embedded NoC

Lightweight embedded systems are designed to perform various programs and are expected to support multiprocessing. In other words, a lightweight embedded system must possess multiprocessing abilities making MMUs an essential component for these systems. This can be done by developing an ULP lightweight processor with a MMU. However, from a technical point of view, it is impractical to develop a new processor since it takes a lot of resources with an increase in the design effort and implementation cost. Hence, this research integrates the functionalities of a MMU into an embedded system without affecting the design of the processor. In order to perform this, the study implements a NoC and embeds a MMU into NoC. NoC offers substantial support for carrying out multipurpose communication on embedded system platforms (Pu et al., 2018) [31]. NoCs are used widely in the design of lightweight embedded systems because of its ability to overcome the drawbacks of traditional bus-based systems (Chen et al., 2015) [32] (Han et al., 2017) [33] (Han et al., 2017) [34]. Based on the analysis that a processor in the embedded system with NoC communicates with other IPs in the network only through a reliable network interface, this work came up with an idea of implementing a MMU in an NI as illustrated in figure 3.2. This approach allows to integrate the functionalities of an MMU into an NoC irrespective of the processor types in the embedded system platforms.

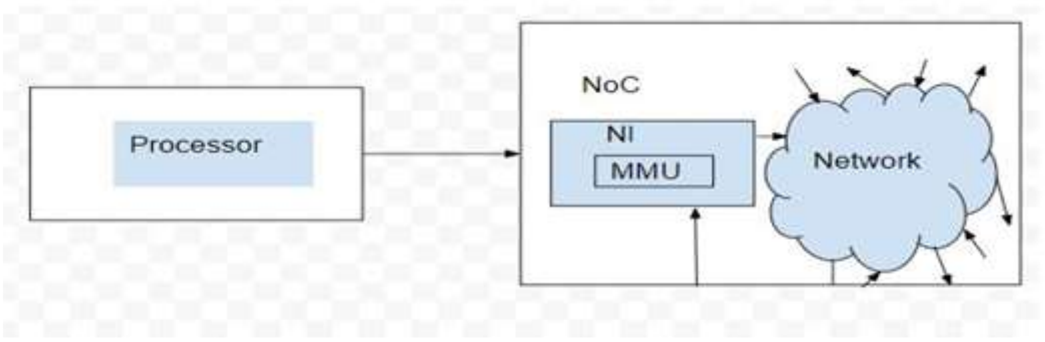

Figure 3.2 Implementation of an MMU on an Network Interface based processor

In this research a module called router logic (bidirectional ring network-on-chip (NoC) router) was used as the building block of a NoC for a multiprocessor. This router logic consists of two virtual channels namely fifo_buffer 0 and fifo_buffer 1 for each physical channel for preventing the deadlock related to routing in the ring. FIFO buffers are employed for storing the incoming data packets from the input ports. The router module determines the structure and interconnections of the router logic. In this research, 4 router logics are connected in the form of a ring with their respective input nodes and output nodes.
The Network Interface Component (NIC) is implemented for creating a path between the processor and the underlying ring network. When viewed from the sender's point of view, the data packets are sent through a single network output channel buffer in the NIC where the sending packets are written. While on the receiver side, the data packets are received through a single network input channel buffer in the NIC, from which the incoming packets are read. It is essential to map the buffers for both input and output network channels along with their status registers and the memory address. This mapping enables the processors to access the memory using basic storing/loading instruction sets. The interface provided by the NIC is similar to that of the memory interface and hence the NIC interface is not affected by other network details such as signal polarity, handshaking etc. Running multiple programs using embedded systems is considered to be crucial in modern communication systems and the multiprocessing capability of the embedded systems are supported by MMUs. For multiprocessing operations, every component in this system requires a specific memory space for performing multiple operations in parallel and this is carried out using MMUs. This study uses a RISC-V processor and instead of connecting the processor directly into Network-on-chip, the MMU is integrated on a Network Interface Component (NIC) and the module is termed as NIC-MMU and a Wishbone Interface was implemented for establishing a

communication between the RISC-V processor and MMU for sending and receiving instruction memory and data memory. Generally, most of the CPUs possess a structured level for storing the cache with a unique instruction set and data-based caches. The Wishbone interface used in this research is used to track the components of an integrated circuit and to allow the interconnection between different components in the circuit. The objective is to connect different cores inside a NoC. The communication based MMU module is the main module whereas all the other modules are instantiated, and this design is validated through the simulation analysis. Furthermore, a uniform and system-level modeling framework is required, wherein performance, power, and area models are simultaneously integrated to enable design space exploration and highly specific optimization of NoC architectures for throughput efficiency, power consumption, and area consumption. A framework that would offer insightful information that would facilitate design cross-examination and simplify NoC design optimization. The schematic of the proposed system architecture is illustrated in figure 3.3.

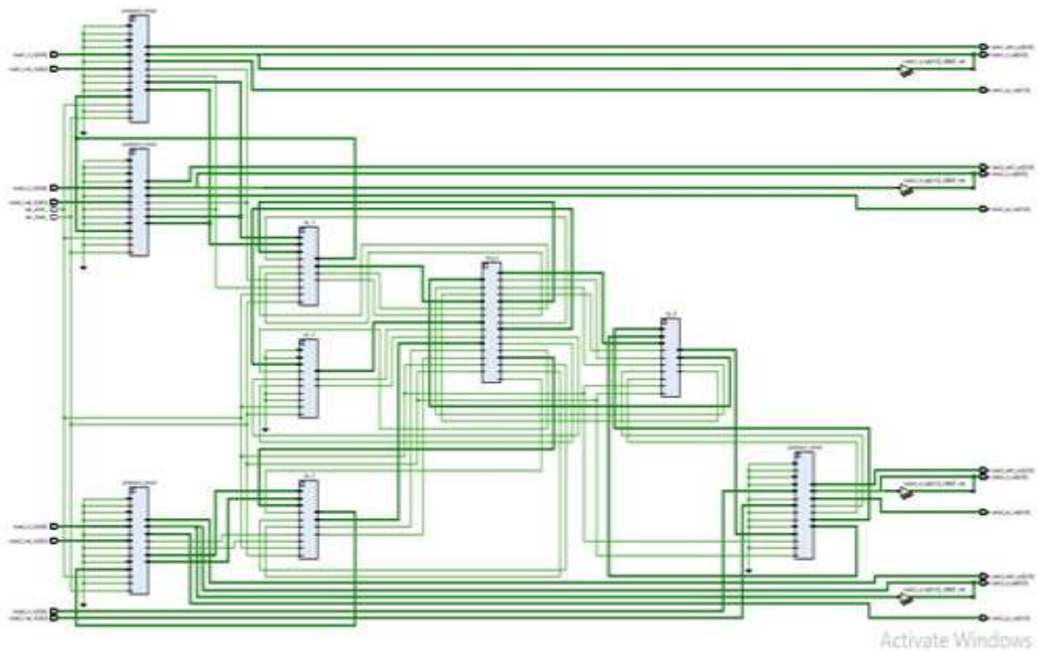

Figure 3.3 Schematic of the proposed system architecture with all modules

The schematic represents all modules considered in this research. The blocks represent the blocks of 4 RISC-V processors which are interfaced with 4 blocks of MMU and are connected to NIC (Network Interface Component). The MMU is embedded on NoC (ring router) and the instruction memory and data memory are sent through one of the RISC-V processors which is connected to one of the nodes (e.g. node 0) and this instruction memory and data memory are transferred to other nodes in such a way that the obtained address and data from instruction memory and data memory are stored in Memory Management Unit (MMU) and then it is accessed through NIC (Network Interface Component) through routers and through other nodes. These are used to link memory and I/O devices to several processors. Multiple types of buses exist, including I/O buses for connecting many devices and local buses for establishing fast links between processors and memory.

3.3 Design of RISC-V processor

The module p_CPU consists of a RISC-V processor which has a four staged pipeline for executing different instruction sets and the schematic of the same is shown in figure 3.4.

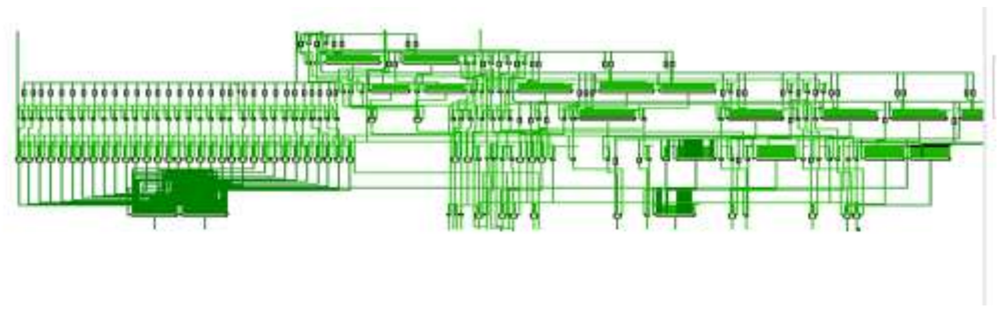

Figure 3.4 Schematic of the RISC-V processor

The processor consists of 12 input and output ports (I/0 ports) with 1463 nets and 1235 cells. Since the load or store instructions make use of only immediate address specifiers, the memory operations are carried out in the stage similar to that of the ALU (Arithmetic Logic Units) functions and hence we use a 4-stage pipeline structure. The 4 stages of the pipeline are:
Pipeline Stage 1. Instruction Fetch (IF)
Pipeline Stage 2: Instruction Decode and Register Fetch (ID)
Pipeline Stage 3: Execution or Memory Access (EX/MEM)
Pipeline Stage 4: Write Back (WB)
The modules such as P_ALU, P_DIV, and P_MUL executes arithmetic and logical functions inside the processor module. The system considers a synchronous active high RESET which is used to clear the data stored in the stage registers. Inside the memory unit, a program counter is implemented with 32 bits which increments the clock value by 4 every time and has a value of 32`h0000_0000 at reset. The program counter must have a positive-edge triggered clock with a synchronous reset. An asynchronous instruction memory consisting of 256 words i.e., 32 bits/word is enough for carrying out sample simulation analysis and it can be considered that the processor supports certain operations only at a particular time (I.e., either READ or WRITE operations). An instruction decoding unit is provided for decoding the fetched instruction and to produce suitable control signals for carrying out accurate and relevant execution of the fetched instruction. The proposed MMU has 3 ports where 2 ports are reserved for performing READ operations and one port is used for executing WRITE instructions. The MMU consists of a fundamental register file which incorporates 32 numbers of 64-bit registers. The MMU performs an asynchronous reading while the WRITE operation is synchronous. Every port in the memory unit consists of a 5bit address and 64-bit data to read/write the contents into a specific register. When the processor receives RESET instruction, the contents present in the register are cleared completely. The register 0 must be hard-wired to 64'h0000_0000_0000_0000 and this register performs READ ONLY i.e., the register 0 cannot be written into the processor memory. The WRITE port must also receive a write enable signal

(wrEn) for controlling the write operation of the MMU. The ports considered for reading will only read the contents from the register using the asynchronous 5-bit address specifier.

3.4 Router Design

The router logic used in this research is the bidirectional ring network-on-chip (NoC) router which plays an important role in the router design and is considered as the building block for the NoC based multiprocessor. The design of the router consists of 399 input and output ports with 1709 nets and 114 cells. Every data packet in the network must be routed using a ring of routers and has a fixed length of 64 bits. Therefore, the size of the data packet is maintained equal with the flit size and the channel width. The router sends an entire packet from the output buffer of one router channel to the input buffer of the next router channel within one complete cycle. Considering the fact that there is no contention, an entire packet can be sent from an input channel buffer to an output channel buffer within a router in one cycle. For routing purposes, a header information is used which is composed of 32 bits of the packet. The remaining bits are used for executing the polarity of the virtual channel and direction. The router design is done so that the module can prioritize the controlling of the instruction execution in the processor. A rotating arbitration technique is employed for performing arbitration among multiple requesters for each set of the output virtual channels. Once the request is granted, the output controller must validate the sender and the priority scheme must be reversed in order to grant the priority to the other requestor.

The channel buffer is designed for providing the buffer to perform queue operations. In this research, the module fifo_buffer is used as a channel buffer and it is implemented as a 64-bit register with a synchronous write and read port. For input channel buffers, when the corresponding signal is flagged, the corresponding data is latched directly into the respective input channel buffer at the rising clk edge without requiring any computation. Decoding the address registers and output channel buffer requesting will be performed in the next cycle.

3.5 Design of a Network Interfaced MMU

The module of a NIC-MMU acts as an interface between the MMU and the NoC-based multiprocessor. The design consists of 297 input and output ports with 508 nets and 22 cells. NIC is a synchronous module with two channels namely network input and network output channel. The network output channel is used for sending the data packets from the processor to the router. The 64-bit data packets from the MMU are sent to the router and are injected into the d_in port and are delivered to the routers through the net_do port. Whereas the network input channel sends the packet from the router to the MMU. The 64-bit packets from the router are injected into the net_di port and are delivered to the MMU via the d_out port. The interface between the NIC and the MMU is very similar to the Router Design. Each channel has two control signals (s – sending, r – ready) for handshaking. The polarity signal from the connected router enables the NIC to inject the packets into the correct virtual channel according to the VC bit of the packet.

When the processing unit stores a data packet in its MMU for forwarding the packet to the NIC_MMU, it must load the value of the network output channel status register into the MMU, and the output channel receives the packet only when the value of the status register is 0. If the network status register shows 1 then the processor does not store the packet which indicates that the channel is occupied. Similarly, when the CPU wants to receive a packet, it must load the input network status register into the MMU, and the operation is similar as that of the output channel. It can determine the optimal number of ports for the input/output of network interfaces in an NoC .

4. Simulation Results

The performance of the proposed NoC-based multiprocessor was verified by simulating the design using a Vivado 2017.2 simulation tool. The schematic of the Network Interface MMU is illustrated in figure 4.1.

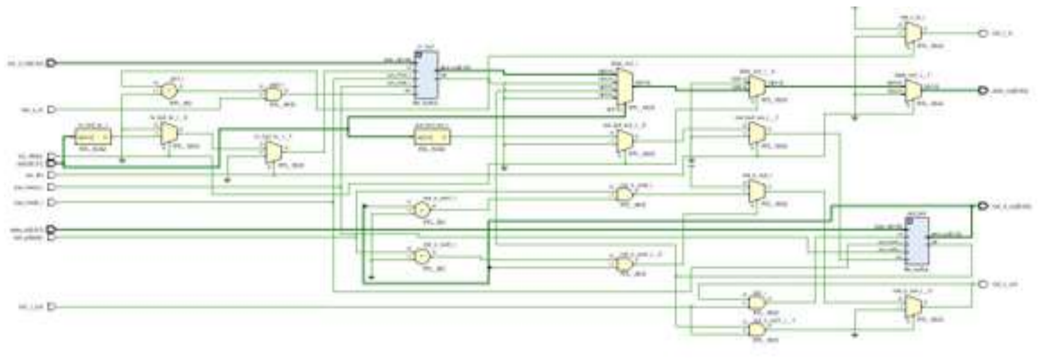

Figure 4.1 Schematic of a Network Interface MMU

The resource utilization and power consumption of FPGA is shown in table 1

Table 1 Resource utilization and Power consumption of FPGA

	NoC	MMU (4 MMUs)
FPGA power utilization (mW)	1.9	0.07
LUT’s (Look up Tables) used	3705	136
Number of Flip Flops used	4940	150

The operating frequency considered for the simulation analysis was 100MHz and the FPGA uses 36 Kb Block RAMs. Each 36Kb Block RAM contains two independently controlled 18 Kb RAMs. The 36 Kb blocks can be connected in a cascaded form to enable a deeper and wider memory implementation, with a minimal timing penalty. Different test cases are written for each module in the simulation part of the design and the verified results can be observed from the waveform given in below figure 4.2.

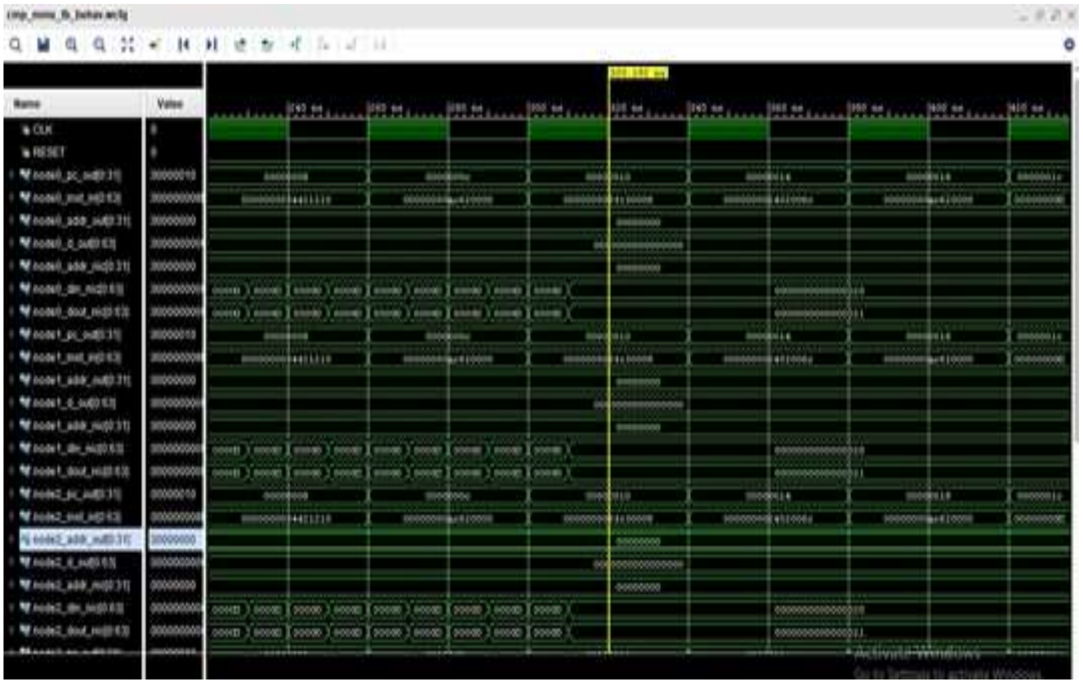

Figure 4.2 Waveforms for different test cases for different module

Table 2. comparison table of existing method

Reference		**NOC**
[32]	ConvOpt-PG	0.69
[14]	MP3	0.971
[33]	ULP NoCs,	0.36
[28]	Gate count	22740

The comparison between our method, NOC ConvOpt-PG, and the current approaches, MP3 and ULP NoCs, is shown in Figure 4.3. The findings unequivocally demonstrate that NOC ConvOpt-PG outperforms other optimizations like MP3 (0.971) and ULP NoCs with a substantial difference (0.69). further demonstrates that NOC ConvOpt-PG consumes just 22.740 gates, which is a strong performance in terms of hardware implementation costs. This comparison shows that the suggested approach is effective for NoC design and may be used to achieve good performance with the least amount of hardware. In terms of NoC design optimization, NOC proposed method shows promise because it works better than cutting-edge methods.

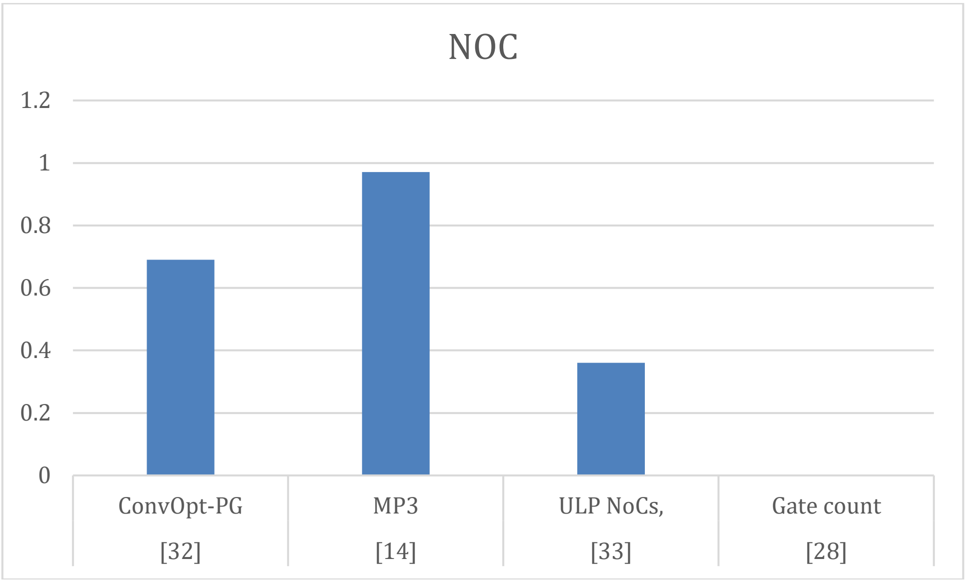

Figure 4.3 comparison of existing method

5. Conclusion

This paper presented an efficient approach for NoC based multiprocessors with reduced power consumption and design cost. The preliminary objective of this research was to design a lightweight embedded system by embedding memory management units (MMUs) into the NoC. The NoC enabled the multiprocessing functionality in the proposed lightweight embedded system. A brief description of the design considerations for the MMU has been presented in the paper along with its integration into the NoC was discussed with detailed router design. The proposed MMU embedded NoC does not affect the processor design and hence this design can be employed for analysing different types of NoC. The MMU based NoC enables the developers and engineers to transform the design of existing lightweight processors which do not possess an MMU to perform multiprocessing. The proposed design was simulated using A Vivado tool and was synthesized using a FPGA platform. The simulation results show that the proposed design approach of the MMU consumes 1.9 mW of power while the power consumption was reduced to 0.07mW after deployment of MMUs into NoC. This validates the effectiveness of the proposed approach with respect to its multiprocessing capability and its ability to reduce the hardware overhead.

References

1. Ojo, M. O., Giordano, S., Procissi, G., & Seitanidis, I. N. (2018). A review of low-end, middle-end, and high-end IoT devices. IEEE Access, 6, 70528-70554.
2. Musaddiq, A., Zikria, Y. B., Hahm, O., Yu, H., Bashir, A. K., & Kim, S. W. (2018). A survey on resource management in IoT operating systems. IEEE Access, 6, 8459-8482.
3. Raza, M. A., & Azeemuddin, S. (2014, January). Multiprocessing on FPGA using light weight processor. In 2014 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT) (pp. 1-6). IEEE.

4. 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
5. 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
6. NXP. K32W0x MCUs for Wireless IoT Applications. Accessed: Jun. 17, 2019. [Online]. Available: https://www.nxp.com/docs/en/factsheet/ K32W0XFS.pdf
7. Samsung. Bio-Processor. Accessed: Jun. 17, 2019. [Online]. Available: https://www.samsung.com/semiconductor/products/bio-processor
8. Bai, L. S., Yang, L., & Dick, R. P. (2009). MEMMU: Memory expansion for MMU-less embedded systems. ACM Transactions on Embedded Computing Systems (TECS), 8(3), 1-33.
9. Chang, H. P., Liu, Y. T., & Yang, S. S. (2014). Surviving sensor node failures by MMU-less incremental checkpointing. Journal of Systems and Software, 87, 74-86.
10. Shalan, M., & Mooney III, V. J. (2002, May). Hardware support for real-time embedded multiprocessor system-on-a-chip memory management. In Proceedings of the tenth international symposium on Hardware/software codesign (pp. 79-84).
11. Monchiero, M., Palermo, G., Silvano, C., & Villa, O. (2007). Exploration of distributed shared memory architectures for NoC-based multiprocessors. Journal of Systems Architecture, 53(10), 719-732.
12. Pu, Y., Shi, C., Samson, G., Park, D., Easton, K., Beraha, R., ... & Attar, R. (2018). A 9-mm 2 ultra-low-power highly integrated 28-nm CMOS SoC for Internet of Things. IEEE Journal of Solid-State Circuits, 53(3), 936-948.
13. Khan, S., Anjum, S., Gulzari, U. A., Umer, T., & Kim, B. S. (2017). Bandwidth-constrained multi-objective segmented brute-force algorithm for efficient mapping of embedded applications on NoC architecture. IEEE Access, 6, 11242-11254.
14. Ali, H., Tariq, U. U., Zheng, Y., Zhai, X., & Liu, L. (2018). Contention & energy-aware real-time task mapping on noc based heterogeneous mpsocs. IEEE Access, 6, 75110-75123.
15. Behera, D., & Jena, U. R. (2020, July). Detailed review on embedded MMU and their performance analysis on test benches. In 2020 International Conference on Computational Intelligence for Smart Power System and Sustainable Energy (CISPSSE) (pp. 1-6). IEEE.
16. Abad, P., Puente, V., Gregorio, J. A., & Prieto, P. (2007, June). Rotary router: an efficient architecture for CMP interconnection networks. In Proceedings of the 34th annual international symposium on Computer architecture (pp. 116-125).
17. Kumar, S., Jantsch, A., Soininen, J. P., Forsell, M., Millberg, M., Oberg, J., ... & Hemani, A. (2002, April). A network on chip architecture and design methodology. In Proceedings IEEE Computer Society Annual Symposium on VLSI. New Paradigms for VLSI
18. Nesbit, K. J., Aggarwal, N., Laudon, J., & Smith, J. E. (2006, December). Fair queuing memory systems. In 2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06) (pp. 208-222). IEEE.
19. Man, C., Bin, X., Fuming, Q., Qingsong, S., Tianzhou, C., & Like, Y. (2010, June). Distributed memory management units architecture for NoC-based CMPs. In 2010 10th IEEE International Conference on Computer and Information Technology (pp. 54-61). IEEE.
20. Chen, X., Lu, Z., Jantsch, A., & Chen, S. (2010, March). Supporting distributed shared memory on multi-core network-on-chips using a dual microcoded controller. In 2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010) (pp. 39-44). IEEE.
21. Chen, X., Lu, Z., Jantsch, A., Chen, S., Guo, Y., Chen, S., ... & Liao, M. (2015). Command-Triggered Microcode Execution for Distributed Shared Memory Based Multi-Core Network-on-Chips. JSW, 10(2), 142-161.

22. Tatas, K., Siozios, K., Soudris, D., & Jantsch, A. (2014). Middleware Memory Management in NoC. In Designing 2D and 3D Network-on-Chip Architectures (pp. 191-208). Springer, New York, NY.
23. Goebel, M., Behnke, I., Elhossini, A., & Juurlink, B. (2018, May). An Application-Specific Memory Management Unit for FPGA-SoCs. In 2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW) (pp. 222-225). IEEE.
24. Siast, J., Łuczak, A., & Domański, M. (2019). Ringnet: A memory-oriented network-on-chip designed for fpga. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 27(6), 1284-1297.
25. Gordon-Ross, A., Abdel-Hafeez, S., & Alsafrjalni, M. H. (2019, July). A One-Cycle FIFO Buffer for Memory Management Units in Manycore Systems. In 2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI) (pp. 265-270). IEEE.
26. Raparti, V. Y., & Pasricha, S. (2019). Approximate NoC and memory controller architectures for GPGPU accelerators. IEEE Transactions on Parallel and Distributed Systems, 31(5), 25-39.
27. Kumar, A., & Reddy, V. K. (2020, July). Advanced Memory Management Unit for 3-D Network on Chip. In 2020 International Conference on Electronics and Sustainable Communication Systems (ICESC) (pp. 1062-1067). IEEE.
28. Jang, H., Han, K., Lee, S., Lee, J. J., & Lee, W. (2019). MMNoC: Embedding Memory Management Units into Network-on-Chip for Lightweight Embedded Systems. IEEE Access, 7, 80011-80019.
29. Musaddiq, A., Zikria, Y. B., Hahm, O., Yu, H., Bashir, A. K., & Kim, S. W. (2018). A survey on resource management in IoT operating systems. IEEE Access, 6, 8459-8482.
30. Shi, B., Li, B., Cui, L., & Ouyang, L. (2018). Vanguard: A cache-level sensitive file integrity monitoring system in virtual machine environment. IEEE Access, 6, 38567-38577.
31. Pu, Y., Shi, C., Samson, G., Park, D., Easton, K., Beraha, R., ... & Attar, R. (2018). A 9-mm 2 ultra-low-power highly integrated 28-nm CMOS SoC for Internet of Things. IEEE Journal of Solid-State Circuits, 53(3), 936-948.
32. Chen, L., Zhu, D., Pedram, M., & Pinkston, T. M. (2015, February). Power punch: Towards non-blocking power-gating of noc routers. In 2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA) (pp. 378-389). IEEE.
33. Han, K., Lee, J. J., Lee, J., Lee, W., & Pedram, M. (2017). TEI-NoC: Optimizing ultralow power NoCs exploiting the temperature effect inversion. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 37(2), 458-471.
34. K. Han, J.-J. Lee, and W. Lee, (2017) ``Converting interfaces on applicationspeci _c network-on-chips," J. Semicond. Technol. Sci., vol. 17, no. 4, pp. 505_513, Aug. 2017.