A 16 Core Processor With Hybrid Inter-Core Communication

K. Kusuma (M.Tech),
Dept. Of. ECE,
JNTUACEA,
Anantapur.

S. Aruna Mastani M.Tech , Ph.D ,
Assistant Professor,
Dept. Of. ECE,
JNTUACEA,
Anantapur.

Abstract: ***A 16-core processor with hybrid (i.e., both message-passing and shared-memory) inter-core communication mechanisms are implemented in 90nm CMOS. Shared-memory communication is supported using the shared memory within each cluster and Message-passing communication is enabled in a 3× 6 Mesh packet-switched network-on-chip. The proposed system consists of a direct memory to memory communication between two clusters which is implemented by using a DMA prototype. The prototype is especially designed for specific functionality. It is observed that the communication time between the source and destination is 620ns in case of shared-memory communication and it gets reduced to 450ns in case of message-passing communication. Compared to shared-memory and message-passing communications the proposed system reduces the communication time to 330ns. Three types of communications are implemented in XILINX 12.2 version software.***

Index Terms—Chip multiprocessor, Direct Memory Access, cluster-based, SIMD, inter-core communication, shared-memory, multi-core, message-passing, network-on-chip (NoC), inter-core synchronization, Memory to Memory.

I. INTRODUCTION

In order to meet performance requirements single core designs were pushed to higher clock speeds, thereby the power requirement grew at a faster rate than the frequency. This power problem was exacerbated by designs that attempted to dynamically extract extra performance from the instruction stream, As we will note later that this led to designs that were complex, unmanageable, and power hungry. To meet these requirements chip designers turned to multi-core processors. A multi-core processor is one which consists of multiple number of processors on a single chip, all these processors work in parallel thereby the overall performance of the multi-core processor increases. To meet power budget many efforts are taken to optimise memory hierarchy and to increase parallelism concurrently.

Inter-core communication plays an important role to balance the power and performance in a multi-core processor. Now a day's multi-core architecture introduces new challenges for effective implementation of inter-core communication. When certain problem is given
to an embedded processor the throughput depends on both computing capability and communication efficiency between cores. To enhance computing capability there are various technologies such as Very long instruction word (VLIW), Single instruction multiple data (SIMD), Super scalar, Reduced Instruction set computer (RISC) etc. But there are no matured solutions for inter-core communication, Hence the research focus on improving the efficiency of inter-core communication.

Shared-memory communication is most often used inter-core communication mechanism due to its simple programming model but it fails to provide sufficient scalability with the increasing number of processors.[3]-[5]. Therefore the designers turned to message-passing communication mechanism which has high scalability even with the increase in number of resources.[6]-[9]. We can obtain high performance by integrating both the inter-core communication mechanisms.[1]. The proposed system introduces a new type of inter-core communication called Memory- to-Memory communication through which the path from source to destination in multiple number of clusters gets reduced thereby the performance gets increases alot compared to previous mechanisms.

This paper is organised as below. Section II describes the key features of the 16-core processor. Section III details the design and implementation of existing methods. Section IV describes the implementing method. Section V presents the measured results. Section VI

concludes the paper. Section VII describes the future work.

II. KEY FEATURES

A. Multi-core Processors.

A processor is an electronic element executes a set of instructions each at a time and produces results. A multi-core processor is a single computing component with two or more independent actual processors (called "cores"), which are the units that read and execute a set of instructions simultaneously. Multi-core processor architecture is now becoming the mainstream of commercial processor architecture in the market. Parallelism and pipelining are implemented in a multi-core processor. Thereby the performance of the multi-core processor is very high when compared to a normal processor. To execute an instruction the processor will take the data either from the shared-memory or from another processor.

B. Inter-core Communication.

It is defined as the communication between multiple number of processors integrated on a single chip. Power and cost budgets limits high computability processors to be integrated on a chip thereby the overall performance of multi-core processor relies highly on inter-core communication. In multi-core processors the data stream flows through several processor cores until getting the results. Thus the throughput is highly relevant to inter-core communication. With the increasing number of cores more challenges are required to achieve efficient inter-core communication.

C. Hybrid Inter-core Communication.

Two types of inter-core communication mechanisms exist for an embedded processor. The first one is Shared-memory communication which is implemented by making use of a shared cache or memory units. Typical examples are cortexA9, UltraSPARC, HYDRA etc. The features of shared-memory communication are simple programming, used for transferring of large blocks of data. It faces several challenges which limits its use in future processors. First its low scalability, more than 8 cores are not allowed to share a single memory. In an 8 core processor the interconnections take area equivalent to 3cores and consume power equivalent to one core. Second cache coherence issues are very complex which results in more hardware overhead.

Because of its high scalability the second type of communication i.e., Message-passing communication attracts many designers. It is implemented by connecting the processors in a Network in certain topology like mesh, bus, ring etc. Typical examples are ASAP, Intel-80 tile etc. In spite of its strong scalability it has complex programming model, and the quality of service (QoS) is not guaranteed.

Shared-memory and Message-passing communications are suitable for two different environments. By combining both the mechanisms it is suitable to work in all scenarios. Shared memory communication is implemented by a cluster based memory hierarchy and Message-passing communication is implemented by arranging the processors in a 2D mesh network on chip. The features of shared-memory communication and message-passing communication are described below in Table-1.

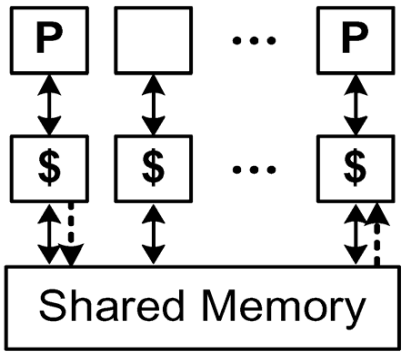
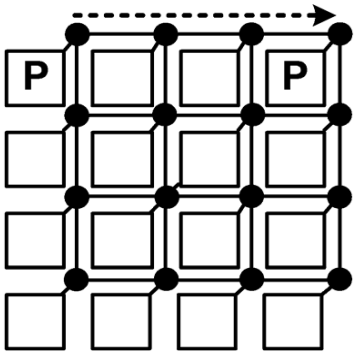
D. Cluster-Based Memory Hierarchy.

In multi-core processors the competition for memory resources increases with increasing core number results in “Memory wall” issues, Memory access latency and cache coherence issues become more complex.chip multiprocessor. Some designers solve this problem by using cache free architectures and some others suggested to partition the cache in to shared and private memory in order to improve the efficiency.

E. Direct Memory Access.

Direct Memory Access (DMA) is one of several methods for coordinating the timing of data transfers between an input/output (I/O) device and the core processing unit or memory in a computer. DMA is one of the faster types of synchronization mechanisms, generally providing significant improvement in terms of both latency and throughput. DMA allows the I/O device to access the memory directly, without using the core. DMA can lead to a significant improvement in performance because data movement is one of the most common operations performed in processing applications.

TABLE-1
Comparison of Shared-Memory and Message-Passing inter-core communications

Method	**Shared-memory**	**Message-passing**
Usage	Large, unsplit data block	Frequent, scattered data
Pro	Simple programming	Better scalability
Con	Lower scalability	Uncertain channel
Medium	Shared cache/memory	Network-on-chip
Scenario	Computation data flow	Control data flow
Graphics		

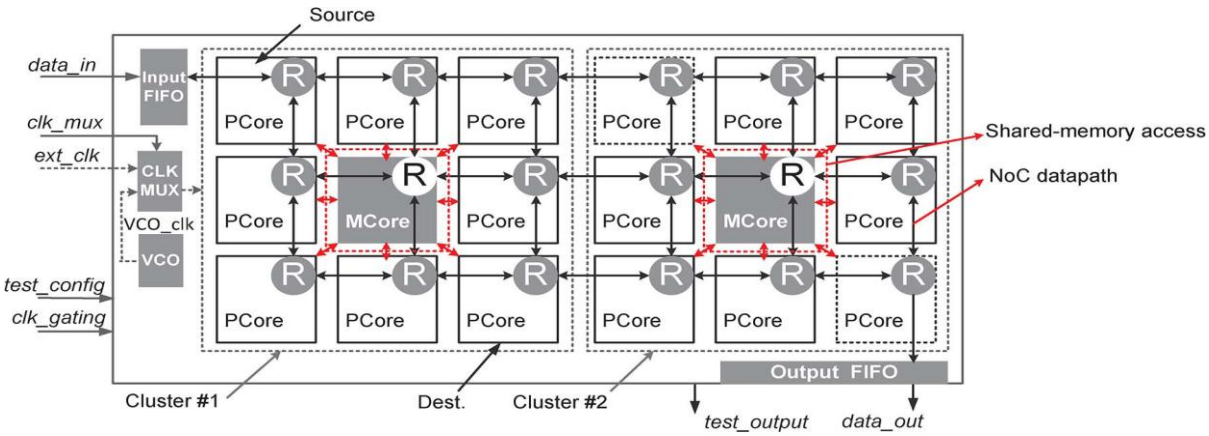


Fig. 1. Architectural Overview of the 16-core processor

III. EXISTING METHODS.

The existing system has 16-core processors which are connected in 3× 6 2D Mesh NoC form that links 16 core processor (PCore) and 2 memory cores (MCore). Cluster-based architecture is employed with two clusters where each cluster contains eight PCores and one MCore. The PCores present in the cluster can able to access the MCore present in the same cluster. A hybrid inter-core communication scheme is employed supporting both shared-memory and message-passing communications. Shared memory in MCore enables shared-memory communications within the cluster, and the NoC enables message-passing communication among all PCores. The architecture overview of existing system is as shown in Fig.1 Data enters and leaves the processor through the input and output FIFO (First In First Out). Each PCore has 2k-word instruction memory and 1k-word private data memory. MCore has 8k-word shared memory with 4 memory banks.

A. *Designing of key modules*

1). *Processor core:*

The architecture of PCore is shown in Fig. 2. The processor core has six-stage pipelined SIMD processor. In Instruction Fetch stage Instructions are fetched from instruction memory according to the program counter. The

decode stage converts the instructions fetched into opcodes and fetches operands from register files. Operations like addition, subtraction, multiplication, and, or, etc. All arithmetic and logical operations are performed and address calculations are done in execution phase. Data memory accessing is done in memory stage. The data is aligned in align stage and written back to register file or output FIFO in write back stage. The six-stage pipelining of the processor is as shown in Fig.3. Typically private data memory access requires 1 clock cycle and shared memory access takes 2 clock cycles because of its contention.

SIMD Instruction Set Architecture (ISA) supports 3 computing modes includes scalar-scalar, scalar-vector and vector- vector. Now a day's most of the processors support three kinds of data widths they are 8b, 16b, and 32b. The proposed processor is of 32b wide. We reconstruct the data path with reconfigurable data width. Power consumption reduces with the increase in data locality so, we extended the register file size to 64 words from 32 words. The benefits of this extended register file are more number of registers is available means more capacity to allocate data so the performance of the processor gets improved. These register files serves as FIFO mapping ports. As we are directly processing with the registers so, no need of load/store instructions thereby the time to access the data from memory gets reduced. Each core processor has a router which consists of 4 FIFOs and a control unit to control the flow mechanism.

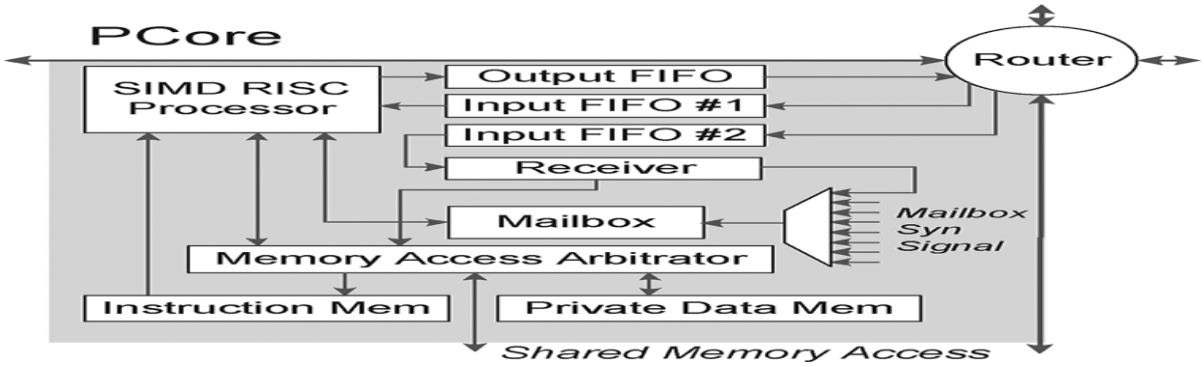


Fig. 2. Architecture overview of PCore

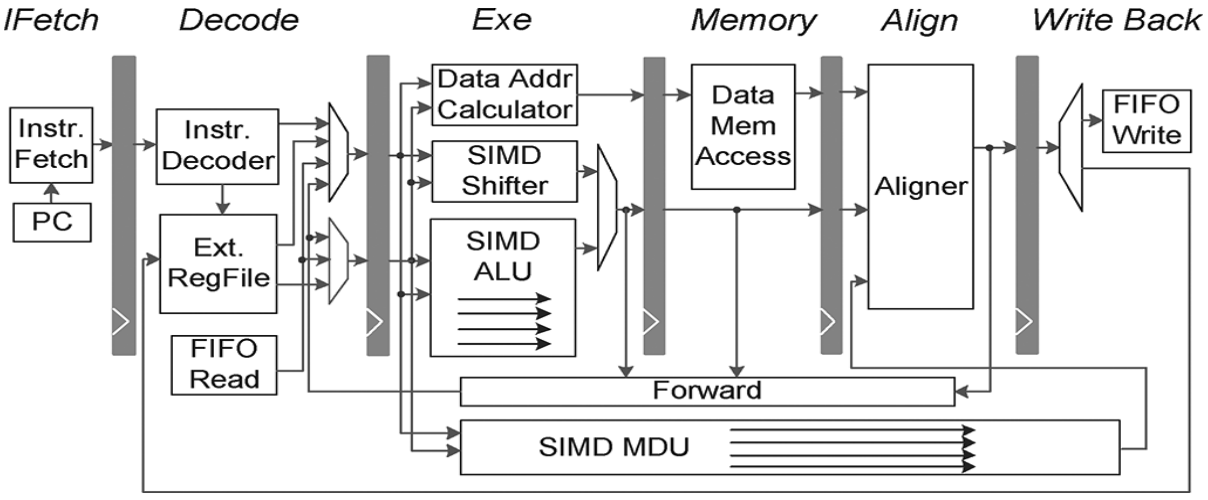


Fig. 3. The six-stage pipelining of the processor core.

2) Memory Core:

A MCore consists of 8k-word shared memory which is partitioned in to four banks. All processor cores can access the MCore directly by hardwires with fixed priority order to obtain high performance and low cost, and to simplify the arbitration logic and optimize the critical path. Because of fixed priority access it may leads to live-lock conditions, but in reality the live-lock is rarely observed. The shared memory is mainly used to map the data between different processors. If the data is not ready in the memory the PCore has to wait till the data is available even though it is having highest priority. The latency of MCore accessing without contention is 2 cycles. It increases if multiple number of processors requests the memory at the same time because the processors with low priority has to wait till it gets its turn.

The architecture overview of MCore is as shown in Fig. 4. It has a multiplexer which is used to select the processor that can access the MCore, And a Decoder is present which is used to select the memory bank on which the processor can store or load the data. A router is designed which performs the same operations as in PCore.

B. *Design of Hybrid Inter-Core Communications.*

A hybrid inter-core communication is employed by integrating both message-passing and shared-memory communication schemes which is implemented in Fig. 5. The 2D Mesh NoC supports the message-passing communication which is highly scalable and is suitable for transferring of frequent and scattered data packets. It is mainly used in control data flow applications. The shared memory in the MCore supports the shared-memory communication inside the cluster which is suitable for large data block transferring. It is mainly used in computational dataflow applications.

1) Shared-Memory Communication:

The processors which are present in the same cluster can access the MCore with fixed priority order. The max number of PCores in a cluster are limited to eight. The processor on the top left corner has highest priority and the PCore on the Bottom right corner has lowest priority. High inter-core synchronization efficiency has been achieved through hardware-aided mailbox mechanism. Shared-Memory communication involves mainly three steps, First the source PCore stores the data in to shared memory, next it sends a synchronization signal to the destination PCore, Finally the Destination PCore access the data from shared memory after the synchronization signal is received. The steps for shared-memory communication are as shown in Fig. 6.

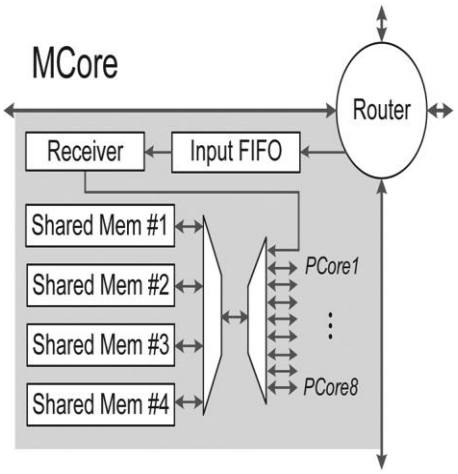


Fig.5.Architecture overview of MCore.

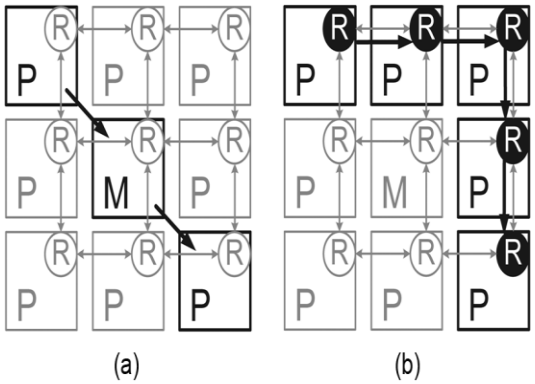


Fig.6. Implementation of the hybrid inter-core communications: (a) Shared-memory via MCore (b) Message-passing via NoC.

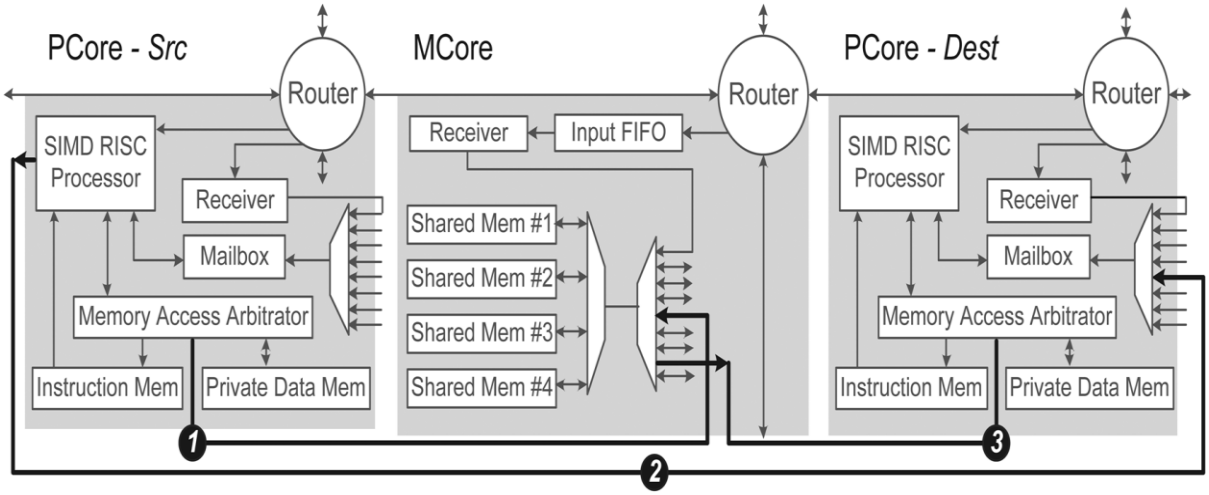


Fig.7. Three steps in a typical shared-memory communication: (1) Src PCore stores data to shared memory in MCore; (2) Src PCore sends synchronization signal to Dest PCore; (3) Dest PCore loads data from shared memory when synchronization signal is received.

2) *Message-Passing Communication:*

A 3×6 2D Mesh NoC supports Message-Passing communication where an XY dimension ordered wormhole routing algorithm is implemented. Even the shared-memory communication is implemented only within the cluster, The Message-passing communication is implemented between any two processors in the chip (i.e., with in the cluster or outside the cluster). It is more scalable and is mainly used for transferring of frequent and scattered data. Apart from its advantages it has 2 bottlenecks. The first one is the uncertainty in the communication channel. The network with heavy traffic will block the data packets in the channel hence the latency gets increased. But with the aid of shared-memory communication within the cluster the traffic load on the network can be reduced. The second bottleneck lies in the data transferring between the processor and router. By using two input FIFOs and one output FIFO between the Processor and router we can solve this problem. One input FIFO is used to receive the data coming from another processor core and second FIFO is used to receive data coming from memory core. When router receives data from another router based on the MSB digit it will decide whether the data comes from PCore or from MCore and then it will store the data on the corresponding FIFO's. The data from one of the input FIFO is moved to extended register files or data memory and the data from second input FIFO is moved to the receiver and then to data memory. The data path in message-passing communication is as shown in Fig. 7.

3) *Hardware-Aided Mailbox Synchronization:*

To support inter-core communication Hardware-aided mailbox synchronization is used rather than software synchronization protocols. Every core i.e., both PCore and MCore has a mailbox and it is accessed by PCores and the receiver in the same cluster. It generates a synchronization signal based on which the SIMD processor will accept the input data. By using a Multiplexer we can select one of the 8 PCores present in a cluster and a receiver. Mailbox has 9 registers of 4 bits each which are used to store the LSB bits of PCores and the receiver. By using address and enable inputs we can select one of the registers and verify its value with the check value if both the values are equal the mailbox will return a valid signal otherwise an invalid signal is generated. The design of the mailbox is as shown in Fig. 8

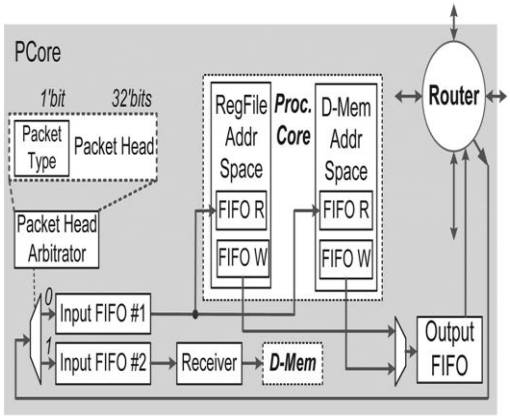


Fig.7. Datapath of the message-passing communication in PCore.

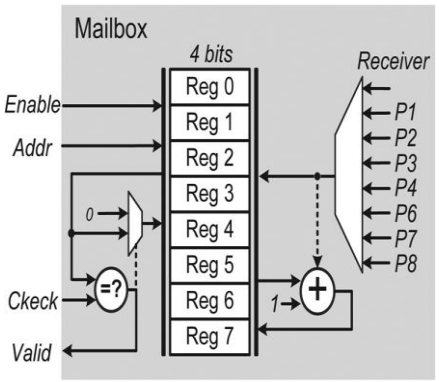


Fig.8. Hardware design of the mailbox synchronization.

IV. PROPOSED SYSTEM

The proposed system introduces a new mechanism for inter-core communication i.e., a Memory-to-Memory communication between the MCores in two clusters. It is implemented by using a memory interface it may be a FIFO, Router, DMA etc. In order to implement this Memory-to-Memory communication first we have to design a processor architecture with 16-core processors which are connected in 3× 6 2D Mesh NoC that links 16 core processor (PCore) and 2 memory cores (MCore). Cluster-based architecture is employed with two clusters where each cluster contains eight PCores and one MCore. The PCores present in the cluster can able to access the MCore present in the same cluster. The designing and implementation of processor cores (PCore) and memory cores (MCore) is explained in existing methods. This portion mainly concentrates on designing the memory interface and implementing memory-to-memory communication.

A. Design and implementation of memory interface:

The Memory interface may be a FIFO, ROUTER, DMA....etc. Here it is a DMA prototype which is mainly designed with specific functionalities. It consists of some memory which may be a register (or) a FIFO and it is used to store the data loaded from the MCore. A controller is required to control the load store operations in the memory. The main purpose of this memory interface is transferring the data between two memory cores in two different clusters directly without the intervention of processor cores.

The memory-to-memory communication between clusters is implemented simply in five steps first the data is loaded in to the source processor (PCore 1) through an input FIFO. Second the data from source processor is loaded directly in to memory core present in the same cluster. Next by using a memory interface the data is loaded in to the interfacing component and this data is transferred from interfacing component to memory core present in the second cluster. Next the data from second MCore is directly loaded by destination Processor core (PCore 16). Lastly the data from destination processor is collected through an output FIFO. The architecture of proposed system is as shown in Fig. 8. If we observe the three communication mechanisms the communication path is greatly reduced from cluster to cluster in case of Inter-memory communication. The path from source to the destination requires only 5 intermediate components hence it is very efficient to use memory to memory communication in multicore processors compared to shared-memory and message-passing communication.

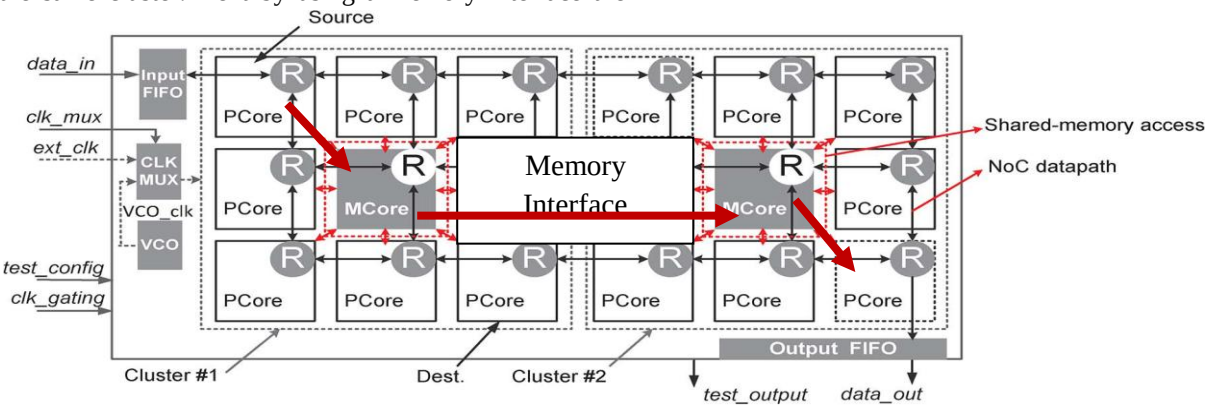


Fig. 8. Architecture overview of proposed system

V. MEASUREMENT AND EVALUATION.

1) Area:

The proposed cache-free architecture can significantly reduce chip area. Moreover, embedded applications usually require limited memory resources, so only 256 KB on-chip memory units are implemented. As a result, we can place more area budgets on execution cores and inter-core communication units. The no of LUT slices required to implement shared-memory, message-passing and memory-to-memory communication are 1279, 1466 and 922 respectively. Atmost the communication mechanisms consumed only 60% of total sliced LUTs. The complete details are as shown in synthesis results.

2) Performance and power:

In multi-core processors the throughput mainly depends on computing capability and communication efficiency between cores. To enhance computing capability there are various technologies such as VLIW, SIMD, Super scalar, RISC etc. In order to achieve high communication efficiency we are proposing this method.

The three communication mechanisms in which two of them are already existing schemes and the last one is newly implemented here. In this paper we are implementing the three communication mechanisms and then we are comparing the throughputs of existing methods with proposed one. It is observed that the shared-memory communication takes 610ns to transfer the data from source processor to destination processor and Message-passing communication required 470ns to transfer the data between source and destination processor. The proposed method i.e., memory-to-memory communication requires only 330ns to transfer the data. Hence from these results it is observed that the proposed inter-memory communication mechanism is very much effective and results in high performance when compared to shared-memory and message-passing communication. The simulation results are as shown in Fig.9, Fig. 10, Fig. 11.

Two key features contribute to the low power consumption. First, cache is discarded in the proposed cluster-based memory hierarchy, thus related hardware overhead is also reduced. Second, the data locality is improved by extended register file and separation of private and shared memory.

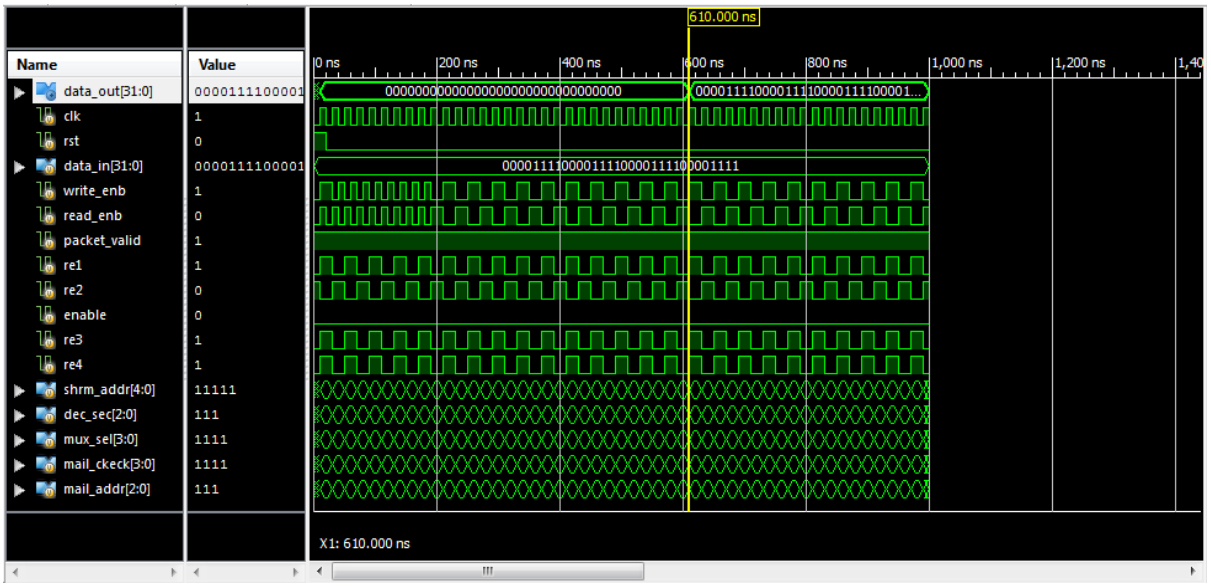


Fig. 9. Simulation results of shared-memory communication.

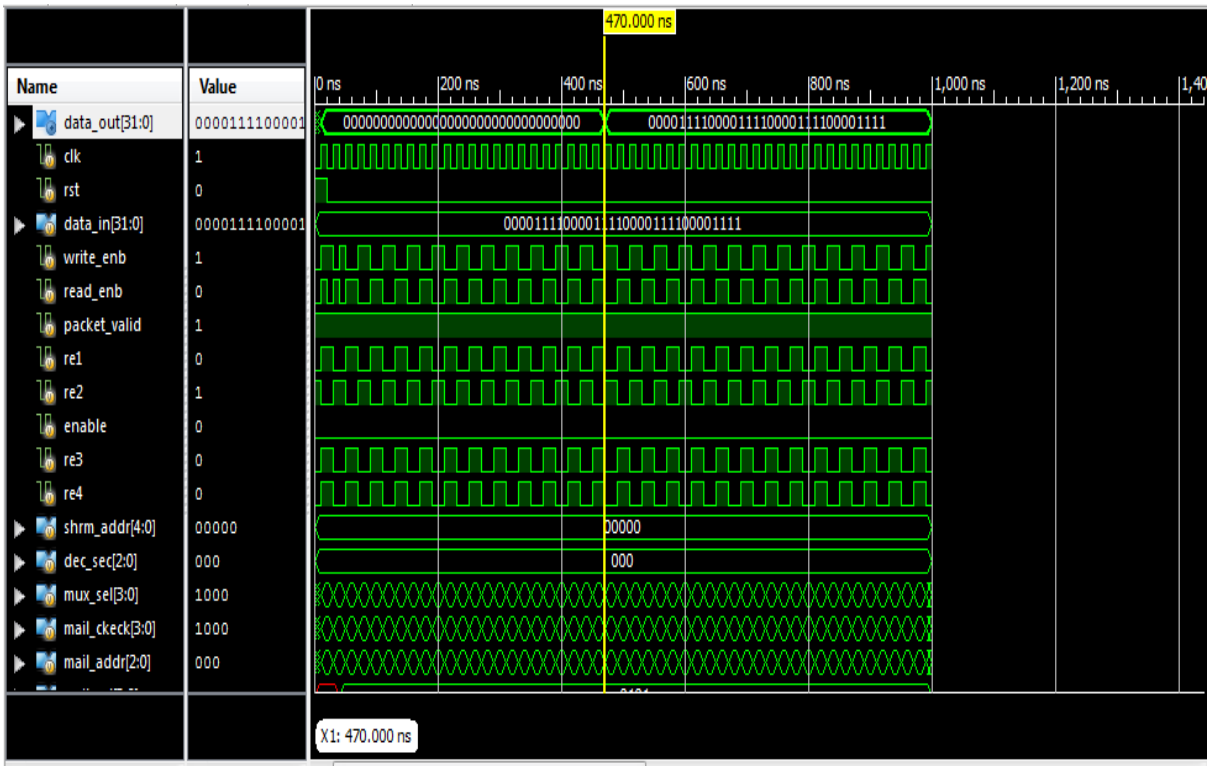


Fig. 10. Simulation results of message-passing communication.

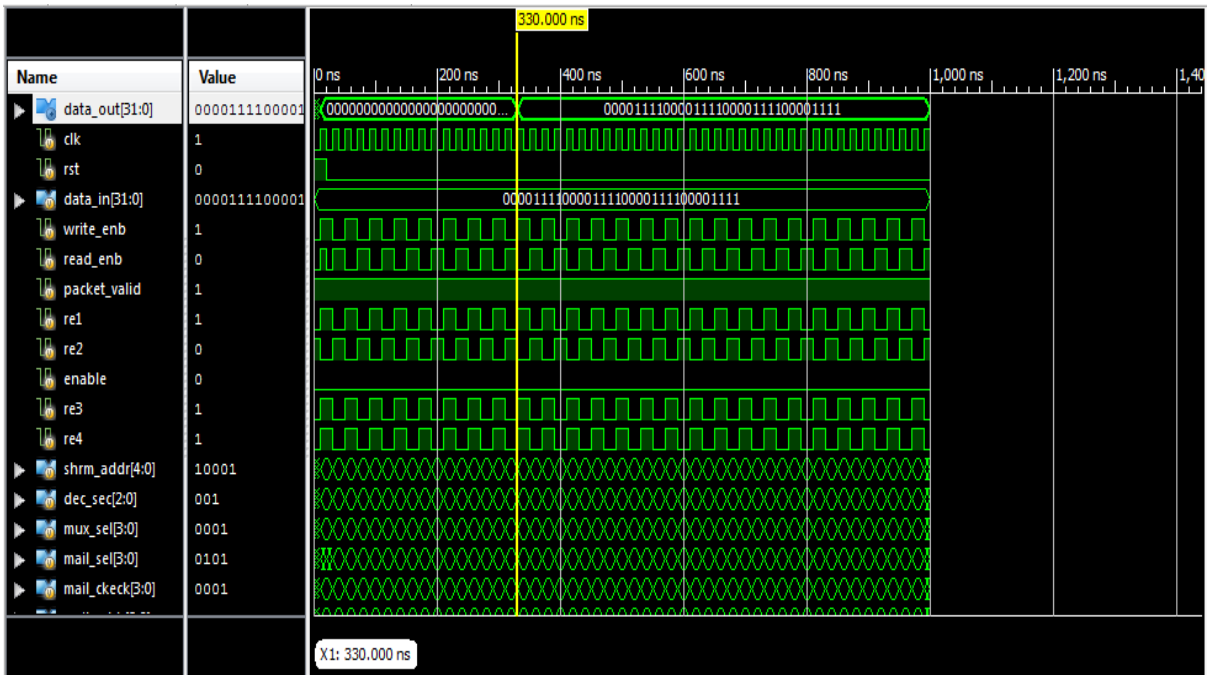


Fig. 11. Simulation results of memory-to-memory communication.

VI. CONCLUSION

A 16-core processor for embedded applications with Memory-to-Memory communications is proposed in this paper and it is compared with existing inter-core communication mechanisms. The processor has 16 processor cores and 2 memory cores. Message-passing communications are supported by the 3×6 2D Mesh NoC, and shared-memory communications are supported by shared memory units in the memory cores. The cluster-based memory hierarchy makes the processor well-suited for most embedded applications. The processor chip has a total 256 KB on-chip memory, while each processor core has an 8 KB instruction memory and a 4 KB private data memory, and each memory core has a 32 KB shared memory. A memory-to-memory communication is implemented using a memory interface called DMA. The DMA used here is a prototype one which performs only specific functions. The proposed system provides high throughput compared to existing methods. The proposed system is implemented in 90nm CMOS using XILINX 12.2 version software.

VII. FUTURE WORK

The performance of the processor gets still increased by implementing the inter-memory communication with exact DMA. It‘s throughput gets increased by using parallel processors.

REFERENCES

[1] Zhiyi Yu, and Ruijin Xiao , ”A 16-Core Processor With Shared-Memory and Message-Passing Communications.”.*Trans.circuit syst.vol.61,No.4,April 2014.*

[2] G. Blake, R. G. Dreslinski, and T. “A survey of multicore processors: A review of their common attributes,” *Signal Process. Mag.*, pp. 26–37, Nov. 2009.

[3] R. Kumar, V. Zyuban, and D., “Interconnections in multi-core architecture: Understanding mechanisms, overheads and scaling,” in *Proc. 32nd Int. Symp. Computer Architecture (ISCA’05)*, 2005, pp. 408–419.

[4] H.-Y., Y.-J. Kim, J.-H. Oh, and L.-S. Kim, “A reconfigurable SIMT processor for mobile ray tracing with contention reduction in shared memory,” *Trans. Circuits Syst. I, Reg. Papers*, no. 60, pt. 4, pp. 938–950, Apr. 2013.

[5] L. Hammond, B.-A. Hubbert, M. Siu, M.-K. Prabhu,M. Chen, and K. Olukolun, “The stanford Hydra CMP,” *Micro*, vol. 20, no. 2, pp. 71–84, 2000.

[6] A. S. Leon, B. Langley, and L. S. “The UltraSPARC T1 processor: CMT reliability,” in *Proc. Custom Integrated Circuits Conf. (CICC’06) Dig. Tech. Papers*, 2006, pp. 555–562.

[7] M.-B. Taylor, J. Kim, J. Miller, D. Wentzlaff, F. Ghodrat, B. Greenwald, H. Hoffman, P. Johnson, J.-W. Lee,W. Lee, A. Ma,A. Saraf,M. Seneski,N. Shnidman, V. Stumpen, M. Frank, S. Amarasinghe, and A. “The Raw microprocessor: A computational fabric for software circuits and general-purpose programs,” *Micro*, vol. 22, no. 2, pp. 25–35, Mar/Apr. 2002.

[8] Tilera Corp., Tilepro64 Processor Tilera Product Brief, 2008 [Online].Available:http://www.tilera.com/pdf/Product-Brief_TILEPro64_Web_v2.pdf

[9] S. R. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, A. Singh, T. Jacob, S. Jain, V. Erraguntla, C. Roberts, Y. Hoskote, N. Borkar, and S. Borkar, “An 80-tile sub-100-WteraFLOPS processor in 65-nm CMOS,” *J. Solid-State Circuit*, vol. 43, no. 1, pp. 29–41, Jan 2008.