A Simple High-Efficient Inter-Core Communication Mechanism for Multi-Core Systems

Heng Quan, Ruijing Xiao, Kaidi You, Bei Huang, Xiaoyang Zeng, Zhiyi Yu*
State Key Laboratory of ASIC and System, Fudan University, Shanghai
Email: 09210720066@fudan.edu.cn, *zhiyiyu@fudan.edu.cn

Abstract— **In this paper, we propose a simple and high efficient inter-core communication mechanism for multi-core systems. To alleviate the high hardware complexity and low communication efficiency originated from the relatively independent processor and NoC designs, we propose a scheme that integrates the computation and communication together as an efficient system. The space of the register file is extended using a configurable logic, to reduce the possibility of memory accesses without increasing operand space. Furthermore, read and write ports of synchronous FIFOs that used as interfaces between processors and network routers are mapped to register file address space by an added configure instruction, so that -the calculation results from computing units can be shared highly efficiently between different cores. Also, for each single core, we adopt SIMD technology to enhance performance of processing multimedia and communication applications. The system is synthesized to achieve 830MHz under 65nm TSMC technology in typical case.**

I. Introduction

Long development cycle and high design cost of ASIC circuits and relatively low performance/power ratio of general purpose processors determine the evolution of modern integrated circuits. Power consumption of single-core processors has been increased greatly as semiconductor technology improves according to Moore' Law, which has become the bottleneck of consumer electronics, such as cellular phones, portable game machines, digital video/audio players and tablet PCs. Designers must turn to other techniques for both performance/power ratio improvements and high data processing efficiency. One attractive option is to explore the possibility of extension based on general purpose processors to achieve ASIC-like computing efficiencies with microprocessor-like application development cost [1].

In the last few years, a fundamental shift in microprocessor design from frequency scaling to increased core counts has facilitated the emergence of multi-core architectures. Recent multi-core designs have proven to optimize performance while achieve higher energy efficiency, such as 48 cores in 45nm technology published in [2] and 167 processors in 65nm CMOS introduced in [3]. Although there have been many research works on Network-on-Chip (NoC) in the last decade, such as 2-D mesh architecture in [5] [6] and Spidergon architecture in [7], low communication efficiency between cores as a result of the relatively independent processor and NoC designs still exists as an urgent problem to solve.

This paper presents a simple high-efficient inter-core communication mechanism for a multi-core system that integrates 32 RISC processors with extensions of register file and computation modules. In the proposed architecture, the original register file size is enlarged to reduce the possibility of memory access to improve performance/power ratio. Furthermore, inter-core communication efficiency is enhanced by mapping synchronous FIFOs to register file by an added configure instruction. Besides, computation modules of single core including ALU, MDU and Shifter are modified to support SIMD operations including 8-bit, 16-bit, and 32-bit data length modes, to accelerate applications of communication and multimedia.

The following of this paper will be organized as follows: part II introduces related multi-core systems and on-chip communication architectures; part III shows the overview of the proposed system architecture; part IV describes the extension of the register file and a high-efficient communication method using FIFO mapping; part V shows the SIMD extensions of computing modules; in part VI, evaluations are given for the proposed communication mechanism and overall system; last section is the summary and conclusion.

II. Related Works

Many multi-core architectures and on-chip communication networks have been proposed in the last several years. Because of its simple topology, low area cost and extensibility, bus architectures are widely used in SoC platforms in the early years from on-chip computer buses as AMBA [8] and CoreConnect [9] to data network as MicroNetwork used in Sonic's Silicon Backplane [10]. Due to the growing bus length and IP number, delay caused by the increasingly intrinsic parasitic resistance and capacitance may become large enough as a bottleneck, thus lead to another solution that uses a hierarchical architecture [11]. More recently, communication-centric approach that decouples the processing units from communication network has become popular, such as SPIN (Scalable, Programmable, Integrated Network) in [12], mesh-based interconnect structure called CLICHÉ (Chip-Level Integration of Communicating Heterogeneous Elements) in [13], 2D torus in [14] and OCTAGON MP-SoC architecture in [15]. The first-generation CELL uses a high bandwidth internal Element Interconnect Bus (EIB) couple its power architecture processor with multiple synergistic processors, IO interface and memory interface. 2D mesh network is used both in the TILE64-processor to connect 64 tile processors arranged in an 8×8 array [16] and in the 167-processor AsAP

This work is supported in part by a grant from Science and Technology Commission of Shanghai Municipality 10706200300 and Shanghai Rising-Star Program 11QA1400500, National Significant Science and Technology Projects – 01 Special 2010ZX01030-001-001-03, and State Key Laboratory of ASIC & System (Fudan University) 09ZD002 and 10KF008.

computational platform [3].

Higher performance of multi-core systems compared to single-core processor comes from instruction, data and task parallelism between cores and within cores, which can be categorized in three ways shown in Figure 1. Algorithms that need large computations can be partitioned into balanced parts and processed in pipeline to improve throughputs. Also in each pipeline stage, independent tasks in the program can be designed to execute in parallel, which reduces the time delay of critical path greatly. Furthermore, for a single core, computation efficiency is raised by increasing instruction level and data level parallelism, such as superscalar or super-pipelined architecture for the former, and SIMD structure for the latter.

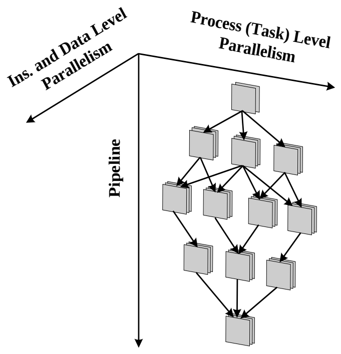

Ins. and Data Level Parallelism
Process (Task) Level Parallelism
Pipeline

Figure 1. Parallelisms in Multi-core Processors

III. PROPOSED SYSTEM ARCHITECTURE

Figure 2 shows the block diagram of the proposed multi-core architecture and its single core. Three ways of parallelism mentioned in above section can be reflected from our design. The overall system is partitioned into several homogeneous clusters to support process (task) level parallelism, and pipeline technology can be used between cores within a cluster or between clusters. For a single core, it is extended to be SIMD supported to improve data level parallelism.

Our system is composed of 32 homogeneous processors and 4 logic circuits (memories). Locally the system can be regarded as 4 clusters, consisted by eight cores and one shared memory each. The processors and memories are connected by 2-D mesh structure using wormhole routing strategy. This kind of hierarchy structure provides high efficiency both for a large task processed globally in the whole system and for a relatively simple task with small amount of computations locally processed in clusters. More related information about our system is available in reference [25] from our group.

Single core is designed based on a MIPS-like classic RISC processor of MIPS32 4KE family [17] [18]. The processor is pipelined in 5 stages, including I stage for instruction fetch, E stage for instruction decode and register file data preparation, M stage for memory access, A stage for data aligner and W stage for write back to register file, as demonstrated in Figure 2.

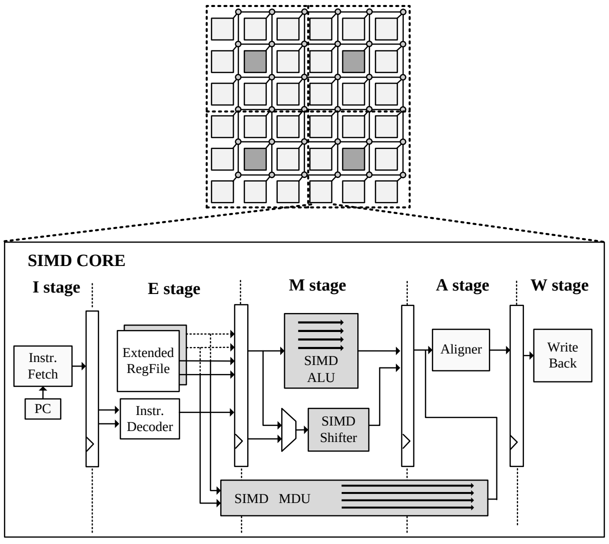


Figure 2. Block diagram of the multi-core processor and its single core

The space of the register file is extended using a configurable logic, to reduce the possibility of memory accesses without increasing operand space. Furthermore, the addresses of read and write ports of FIFOs are mapped to register file by an added configure instruction, which are designed to be asynchronous to function as the input and output interfaces of the system, and synchronous as the intermediate buffers between processors and routers. Also, computing modules including ALU, MDU and Shifter are designed to be SIMD supported. As a result multiple 8-bit and 16-bit data can be processed in parallel to improve performance for communication and multimedia applications. The next two sections will explain the above mentioned design in details.

IV. CONFIGURATION OF REGFILE AND INTER-PROCESSOR COMMUNICATION MECHANISM

Some registers in MIPS system are designed for special usage [4], such as $0, $1, $26, $27, $28, $29, $30, $31. Most algorithms and applications need more than 24 registers to store temporary data, leading to high possibilities of memory access, which causes high power consumption and low performance.

An extended register file scheme is used in our design. We enlarge the size of register file to support data swap. Take R-type instruction (Figure 3) for example, 5-bit rs, rt and rd are used for index of registers to store two source and one destination operands. In order to not increasing the instruction width for extra register index, a configure instruction is added to realize the switch between standard and extended registers.

R-type(Register)

31 26	25 21	20 16	15 11	10 6	5 0
op	rs	rt	rd	sa	funct

Figure 3. R-type Instruction

The space of original register file is doubled with an extended file (shadow file), and is organized as 4 pairs of groups, each of which contains eight 32-bit registers. Read and write ports of FIFO are mapped to $24 and $25 registers. Accesses are switched by an added configure instruction. Figure 4 demonstrates read and write configured schemes of the proposed register file.

The MSB of 5-bit Configuration decoded from configure instruction determines whether $24 and $25 registers are mapped as ports of FIFO. The least 4 bits represents switch signals between 4 pairs of register from (#1, #5) to (#4, #8) respectively. After that, the 5-bit address information included in the instructions indexes the output data from the target register.

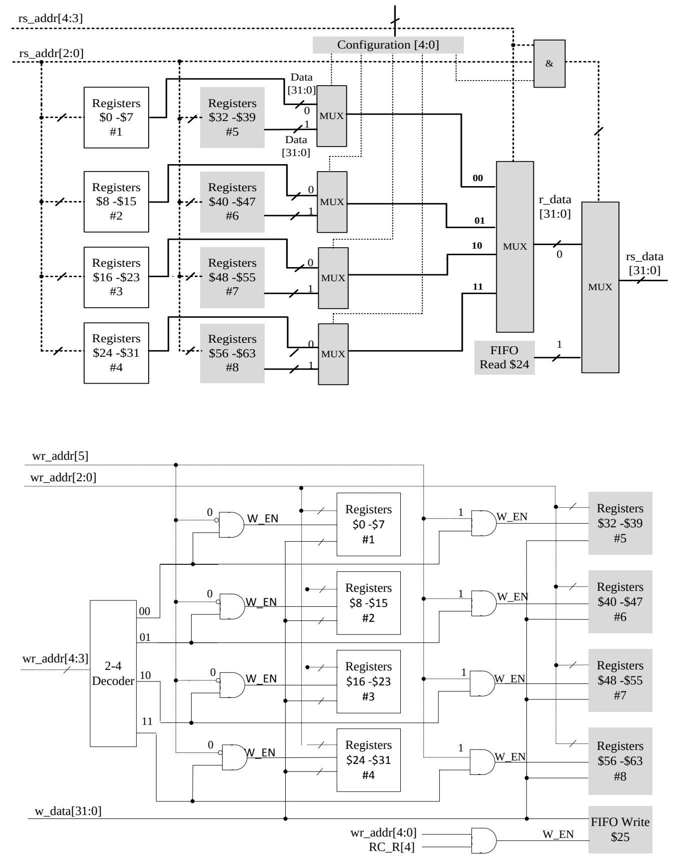


Figure 4. Read and write structures of the register file

Applications are mapped to the system to take full use of its high parallelism and throughput. As demonstrated in Figure 5, appointed task is mapping to four cores on the right with dark color, the parallel results from SIMD execution units (ALU/MDU/Shifter) in the first core (up left) is transferred through read port of FIFO mapped in register file through intermediate routers to other SIMD execution units in neighbor cores. Data received by the write port of FIFO in destination core can be directly calculated as an operand by configuring the register file into normal state. Inter-core communication efficiency is highly increased as a result of this mapping mechanism.

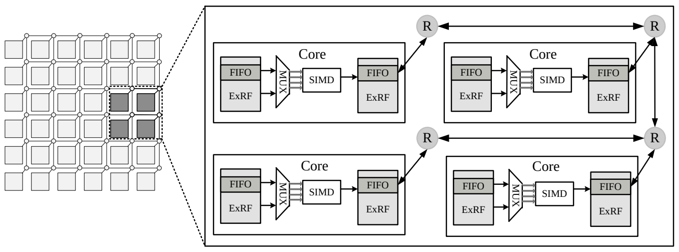


Figure 5. Inter-processor Communication Mechanism

V. Extensions On Computing Modules

8-bit and 16-bit data are widely used in communication and multimedia applications, such as RS FEC decoding algorithm and H.264 decoder. To improve high performance/power ratio, key computing modules including ALU, MDU and Shifter are extended to support multiple modes of calculations, as a result, four 8-bit and two 16-bit data can be operated in one instruction in parallel.

In most cases, the efficiency of multiplication operation determines the performance of a system. The module adopted in the proposed design can operate in four modes by using a reconfigurable booth array [19] (shown in Figure 6), which represents the partial products of 8-bit×8-bit, 16-bit×16-bit, 32-bit×16-bit and 32-bit×32-bit operations (32-bit×32-bit is realized by executing 32-bit×16-bit twice, so the result would be obtained after one more clock cycle). The reconfigurable MDU module is realized in 4 steps, performing Booth encoding, Wallace tree compressing, accumulation and writing back to register, which corresponds with stage E, M, A and W in main pipeline stream of the processor. It is worth noting that the operation of Booth encoding is executed in the second half cycle of E level, while data read from register file is operated in the first half. So the design is actually pipelined in 3 stages and can achieve nearly 850MHz in TSMC 65nm technology.

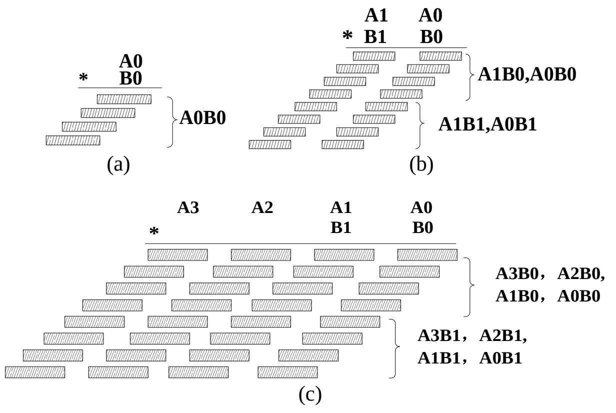


Figure 6. Reconfigurable Booth encoding unit

Furthermore, according to the features of communication applications, the performance would be much better if multiple-to-one operations are provided. For instance, multiply operations between four different 8-bit operands and one 8-bit operand is common. Thus, two modes of SIMD operations - scalar and vector are introduced [19].

Figure 7 shows the architecture of the extended SIMD ALU which can operate 4-bit, 8-bit, 16-bit, 32-bit saturated addition. Where Csa4_pg stands for 4-bit carry propagation signal [20] and carry generation signal generator unit. Then the Carry Generator module calculates these 16 pairs of data to get carry out signals into Carry Select Adders to select addition results. The parallelism between Carry Generator and Carry Select Adders reduces the latency to a certain extent.

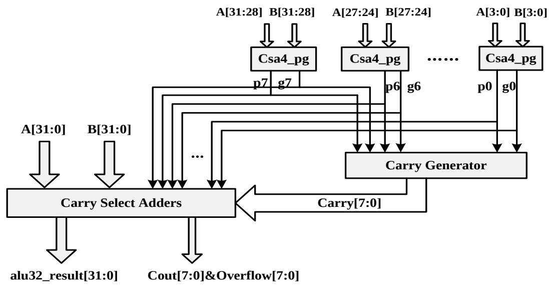


Figure 7. Architecture of SIMD ALU

Communication algorithms also require for 8-bit shift operations. The SIMD extended Shifter supports SLL (shift left logical), SRL (shift right logical) and SRA (shift right arithmetic) for 8-bit and 16-bit. Figure 8 demonstrates the operation of four 8-bit data left shifted 3-bit in instruction psll.o.

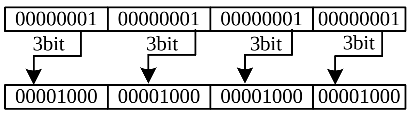
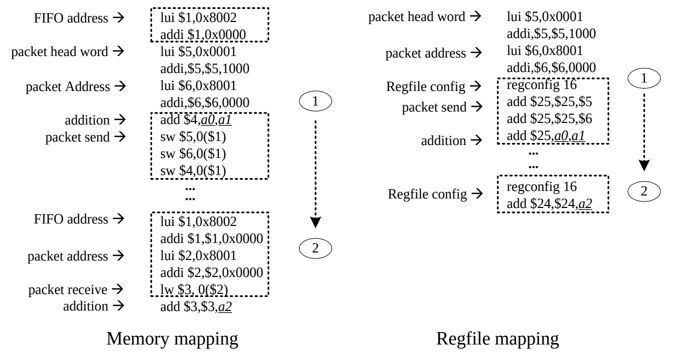


Figure 8. Demonstration of PSLL.O

VI. SYSTEM EVALUATION

A comparison has been made between regular communication in which FIFO ports are mapping to memory address space and to register file as described in this paper. Figure 9 shows the difference of assembler codes and efficiency evaluation between the two mapping solutions. It compares the codes that the addition result of *a0* and *a1* is generated in source core (1) and sent to destination core (2) to operate addition with *a2*. It is quite obvious that the proposed communication mechanism evolves much fewer instructions than regular mechanism. The efficiency differs greater as more data need to be processed.



```
FIFO address →       lui $1,0x8002
                     addi $1,0x0000
packet head word →   lui $5,0x0001
                     addi,$5,$5,1000
packet Address →     lui $6,0x8001
                     addi,$6,$6,0000
addition →           add $4,a0,a1
packet send →        sw $5,0($1)
                     sw $6,0($1)
                     sw $4,0($1)
                     ...
                     ...
FIFO address →       lui $1,0x8002
                     addi $1,$1,0x0000
packet address →     lui $2,0x8001
                     addi $2,$2,0x0000
packet receive →     lw $3, 0($2)
addition →           add $3,$3,a2
```

```
packet head word →   lui $5,0x0001
                     addi,$5,$5,1000
packet address →     lui $6,0x8001
                     addi,$6,$6,0000
Regfile config →     regconfig 16
packet send →        add $25,$25,$5
                     add $25,$25,$6
addition →           add $25,a0,a1
                     ...
                     ...
Regfile config →     regconfig 16
                     add $24,$24,a2
```

Figure 9. The code differences between two FIFO mapping solutions

Figure 10 demonstrates the relation between execution time in clock cycles with communication data packet length. Despite the same communication cost between source and destination processors, as processing data increase, operation cost of store word and load word in regular way will become even greater, while this cost is eliminated in the proposed mechanism. For example, 14 instructions will be used to transfer one word for regular way while 10 is needed in our design. The result is evaluated in condition that only data transfer is included without concerning extra computation operation. The differences in efficiency mainly originated from the cost of FIFO address generation and data transfers between memory and register file (as demonstrated by dashed box in Figure 9), the latter of which will differ much larger as data length increases (but within packet limit).

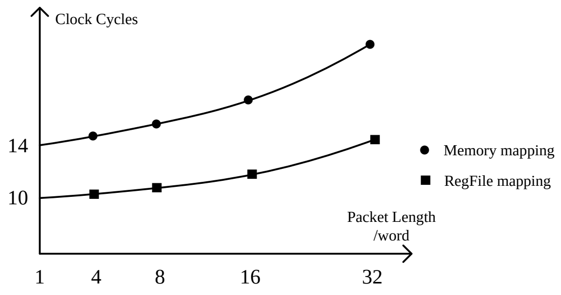


Figure 10. Relative execution clock cycles with communication data packet length

RS (255,239,8) FEC decoding algorithm is used to test the functionality and evaluate the performance of overall system. Table I shows the comparisons with other solutions. The overall latency, 1955 clock cycles, has less advantage than others because our core has much fewer extended GF instructions compared with EVP [21] and TI DSP [22], and the parallelism degree of our SIMD core (4) is lower than EVP (32) and TI DSP (8). However, although in the hypothetical case of working under 830 MHz operation clock frequency, the throughput of our implementation can reach 5157 Mbps in the worst case of 8-error incoming codeword, which is best among these platforms.

TABLE I
COMPARISON OF LATENCY AND THROUGHPUT UNDER THE WORST CASE (8 ERRORS) CONDITION

Latency and Throughput	RS algorithms				
	Ours RS(255,239)	***Subhead EVP [21] RS(255,239)***	***TI [22] RS(204,188)***	***Pal-Mult (0-p) [23] RS(255,251)***	***Starcore [24] RS(255,239)***
Latency (clock cycles)	1955	787	1213	1567	14334
Data reception rate (1/throughput) (clock cycles)	320	787	1213	1567	14334
clock frequency (MHz)	830	300	600~ 800	50	300*
Worst case Throughput (Mbps)	5157	778	783~ 1045	65	43

*The clock frequency of Starcore is not released in [24] and is obtained from a public web.

VII. CONCLUSION

In this paper, we propose a simple high-efficient inter-core communication mechanism used for multi-core system. Based on the extension and configuration adopted in register file, FIFO that used as an interface between processor and network is mapped to register file by a configure instruction. As a result, calculation results from computing units are enabled to be transferred highly efficiently between different cores in the network. This communication mechanism provides an efficient solution for network design in multi-core systems. Also, with SIMD extended, basic single processor based on MIPS32 4KE family is enabled to enhance performance of communication and multimedia applications. The overall system is synthesized to achieve 830MHz under 65nm TSMC technology in typical case.

ACKNOWLEDGMENT

Special thanks should be given to Peng Ou for system design coordination and Yan Ying for their test and evaluation work.

REFERENCES

[1] Rehan Hameed, Wajahat Qadeer, Megan Wachs, Omid Azizi, Alex Solomatnikov, Benjamin C. Lee, "Understanding Sources of Inefficiency in General-Purpose Chips", *ISCA'10*, June 19–23, 2010.

[2] Jason Howard, Saurabh Dighe, Sriram R. Vangal, Gregory Ruhl, "A 48-Core IA-32 Processor in 45 nm CMOS Using On-Die Message-Passing and DVFS for Performance and Power Scaling", *IEEE Journal Of Solid-State Circuits*, vol. 46, No. 1, January 2011.

[3] Dean N. Truong, Wayne H. Cheng, Tinoosh Mohsenin, Zhiyi Yu, et al., "A 167-Processor Computational Platform in 65 nm CMOS", *IEEE Journal Of Solid-State Circuits*, vol. 44, No. 4, April 2009.

[4] Dominic Sweetman, "See MIPS Run" Second Edition, ISBN : 9780120884216, Morgan Kaufmann.

[5] Shashi Kumar1, Axel Jantsch1, Juha-Pekka Soininen2, Martti Forsell2, Mikael Millberg1, Johny Öberg, et al "A Network on Chip Architecture and Design Methodology", *Proceedings of the IEEE Computer Society Annual Symposium on VLSI* (ISVLSI.02).

[6] M. Millberg et al., "The Nostrum backbone - acommunication protocol stack for networks on chip", *VLSI Design Conference,* 2004

[7] M. Coppola, R. Locatelli, G. Maruccia, L. Pieralisi, M.D.Grammatikakis, "Spidergon: A NoC Modeling Paradigm", book chapter in *Model Driven Engineering for Distributed Real-time Embedded Systems*, ISBN: 1905209320, Aug. 2005.

[8] ARM, Limited, *AMBA Specification*, Revision 2.0, May 1999, available from httu://www.arm.com.

[9] IBM, *Coreconnect Bus Architecture*, 1999, available from httr://www.chius.ibm.com/r,roducts/coreconnect.

[10] D. Wingard, " MicroNetwork-Based Integration for SoCs", *Proc.DAC 2001*, pp. 673-677, Las Vegas, Nevada, USA, June 18-22, 2001.

[11] C. Hsieh and M. Pedram, "Architectural Energy Optimization by Bus Splitting," *IEEE Trans. Computer-Aided Design*, vol. 21, no. 4, pp. 408-414, Apr. 2002.

[12] P. Guerrier and A. Greiner, "A Generic Architecture for On-Chip Packet-Switched Interconnections," *Proc. Design and Test in Europe (DATE)*, pp. 250-256, Mar. 2000.

[13] S. Kumar et al., "A Network on Chip Architecture and DesignMethodology," *Proc. Int'l Symp. VLSI (ISVLSI)*, pp. 117-124, 2002.

[14] W.J. Dally and B. Towles, "Route Packets, Not Wires: On-Chip Interconnection Networks," *Proc. Design Automation Conf. (DAC)*, pp. 683-689, 2001.

[15] F. Karim et al., "An Interconnect Architecture for Networking Systems on Chips," *IEEE Micro*, vol. 22, no. 5, pp. 36-45, Sept./Oct. 2002.

[16] Shane Bell, Bruce Edwards, John Amann, Rich Conlin et al.,"TILE64 Processor: A 64-Core SoC with Mesh Interconnect", *Solid-State Circuits Conference*, 2008. *ISSCC* 2008. *Digest of Technical Papers. IEEE International*, Page(s): 88 – 598.

[17] "MIPS32 4KE™ Processor Core Family Software User's Manual", Document Number: MD00103, Revision 2.02, January 5, 2004

[18] "MIPS32® Architecture For Programmers Volume II: The MIPS32® Instruction Set", Document Number: MD00086, Revision 2.50, July 1, 2005

[19] Heng Quan, Ruijin Xiao, Kaidi You, Xiaoyang Zeng, Zhiyi Yu, "A novel vector/SIMD multiply-accumulate unit based on reconfigurable booth array" , 2010 10th *IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT)*, Page(s): 524 – 526

[20] Rabaey J M, Chandrakasan A, Nikolic,B. Digital integrated circuits:a design perspective.2nd ed. *New Jersey:Prentice Hall*, 2003.

[21] Akash Kumar1, Kees van Berkel, "Vectorization of Reed Solomon decoding and mapping on the EVP," *Design, Automation and Test in Europe*, 2008, pp. 450 – 455.

[22] J. Sankaran, "Reed Solomon decoder: TMS320C64x implementation," *Texas Instruments Inc*, SPRA686, Dec. 2000.

[23] Song, L., Parhi, K.K., Kuroda, I., Nishitani T., "Hardware/software codesign of finite field datapath for low-energy Reed-Solomon codecs," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2000*, Volume: 8, Issue: 2, pp. 160 – 172.

[24] Dana Taipale, Iantha E. Scheiwe, Tina M. Redheendran, "Reed Solomon decoding on the StarCore processor," *Motorola Semiconductors Inc*, AN1841/D, May 2000.

[25] Ruijin Xiao, Heng Quan, Kaidi You, Bei Huang, Xiaoyang Zeng, Zhiyi Yu, "A novel multi-core processor for communication application", 2010 10th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT), pp. 236 – 238.