Design of 16-bit MIPS Processor for Multi-Core SoC

Ms. Jinal K.Tapar Ms. Shrushti K. Tapar Prof. Ashish E. Bhande

Abstract— **This being the era of fast, high performance computing, there is the need of having efficient optimizations in the processor architecture and at the same time in memory hierarchy too. Each and every day, the advancement of applications in communication and multimedia systems are compelling to increase number of cores in the main processor viz., dual-core, quad-core, octa-core and so on. Thus, a MPSoC with 8-cores supporting both message-passing and shared-memory inter-core communication mechanisms is to be implemented. It is well known that processor is a heart of any computing device. Thus, a 16-bit RISC type processor supporting the MIPS III instruction set architecture (ISA) with special support for detecting the data dependency cases like RAW is synthesized and simulated using Xilinx ISE 14.5 and ModelSim tools. The proposed processing node is 16-bit, 6-stage pipelined instead of traditional 5-stage pipelined. It not only detects data hazard but also takes proper corrective measure to avoid multiple stalls in the pipelining of stages. All the codes for processor design are written in Verilog HDL. Each of these cores is to be connected to the rest of the network of processors by a parameterized wormhole 2-D 3x3 mesh network-on-chip (NoC).**

Index Terms— **Data hazard, ISA, MPSoC, message-passing, MIPS, network-on-chip, RISC, shared memory, SIMD, 2-D Mesh, virtual channel router.**

I. INTRODUCTION

The computing trends are advancing by leaps and bounds. It has evolved in upward graph right from mainframes to wearable gadgets, which is everywhere computing, in terms of all parameters including size, power, latency, speed and applications too. Thus, processor design and its co-supporters that are memory sub system, I/O peripherals, software etc. also need to be updated with the current requirement. The trend of increasing a processors' speed to get a boost in performance is a way of the past. Multi-core processors are the new direction manufacturers are focusing on. Using multiple cores on a single chip is advantageous in raw processing power, but nothing comes for free. With more than one head, there arises the need of proper synchronization [12] among all the heads. Shared memories and message-passing are the well known techniques employed for inter-core communications. But still using them both as hybrid inter-core communication method is among the recent experiments that researchers are trying out to exploit advantages of both for current requirements.

Generally, a processor may be defined as any unit that read, execute program instructions and perform desired computing. Thus, a multi-core processor is a single computing chip with more than one independent real processing elements specifically termed as "cores". The multiple cores can run multiple instructions like add, move, branch etc. simultaneously, leading to fast execution speeds. The simultaneous working of these different cores on a chip achieves the goal of "parallel computing". Computer architecture courses in university and technical schools around the world often study the MIPS architecture due to its various basic and easy-to-understand design characteristics.

The MIPS instruction set architecture (ISA) is a RISC (Reduced instruction set computer) based microprocessor architecture that was developed by MIPS Computer Systems Inc. in the early 1980s by Patterson [10]. MIPS is now an industry standard and the performance leader within the embedded industry. Their designs can be found in Canon digital cameras, Windows CE devices, Cisco Routers, Sony Play Station 2 game consoles, and many more products used in our everyday lives. By the late 1990s it was estimated that one in three of all RISC chips produced was a MIPS-based design. MIPS RISC microprocessor architecture characteristics include: fix-length straightforward decoded instruction format, memory accesses limited to load and store instructions, hardwired control unit, a large general purpose register file, and all operations are done within the registers of the microprocessor.

II. RELATED WORK

The need of the multi-core processors is already figured out in the introduction section. It is also cleared that the basic building unit of this MPSoC (multi processor system on chip), is the processor module design, here referred to as "P core". Many pros and cons of the multi-core processors have been discussed by the system designers. The following section will point out the evolution of the project's work and research till date:

Rogers *etal.* [2] had put light on the issue of 'memory wall'. They explained that in spite of very sophisticated developments in improvement of processor speed, memory accesses operations consume longer time cycles and thus the associated latency reduces the overall performance. It is observed that CPU speed is improving at the rate of 50% per twice year whereas DRAM speed is sliding up at the rate of 10% only. Thus, it was now clear with the designers that further increasing the clock speed will widen the gap between CPU and memory speed. So the trend of shifting from single core of high frequency to multi-core processor running at smaller frequencies without compromising performance started gaining strength among prime manufacturers.

Geoffrey Blake [1] and others reviewed a commendable survey and comparison of various multi-core processors right from the Intel's core2 duo, xeon to icore 7 and ARMs, Toshibas to UltraSPARC chips. They pointed out that main advantage to multi-core systems is that raw performance increase can come from increasing the number of cores rather than frequency. But nothing comes free; For taking most of the multi-core processors, there must be proper synchronization [12] amongst all the cores on chip. 'Shared- memories', whereby all the cores access to single main memory and the 'message-passing', whereby all the cores are free to communicate with each other like on internet, are the well acquainted inter-core communication methods. Matteo Monchiero [12] suggested optimized synchronization techniques for shared memory on-chip multiprocessors (CMPs) based on network-on-chip (NoC) and targeted at future mobile systems. The proposed solution is based on the idea of locally performing synchronization operations. For traditional Symmetric Multi-Processing (SMP) processors, shared cache or memory units can support shared-memory inter-core communications. Typical examples are MRTP, Hydra [4], UltraSPARC T1 [5] and Cortex-A9 [8]. Although with simple programming, shared-memory communications face several challenges limiting its massive use in future many-core processors. Conversely, message-passing communication mechanism attracts lots of attention recently because of its better scalability. Typical examples are RAW [6], TILE64 [7], and ASAP [8]. They adopt NoC (network on chip) as the channels to link massive cores, and it's convenient to add or reduce cores underlying certain topology.

The paper [3] and [9] is the inspiration for this project. It proposes promising solution for high performance, fast parallelism supporting multi-core chips for embedded application like for DSP processors and network communication applications. In our previous review paper [13] published by us , we had proposed an octa core processor with hybrid inter-core communication connected to each other through a 3x3 2D mesh router.

Thus, we are trying to take step forward in achieving the goal of project proposed by us in [13] by designing its very important block: 'The processing element'. Hereafter, the designing and implementation of 16 bit RISC processor is detailed in coming sections.

III. System Overview

The 8-core processor proposed [13] has a 3x3 2D Mesh NoC that links eight processor cores (PCore) based on MIPS III instruction set architecture and a memory cores (MCore). 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 among all PCores. The cluster comprises eight Pcores and one MCore, shared memory in MCore can be accessed by PCores in the same cluster. Data enters the processor through the input First In First Out (FIFO), and exits through the output FIFO. An on-chip oscillator generates the system clock, necessary for circuit functioning. Fig. 1 depicts the architecture overview and key features of the proposed processor [13].

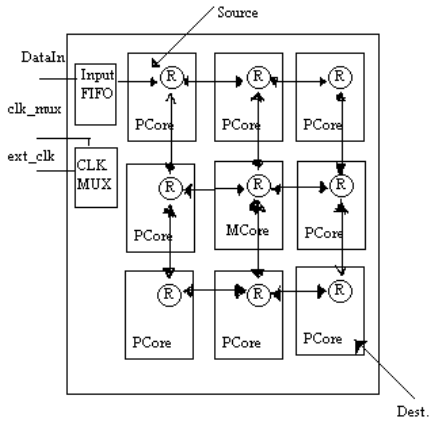

Fig.1 Architecture overview of proposed 8-core processor

The PCore includes a typical Reduced Instruction Set Computer (RISC) style processor core with six-stage pipeline, a 256-word instruction memory, a 256-word private data memory, a router and interfaces for inter-core communications. The MCore includes an 1k-word shared memory with four banks. Detailed design and implementation of 16 bit RISC processor of Pcore is discussed in the following section.

IV. Processor Overview

As mentioned before MIPS is a RISC microprocessor architecture. The MIPS Architecture defines thirty-two, 32-bit general purpose registers (GPRs). For simplicity, here we have taken only eight (8) registers. Register $R0 is hard-wired and always contains the value zero. A major aspect of the MIPS design is to ensure that all instructions take only one cycle to complete, thereby removing any needs for interlocking. The design of the MIPS [11] processor demands the elimination of instructions that would take multiple steps to complete.

The basic MIPS architecture pipeline implemented here is seen in Fig.2. The ideal CPI (cycles per instruction) for this designed processor [11] is 1.0 in case any data stalls doesn't occurs. A pipeline is a set of data processing elements connected in series, so that the output of one element is the input of the next one. These elements or stages of a pipeline are executed in parallel time-sliced fashion with some amount of buffer storage inserted between stages. The stages in this pipeline are: instruction fetch, instruction decode, execution, memory, align and write back. These pipeline stages are the basic stages implemented in this MIPS processor.

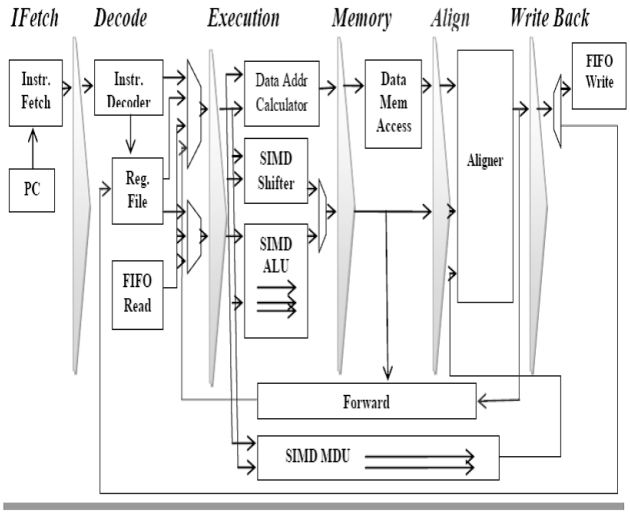

Fig.2 Six-stage pipeline of implemented processor

A. Instruction Fetch Stage

The Instruction Fetch (IF) stage directs the flow of the program and retrieves the instructions from memory. The IF stage consists of the program counter (PC) register, the synchronous instruction memory, and passed signals from the instruction decode stage that has calculated logic for branches, jumps as well as signals for stalling the PC.

B. Instruction Decode Stage

The instruction decode (ID) stage holds most of the control logic of the pipeline as it decodes what needs to happen for each instruction. The ID stage contains the register file, branch logic module, next PC logic module, and controller. The register file is a collection of 7 general purpose registers and a zero register that cannot be modified. It is an 8-entry 16-bit register file, with 1 synchronized write port and 2 synchronized read port. For Register 0, read data from it will always be 0, and write operations will also be discarded.

The controller is the component that decodes the instructions and sets signals to allow for proper execution. The controller only uses the opcode (the four MSB of the instruction) to direct data to proper arithmetic or logical function unit indicated by instruction.

C. Instruction execution

This stage is related to actual computation or manipulation of data operands in the ALU module. The shifter module does the shifting of data by the amount specified by instruction. Data address for branching and load-store instructions is calculated in this stage.

D. Memory access

This stage mainly deals with load ld and store sd instructions to calculate the memory address from where the data is to be accessed and kept ready within registers or where the results are to be written back

E. Hazard Detection or Aligner

The hazard detection module has two purposes. The first is to forward data from instructions currently in the pipeline that have not written back to the register file to an instruction in the ID stage that needs the value. The second is to stall the pipeline when forwarding the correct value is not available for that clock cycle. The hazard controller prevents read after write (RAW) hazards [10] with the data forwarding. The write after write (WAW) and write after read (WAR) hazards cannot happen in this implementation since all instructions are executed in order according to the program being run. The data forwarding problem occurs when an instruction writes to a register and then next instruction uses that register as one of its operands. For better understanding of this unit, take the instruction sequence given below

add R1, R2, R3 ------ R1 = R2 + R3
or R4, R1, R3 ----- R4 = R1 | R3
sub R5, R1, R2 ------ R5 = R1 – R2
and R6, R1, R1 ------ R6 = R1 & R1

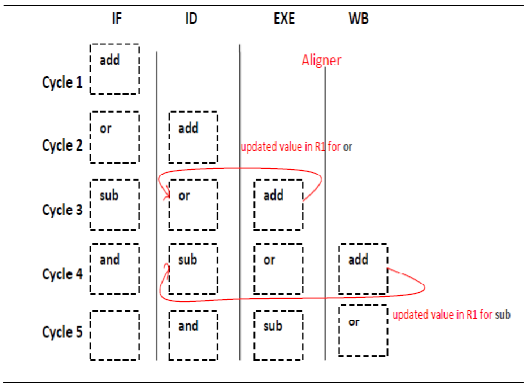

Fig.3 Forwarding illustration in the pipeline

Here, 2nd and 3rd instructions are dependent on the output of first instruction i.e content of R1 register. But in pipeline updated content of R1 will be available only after write back stage; otherwise outcome of second and third instructions would be wrong with the earlier values of R1. This is read after write (RAW) data hazard. To cope with this problem, align stage will readily forward the latest computed contents of R1 to decode stage register file.

F. Write back

Finally, the addresses passed by memory stage are used by this write back stage and results are written back to specified memory locations in the main memory.

V. INSTRUCTION SET

The instruction set is a vital aspect of any designed processor because all the applications, programs and supporting software are fabricated depending mainly on the instruction set of the processor The instruction set of MIPS is vast. But broadly, classifying MIPS only has three instruction types: I-type is used for the Load and Stores instructions, R-type is used for Arithmetic instructions, and J-type is used for the Jump instructions.

Table I.
Instruction set structure

Instruction	15:12 Op-code	11:9 Dest	8:6 Src 1	5:3 Src 2	2:0
NOP	0000	Rd	Rs1	Rs2	0000
ADD	0001	Rd	Rs1	Rs2	0000
SUB	0010	Rd	Rs1	Rs2	0000
AND	0011	Rd	Rs1	Rs2	0000
OR	0100	Rd	Rs1	Rs2	0000
XOR	0101	Rd	Rs1	Rs2	0000
SL	0110	Rd	Rs1	Rs2	0000
SR	0111	Rd	Rs1	Rs2	0000
SRU	1000	Rd	Rs1	Rs2	0000
ADDI	1001	Rd	Rs1	Immediate	
LD	1010	Rd	address		
ST	1011	Rd	address		
BZ	1100	000	Rs1	Label addr	

Basic 13 instructions implemented for this processor and description of each of the fields used in the three different instruction types is provided in the Table 1. Each instruction is 16 bits wide; Four bits 15:12 of instruction is the opcode; Three bits 11:9 represents the destination register where final result is stored; Three bits 8:6 and 5:3 each specifies the source registers for the two operands. In case of load and store instructions nine bits 8:0 are available for specifying address location.

MIPS is a load/store architecture [10], meaning that all operations are performed on operands held in the processor registers and the main memory can only be accessed through the load and store instructions (e.g ld, sd). A load instruction loads a value from memory into a register. A store instruction stores a value from a register to memory. Thus, ld and sd works with memory; the add, subtract cover the arithmetic operations; the and, or, and xor cover the logical operations; the sll, srl, and sra cover shifting needs; the set less than and branch instructions cover comparisons; the branch and jump instructions cover loops. The instructions left out of this processor included instructions that work with floating point values, caches, and variations of what was implemented (branch, jump, memory access, etc.)

VI. SYNTHESIS AND IMPLEMENTATION

All the codes for synthesizing 16-bit MIPS core are being written in Verilog HDL. Xilinx's ISE 14.5 design suite tool is used for the same. Inbuilt simulation environment of ISim provided by Xilinx's tool is used for simulation. All the basic building blocks of processor core are being written and verified by corresponding modules and test benches independently. Hierarchy between various modules is manifested in the fig.4.

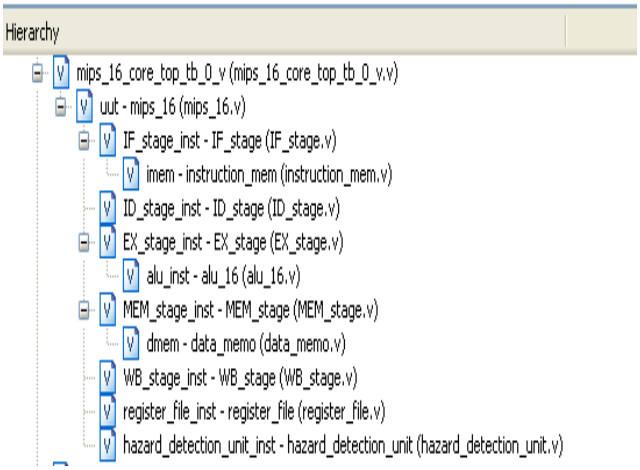

Fig.4 Design Hierarchy of different modules

In the synthesis of any module, actually the prime goal is to convert the high level behavioral lines of HDL into basic digital hardware components like flip-flops, multiplexers, adders, registers etc. The resultant combinations of this components developed by running module under consideration is called as RTL schematic (Register Transfer Logic). For this 16-bit MIPS processor, the synthesized RTL schematic is shown in the fig. 5. All the basic stages discussed in section IV and their interconnections can be seen in the realized RTL structure. The width size of each signal, inner composition of each top level block can be viewed by further opening of each top block. All this is done by synthesis of written code files shown in fig. 4.

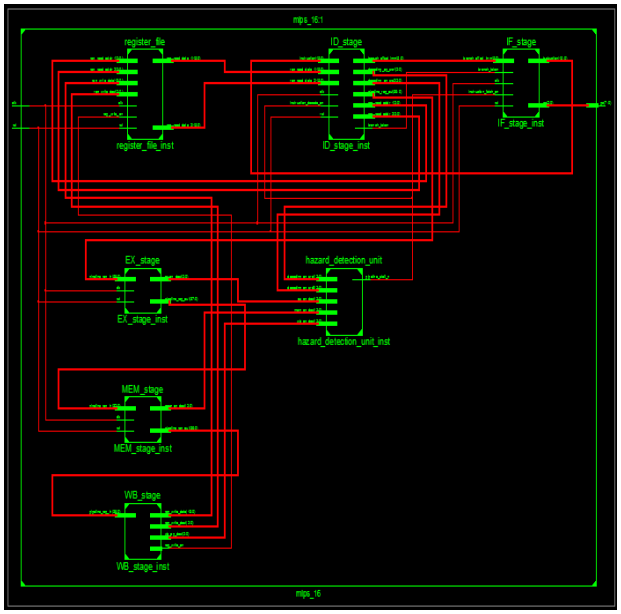

Fig.5 RTL Schematic of 16-bit MIPS Processor

VII. PROGRAM FOR TEST

After synthesizing fully functional processor, its necessary to test its functionality by loading some useful program code constituting of processor's fabricated instruction set mentioned in section V . So, we have chosen here the function of multiplication of two integers by add and shift method. There are two important reasons behind choosing this particular function for testing our processor. First one is that although multiplication operator is available with verilog operator set, its soft core module is provided internally by various FPGAs. In real it's unsynthesizable. Secondly, multiplication is of great scientific importance for almost all of the multimedia processing applications like Fast Fourier Transform (FFT), H.264 decoder for video processing and other image processing tasks also. Thus, the program code in assembly language is composed as shown in figure 6.

This program is good for our understanding; but it needs to be converted to machine level language i.e. in binary format and be loaded in the memory ROM of the processor for testing. At present as we haven't developed any of the assembler and related software tool chain for mapping of application programs onto processor, we have to load binary program manually into rom. The technique of converting each instructions into corresponding binary format and then into hexadecimal format is illustrated in the fig. 7.

```
/*   Multiply R3=R1*R2 by add and shift
*/

      ADDI      R1,R0,28        /R1=28
            ADDI R2,R0,17       /R2=17
            ADDI R6,R0,1
            ADDI R7,R0,0
            ADDI R5,R0,16       /R5=16
L1:       ADDI  R7,R7,1         /R7++
            AND   R4,R6,R2
            BZ    R4,L2          /don't
need add, skip
            NOP
            ADD  R3,R3,R1       /accumulate
L2:       SL     R1,R1,R6       /R1=R1<<1
            SRU  R2,R2,R6       /R2=R2>>1
            SUB  R4,R5,R7       /R4=R5-R7
            BZ   R4,L3           /shift
over, go to stop
            NOP
            BZ   R0,L1          /continue
            NOP
L3:         BZ    R0,L3         /stop here
```

Fig.6 Program for multiplication

Different fields of instruction were clarified in the table I. Using that data here each fields are filled by their corresponding binary representation. For example, the opcode for ADDI is 1001 from table I; then register R1 and R0 are given pointer as 001 and 000 respectively. Finally this being immediate instruction, data operand 28, taken here, is converted into binary. Thus, this full binary coded instruction is converted back to hexadecimal for ease of loading to instruction memory.

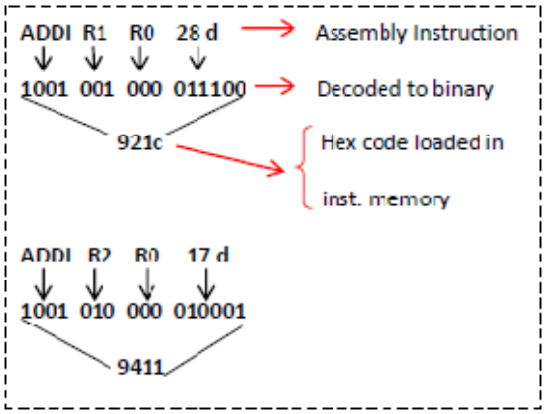

Fig. 7 Illustration of assembly to binary mapping

In this way, each instruction of whole code is converted to hexadecimal number and loaded into memory using '$readmemh' directive of verilog.

RESULTS

The simulation waveform generated for above test is shown in figure 8. The console data is printed by test bench for easy verification of results. A snapshot of data in the console window of ISim is shown in figure 10.

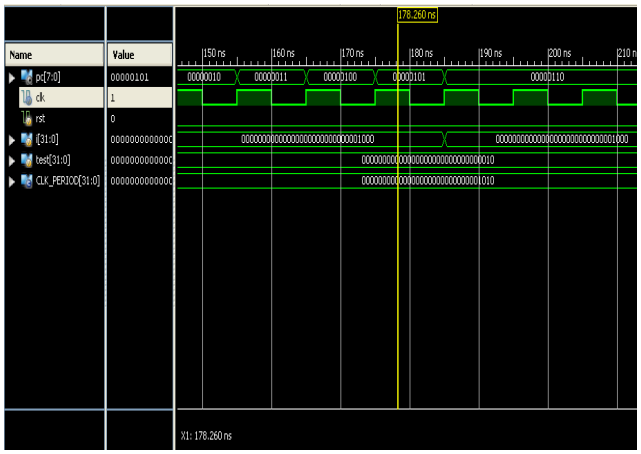

Fig. 8 Simulation results of multiply R3=R1*R2

The register R1 is loaded with first operand 28(multiplicand) in decimal, R2 is loaded with number 17(multiplier) in decimal. As it is seen in the code that in each iteration, LSB of multiplier is checked; if it is 1 then multiplicand is added with itself, accumulated and shifted by 1 bit; otherwise accumulation register is shifted by 1 bit without addition. After 'multiplier' times of iteration, we finally, get the result in R3 register.

```
# ****************************
# mips_16 core test
# ****************************
# rom load successfully
# running test2
# multiply R3=R1*R2
# display_all_regs:
# --------------------------------
# R0    R1    R2    R3    R4    R5    R6    R7
#  0     0     0     0     0     0     0     0
# --------------------------------
# running test2
# current pc:  0 ,instruction: 921c
# current pc:  1 ,instruction: 9411
# current pc:  2 ,instruction: 9c01
# current pc:  3 ,instruction: 9e00
# current pc:  4 ,instruction: 9a10
# current pc:  5 ,instruction: 9fc1
# current pc :  6
# display_all_regs:
# --------------------------------
# R0    R1    R2    R3    R4    R5    R6    R7
#  0    28    17     0     0     0     0     0

# --------------------------------
# current pc:  6 ,instruction: 3990
                .
                .
                .
                .
                .
# current pc:  17 ,instruction: c03f
# current pc:  18 ,instruction: 0000
# display_all_regs:
# --------------------------------
# R0    R1    R2    R3    R4    R5    R6    R7
#  0     0     0    476    0     16    1     16

# --------------------------------
```

Fig. 9 Simulation Result data

Eventually, it can be inferred from above data that processor functions as designed and gives proper results.

So,

R1 ⇐ 28d

R2 ⇐ 17d

R3 ⇐ 476d

Thus,

R3=R1*R2

FUTURE WORK

The eight core architecture connected with 3x3 mesh router is synthesized and ready for simulation. We are trying to run matrix multiplication program on this multi core platform.

REFERENCES

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

[2] B. Rogers, A. Krishna, G. Bell, and K. Vu, "Scaling the bandwidth wall: Challenges and avenues for CMP scaling," in *Proc. ACM Int. Symp. Computer Architecture (ISCA'09)*, 2009, pp. 371–382..

[3] Zhiyi Yu, Member, IEEE, Ruijin Xiao, Student Member, IEEE, Kaidi You, Heng Quan, Peng Ou, Zheng Yu, Maofei He, Jiajie Zhang, Yan Ying, Haofan Yang, Jun Han, Xu Cheng, Zhang Zhang, Ming'e Jing, and Xiaoyang Zeng, Member, IEEE "A 16-Core Processor With Shared-Memory and Message-Passing Communications", IEEE transactions on circuits and systems—i: regular papers, vol. 61, no. 4, april 2014

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

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

[6] 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. Agarwal, "The Raw microprocessor: A computational fabric for software circuits and general-purpose programs," *IEEE Micro*, vol. 22, no. 2, pp. 25–35, Mar/Apr. 2002.

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

[8] M. P. Singh, M. K. Jain, "Evolution of Processor Architecture in Mobile Phones", International Journal of Computer Applications (0975 – 8887) Volume 90 – No 4, March 2014

[9] John David Kubiatowicz S.B., Massachusetts Institute of Technology (1987) S.M., Massachusetts Institute of Technology (1993), "Integrated Shared-Memory and Message-Passing Communication in the Alewife Multiprocessor", Doctor of Philosophy at the MASSACHUSETTS INSTITUTE OF TECHNOLOGY, February 1998

[10]Patterson, David; Hennessy, John (1993). "Computer Organization and Design"(3rd Ed.). Morgan Kaufmann. ISBN: 1-55860-604-1

[11]Patterson, David; Hennessy, John (1996). "Computer Architecture: A Quantitative Approach" (5th Ed.). Morgan Kaufmann. ISBN 978-1-55-860329-5

[12]Matteo Monchiero, Student Member, IEEE, Gianluca Palermo, Member, IEEE, Cristina Silvano, member, Ville "Efficient Synchronization for Embedded On-Chip Multiprocessors",IEEE transactions on very large scale integration (VLSI) systems, vol. 14, no. 10, october 2006

[13]Jinal Tapar., PG student, Prof. A E. Bhande, Asst. Prof, "A Multi-Core Processor with Efficient Memory", *IJAFRC, volume 2, issue 2,February 2015,ISSN 2348 4853.*