FPGA Implementation of Pipelined 16-bit RISC Processor with Floating Point Unit

[1] C Kowsalya, M Tech Student, [2] K PrasadBabu, Associate professor, [3] B Naga Rajesh, Assistant Professor

[1,2,3] Department of Electronics and Communication Engineering,Dr.K.V.Subba Reddy Women's Engineering College,
Kurnool, Andhra Pradesh,, India

Abstract: *This paper describes implementation of 16-bit RISC processor with Floating Point mode design using Verilog Hardware Description Language (HDL). The architecture proposed is Harvard architecture having separate instruction and data memory. The important feature of proposed processor is pipelining, Floating Point Mode, Pipelining is used for improving the performance; such that the design is optimized for every clock cycle.one instruction is executed per clock. The instruction set finalized consists of 16*

instructions, which is very compact, simple and easy to learn, which another key feature of the design is. The processor has ALU, Instruction Memory, Data Memory, Program Counter, Eight 16bit general purpose registers, Floating Point Adder/Substractor, 4-bit flag register and priority based three vectored interrupts. The processor code is synthesized for Xilinx Spartan 3A Starter Board FPGA.

***Index Terms* – RISC, HDL, FPGA, Verilog, Xilinx, Spartan, Harvard, ALU.**

I. INTRODUCTION

Due to betterment of field programmable gate array, have reached a point where architecture of processor can be modified by programming [1]. The main difference between the reconfigurable processor and ordinary microprocessors is the potential to make significant changes to the data path itself besides the control flow. On the other hand, the main distinction with application specific integrated circuits (ASIC's) is the feasibility to redesign the hardware during runtime by loading a new circuit on the reconfigurable fabric [2]. To minimize the amount of area required, complexity of instruction set, instruction cycle and cost during the implementation of the design Reduced Instruction Set Computer design technique is used. The usage of RISC processor is increasing extensively in all fields with expeditious development of the silicon technology and fall in the price of the integrated circuit. Memory is accessed by the load and store operations. Other operations are performed on register to register basis, so that this feature makes the instruction set design more clear as it allows execution of instructions at one instruction per cycle rate. Multiple instructions are executed simultaneously with the help of pipelining implementation technique. Clock power is important component of overall dynamic power consumption, which should be minimized in design in order to reduce the power consumption. One of the methods which are used to reduce the clock power is clock-gating (ANDing) which dynamically terminates the clock signals in unused modules of the total hardware. Universal Asynchronous Receiver Transmitter (UART) which is a type of serial communication mostly used for short distance communication, low speed, low-cost data exchange between the computer and other peripherals. It has advantages of high reliability, less transmission line and long transmission distance, and hence extensively used as a mode of communication between the computer and the peripherals.

II. BACKGROUND

In outlining a chip, there are a few parameters that must be considered. Some of these parameters include: speed review, throughput, number of bits that the microchip manages at once, number of instructions the chip can execute, and different contemplations that contributes for the execution of the chip. Too, factors that incorporate multifaceted nature, possibility for usage, configuration structure, and capacity to be actualized in the accessible apparatuses were likewise considered. So as to deliver and to know the contemplations for these parameters, the accompanying investigations also, chip designs were looked into and examined.

1) Pipelined vs Non-pipelined: Normally pipeline implemented processors are faster than Non-pipelined processors. In Non-Pipelined processors one instruction is executed per clock cycle but in pipeline implemented processor every part of it is kept busy. Many instructions are executed per clock cycle in pipelining technique by dividing a single instruction into many number of instructions.

Pipelining increases the CPU throughput but increases the latency due to overhead of the pipelining process itself. Hardware or software implementation of pipeline technique is possible. Pipeline is a series of stages where some work is done at each stage and work is not completed until it has passed through all the stages.

2) Architecture comparison- RISC vs CISC: RISC stands for Reduced Instruction Set Computer whereas Complex Instruction Set Computer is an abbreviation of CISC.CISC had large number of instructions wherein some instructions for special tasks where used infrequently. Compilers became more prevalent. RISC whereas is designed to perform smaller number of types of computer instruction so it is able to operate at a higher speed than CISC. RISC has less number of instructions may be 128 or less and hence few addressing modes. Memory access instructions were limited to LOAD and STORE. All operations were done within the registers of the CPU. The most important feature of RISC compared with that of CISC is Single cycle Instruction execution i.e done by overlapping fetch, decode and execute phases of two or three instructions known as pipelining.Clock

gating is used for specific modules which be clocked, only when required for efficient reduction of the power. Loading to the registers is taken place during falling edge of the clock and all the control signals are generated during the rising edge of the clock. The modules which use clock gating are data memory and general purpose registers.

III. PROPOSED ARCHITECTURE

The proposed architecture is shown in figure 1. It is based upon the Harvard architecture which has 2 different storage and 2 different pathways for the instructions to be executed and the corresponding data to be fetched. It consist of eight 8-bit general purpose registers for data saving purpose. It also consists of ALU(Arithmetic Logic Unit) which performs Arithmetic and Logic tasks on the operands which are present in the instructions. The proposed system also consist of RAM(Random Access Memory) for the purpose of storing the transitional outcomes and ROM(Read Only Memory) to store the instructions. It also consist of CU(Control Unit) that assists the processor what operation to be performed during subsequent execution of the instruction. For the current instruction to be executed it has PC(Program Counter) which acts as a counter and its related enlist. With the help of pipelining computer architecture allows the next instruction to be fetched while the processor is performing on another instruction. Pipelining is the standard trait used in RISC processor for improving the performance and to reduce execution time per instruction. In the proposed processor, pipelining makes the architecture more efficient when tested on Xilinx Spartan 3A Starter Board FPGA with performance of at 73.364 MHz.

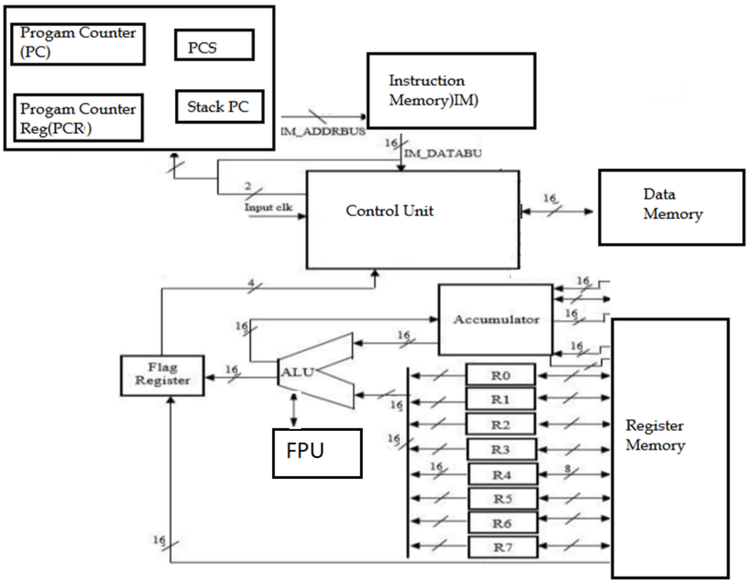


Fig1:RISC Architecture with FPU

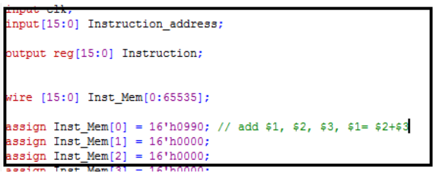
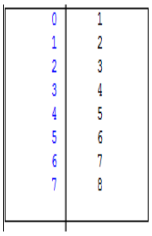
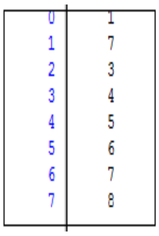
The proposed RISC architecture consists of several functional blocks like PCU(Program Counter Unit), CU(Control Unit), DM(Data Memory), IM(Instruction Memory), ALU(Arithmetic Logic Unit), GPR(General Purpose Registers), Accumulator, Register memory.

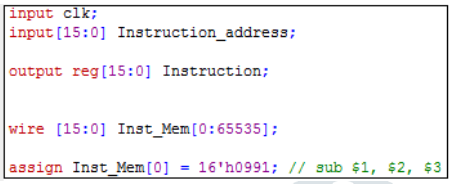
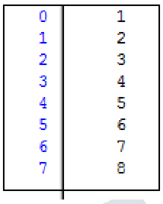
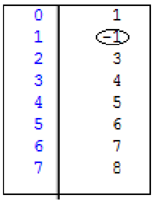
MNEMONIC	DESCRIPTION
add	Add the contents of specified registers and store it in specified destination
sub	Subtarct the contents of specified registers and store it in specified destination
and	AND the contents of specified registers and store it in specified destination
or	OR the contents of specified registers and store it in specified destination
slt	Slt in MIPS is used for a specific condition like if one value is less than another value then set the value of a particular register
sw	To store the value in memory from register
lw	To load the value from memory to register
beq	Branch if accumulator is zero
addi	Add immediate data with specified register
Jr $t	Jump to the address specified by $t
slti	Set if value is less than immediate data
Addfp	Same as above Add, Sub but for floating point mode, Mult is added for floating point operation
Subfp	
Multfp	

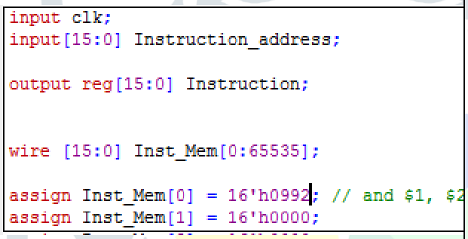
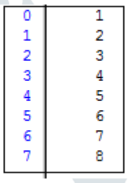
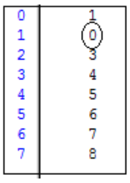
IV. RESULTS AND DISCUSSION

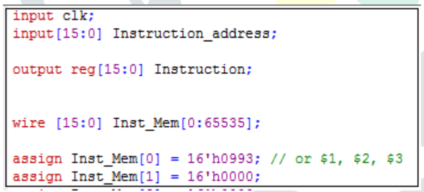
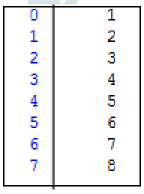
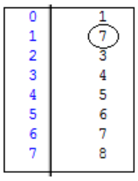
The simulation results have been performed by Modelsim and synthesis using Xilinx ISE. Instructions are given in the form of opcode. The operations will be performed according to the instructions given. The instructions that can be executed with the help of this processor are as follows.

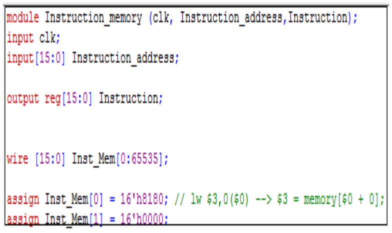
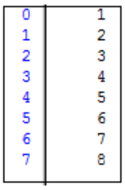
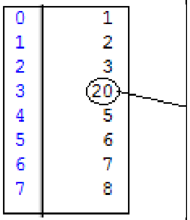
		Before Execution	After Execution

Addition: The instruction given to perform addition operation is as shown in fig. The addition operation is performed and the result is stored in the register. From the fig we can see that the values in the register 2 and register 3 are added and the result is stored in the register 1.	
	`input clk;`
	`input[15:0] Instruction_address;`
	`output reg[15:0] Instruction;`
	`wire [15:0] Inst_Mem[0:65535];`
	`assign Inst_Mem[0] = 16'h0990; // add $1, $2, $3, $1= $2+$3`
	`assign Inst_Mem[1] = 16'h0000;`
	`assign Inst_Mem[2] = 16'h0000;`
	`assign Inst_Mem[3] = 16'h0000;`
		
		0 1
		1 2
		2 3
		3 4
		4 5
		5 6
		6 7
		7 8
			
			0 1
			1 7
			2 3
			3 4
			4 5
			5 6
			6 7
			7 8

Subtraction: The instruction given to perform subtraction operation is as shown in fig. The subtraction operation is performed and the result is stored in the register. From the fig we can see that the values in the register 2 and register 3 are subtracted and the result is stored in the register 1.	
	`input clk;`
	`input[15:0] Instruction_address;`
	`output reg[15:0] Instruction;`
	`wire [15:0] Inst_Mem[0:65535];`
	`assign Inst_Mem[0] = 16'h0991; // sub $1, $2, $3`
		
		0 1
		1 2
		2 3
		3 4
		4 5
		5 6
		6 7
		7 8
			
			0 1
			1 -1
			2 3
			3 4
			4 5
			5 6
			6 7
			7 8

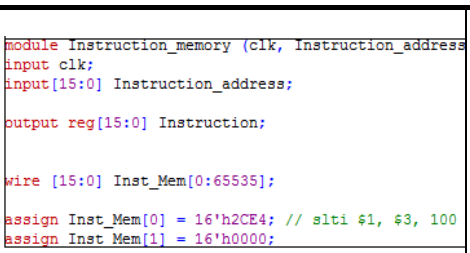
AND: The instruction given to perform and operation is as shown in fig. The AND operation is performed and the result is stored in the register. From the fig we can see that the values in the register 2 and register 3 are ANDed and the result is stored in the register 1.	
	`input clk;`
	`input[15:0] Instruction_address;`
	`output reg[15:0] Instruction;`
	`wire [15:0] Inst_Mem[0:65535];`
	`assign Inst_Mem[0] = 16'h0992; // and $1, $2`
	`assign Inst_Mem[1] = 16'h0000;`
		
		0 1
		1 2
		2 3
		3 4
		4 5
		5 6
		6 7
		7 8
			
			0 1
			1 0
			2 3
			3 4
			4 5
			5 6
			6 7
			7 8

OR: The instruction given to perform OR operation is as shown in fig. The OR operation is performed and the result is stored in the register. From the fig we can see that the values in the register 2 and register 3 are ORed and the result is stored in the register 1.	
	`input clk;`
	`input[15:0] Instruction_address;`
	`output reg[15:0] Instruction;`
	`wire [15:0] Inst_Mem[0:65535];`
	`assign Inst_Mem[0] = 16'h0993; // or $1, $2, $3`
	`assign Inst_Mem[1] = 16'h0000;`
		
		0 1
		1 2
		2 3
		3 4
		4 5
		5 6
		6 7
		7 8
			
			0 1
			1 7
			2 3
			3 4
			4 5
			5 6
			6 7
			7 8

Load Word (LW): The contents stored in the data memory are transferred to the register. The offset and base address of the memory is required and then the particular locations of memory are transferred into the register. The instruction for the load word is as shown in the fig.	
	`module Instruction_memory (clk, Instruction_address,Instruction);`
	`input clk;`
	`input[15:0] Instruction_address;`
	`output reg[15:0] Instruction;`
	`wire [15:0] Inst_Mem[0:65535];`
	`assign Inst_Mem[0] = 16'h8180; // lw $3,0($0) --> $3 = memory[$0 + 0];`
	`assign Inst_Mem[1] = 16'h0000;`
	The values stored in the data memory can be read and stored in a register. The offset is added with the content of any one of the required register and that content of the memory is transferred to the register
		
		0 1
		1 2
		2 3
		3 4
		4 5
		5 6
		6 7
		7 8
			
			0 1
			1 2
			2 3
			3 20
			4 5
			5 6
			6 7
			7 8

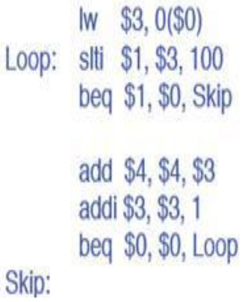
Looping:Set if Less Than Integer (slti):

It is used to record the result of a less than comparison with a constant. It compares the content of the register and immediate and records the Boolean result of the comparison in the destination register. If the contents of the register is less than the immediate the result is 1 (true) or else 0 (false). The instruction for the slti operation is as shown in the fig.

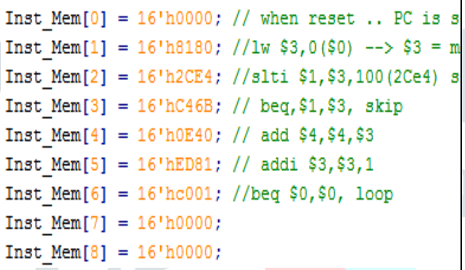


```
module Instruction_memory (clk, Instruction_address
input clk;
input[15:0] Instruction_address;

output reg[15:0] Instruction;


wire [15:0] Inst_Mem[0:65535];

assign Inst_Mem[0] = 16'h2CE4; // slti $1, $3, 100
assign Inst Mem[1] = 16'h0000;
```

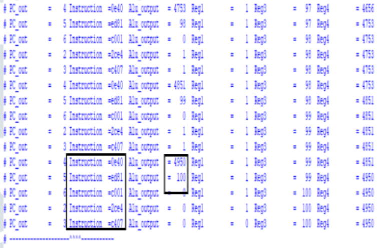
Test Code: A code snippet, is taken which has both branching and looping instructions. The performance of the processor is evaluated using this piece of code shown below



```
       lw   $3, 0($0)
Loop:  slti $1, $3, 100
       beq  $1, $0, Skip

       add  $4, $4, $3
       addi $3, $3, 1
       beq  $0, $0, Loop
Skip:
```

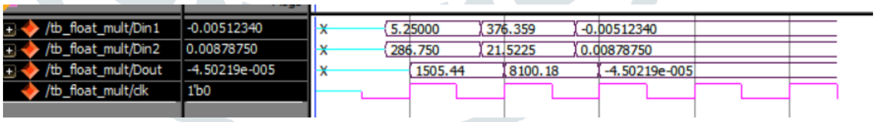


```
Inst_Mem[0] = 16'h0000; // when reset .. PC is s
Inst_Mem[1] = 16'h8180; //lw $3,0($0) --> $3 = m
Inst_Mem[2] = 16'h2CE4; //slti $1,$3,100(2Ce4) s
Inst_Mem[3] = 16'hC46B; // beq,$1,$3, skip
Inst_Mem[4] = 16'h0E40; // add $4,$4,$3
Inst_Mem[5] = 16'hED81; // addi $3,$3,1
Inst_Mem[6] = 16'hc001; //beq $0,$0, loop
Inst_Mem[7] = 16'h0000;
Inst_Mem[8] = 16'h0000;
```



```
# PC_out   =   4 Instruction =0e40 Alu_output = 4753 Reg1   =   1 Reg3   =  97 Reg4   = 4656
# PC_out   =   5 Instruction =ed81 Alu_output =   98 Reg1   =   1 Reg3   =  97 Reg4   = 4753
# PC_out   =   6 Instruction =c001 Alu_output =    0 Reg1   =   1 Reg3   =  98 Reg4   = 4753
# PC_out   =   2 Instruction =2ce4 Alu_output =    1 Reg1   =   1 Reg3   =  98 Reg4   = 4753
# PC_out   =   3 Instruction =c407 Alu_output =    1 Reg1   =   1 Reg3   =  98 Reg4   = 4753
# PC_out   =   4 Instruction =0e40 Alu_output = 4851 Reg1   =   1 Reg3   =  98 Reg4   = 4753
# PC_out   =   5 Instruction =ed81 Alu_output =   99 Reg1   =   1 Reg3   =  98 Reg4   = 4851
# PC_out   =   6 Instruction =c001 Alu_output =    0 Reg1   =   1 Reg3   =  99 Reg4   = 4851
# PC_out   =   2 Instruction =2ce4 Alu_output =    1 Reg1   =   1 Reg3   =  99 Reg4   = 4851
# PC_out   =   3 Instruction =c407 Alu_output =    1 Reg1   =   1 Reg3   =  99 Reg4   = 4851
# PC_out   =   4 Instruction =0e40 Alu_output = 4950 Reg1   =   1 Reg3   =  99 Reg4   = 4851
# PC_out   =   5 Instruction =ed81 Alu_output =  100 Reg1   =   1 Reg3   =  99 Reg4   = 4950
# PC_out   =   6 Instruction =c001 Alu_output =    0 Reg1   =   1 Reg3   = 100 Reg4   = 4950
# PC_out   =   2 Instruction =2ce4 Alu_output =    0 Reg1   =   1 Reg3   = 100 Reg4   = 4950
# PC_out   =   3 Instruction =c407 Alu_output =    0 Reg1   =   0 Reg3   = 100 Reg4   = 4950
# -----------------^^^^----------
```

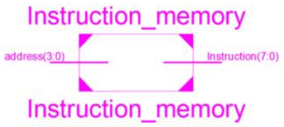
Floating Point Mode



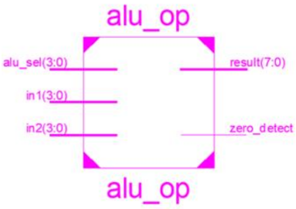
```
/tb_float_mult/Din1   -0.00512340      X  5.25000   376.359   -0.00512340
/tb_float_mult/Din2   0.00878750       X  286.750   21.5225   0.00878750
/tb_float_mult/Dout   -4.50219e-005    X    1505.44   8100.18   -4.50219e-005
/tb_float_mult/clk    1'b0
```

FPGA Implementation: To test the complete code on the evaluation kit was not possible, hence the code was split into 3 functional modules for testing on FPGA i.e Instruction Memory, ALU and Store and Load operation. Since the evaluation FPGA has 8 LEDs, 8 DIP switches and 5 Switch's, the code was accordingly modified so that the output could be seen on FPGA.

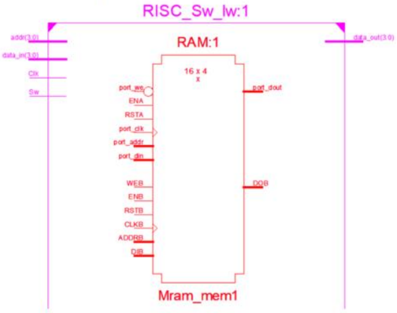
1. Instruction Memory



2. ALU



3. Load Store Operations



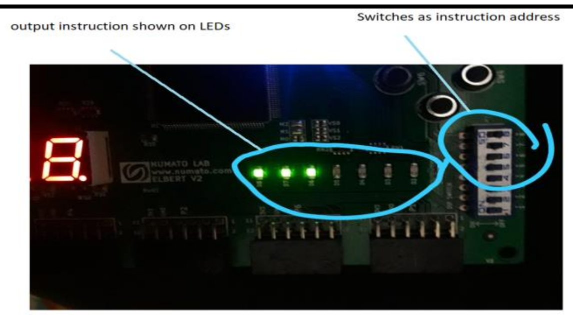

output instruction shown on LEDs
Switches as instruction address

Fig2: Implementation snap shots of FPGA results

V. CONCLUSION AND FUTURE SCOPE

Hence we have designed a 8-bit RISC processor incorporated with pipelining technique for performance enhancement, simulated using modelsim and have implemented the same on Xilinx Spartan 3E starter board FPGA at speed of more than 60MHz. The complete instruction set were tested individually and at last a code snippet is tested which consisted of all type of instructions. Pipelining technique had yield better results with respect to the speed of the execution of the processor. The proposed processor has the uplift of lower power dissipation, occupies lesser area and achieves faster concurrent programming execution.

REFERENCES

[1]Patterson and Hennessy, “Computer Organization and Design”, the hardware/software interface, 3rd edition, Pearson, 2004.

[2]Jikku Jeemon, “Pipelined 8-bit RISC Processor Design Using Verilog HDL on FPGA”, *IEEE International Conference On Recent Trends In Electronics Information Communication Technology, May 20-21, 2016*, India .

[3]Jikku Jeemon, “Low Power Pipelined 8-bit RISC Processor Design and Implementation on FPGA”, *International Conference on Control Instrumentation, Communication and Computational Technologies (lCCICCT),2015*.

[4]E. Ayeh, K. Agbedanu, Y. Morita, O. Adamo, and P. Guturu, “FPGA Implementation of an 8-bit Simple Processor” *IEEE region 5 conference,2008*, Kenya.

[5]Xuan Guan, Yunsi Fei, and Hai Lin, “Hierarchical Design of an Application-Specific Instruction Set Processor for High-Throughput and Scalable FFT Processing”, *IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, VOL. 20, NO. 3, MARCH*, 2012.

[6]Antonio, Zavala, Avante, Duarte, and Valencia , “RISC-Based Architecture for Computer Hardware Introduction”, *3rd International Conference on Computer Research and Development (ICCRD)*, 2011.

[7]Erich Wenger and Johann Großschadl, “ An 8-bit AVR-Based Elliptic Curve Cryptographic RISC Processor for the Internet of Things”, *45th Annual IEEE/ACM International Symposium on Microarchitecture Workshops*, 2012.

[8]Moisés Herrera and Francisco Vivero, “Asynchronous 8-Bit Processor Mapped into an FPGA Device”, *IEEE Colombian Conference on Communications and Computing (COLCOM)*, 2014.

[9]Lopamudra Samal and Chiranjibi Samal , “Designing a low power 8-bit Application Specific Processor”, *International conference on Green Computing Communication and Electrical Engineering (ICGCCEE)*, 2014.

[10]Austria, Sambile, Villega and Tabing, “Design of an 8-Bit Five Stage Pipelined RISC Microprocessor for Sensor Platform Application”, *Proc. of the 2017 IEEE Region 10 Conference (TENCON), Malaysia, November 5-8, 2017.*