# Modification of Instruction Set Architecture in a UTeMRISCII Processor

Ahmad Jamal Salim[#1], Nur Raihana Samsudin[*2], Sani Irwan Md Salim[#3] , Soo Yew Guan[#4]

[#]*Faculty of Electronic Engineering and Computer Engineering, Universiti Teknikal Malaysia Melaka*
*Hang Tuah Jaya, 76100 Durian Tunggal, Melaka, Malaysia*

***Abstract*— The development of application specific instruction set processor (ASIP) is a methodology in designing the processor system. The designing of processor system is focused on the internal architecture of the processor. By using the ASIP design, it can offer the optimum performance and also the flexibility in a processor architecture, but with limited application. However, by implementing the processor on Field Programmable Gate Array (FPGA), it could further extend the opportunity to reconfigure the architecture instantly. Therefore, this paper is about the implementation of the modification of a 16-bit wide instruction set for a simple 8-bit soft-core RISC processor called UTeMRISCII. The purpose of the project is to improve the ability of the processor by adding a new instruction set that can be able to perform basic digital signal processing (DSP) algorithm. For verification, a multiply-accumulate (MAC) instruction is created as the new customized instruction. The modification of the instruction set architecture is achieved by using Hardware Description Language (HDL) implementation. To validate the operation of new customized instruction in the software platform, the CPUSim software is used as the simulator to observe the output. Meanwhile, in the hardware platform, the new customized instruction is translated into processor design and verified using the Xilinx ISE software. The Xilinx Virtex-6 board is used to implement the processor. The simulation and hardware synthesis results proved that the new MAC instruction implementation performed correctly and produces correct outputs during the processor execution.**

***Keywords*— ASIP; modification instruction set; Multiply-accumulate.**

## I. Introduction

The application specific instruction set processor (ASIP) has the flexibility of general purpose processors, in using additional instruction set and programming field through its design [1, 2]. Therefore, to implement ASIP technique, a customized instruction sets is introduced through the instruction set extensions. The instruction set extensions technique is adapted for specific application. The instruction set extensions can be categorized into two approaches, which are complete customization and partial customization [2]. The complete customization approach requires a complete build on the instruction set architecture of a processor. Meanwhile, partial customization approach involves the extension of the existing instruction set architecture where a limited number of new instructions are added to tune it to perform specific functions [3]. Basically, both approaches are targeted is to simplify the processor design which takes into account the most useful instructions for the specific application and discarded the others in order to maximize the execution time.

The Reduced Instruction Set Computer (RISC) architecture is a CPU design that provides higher performance with much faster execution of each instruction [4, 5, 6]. In determining the performance parameters for RISC processors, improvement of maximum operating frequency and clock cycle per instruction have been considered. Therefore, from that point, the RISC processor provides the best platform in implementing instruction set extension approaches.

Implementation of a simple RISC processor on an FPGA platform would enable more opportunities to enhance the processor’s capability by adoption of ASIP design methodologies [7, 8]. In this platform, the internal architecture of RISC processor is described in Hardware Description Language (HDL) [9] that make it possible to make modification on memory allocation, register array and instruction decoder. Therefore, the designers are in control to determine the performance trade-off, resource utilization and detail configuration of the required instruction set.

For implementation of DSP application, FPGA-based DSP system design also incorporates hardware multipliers and memory blocks. Embedded DSP processors could be integrated into micro-controller platforms for a complete digital signal controller (DSC) system design [10]. Performance parameters measured including the area of the resulting architecture and the number of clock cycles are needed to evaluate the DSP operation.

The multiply-accumulate (MAC) unit is a common digital block used extensively in a microprocessor and digital signal processors for data-intensive applications [11]. For example, many filters, orthogonal frequency-division multiplexing algorithms, and channel estimators require FIR or FFT/IFFT computations that MAC units can accelerate efficiently. Inputs are fed to the multiplier, and successive products are summed by the accumulate adder [12]. Different architecture in the implementation of a MAC on the FPGA platform as hardware emulation of multipliers and accumulators have also been proposed with the intention of providing speed increased and area reduction [13].

## II. Methodology

### A. *UTeMRISCII Architecture*

An architecture of UTeMRISCII consists of several block modules which are compatible with an 8-bit RISC micro-controller. Fig.1 shows the module implementations that are required in UTeMRISCII processor including the corresponding ROM and RAM.

The Arithmetic and Logic Unit (ALU) module acts as the heart of the processor where the function is to control and synchronize the data path according to the programmed operations. The RISC processor core controls the data flow, synchronization and to fetch and execute the instruction set and data.

To generate the ROM and RAM, the CoreGen application module is utilized. The ROM functions as the instruction/program code storage while the RAM functions as the data/register storage. Normally, block ROM instantiation option is sufficient in generating memory blocks, however, for RAM, a distributed memory block option is used due to the asynchronous read and write design requirements.

Multiply and Accumulate instruction set is designated as a new instruction set of the processor. The MAC instruction executes the multiply and accumulate operation and the new instruction op-code is embedded to the instruction decoder module. Therefore, when the MAC instruction is fetched, the instruction decoder will be able to decode the instruction and the arithmetic execution will be performed by the ALU module.

Besides that, this instruction will also check for overflow condition. The overflow flag is set if the result of a MAC instruction is out of range of the signed arithmetic number between -32,768 to 32, 767.

Modification of the Instruction Set Architecture (ISA) is followed by an instruction modification flow chart as shown in Fig. 2. From this flowchart, the modification is started with the development of the UTeMRISCII architecture which is capable to accommodate a new instruction set. Then, the MAC instruction is created by modifying the ALU and the IDEC modules. A test program in assembly language file (.ASM) is developed which include the new MAC instruction. For verification purpose, the test program is simulated using CPUSim software. The (.ASM) file is then compiled and assembled to generate the hexadecimal file (.HEX). Subsequently, the .HEX file is converted to a coefficient file (.COE). This file is then used as an initialization file to be loaded to the ROM module.

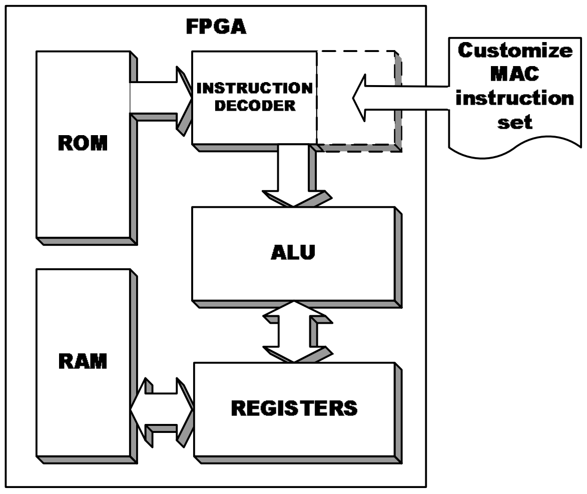


Fig. 1 Block Modules of UTeMRISCII Processor

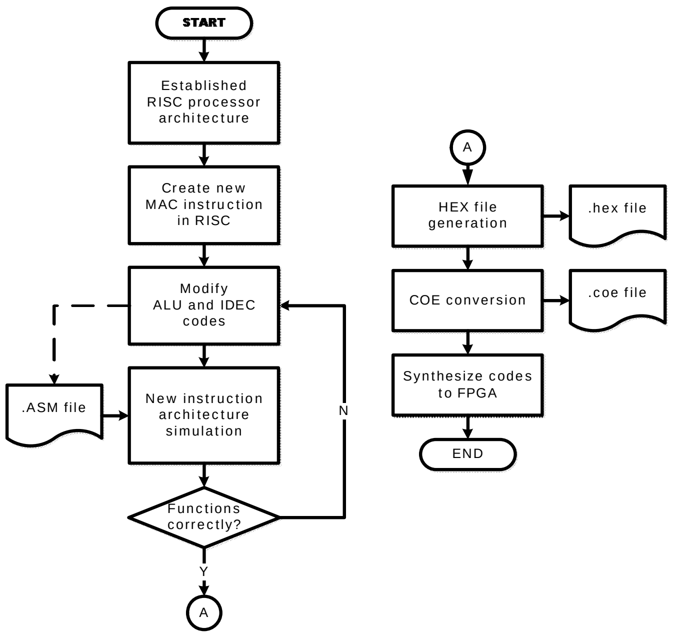

START
Established RISC processor architecture
Create new MAC instruction in RISC
Modify ALU and IDEC codes
.ASM file
New instruction architecture simulation
Functions correctly?
N
Y
A
A
HEX file generation
.hex file
COE conversion
.coe file
Synthesize codes to FPGA
END

Fig. 2 Instruction Set Modification Flow Chart

### B. *MAC Architecture*

The architecture of the MAC operation is shown in Fig. 3. Input signals a and b are the 8-bit multiplicand and multiplier respectively. The multiplier module computes the multiplication operation and produces a 16-bit result. The result is then accumulated with the previous MAC result and stored in the accumulator register. The final output is stored in two 8-bit register pair.

In the 8-bit UTeMRISCII processor architecture, the multiplier and adder function is configured and implemented in the ALU processor. Its operation utilizes a signed 2's complement multiplier and a full adder. The operation requires one clock cycle to implement the multiplication and accumulation process. In the clock cycle, the multiplier provides a double precision result of two single precision operand, while the accumulator performs arithmetic functions with double precision input and output.

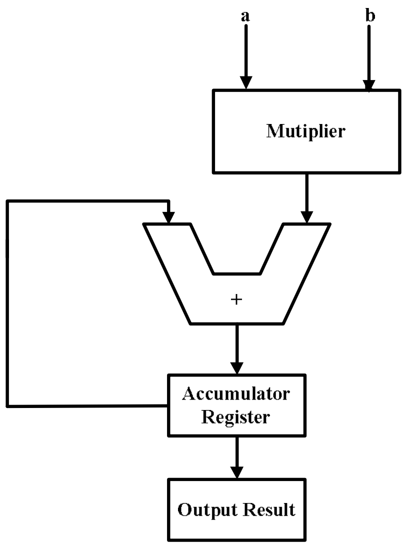


Fig. 3 Basic MAC Hardware Architecture

## C. *Instruction Set Modification*

For the UTeMRISCII processor, the width of the instruction set is 16-bit wide, which consists of a combination of opcode and literal bit or file register or memory address. Direct and indirect addressing modes are also supported in this processor. The format of the instruction set architecture of the RISC processor is shown in Fig.4. There are three instruction set format utilized in the UTEMRISCII. The byte-oriented file register consists of instruction opcode, file register/memory address and directional bit. The directional bit indicates the destination of the ALU output where bit ‘1’ will store the output back to the file register/memory and bit ‘0’ will store the output to working register. The bit-oriented file register operation involved bit testing indicated by select bit (‘b’) on the corresponding file register/memory address. Literal value operation only involved opcode and 10-bit literal value (‘k’). The literal value indicates either raw data or label that contained memory location address (especially in goto and call instruction).

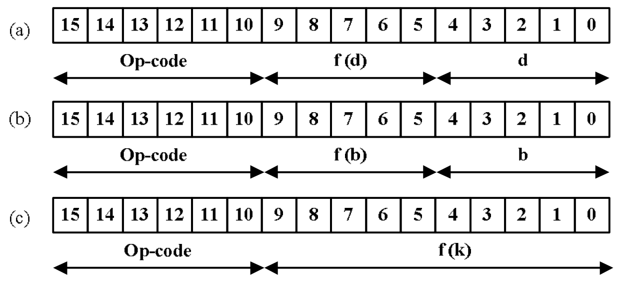
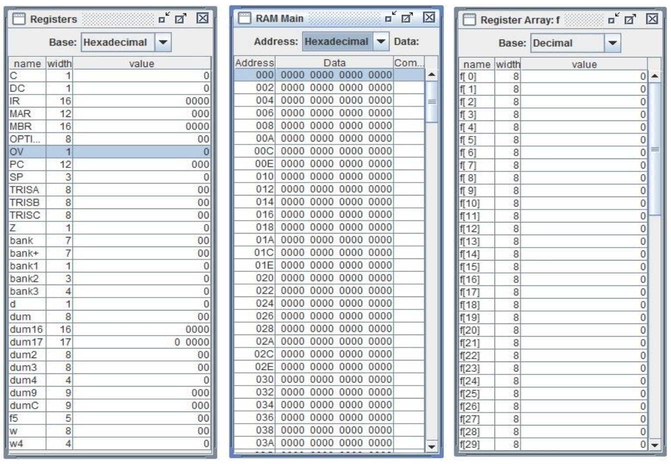


Fig.4 General Instruction Format (a) Byte-oriented File Register Operation, (b) Bit-oriented File Register Operation and (c) Literal Value Operation

## D. *Instruction Set Simulation*

The CPUSim software is a tool that is used to verify and simulate the new instruction set. CPUSim software is a highly customized Java application computer architecture simulator [14] , which provides features to insert customized instruction in assembly language with the ability to perform simulation at microcode level to a variety of CPU architectures including RISC processors. Therefore, the new MAC instruction is configured inside the simulator. Fig. 5 shows the layout of the CPUSim simulator window.


Registers

Base: Hexadecimal

| name | width | value |
|---|---|---|
| C | 1 | 0 |
| DC | 1 | 0 |
| IR | 16 | 0000 |
| MAR | 12 | 000 |
| MBR | 16 | 0000 |
| OPTI... | 8 | 00 |
| OV | 1 | 0 |
| PC | 12 | 000 |
| SP | 3 | 0 |
| TRISA | 8 | 00 |
| TRISB | 8 | 00 |
| TRISC | 8 | 00 |
| Z | 1 | 0 |
| bank | 7 | 00 |
| bank+ | 7 | 00 |
| bank1 | 1 | 0 |
| bank2 | 3 | 0 |
| bank3 | 4 | 0 |
| d | 1 | 0 |
| dum | 8 | 00 |
| dum16 | 16 | 0000 |
| dum17 | 17 | 0 0000 |
| dum2 | 8 | 00 |
| dum3 | 8 | 00 |
| dum4 | 4 | 0 |
| dum9 | 9 | 000 |
| dumC | 9 | 000 |
| f5 | 5 | 00 |
| w | 8 | 00 |
| w4 | 4 | 0 |

RAM Main

Address: Hexadecimal Data:

| Address | Data | Com... |
|---|---|---|
| 000 | 0000 0000 0000 0000 | |
| 002 | 0000 0000 0000 0000 | |
| 004 | 0000 0000 0000 0000 | |
| 006 | 0000 0000 0000 0000 | |
| 008 | 0000 0000 0000 0000 | |
| 00A | 0000 0000 0000 0000 | |
| 00C | 0000 0000 0000 0000 | |
| 00E | 0000 0000 0000 0000 | |
| 010 | 0000 0000 0000 0000 | |
| 012 | 0000 0000 0000 0000 | |
| 014 | 0000 0000 0000 0000 | |
| 016 | 0000 0000 0000 0000 | |
| 018 | 0000 0000 0000 0000 | |
| 01A | 0000 0000 0000 0000 | |
| 01C | 0000 0000 0000 0000 | |
| 01E | 0000 0000 0000 0000 | |
| 020 | 0000 0000 0000 0000 | |
| 022 | 0000 0000 0000 0000 | |
| 024 | 0000 0000 0000 0000 | |
| 026 | 0000 0000 0000 0000 | |
| 028 | 0000 0000 0000 0000 | |
| 02A | 0000 0000 0000 0000 | |
| 02C | 0000 0000 0000 0000 | |
| 02E | 0000 0000 0000 0000 | |
| 030 | 0000 0000 0000 0000 | |
| 032 | 0000 0000 0000 0000 | |
| 034 | 0000 0000 0000 0000 | |
| 036 | 0000 0000 0000 0000 | |
| 038 | 0000 0000 0000 0000 | |
| 03A | 0000 0000 0000 0000 | |

Register Array: f

Base: Decimal

| name | width | value |
|---|---|---|
| f[ 0] | 8 | 0 |
| f[ 1] | 8 | 0 |
| f[ 2] | 8 | 0 |
| f[ 3] | 8 | 0 |
| f[ 4] | 8 | 0 |
| f[ 5] | 8 | 0 |
| f[ 6] | 8 | 0 |
| f[ 7] | 8 | 0 |
| f[ 8] | 8 | 0 |
| f[ 9] | 8 | 0 |
| f[10] | 8 | 0 |
| f[11] | 8 | 0 |
| f[12] | 8 | 0 |
| f[13] | 8 | 0 |
| f[14] | 8 | 0 |
| f[15] | 8 | 0 |
| f[16] | 8 | 0 |
| f[17] | 8 | 0 |
| f[18] | 8 | 0 |
| f[19] | 8 | 0 |
| f[20] | 8 | 0 |
| f[21] | 8 | 0 |
| f[22] | 8 | 0 |
| f[23] | 8 | 0 |
| f[24] | 8 | 0 |
| f[25] | 8 | 0 |
| f[26] | 8 | 0 |
| f[27] | 8 | 0 |
| f[28] | 8 | 0 |
| f[29] | 8 | 0 |

Fig. 5 Layout of CPUSim simulator window: (a) RAM of test program, (b) Register and (c) Memory Register windows

## E. *FPGA Implementation*

Implementation of FPGA is a part of the process to verify the design of the MAC instruction on FPGA chip. All logic modules are then synthesized, placed and routed and executed in Virtex-6 FPGA chip. During the FPGA implementation, key parameters, such as the maximum clock frequency and resource utilization are observed together. The coefficient file as mentioned in the previous section, is used as the initialization file to the ROM module. During the operation, ALU fetched the MAC instruction from instruction decoder module and then execute the corresponding MAC sequences as per programmed. In Hardware Description Languange (HDL) simulation, the design is simulated through the testbench environment. The testbench is used to verify the correctness of design before simulation using the ISim simulator. Afterwards, the design is run through several processes including translate, mapping and “place and route” to generate the configuration file. The configuration file is developed to program the configuration device before hardware implementation process. To validate and debug the design, the ChipScope analyzer is utilized. Before the debugging process, the design is implemented in the FPGA chip on the Virtex-6. The Integrated Logic Analyzer (ILA) core is inserted within the project design in order to observe the internal signals of the UTeMRISCII processor. The internal signals are then is displayed as waveforms at the ChipScope Analyzer window.

## III. Result and Discussion

The overall result of MAC architecture is shown in Fig. 6. The input of a and b contain the values of working register (w) and File Select Register (FSR) respectively. The values of a and b are in two's complement. Multiplication is done in unsigned numbers (a1 and b1) and the result is stored in m1 register. The result is presented back in signed number format in register m1. Next, to perform accumulate operation, the value of m2 is added to the previous value of accumulator (accuold) and stored the result in accu register. The accumulated result is stored into register pair y and y2, where y is stored as the higher 8 bit data, while y2 is stored as the lower 8-bit data. Indirectly, the data in register y and y2 are stored in w and FSR. Therefore, the MAC instruction is repeated in loops until the result triggers the overflow flag.

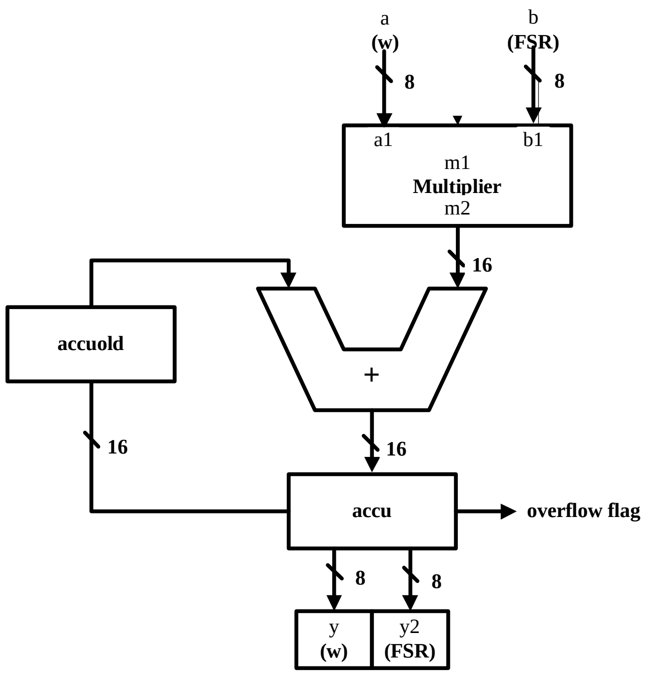
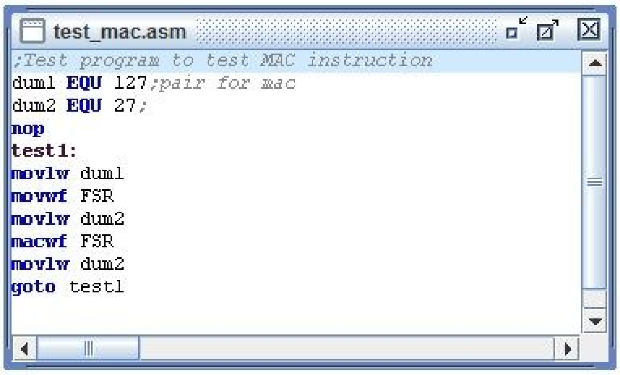


Fig. 6 MAC hardware architecture

### F. CPUSim simulation

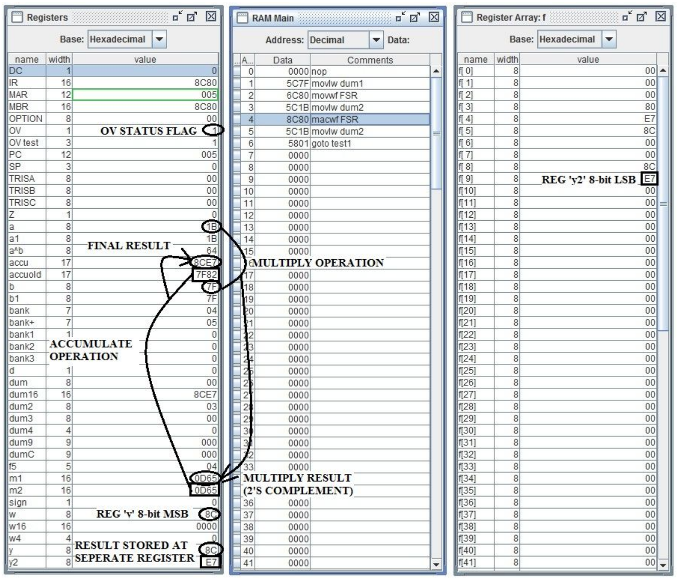
No In CPUSim simulator, the new MAC's machine instruction is defined by assembling the instruction in the test program at the main window. The simulator is then run in debug mode where all the instructions in the assembly language program are executed sequentially. The outputs of the simulation are observed at register level through its register windows. Fig. 7 shows the assembly language program that includes the MAC instruction. The MAC instruction then went through a continuous loop to verify its operation. The assembly language program is compiled to generate the hexadecimal file.

Fig. 8 shows the correct implementation of the MAC instruction where the values of a and b is generated from dum2 and dum1 with values in hexadecimal are 1B and 7F respectively. The values of a and b are converted to its magnitude value before the multiplication is done. The multiplication result is stored in m1 (0D65). The result is converted back to its signed value and stored in register m2. Next, accumulate operation is performed by adding value of m2 (0D65) with value of previous accumulate result or accuold (7F82), which is also the value of (y, y2) from the previous MAC iteration. Then the accumulation result (8CE7) is updated to accu register, and result is separated in 8-bit data stored to y (8C) and y2 (E7) registers. However, overflow flag status is indicating '1' in this iteration, which means the final result of this iteration has exceeded the range of -32,768 to 32, 767 (&H8000 to &H7FFF) sign bit number.



```
;Test program to test MAC instruction
dum1 EQU 127;pair for mac
dum2 EQU 27;
nop
test1:
movlw dum1
movwf FSR
movlw dum2
macwf FSR
movlw dum2
goto test1
```

Fig. 7 Assembly language program for MAC instruction execution



| name | width | value |
|---|---|---|
| DC | 1 | 0 |
| IR | 16 | 8C80 |
| MAR | 12 | 005 |
| MBR | 16 | 8C80 |
| OPTION | 8 | 00 |
| OV | 1 | 1 |
| OV test | 3 | 1 |
| PC | 12 | 005 |
| SP | 3 | 0 |
| TRISA | 8 | 00 |
| TRISB | 8 | 00 |
| TRISC | 8 | 00 |
| Z | 1 | 0 |
| a | 8 | 1B |
| a1 | 8 | 1B |
| a^b | 8 | 64 |
| accu | 17 | 8CE7 |
| accuold | 17 | 7F82 |
| b | 8 | 7F |
| b1 | 8 | 7F |
| bank | 7 | 04 |
| bank+ | 7 | 05 |
| bank1 | 1 | 0 |
| bank2 | 1 | 0 |
| bank3 | 1 | 0 |
| d | 1 | 0 |
| dum | 8 | 00 |
| dum16 | 16 | 8CE7 |
| dum2 | 8 | 03 |
| dum3 | 8 | 00 |
| dum4 | 4 | 0 |
| dum9 | 9 | 000 |
| dumC | 9 | 000 |
| f5 | 5 | 04 |
| m1 | 16 | 0D65 |
| m2 | 16 | 0D65 |
| sign | 1 | 0 |
| w | 8 | 8C |
| w16 | 16 | 0000 |
| w4 | 4 | 0 |
| y | 8 | 8C |
| y2 | 8 | E7 |

| A... | Data | Comments |
|---|---|---|
| 0 | 0000 | nop |
| 1 | 5C7F | movlw dum1 |
| 2 | 6C80 | movwf FSR |
| 3 | 5C1B | movlw dum2 |
| 4 | 8C80 | macwf FSR |
| 5 | 5C1B | movlw dum2 |
| 6 | 5801 | goto test1 |
| 7 | 0000 | |
| 8 | 0000 | |
| 9 | 0000 | |
| 10 | 0000 | |
| 11 | 0000 | |
| 12 | 0000 | |
| 13 | 0000 | |
| 14 | 0000 | |
| 15 | 0000 | |
| [illegible] | [illegible] | |
| 17 | 0000 | |
| 18 | 0000 | |
| 19 | 0000 | |
| 20 | 0000 | |
| 21 | 0000 | |
| 22 | 0000 | |
| 23 | 0000 | |
| 24 | 0000 | |
| 25 | 0000 | |
| 26 | 0000 | |
| 27 | 0000 | |
| 28 | 0000 | |
| 29 | 0000 | |
| 30 | 0000 | |
| 31 | 0000 | |
| 32 | 0000 | |
| 33 | 0000 | |
| [illegible] | [illegible] | |
| [illegible] | [illegible] | |
| 36 | 0000 | |
| 37 | 0000 | |
| 38 | 0000 | |
| 39 | 0000 | |
| 40 | 0000 | |
| 41 | 0000 | |

| name | width | value |
|---|---|---|
| f[ 0] | 8 | 00 |
| f[ 1] | 8 | 00 |
| f[ 2] | 8 | 00 |
| f[ 3] | 8 | 80 |
| f[ 4] | 8 | E7 |
| f[ 5] | 8 | 8C |
| f[ 6] | 8 | 00 |
| f[ 7] | 8 | 00 |
| f[ 8] | 8 | 8C |
| f[ 9] | 8 | E7 |
| f[10] | 8 | 00 |
| f[11] | 8 | 00 |
| f[12] | 8 | 00 |
| f[13] | 8 | 00 |
| f[14] | 8 | 00 |
| f[15] | 8 | 00 |
| f[16] | 8 | 00 |
| f[17] | 8 | 00 |
| f[18] | 8 | 00 |
| f[19] | 8 | 00 |
| f[20] | 8 | 00 |
| f[21] | 8 | 00 |
| f[22] | 8 | 00 |
| f[23] | 8 | 00 |
| f[24] | 8 | 00 |
| f[25] | 8 | 00 |
| f[26] | 8 | 00 |
| f[27] | 8 | 00 |
| f[28] | 8 | 00 |
| f[29] | 8 | 00 |
| f[30] | 8 | 00 |
| f[31] | 8 | 00 |
| f[32] | 8 | 00 |
| f[33] | 8 | 00 |
| f[34] | 8 | 00 |
| f[35] | 8 | 00 |
| f[36] | 8 | 00 |
| f[37] | 8 | 00 |
| f[38] | 8 | 00 |
| f[39] | 8 | 00 |
| f[40] | 8 | 00 |
| f[41] | 8 | 00 |

Fig. 8 CPUSim output result of MAC instruction implementation

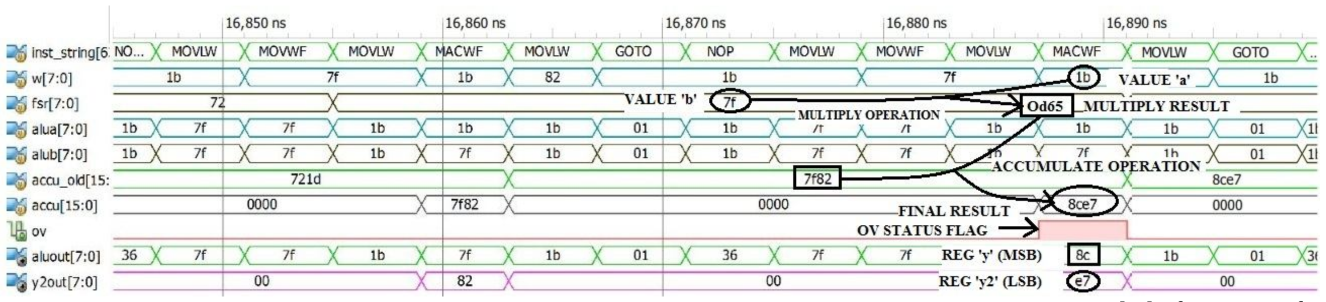


Fig. 9 ISim output result of MAC instruction implementation

e a good platform to perform

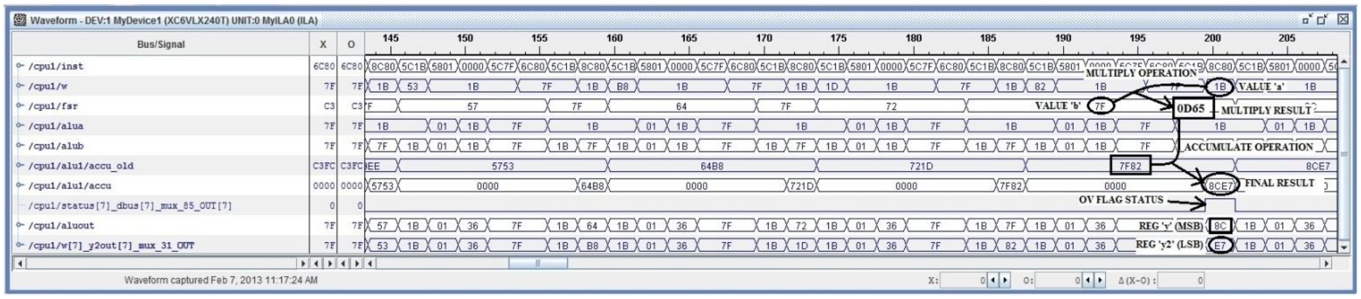


Fig. 10 ChipScope output result of MAC instruction implementation

the low-end processor.

## B. Xilinx ISE.

The UTeMRISCII processor block modules are initiated and synthesized using Xilinx Virtex-6 FPGA platform with the Xilinx ISE integrated environment. The integrated ISim simulator is used to validate the design with the MAC instruction. Fig. 9 is the waveform result of the ISim simulator where the result is found to match with the result of CPUSim simulation. In CPUSim software, the input of design is defined as a and b, while in ISim simulator the input is defined as w and f. The final result of y and y2 is labeled as aluout and y2out respectively.

## C. ChipScope Analyzer

The ChipScope analyzer is used to debug the design through the Integrated Logic Analyzer (ILA) core. The waveform window by Chipscope Analyzer is used to display the output result of the design which perform the MAC instruction in the FPGA chip. Fig. 10 shows the correct output result that was generated from ChipScope Analayzer, and matched with the CPUSim simulation and ISim simulation results.

## IV. Conclusion

This paper describes the capability of a simple UTeMRISCII processor architecture to execute a new instruction through an ISA modification. The soft-core processor which is implemented in an FPGA platform provides the flexibility and compatibility in application specific processor design. The new MAC instruction is successfully simulated, implemented and verified with accurate result throughout the whole process.

## Acknowledgment

The authors would like to thank Universiti Teknikal Malaysia Melaka and Ministry of Higher Education Malaysia for the financial support given through the research grant number FRGS/2012/FKEKK/TK02/02/1/F00126.

## References

[1] J. Ball, "Designing Soft-Core Processors for FPGAs Processor Design," in Processor Design: System-on-Chip Computing for ASICs andFPGAs, J. Nurmi, Ed., 1st ed: Springer Netherlands, 2007, pp. 229-256.

[2] C. Galuzzi and K. Bertels, "The Instruction-Set Extension Problem: A Survey," ACM Trans. Reconfigurable Technol. Syst., vol. 4, pp. 1-28, 2011.

[3] L. Barthe, L. V. Cargnini, P. Benoit, and L. Torres, "The SecretBlaze: A Configurable and Cost-Effective Open-Source Soft-Core Processor," in IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011, pp. 310 313.

[4] P. S. Mane, *et al.*, "Implementation of RISC Processor on FPGA," in *Industrial Technology, 2006. ICIT 2006. IEEE International Conference on*, 2006, pp. 2096-2100.

[5] L. Barthe, *et al.*, "The SecretBlaze: A Configurable and Cost-Effective Open-Source Soft-Core Processor," in *IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW)*, 2011, pp. 310-313.

[6] P. S. Mane, I. Gupta, and M. K. Vasantha, "Implementation of RISC Processor on FPGA," in IEEE International Conference on Industrial Technology 2006, pp. 2096-2100.

[7] J. S. Lee and M. H. Sunwoo, "Design of New DSP Instructions and Their Hardware Architecture for High-Speed FFT," The Journal of VLSI Signal Processing, vol. 33, pp. 247-254, 2003.

[8] W. Wenxiang, L. Ling, Z. Guangfei, L. Dong, and Q. Ji, "An Application Specific Instruction Set Processor optimized for FFT," in IEEE 54th International Midwest Symposium on Circuits and Systems (MWSCAS), 2011, pp. 1-4.

[9] T. Coonan. (20 December 2011). Verilog Synthetic PIC. Available:http://www.mindspring.com/~tcoonan/newpic.html

[10] M. M. Mansour, *et al*. (2011) Trends in Design and Implementation of Signal Processing Systems [In the Spotlight]. *IEEE Signal Processing Magazine*. 192-193.

[11] H. Tung Thanh, M. Själander, and P. Larsson-Edefors, "A High-Speed, Energy-Efficient Two-Cycle MultiplyAccumulate(MAC) Architecture and Its Application to aDouble-Throughput MAC Unit," *IEEE Transactions on Circuits and Systems I: Regular Papers,* vol. 57, pp. 30733081,2010.

[12] C. Lo Sing, A. Miri, and Y. Tet Hin, "Efficient FPGAimplementation of FFT based multipliers," in *Canadian Conference on Electrical and Computer Engineering*, 2005,pp. 1300-1303.

[13] A. Abdelgawad and M. Bayoumi, "High Speed and Area Efficient MultiplyAccumulate(MAC)UnitforDigitalSignalProssingApplications," in *IEEE International Symposium on Circuits and Systems* 2007, pp. 3199-3202.

[14] D. Skrien, "CPU Sim 3.1: A tool for simulating computer architectures for computer organization classes," *Journal onEducational Resources in Computing (JERIC),* vol. 1, pp. 46-59, 2001.