International Journal of Engineering & Technology, 7 (4.36) (2018) 373-382

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper

Design & Simulation Of 64-Bit Hybrid Processor Instruction Set Using Verilog

[1]Harish M S & [2]Jayadevappa D

[1]Research Scholar, Dept. of Electronics Engineering, Jain University, Bengaluru, India
[2]Professor, Dept. of E&IE, JSS Academy of Technical Education, Bengaluru, India
**Corresponding author E-mail: devappa22gmail.com*

Abstract

As a part of my ongoing research on implementation of multi core hybrid processor on FPGA, I have developed data flow designs for most popularly used 20 processor instructions. I have made digital design, wrote code in Verilog HDL and simulated all the 20 instructions using Xilinx ISE 14.5. The data flow designs, symbolic representation and simulation results are explained in detail in this technical paper. This is partial implementation of Hybrid Processor & the other sub modules implementation on Xilinx FPGA will be published in my subsequent technical paper.

Keywords*. Image Segmentation, MRI, Contourlet transform, Active contours.*

1. Introduction

As it was discussed in my previous paper [7], FPGA is emerging as a rapid prototype for its several advantages as was listed [7]. Many researchers have undertaken several partial implementation of RISC processor using FPGA. Processor design can be done using any one of the following approaches.

Architecture based (Top down approach: The basic core architecture is kept as reference and instructions are derived based on this basic architecture. This means, better the architecture design, powerful the instructions.

Instruction based (Bottom up approach): here the approach is to list out the most used instructions of popular processors & implement them one by one. By putting all these implementation of instructions together, will emerge architecture at the top level. Hence in this approach incremental growth of instruction implementation will lead to mega processor architecture.

State Machine approach: In this approach, each instruction is treated as a fixed state machine. This means various instructions leads to various field state machines. The processor states can be like- FETCH, DECODE, EXECUTE, READ, WRITE, INTERRUPT, SEND, RECEIVE, ROTATE, IN , OUT etc.

Fixed Function Special Processors: This approach is used for designing custom processor with very specific & dedicated applications. Example: Video processor, audio processor, mobile processor, data acquisition, instrumentation & measurements – all these applications utilize processors only for high speed data processing, dedicated fixed functions or instructions.

All general purpose Processors will have several features set which are rarely used. Only 10% instructions are used 90% of the times. Hence this leads to enormous overhead or wastage of logic/ features /architecture /area/ cost/ delay etc leading to performance degradation of the overall processor application.

Hence there is a strong need to develop application specific / application driven/customized processor for specific applications or fixed task based high performance processor to provide high performance & fixed functionality. Also, present day Processors needs features of flexibility - to add or remove features or instructions or functionality and plug & play or modular architecture. My proposed "Customized and Scalable Hybrid Processor design" is the innovative 5^{th} approach.

2. Other Related Research Work

Till date, several attempts has been made to realize subset of RISC instruction set and several attempts has been made by various researchers to propose some minimal RISC architectures for various bit length like 8 bit, 32 bit, 64 bit using Verilog and VHDL on FPGA.

Mrudul S. Ghaturle, Prof. R. D. Kadam [1] et al, have claimed to have design and simulated decoder unit of 32 bit RISC processor to support R, I,, J, I/O type instructions with data path diagrams. They have indicated simulation results and RTL schematic of the decoder.

Mohammad Gousuddin H Maniya, SujathaHiremath [2] et al, have implemented a 64 bit CISC processor on FPGA to support short and long instruction formats and have implemented a double precision floating point multiplier and have indicated the resulting simulation waveforms.

PriyavratBhardwaj [3] et al, has proposed a MIPS Instruction Set Architecture (ISA) with instruction set categorized into Register, Immediate, Jump (RJI) instruction with 32 bit op-codes and have proposed data path diagrams for RJI instructions. They have also suggested 5 stage pipelined architecture with instruction fetch, decode, execute memory read and write operation. They claim to have implemented this 32 bit RISC processor on FPGA and have indicated RTL design schematic of that MIPS architecture.

Anu Mariam John, ShilpiVarshney [4] et al, have implemented 32 bit CISC processor for multiplication operation. They have used booth multiplier with CISC architecture. They have used Verilog HDL and Xilinx Spartan 3 board with a maximum clock speed of 177MHz and result is stored in 64 bit register.

AkshathaRai K, Basavaraj H[5] et al, have suggested an innovative design for the implementation of dual core RISC architecture and have proposed a pipelined architecture with separate data and program memory. They have implemented a 17 bit RISC processor on Xilinx FPGA and have indicated RTL schematic and simulation results for memory read and write memory.

S. Suresh, R. Ganesh [6] et al, have proposed 8 bit single cycle processor with 10 bit address bus and four stage pipelined data flow (instruction fetch, instruction decoder and operand fetch, execute and write back). They have indicated simulation results for addition, subtraction and multiplication with RTL schematics.

Saraswthi P, M K Chandrasen [7] et al, supposed to have implemented 32 bit CISC CPU architecture on FPGA with architecture logic unit, accumulator, 32 bit memory unit, 32 bit MUX, instruction register, program counter and indicated simulation screenshots.

WojciechWójcik, JacekDługopolski [8] et al, has attempted to implement a multi core processor on FPGA using parallel processing characteristics. They have also experimented on number of parallel processors leading to the overall speed of processor operations and also characterized problem size versus efficiency of processors.

Vijay R. Wadhankar, VaishaliTehre [9] et al, have attempted to implement RISC processor on FPGA and suggested a new architecture and specific design for instruction and control unit. They have shown simulation results of control unit for memory read and write operations.

A. Key findings of survey

After going through several technical papers, my observation is that there is no clear cut approach on whether fixed architectures will lead to powerful instruction set implementations (top down approach) or set of powerful and useful instructions will lead to an open ended architecture(bottom up approach). There is a big dilemma for Processor system designers.

During my exhaustive survey, about various types of Processors, their functionality, feature set, instruction set, interrupts, associated special features, Processor design approach, platforms for implementation etc., In my already published 3 survey papers [10] [11] [12].

Many of the above said processor implementation attempts, I did not come across any complete processor architecture to support contemporary instruction set implementation on FPGA using popular HDL (Verilog/VHDL). Also, none of the other related research works have explained the processor design implementation on FPGA at micro level or at data flow level or at instruction implementation levels.

Hence there is a strong and serious need to attempt design, simulation, implementation & prototype testing ofa scalable general purpose Processor architecture to support required useful instruction set implementation on FPGA meaningful and serious approach is required to realize.

3. Proposed Methodology

Both RISC & CISC Processor Architecture have their own merits & demerits & neither RISC nor CISC standalone Processor can produce a complete solution to the present day computational needs, hence there is a strong need of Hybrid Processor. Our proposed architecture will utilize all the best features of both RISC and CISC.

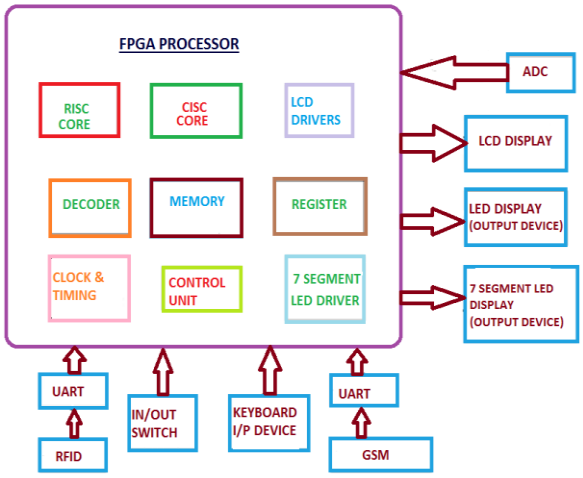


Fig. 1 Proposed Multi core Hybrid Processor Architecture.

Keeping the above technical issues in mind, an efficient Hybrid Processor with the best features of RISC and CISC is proposed & the top level Systems design is as indicated below.

A. Research Design of Hybrid Multi Core Processor

Instruction set and Op-code assignment: The table indicates the list of instruction implemented on FPGA. The table also indicates the operation function of each instruction, along with respective op-codes and instruction decoder output. As indicated in the figure 1, for each instruction respective code is applied to the instruction decoder. The instruction decoder, depending on the op-code will enable only 1 out of 30 outputs which in turn will enable the respective instruction dataflow logic for hardware.

Table 1. Instruction set and Op-code assignment

Sl. No.	Instructions	Operation function	Opcode	Instruction decoder output
1	reset	Reset the registers	000000	I_1E=1 & other I_nE=0
2	Load acc	load data into accumulator	101000	I_2E=1 & other I_nE=0
3	read acc	read data from accumulator	000111	I_3E=1 & other I_nE=0
4	Load regA	load data into reg A	000001	I_4E=1 & other I_nE=0

5	movacc,regA	move data from reg A to acc	001011	I_5E=1 & other I_nE=0
6	Add acc, regA	add accumulator with reg A	010001	I_6E=1 & other I_nE=0
7	sub acc, regA	Sub accumulator with reg A	010100	I_7E=1 & other I_nE=0
8	NOT acc	Not accumulator	011010	I_8E=1 & other I_nE=0
9	AND acc, regA	AND accumulator with reg A	010111	I_9E=1 & other I_nE=0
10	OR acc, regA	OR accumulator with reg A	011011	$I_{10}E$=1 & other I_nE=0
11	EXOR acc, regA	XOR accumulator with reg A	011110	$I_{11}E$=1 & other I_nE=0
12	Exchange regA, regB	Exchange the contents of reg A and reg B	000100	$I_{12}E$=1 & other I_nE=0
13	INC acc	Increment the content of accumulator	110000	$I_{13}E$=1 & other I_nE=0
14	DEC acc	decrement the content of accumulator	110001	$I_{14}E$=1 & other I_nE=0
15	Shift left acc, n	Shift accumulator left by n	100001	$I_{15}E$=1 & other I_nE=0
16	Shift right acc, n	Shift accumulator right by n	100010	$I_{16}E$=1 & other I_nE=0
17	Rotate left acc, n	rotate accumulator left by n	100011	$I_{17}E$=1 & other I_nE=0
18	Rotate right acc, n	rotate accumulator right by n	100100	$I_{18}E$=1 & other I_nE=0
19	Addcacc, regA	add accumulator with reg A and carry	101010	$I_{19}E$=1 & other I_nE=0
20	Subbacc, regA	Sub accumulator with reg A and carry	101101	$I_{20}E$=1 & other I_nE=0
21	comp acc, reg A	Compare the contents of accumulator and regA	100110	$I_{21}E$=1 & other I_nE=0

Instruction Decoder: The above figure indicates an instruction decoder indigenously designed to handle 2^5 instructions. Based on the 5 bit command or Opcode any one of the 32 outputs of the decoder will get enabled i.e., for each Opcode applied as command to the instruction decoder, one particular output of the decoder will go high or enabled and remaining 31 outputs will be held low or disabled. The whole scheme works as per the command and enable assignments done in the table.

These single enables of each command will in turn enable required logic to execute the corresponding instructions as per the assignment table I. This instruction decoder design is fully scalable and can support hundreds of instructions with the increase in number of command bits. The number of instructions will be equal to the number of decoder outputs.

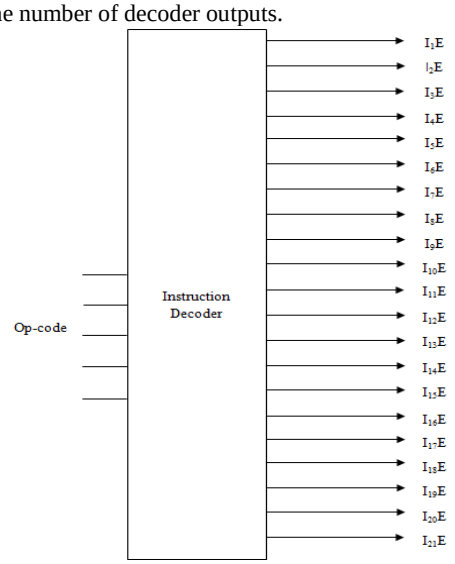


Fig. 2. Instruction Decoder

4. Experimental Results

Popular Instructions set Implementation using VERILOG HDL

- LOAD Accumulator, datain
- READ Accumulator, dataout

In all the simulation waveform, clk is clock, command [5:0] is 6 bit Opcode or command or each instruction, datain1 [63:0] is 64 bit data input bus, accumulator [64:0] is 65 bit accumulator and dataout[63:0] is 64 bit data output bus. When reset is applied 0's are applied to datain1, accumulator, and dataout to clear all the registers.

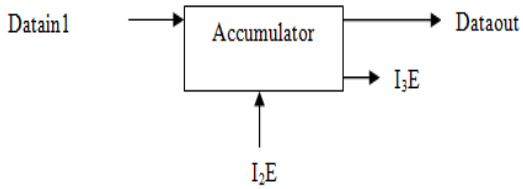
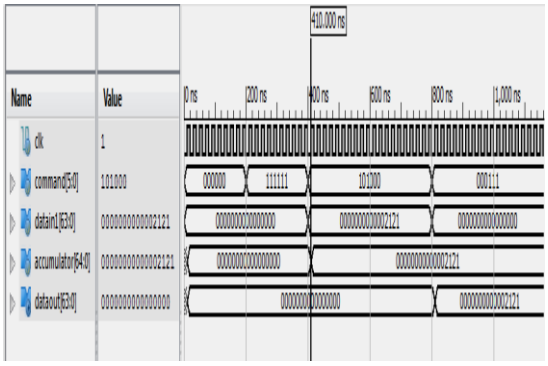


Fig. 3. Digital design for Instruction - Load Accumulator, Datain and Read Accumulator

Design shows the digital design required to implement load and read accumulator instructions.

As seen from the waveform, clock can be of any suitable frequency. Command [5:0] is 6 bit and is unique to specific instructions. The command bit assignment is as per the table.

- Between 0 to 200ns, reset is applied to command, datain1 [63:0], accumulator [64:0] and dataout [63:0].
- Between 200 to 400ns, command [5:0] is loaded with 111111 and no operation is performed.
- Between 400 to 800ns, command [5:0] is loaded with 101000 and datain1[63:0] is loaded with 2121 which indicates load accumulator instructions operation/execution needs to be performed by the processor. Command [5:0] 101000 will enable data loading to accumulator. After some latency, datain1 [63:0] value is moved to accumulator [64:0].
- After 800ns onwards, command [5:0] is now changed to 000111, which indicates READ accumulator instruction operation need to be performed by the processor and accordingly the already loaded accumulator data will be moved to dataout.



Digital Design for Instruction Move Accumulator (Move Accumulator, Reg A):

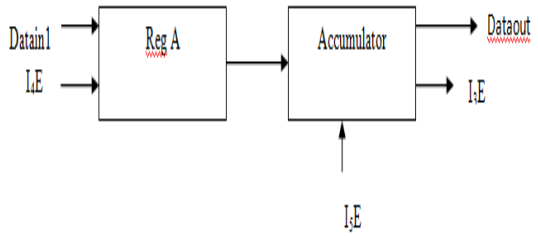
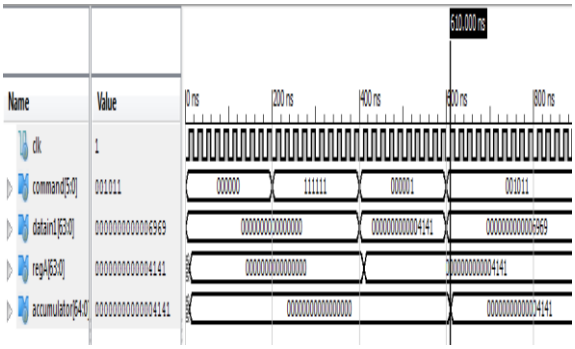


Fig. 4 Digital design for Instruction - Move Accumulator, Reg A.



The above figure indicates the simulation waveform for move accumulator, regA instruction.

- Between 0 to 200ns, reset is applied to command, datain1 [63:0], accumulator [64:0] and dataout [63:0].
- Between 200 to 400ns, command [5:0] is loaded with 111111 and no operation is performed.
- Between 400 to 600ns, command [5:0] is loaded with 000001 and datain1 [63:0] is loaded with the value 4141, which indicates move register A instruction operation.
- After 600ns, command is changed to 001011, which performs move register A to accumulator instruction operation.

Add Acc, RegA (accumulator ← accumulator + regA):

- As shown in the waveform, between 0 to 300ns, reset is applied to command, datain1 [63:0], accumulator [64:0] and dataout [63:0].
- Between 300 to 500ns, command [5:0] is applied with Opcode 000001 and datain1 [63:0] is applied with the value 4141 which indicates load register A instruction operation.
- Between 500 to 700ns, command is loaded with 101000 and datain1 [63:0] is loaded with the value 2121 which performs load accumulator operation.
- Between 700ns to 720ns, command [5:0] is applied with010001, which performs addition of the contents of register A and accumulator and the result is stored in the accumulator.
- After 720ns, command [5:0] signal is loaded with 000111 and the contents of the accumulator are moved to dataout [63:0].

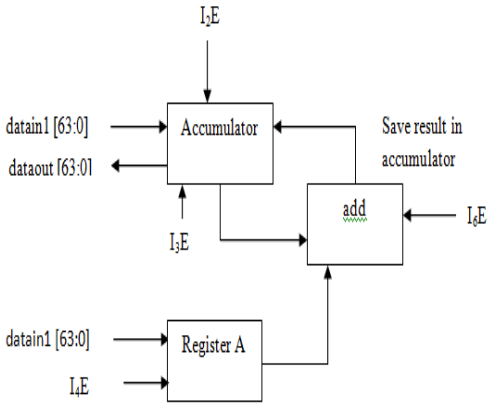
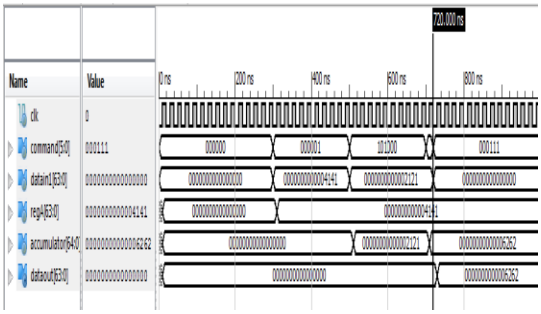


Fig. 5 Digital design for Instruction - Add Accumulator, Reg A



1. Sub Acc, RegA (accumulator ← accumulator - regA):

- As shown in the waveform, between 0 to 300ns, reset is applied to command, datain1 [63:0], accumulator [64:0] and dataout [63:0].
- Between 300 to 500ns, command [5:0] is applied with Opcode 000001 and datain1 [63:0] is applied with the value 2121 which indicates load register A instruction operation.
- Between 500 to 700ns, command is loaded with 101000 and datain1 [63:0] is loaded with the value 4141 which performs load accumulator operation.
- Between 700ns to 720ns, command [5:0] is applied with 010100, which performs subtraction of the contents of register A and accumulator and the result is stored in the accumulator.
- After 720ns, command [5:0] signal is loaded with 000111 and the content of the accumulator is moved to dataout [63:0].

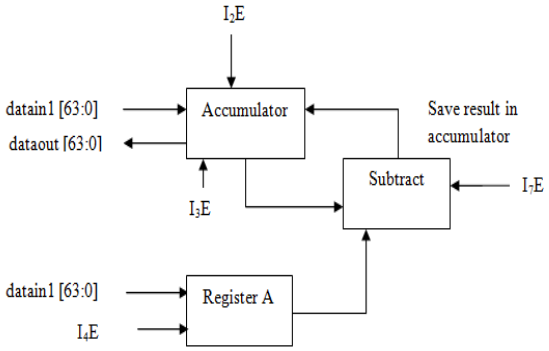
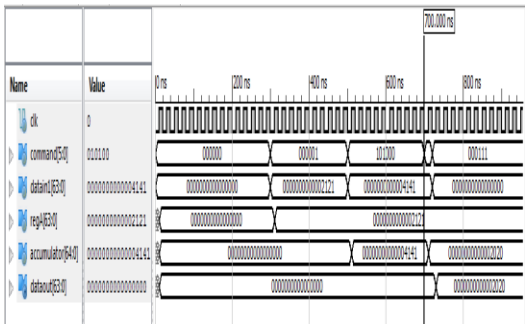


Fig. 6 Digital design for Instruction - Subtract Accumulator, Reg A



2. Not Acc (accumulator ← ~ accumulator):

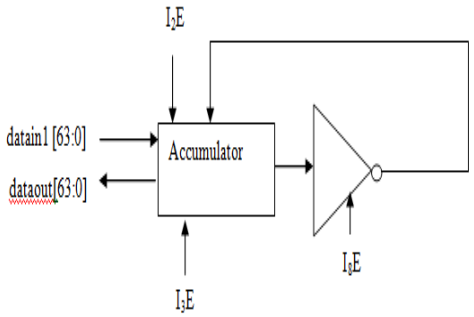
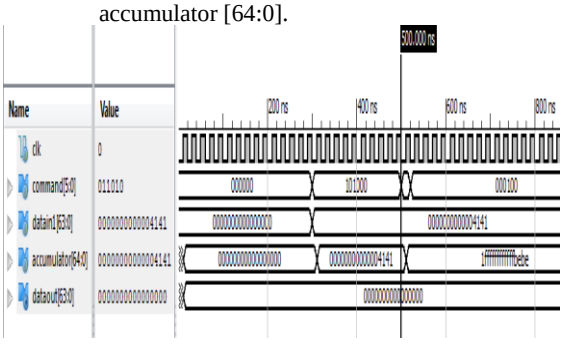


Fig. 7 Digital design for Instruction - NOT Accumulator

- As shown in the waveform, between 0 to 300ns, reset is applied to command, datain1 [63:0], accumulator [64:0] and dataout [63:0].
- Between 300 to 500ns, command [5:0] is applied with the Opcode 101000 and datain1 [63:0] is loaded with the value 4141, which indicates load accumulator operation.
- Between 500 to 520ns, command 011010 is applied which performs NOT operation of the content of the accumulator [64:0].



3. AND ACC, RegA (accumulator ← accumulator and regA):

- As shown in the waveform, between 0 to 300ns, reset is applied to command, datain1 [63:0], accumulator [64:0] and dataout [63:0].
- Between 300 to 500ns, command [5:0] is applied with the Opcode 000001 and datain1 [63:0] is loaded with the value 2121, which indicates load register A operation.
- Between 500 to 700ns, command [5:0] is applied with the Opcode 101000 and datain1 [63:0] is loaded with the value 4141, which indicates load accumulator operation.
- Between 700 to 720ns, command 010111 is applied which performs AND operation of the contents of the accumulator [64:0] and register A [63:0].
- After 720ns, command [5:0] signal is loaded with 000111 and the content of the accumulator is moved to dataout [63:0].

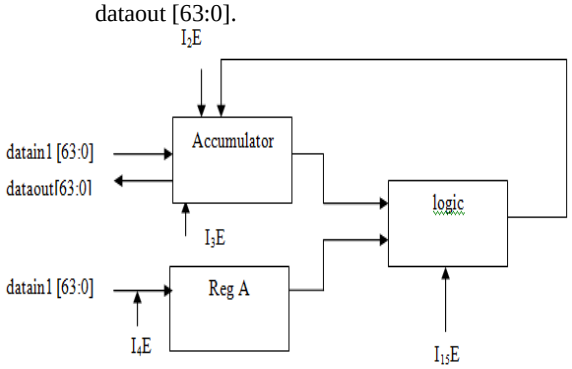
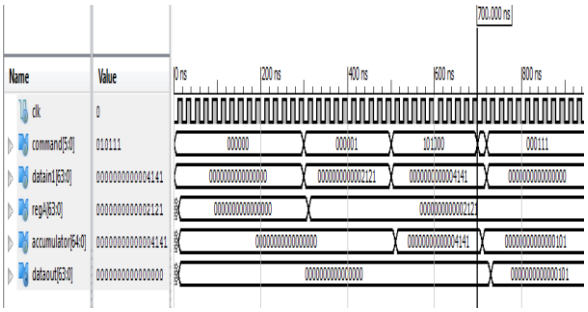
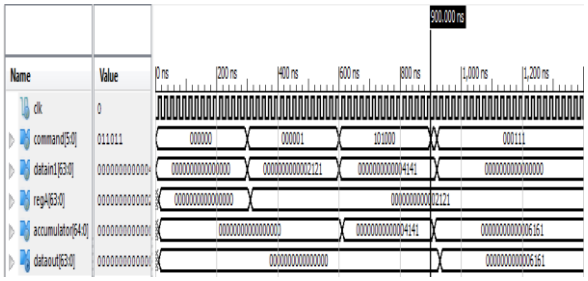


Fig. 8 Digital design for Instruction - AND Accumulator, Reg A



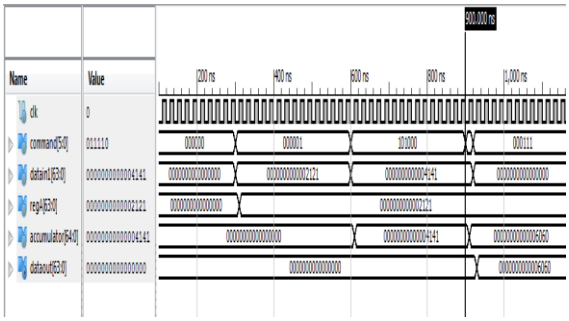
4. OR ACC, Reg A (Accumulator ← Accumulator|regA)



- As shown in the figure, between 0 to 300ns, reset is applied to command, datain1 [63:0], accumulator [64:0] and dataout [63:0].
- Between 300 to 500ns, command [5:0] is applied with the Opcode 000001 and datain1 [63:0] is loaded with the value 2121, which indicates load register A operation.
- Between 500 to 700ns, command [5:0] is applied with the Opcode 101000 and datain1 [63:0] is loaded with the value 4141, which indicates load accumulator operation.
- Between 700 to 720ns, command 011011 is applied which performs OR operation of the contents of the accumulator [64:0] and register A [63:0].
- After 720ns, command [5:0] signal is loaded with 000111 and the content of the accumulator is moved to dataout [63:0].

5. EXOR Acc, RegA (Accumulator ← Accumulator^ regA):

- As shown in the waveform, between 0 to 300ns, reset is applied to command, datain1 [63:0], accumulator [64:0] and dataout [63:0].
- Between 300 to 500ns, command [5:0] is applied with the Opcode 000001 and datain1 [63:0] is loaded with the value 2121, which indicates load register A operation.
- Between 500 to 700ns, command [5:0] is applied with the Opcode 101000 and datain1 [63:0] is loaded with the value 4141, which indicates load accumulator operation.
- Between 700 to 720ns, command 0111110 is applied which performs OR operation of the contents of the accumulator [64:0] and register A [63:0].
- After 720ns, command [5:0] signal is loaded with 000111 and the content of the accumulator is moved to dataout [63:0].



6. Exchange reg A, reg B (regA ↔ regB):

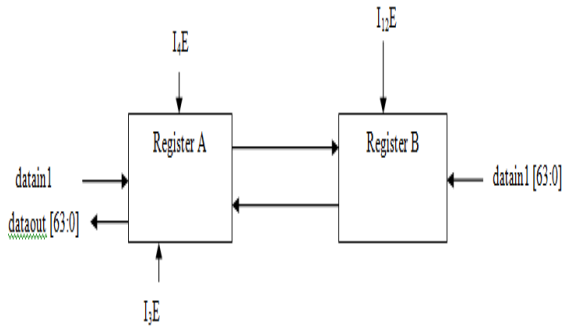
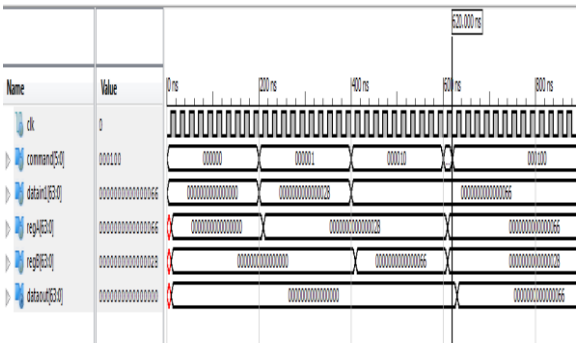


Fig. 9 Digital design for Instruction - Exchange Reg A, Reg B

- As shown in the waveform, between 0 to 200ns, reset is applied to command, datain1 [63:0], accumulator [64:0] and dataout [63:0].
- Between 200 to 400ns, command [5:0] is applied with the Opcode 000001 and datain1 [63:0] is loaded with the value 28, which indicates load register A operation.
- Between 400 to 600ns, command [5:0] is applied with the Opcode 000010 and datain1 [63:0] is loaded with the value 66, which indicates load register B operation.
- Between 600 to 620ns, command [5:0] is loaded with which performs the exchange of the contents of register A and register B.
- After 620ns, command [5:0] signal is loaded with 000100 and the content of the register A is moved to dataout [63:0].



7. Incacc (Accumulator ← Accumulator + 1)

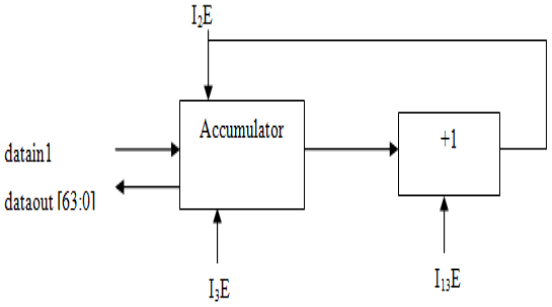
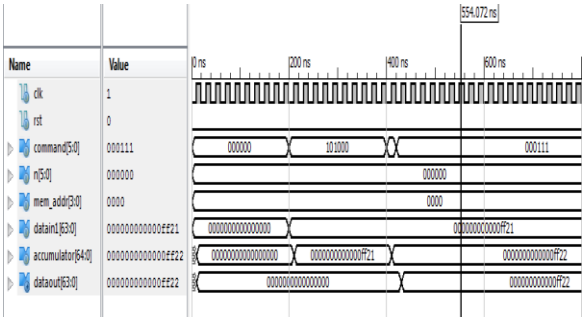


Fig. 10 Digital design for Instruction - Increment Accumulator



- As shown in the waveform, between 0 to 200ns, reset is applied to command, datain1 [63:0], accumulator [64:0] and dataout [63:0].
- Between 200 to 400ns, command [5:0] is applied with the Opcode 101000 and datain1 [63:0] is loaded with

the value ff21, which indicates load accumulator operation.

- Between 400 to 420ns, command is loaded with 110000 which performs increment operation and the content of the accumulator [64:0]is incremented by 1 and the result is stored in the accumulator.
- After 420ns, command [5:0] signal is loaded with 000111 and the content of the accumulator is moved to dataout [63:0].

8. Dec acc (Accumulator ← Accumulator - 1):

- As shown in the waveform, between 0 to 200ns, reset is applied to command, datain1 [63:0], accumulator [64:0] and dataout [63:0].
- Between 200 to 400ns, command [5:0] is applied with the Opcode 101000 and datain1 [63:0] is loaded with the value ff21, which indicates load accumulator operation.
- Between 400 to 420ns, command is loaded with 110001 whichperform decrement operation and the content of the accumulator [64:0] is decremented by 1 and the result is stored in the accumulator.
- After 420ns, command [5:0] signal is loaded with 000111 and the content of the accumulator is moved to dataout [63:0].

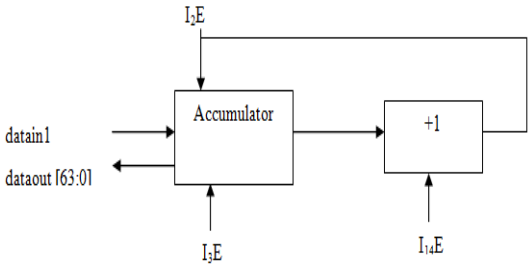
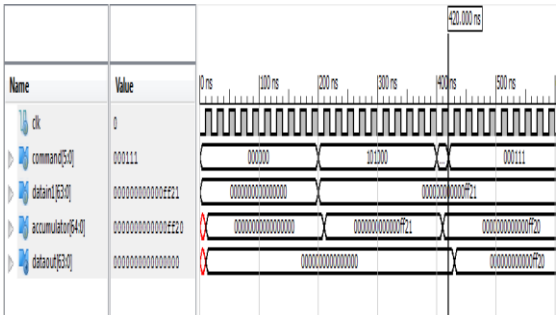


Fig. 11 Digital design for Instruction - Decrement Accumulator



9. Shift left acc, n(CF ← MSB ; MSB ← MSB-1; LSB ← 0)

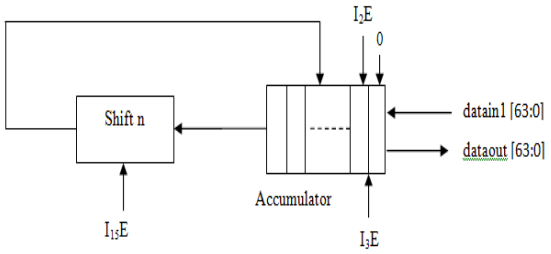
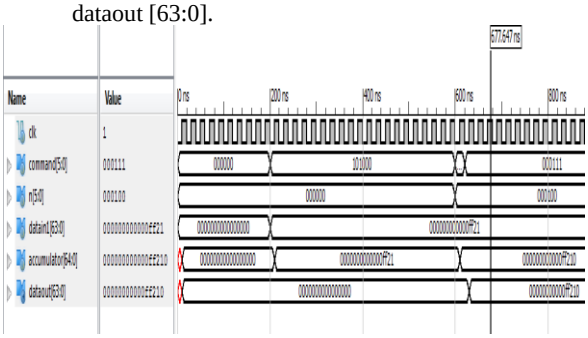


Fig. 12 Digital design for Instruction - Shift left Accumulator, n

- As shown in the waveform, between 0 to 200ns, reset is applied to command, datain1 [63:0], accumulator [64:0] and dataout [63:0].
- Between 200 to 600ns, command [5:0] is applied with the Opcode 101000 and datain1 [63:0] is loaded with the value ff21, which indicates load accumulator operation.
- Between 600 to 620ns, n[5:0] is loaded with the value 4 and command [5:0] is applied with 100001, which indicates shift left operation of the content of the accumulator by value 4 and the result is stored in the accumulator.
- After 620ns, command [5:0] signal is loaded with 000111 and the content of the accumulator is moved to dataout [63:0].



10. Shift right acc,n (CF ←LSB ; LSB ←LSB+1; MSB ← 0)

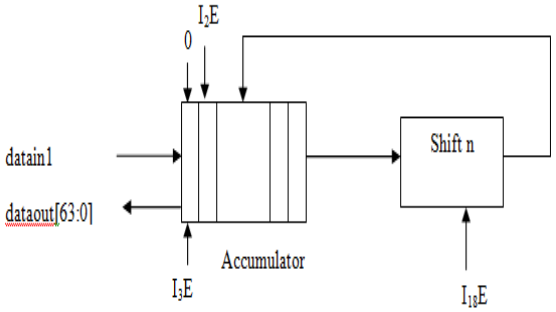
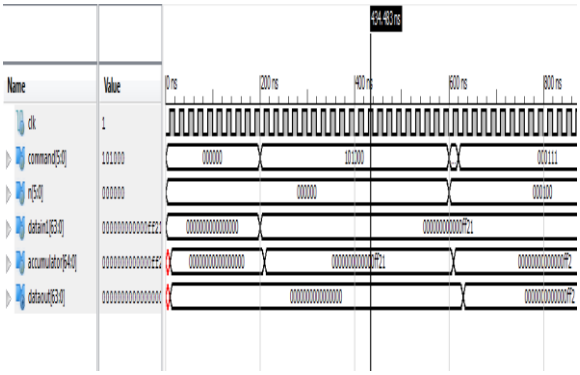


Fig. 13 Digital design for Shift right Accumulator, n



- As shown in the waveform, between 0 to 200ns, reset is applied to command, datain1 [63:0], accumulator [64:0] and dataout [63:0].
- Between 200 to 600ns, command [5:0] is applied with the Opcode 101000 and datain1 [63:0] is loaded with the value ff21, which indicates load accumulator operation.
- Between 600 to 620ns, n [5:0] is loaded with the value 4 and command [5:0] is applied with 100010, which indicates shift right operation of the content of the accumulator by value 4 and the result is stored in the accumulator.
- After 620ns, command [5:0] signal is loaded with 000111 and the content of the accumulator is moved to dataout [63:0].

11. Rotate right acc, n(CF ←MSB ; MSB ←MSB-1; LSB ←CF)

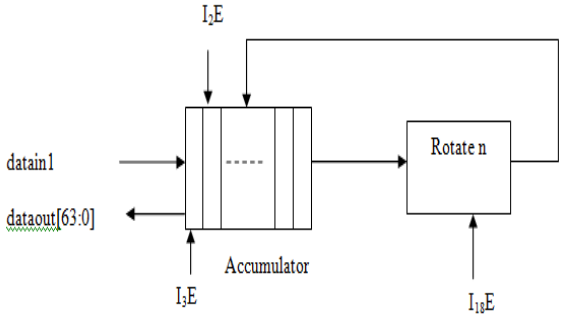
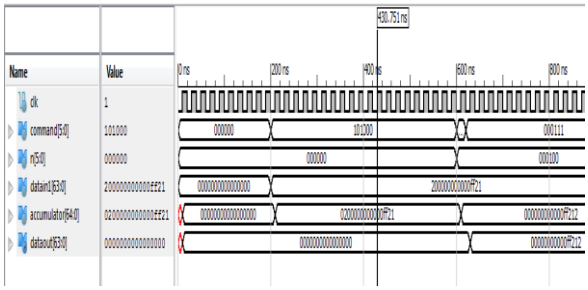


Fig. 14 Digital design for Instruction - Rotate right Accumulator, n



- As shown in the waveform, between 0 to 200ns, reset is applied to command, datain1 [63:0], accumulator [64:0] and dataout [63:0].
- Between 200 to 600ns, command [5:0] is applied with the Opcode 101000 and datain1 [63:0] is loaded with the value 200000000000ff21, which indicates load accumulator operation.
- Between 600 to 620ns, n[5:0] is loaded with the value 4 and command [5:0] is applied with 100011 , which indicates rotate right operation of the content of the accumulator by value 4 and the result is stored in the accumulator.
- After 620ns, command [5:0] signal is loaded with 000111 and the content of the Accumulator is moved to dataout [63:0].

12. Rotate left acc, n(CF ←LSB ; MSB ←CF; LSB ←LSB+1)

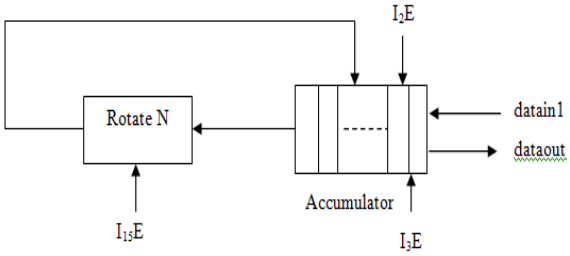


Fig. 15 Digital design for Instruction - Rotate right Accumulator, n

- As shown in the waveform, between 0 to 200ns, reset is applied to command, datain1 [63:0], accumulator [64:0] and dataout [63:0].
- Between 200 to 600ns, command [5:0] is applied with the Opcode 101000 and datain1 [63:0] is loaded with the value 200000000000ff21, which indicates load accumulator operation.
- Between 600 to 620ns, , datin1 [63:0] is applied with the value ff21, n[5:0] is loaded with the value 4 and command [5:0] is applied with 100100, which indicates rotate left operation of the content of the accumulator by value 4 and the result is stored in the accumulator.
- After 620ns, command [5:0] signal is loaded with 000111 and the content of the accumulator is moved to dataout [63:0].

13. Addcacc, regA (accumulator ← accumulator + regA + carry)

The below figure indicates the simulation wave form of addition with carry instruction operation.

- Between 0 to 200ns, reset is applied to command, datain1 [63:0], accumulator [64:0] and dataout [63:0].

- Between 200 to 400ns, command [5:0] is applied with the Opcode 000001 and datain1 [63:0] is loaded with the value ff000000000028, which indicates load register A operation.
- Between 400 to 600ns, command [5:0] is applied with the Opcode 101000 and datain1 [63:0] is loaded with the value f000000000ff21, which indicates load accumulator operation.
- Between 600 to 620ns, command [5:0] is applied with the value 010001 which perform addition operation of the content of register A and accumulator where the carry is generated and the carry flag is set high and the result is stored in the accumulator.
- Between 620 t0 800ns, command [5:0] signal is loaded with 000111 and the content of the accumulator is moved to dataout [63:0].
- Between 800 to 1020ns, datain1 [63:0] is loaded with the new value f0000000000004 and the command [5:0] is loaded with 000001 which indicates load register A operation.
- Between 1020 to 1040ns, command [5:0] is loaded with 101010 which indicates add with carry instruction operation where the content of accumulator and register A is added with the carry flag and the result is stored in the accumulator.
- After 1040ns, command [5:0] signal is loaded with 000111 and the content of the Accumulator is moved to dataout [63:0].

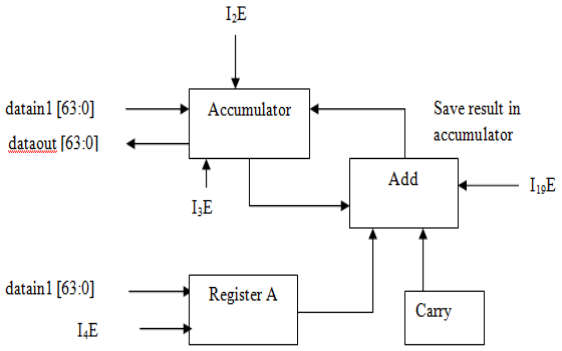
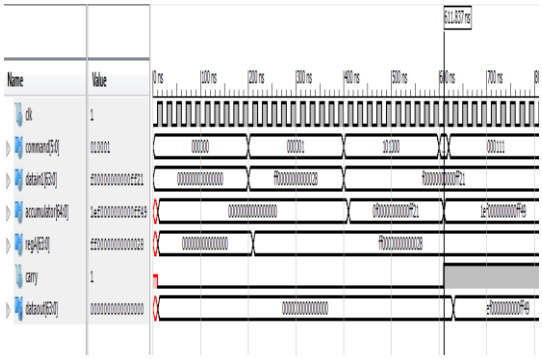
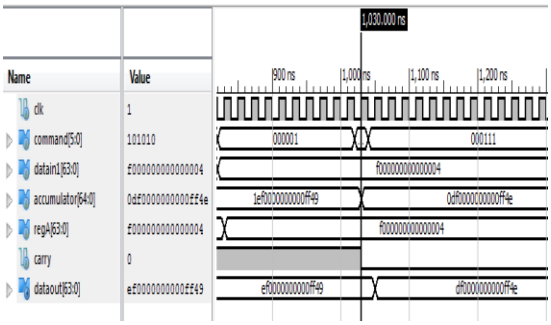


Fig. 16 Digital design for Instruction - Add with carry Accumulator, RegA





14. Subbacc, regA (accumulator ← accumulator -regA - carry)

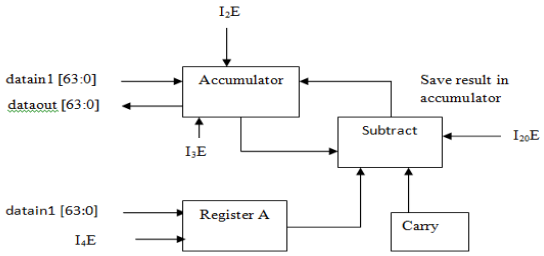
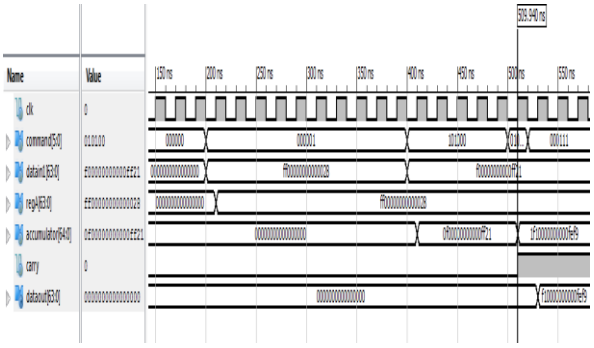
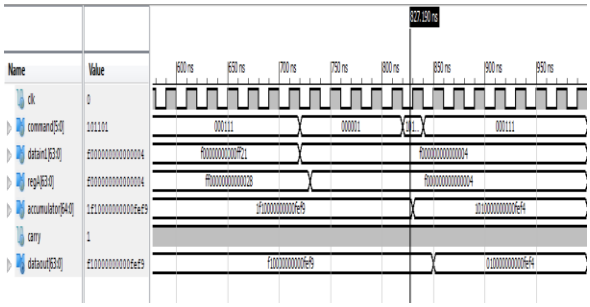


Fig. 17 Digital design for Instruction - Subtract with carry Accumulator, Reg A





- As shown in the waveform, between 0 to 200ns, reset is applied to command, datain1 [63:0], accumulator [64:0] and dataout [63:0].
- Between 200 to 400ns, command [5:0] is applied with the Opcode 000001 and datain1 [63:0] is loaded with the value ff000000000028, which indicates load register A operation.
- Between 400 to 500ns, command [5:0] is applied with the Opcode 101000 and datain1 [63:0] is loaded with the value f000000000ff21, which indicates load accumulator operation.

- Between 520 to 520ns, command [5:0] is applied with the value 010001 which perform subtraction operation of the content of register A and accumulator where the carry is generated and the carry flag is set high and the result is stored in the accumulator.
- Between 520 t0 720ns, command [5:0] signal is loaded with 000111 and the content of the accumulator is moved to dataout [63:0].
- Between 720 to 820ns, datain1 [63:0] is loaded with the new value f0000000000004 and the command [5:0] is loaded with 000001 which indicates load register A operation.
- Between 820 to 840ns, command [5:0] is loaded with 101101 which indicates subtract with carry instruction operation where the content of accumulator and register A is subtracted with the carry flag and the result is stored in the accumulator.
- After 840ns, command [5:0] signal is loaded with 000111 and the content of the accumulator is moved to dataout [63:0].

17. Compare Acc, regA (modify flags ← accumulator – regA)

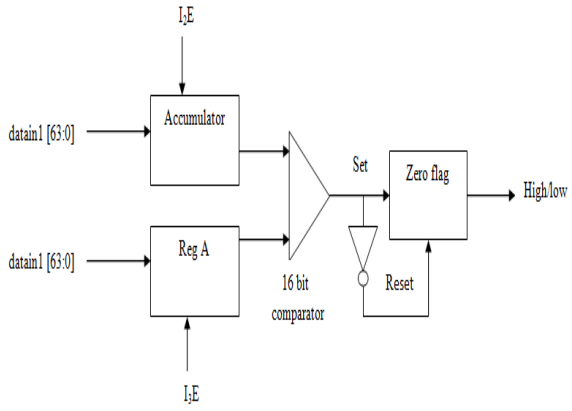
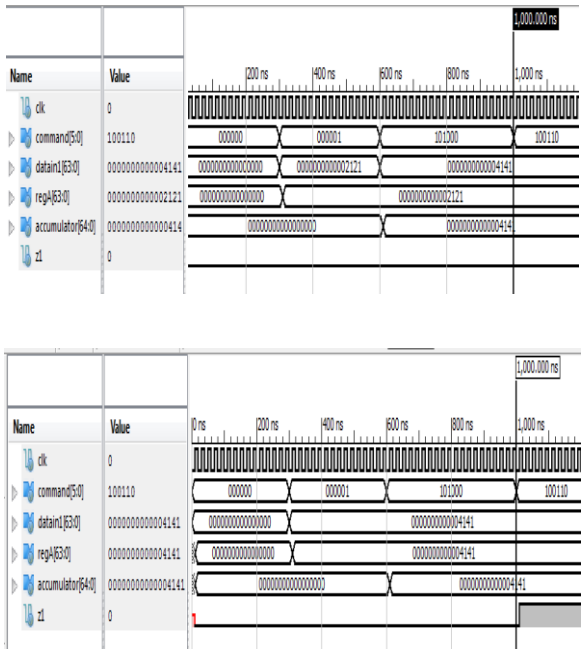


Fig. 18 Digital design for Instruction - Compare Accumulator, Reg A



- As shown in the waveform, between 0 to 300ns, reset is applied to command, datain1 [63:0], accumulator [64:0] and dataout [63:0].
- Between 300 to 600ns, command [5:0] is applied with the Opcode 000001 and datain1 [63:0] is loaded with the value 2121, which indicates load register A operation.
- Between 600 to 1000ns, command [5:0] is applied with the Opcode 101000 and datain1 [63:0] is loaded with the value 4141, which indicates load accumulator operation.
- After 1000ns, command [5:0] is applied with 100110, which indicates the compare instruction operation. Here the contents of register A and accumulator is compared if the values are same then the zero flag is set high and if it is different then the zero flag will be low.

5. Conclusion

There are several popular approaches to design and develop a contemporary processor with several useful instructions. And in this paper, I have evolved a fully scalable and open-ended Processor architecture for basic popular instructions of the processor that can be improvised into a complex next generation multi core Processor. My approach to processor design is-Instruction leading to Architecture. I have taken a subset of 20 popular instructions, along with an innovatively developed command driven Instruction Decoder to fetch and execute each instruction. As a part of my experimentation of implementation of Hybrid Processor on FPGA, I have achieved further results with respect to Interfacing Hardware modules with my Processor Core with Special Interrupt driven Instructions (additional instructions and hardware interfacing) and those will be published soon in my next research paper, as my continued ongoing Research process & methodology of Implementation on the proposed Xilinx FPGA target device.

References

[1]. Mrudul S. Ghaturle and R. D. Kadam "Design and Simulation of Decoder Unit of 32-Bit RISC Processor", International Journal for Research in Applied Science & Engineering Technology (IJRASET), ISSN: 2321-9653 Vol. 5 Issue 7, July 2017.

[2]. Mohammad Gousuddin H Maniya, Sujatha Hiremath, "Design and Implementation o 64 bit RISC Processor on FPGA", International Journal of Advancement in Engineering Technology, Management and Applied Science, ISSN: 2349-3224 Vol. 3, Issue 2, May 2016.

[3]. Priyavrat Bhardwaj, "Design& Simulation of A 32-Bit RISC Based MIPS Processor Using Verilog", International Journal of Research in Engineering and Technology, Vol. 05, Issue 11, Nov. 2016.

[4]. Anu Mariam John, ShilpiVarshney, "FPGA Implementation of 32-bit MIPS Processor with CISC Multiplication Operaton", International Journal of Engineering Research and Technology (IJERT) ISSN: 2278-0181, Vol. 4 Issue, issue 11, Nov. 2015.

[5]. Akshatha Rai K and Basavaraj H J, "Novel Design of Dual Core RISC Architecture Implementation", Proceedings of 3rd IRF International Conference, 7th March 2015, Mysore, ISBN: 978-93-82702-74-0.

[6]. S.Suresh and R.Ganesh, "FPGA Implementation of MIPS RISC Processor", International Journal of Engineering Research and Technology, Vol. 3, Issue 1, January 2014.

[7]. Saraswthi P and M K Chandrasen, "Implementation Of FPGA Based 32-Bit CISCCPU Design International Journal of Advanced Research in Computer and Communication Engineering, Vol. 3, Issue 2, Feb. 2014.

[8]. Wojciech Wójcik, Jacek Długopolski, "FPGA-Based Multi-Core Processor", Computer Science, 14 (3) 2013. http://dx.doi.org/10.7494/csci.2013.14.3.459.

[9]. Vijay R. Wadhankar and VaishaliTehre, "A FPGA Implementation of a RISC Processor for Computer Architecture", National Conference on Innovative Paradigms in Engineering & Technology (NCIPET-2012).

[10]. Harisha M. S and D. Jayadevappa, "Innovative Architecture for FPGA based Multicore Hybrid Processor", International Journal of Scientific & Engineering Research, Vol. 7, Issue 6, June 2016, ISSN 2229-5518.

[11]. Harisha M. S and D. Jayadevappa "A Survey of Various Processor Types and Design Architectures", International Journal of Emerging Technology and Advanced Engineering ISSN 2250-2459, Vol. 8, Issue 2, Feb. 2018.

[12]. Harisha M. S and D. Jayadevappa "A Comprehensive Survey of Various Processor types and Latest Architectures", International Journal of Research and Scientific Innovation (IJRSI), Vol. 5, Issue 4, April 2018, ISSN 2321–2705.