*IOSR Journal of VLSI and Signal Processing (IOSR-JVSP)*
*Volume 3, Issue 5 (Nov. – Dec. 2013), PP 31-42*
*e-ISSN: 2319 – 4200, p-ISSN No. : 2319 – 4197*
*www.iosrjournals.org*

# VHDL Design and Synthesis of 64 bit RISC Processor System on Chip (SoC)

Navneet kaur[1], Adesh Kumar[2], Lipika Gupta[3]
[1]*(M.Tech Scholar, VLSI Design & Microelectronics, Department of Electronics &Communication Engineering, Chitkara University, Himachal Pradesh India)*
[2]*(Assistant Professor, Department of Electronics, Instrumentation & Control Engineering, University of Petroleum and Energy Studies, Dehradun India)*
[3]*(Assistant Professor, VLSI Design & Microelectronics, Department of Electronics &Communication Engineering, Chitkara University, Himachal Pradesh India)*

***Abstract:*** *The paper presents the design and synthesis of 64 bit Reduced Instruction Set Computer (RISC) on Spartan-3E FPGA. A computer using few instructions with simple constructs so they can be executed at much faster rate within the CPU without having to use the memory very often. This type of computer is classified as a reduced instruction set of computer is called RISC. One advantage of RISC is that they can execute their instructions very fast because the instructions are so simple. Another more important advantage is that RISC chips require fewer transistors, makes them cheaper to design and produce. The system on chip (SOC) design of 64 bit RISC processor consist of ALU, Shifter, comparator, RAM Memory, Control Unit, Program counter. Top design approach is used to configure the design. The design is synthesized on Spartan -3E FPGA for 33 instructions. VHDL programming language is used to develop the RISC in Xilinx 14.2 ISE design suit and functional simulated on Modelsim 10.1 b software.*

***Keywords:*** *Complex Instruction set Computer (CISC), Field Programmable Gate Array (FPGA), and System on chip (SOC), Reduced Instruction Set Computer (RISC)*

## I. Introducton

An important aspect of computer architecture is design of the instruction set for the processor. The instruction set chosen for particular computer determines the way in which the machine language programs are constructed. A computer with large number of instructions is classified as a complex instruction set computer, abbreviated CISC. A computer using few instructions with simple constructs so they can be executed at much faster rate within the CPU without having to use the memory very often. This type of computer is classified as a reduced instruction set of computer, abbreviated RISC. Until the mid1980's, the tendency among the computer manufacturers was to build increasingly complex CPUs that had ever larger set of instructions. At that time however, a number of complex manufacturers decided to reverse the trend by building CPU capable of executing only a very limited set of instructions. One advantage of RISC is that they can execute their instructions very fast because the instructions are so simple. Another more important advantage is that RISC chips [1] [5] require fewer transistors for cheaper design and produce. Since the RISC computers, conventional computers have been referred to as CISCs. The main characteristics of RISC[1] are, reduced instruction set, less complex, simple instructions, few addressing modes, Hardwired control unit and machine instructions and instruction pipelining.

The RISC architecture [8] of 64 bit processor is shown in the figure 1. The major parts of 64 bits RISC are data bus (64 bits), address bus (64 bits), ALU, Control unit, RAM Array. Control unit [3] of the microprocessor generates signals within microprocessor to carry out the instruction, which has been decoded. Control unit reality causes certain connections between blocks of the microprocessor to be opened or closed. Arithmetic Logic Unit (ALU) [8] performs the actual numerical and logic operation such as addition, subtraction, AND, OR etc. uses data from memory and from Accumulator to perform arithmetic. The ALU includes combined flip-flops, which are set or reset after an operation according to data conditions of the result in the accumulator and other registers. They are called Zero (Z), Carry (CY), Sign (S), Parity (P), and Auxiliary Carry (AC) flags.

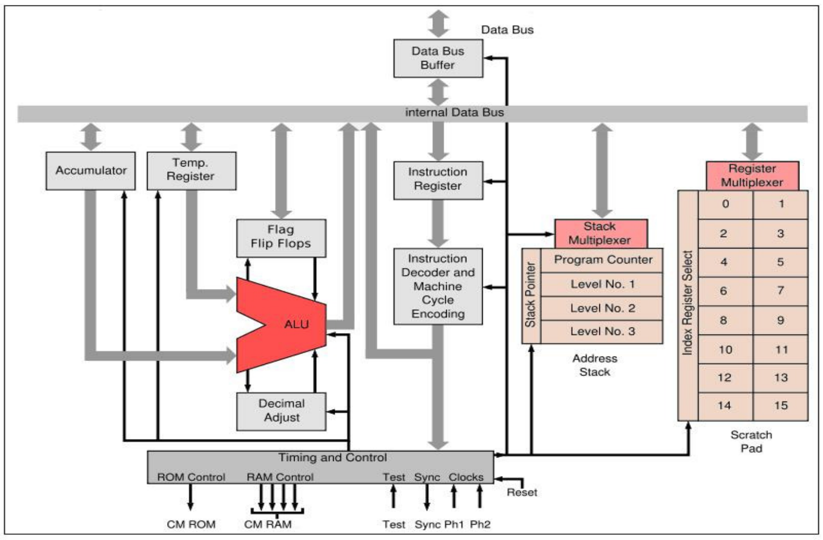


Fig.1 RISC processor architecture (64 bits) [5]

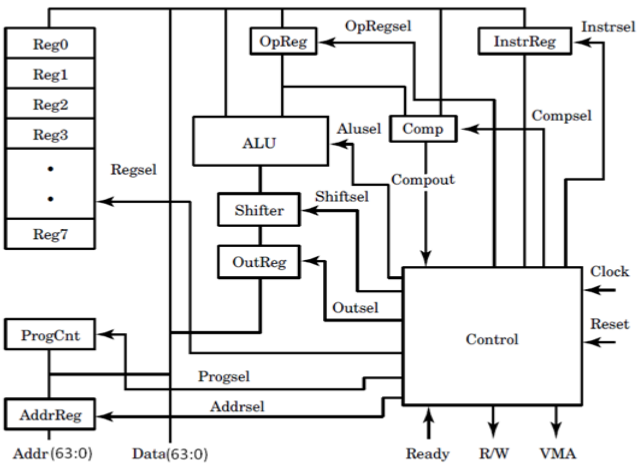


Fig. 2 Datapath architecture of 64 bit RISC processor

The most commonly used flags are Zero, Carry, and Sign. RISC processor uses these flags to test data conditions. For an example the addition of two numbers is done, the sum in the accumulator is larger than 64 bits, the flip-flop uses to indicate a carry called the Carry flag (CY) is set to 1. When an arithmetic operation results in zero, the flip-flop called the Zero (Z) flag is set to one. Parity (P) flags checks weather the result has even parity or odd parity, for odd parity p = 0 and even parity P =1. Auxiliary carry (AC) flag checks weather there is an intermediate carry from one bye to another byte or nibble. Sign (S) flags check the result is a positive number or negative number. These flags have critical importance in the decision making process of the microprocessor. These registers are used by the microprocessor to sequence the execution of the instructions. The main role of the program counter [4] is to point to the memory address from which the next byte is to be fetched. When the machine code of any instruction is fetched, program counter is incremented automatically by one to point to the next memory location. Instruction Register/Decoder is the temporary store for the current instruction of a program. Current instruction sent here from memory prior to execution. Decoder then takes instruction and 'decodes' or interprets the instruction. After the instruction decoding, then passed to next stage. Bidirectional data bus is used to carry data in binary form between microprocessor and other external peripherals, such as memory. The data bus typically consists of 64 wires. Data bus used to transmit data, information, results of arithmetic operations, etc between memory and the microprocessor.

Size of the data bus determines what arithmetic can be done. Control bus is the bus used to perform control operations which are specific functions for coordinating and controlling microprocessor operations, the general control functions are memory read, memory write, I/O read, I/O write. They control whether memory is being written to or data stored in memory, read from or data taken out of memory 1 = Read and 0 = Write with clock pulse. Typically microprocessor has 10 control lines. It cannot function correctly without these vital control signals. The function of control bus is to control signals partly unidirectional, partly bi-directional. Controls signals are generic like "read or write". These operations tell memory that we are reading from a location, basically specified on the address bus, or writing to a location specified. There are other signals to control and coordinate the operation of the system

## II. Data Path Architecure

The data path architecture of 64 bit RISC structure is shown in figure 2. The structure is configured supporting to pipeline structure, with 64 bits datalines, 64 bits address lines. It supports 33 instructions. The major parts of RISC are ALU, shifter, comparator, RAM array and control unit. The functionality of data path design can be understood with the help of main unit. All the arithmetic and logical functions are performed using ALU. Control unit performs the all functions relating to memory or interface device between processor and memory. *AddrReg* is the register used to hold the address, *ProgCnt* is used to hold the address of next instruction. Instruction register is used to fetch the opcode of the instruction which is going to be executed by the microprocessor. The result of every arithmetic and logical instruction is stored in accumulator which is inside ALU. ALU is of 64 bits, if the result is more than 64 bits, it is stored in Opreg which is a temporary register used to hold the result beyond 64 bits such in multiplication, division, addition with carry and subtraction with borrow. The shifter is used to perform shifting operations shift left, shift right, rotate left and rotate right directly in Accumulator. Comparator is used to perform camparison operations synchronized with the control unit of the microprocessor. Ready pin is a pin which indicates that processor is ready to transfer or receive the data within memory or I/O device. R/W signal is applicable for memory write and read operation, clock signal is used to provide a clock pulse with 50% duty cycle and Reset is synchronized with clock pulse, a control signal. RAM architecture has 8 general purpose registers (Reg 0 … Reg 7) for data write and read operations or storage permanently.

Control unit has an output signal *Alusel* to the ALU, *shiftsel* to shifter, *Outsel* to Output register. *Compsel* to comparator, *opregsel* to operational register, *Instrsel* to instruction register, *progsel* to programcounter, *addrsel* to address register and *regsel* is used to select the resister form regster array $R_0$ –$R_7$ .*Compout* is an output from comparator to control unit. The functionality of the architecure can be understood with the help of an example. Let the processor want to execute the instruction MOV A, B, the opcode of the instruction is in instruction register, which is directly connected to control unit. The data first goes to ALU unit, which is directly connected to RAM array, the data is shifted into shifter and compared with the comparator unit, after the comparison the data is shifted to address/ data bus which is common. There is a VMA signal which means Valid Memory Address, this signal behave a Address Latch Enable(ALE) when ALE=1 the address of source register is passed on address bus. As the data bus has 64 bit data, then this data should be copied to destination register. Based on the operation of MOV instruction the address of destination register is on the address bus and stored in address register. Address register is directly connected to control unit and address bus. Clock is signal at which processor is synchronized to work on rising and falling edge with reset. If Ready =1, only then the processor control unit is able to fetch the opcode of aparticular instruction. Program counter is used to point the address of the instruction of next instruction and is hold by the *ProgCnt* unit of the architecture. For memory read and memory write operations , if R/W =1, read operation is performed else write operation is performed.

### 2.1 ALU

The first entity is the ALU, performs the Arithmetic or logical operations. The 64 bits inputs are *A* and *B* the operations are selected by selection logic *sel* (5 bits) and corresponding operation result is taken by *Y* (64 bits). If the output is greater that 64 bits such as in the instructions multiplication , division, there are two outputs $Y_1$ and $Y_2$ of 64 bits. Figure 3 (a) and (b) shows the entity of ALU and table 1 shows the operation of ALU.

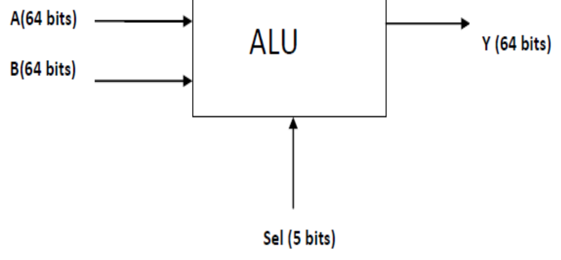


Fig.3 (a) Entity ALU (64 bits)

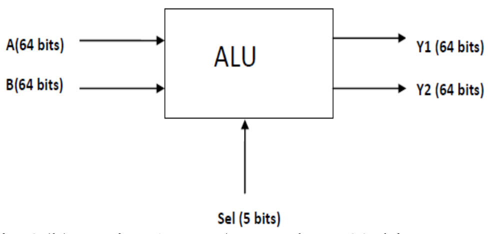


Fig.3(b) Entity ALU (more than 64 bits output)

Table 1 ALU operations

| Sel | Operation |
|---|---|
| 00000 | Addition of a and b |
| 00001 | Subtraction of a and b |
| 00010 | Multiplication of a and b |
| 00011 | Division of a and b |
| 00100 | Addition of a and b with carry |
| 00101 | Subtraction of a and b with borrow |
| 00110 | AND operation of A and B |
| 00111 | OR operation of A and B |
| 01000 | NOT operation of A |
| 01001 | NOT operation of B |
| 01010 | NAND operation of A and B |
| 01011 | NOR operation of A and B |
| 01100 | XOR operation of A and B |
| 01101 | XNOR operation of A and B |
| 01110 | Increment of A |
| 01111 | Increment of B |
| 10000 | Decrement of A |
| 10001 | Decrement of B |
| 10010 | Pass value of A |
| 10011 | Pass value of B |
| 10100 | Zero operation |

**2.2 Shift unit Design**

The shift unit has two inputs *A* and *B* of 64 bits and one output *Y* also of 64 bits. The shift operations of A and B are*shift left*,*shift right* and rotate operations of A and B are*rotate left*, *rotate right*are performed using the selection logic *Sel* which is of 4 bits. Table 2 lists the details of shift and rotate operations for shift unit entity.

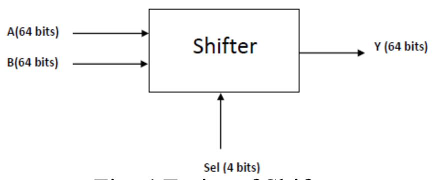


Fig. 4 Entity of Shifter

Table 2 Shift operations of 64 bits RISC

| Sel | Operation |
|---|---|
| 0000 | Shift left A by 1 bit |
| 0001 | Shift right A by 1 bit |
| 0010 | Rotate left A by 1 bit |
| 0011 | Rotate right A by 1 bit |
| 0100 | Shift left B by 1 bit |
| 0101 | Shift right B by 1 bit |
| 0110 | Rotate left B by 1 bit |
| 0111 | Rotate right B by 1 bit |
| 1000 | Shift pass operation of A |
| 1001 | Shift pass operation of B |

**2.3 Comparator Design**

The comparator compares two values *A* and *B* of 64 bits and returns results *Y* of 1 bit either *Y*= '1' or *Y* = '0', depending on the comparison requested and the values being compared. The comparison type is determined by the value on input port *sel*. For instance, to compare if inputs *A*and *B*are equal, apply the value

*EQ* to port *sel*. If ports *A* and *B* have the same value, port *Y* returns '1'. If the values are not equal, '0' is returned. All operations work on two input values and return a single bit result. This bit is used to control the flow of operation within the processor while executing instructions. There are six comparison operations to check as listed in table 3. The compare operations are equal, not equal, greater than, less than, less than equal, greater than equal.

Table 3 Compare operation

| **Sel** | **Operation** |
|---|---|
| 0000 (EQ), Equal | Compare A is equal to B ( A = B), If Yes Y = '1' Else Y = '0' |
| 0001(NEQ), Not Equal | Compare A is equal to B (A≠B) , If Yes Y = '1' Else Y = '0' |
| 0010(GT), Greater than | Compare A is greater than B ( A > B) , If Yes Y = '1' Else Y = '0' |
| 0011(LT), Less than | Compare A is less than B ( A < B) , If Yes Y = '1' Else Y = '0' |
| 0100(GTE), Greater than Equal | Compare A is greater than B ( A ≥B) , If Yes Y = '1' Else Y = '0' |
| 0101(LTE) Less Than Equal | Compare A is less than B ( A ≤ B) , If Yes Y = '1' Else Y = '0' |

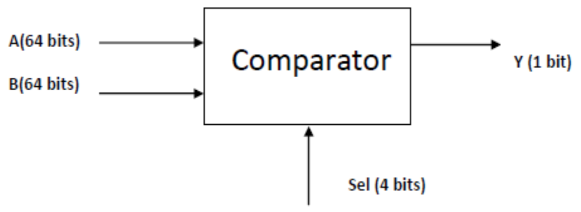


Fig.5 Entity of Comparator

**2.4 Register Array Design (RAM)**

The process of reading the source and destination operands or words with respect to register address is called memory read operation. After reading the data or operands data is stored at particular memory location or address of register, is called the write operation. In the processor operation first memory read operation occurs and then write operation. Figure 6 shows the memory read and write operations with respect to clock signal. A memory unit receives the address from a register, called the address register (AR). The data transferred to another register, called data register (DR). Then read operation is represented as

***Memory Read: DR←M [AR]*** $_{\textbf{\textit{source}}}$

In write operation the contents of data register are transferred to memory unit at specified address. Assuming the input data is in R2 register and address is in AR register.

***Memory Write: M [AR]*** $_{\textbf{\textit{destination}}}$ ***←DR***

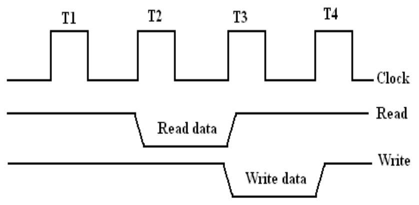


Fig.6 Memory read and write operation

**Example**: Let us consider an example of RAM memory array which has 8 registers (R0- R7) with addresses (000,001,010,011,100,101,110,111) and having the data contents 00H, 50H, 00H, 00H, 00H, 00H, 00H,00H.During the first cycle of clock, let we want to fetch the instruction from R1 (001) register which is called source memory address M[AR] $_{\text{source}}$ register having data 50H stored temporarily in R2 (010) register called memory data register (DR), for memory read Now R2 Register will have 50H. R2 data is stored in R5 (101) register or M [AR] $_{\text{destination}}$ during memory write operation. Now R5 Register will have 50H and R2 register will have 00H.

*Memory read: R2 ←* ***R1 or DR (address = 010) ←M [AR]*** $_{\textbf{\textit{source}}}$ ***(address = 001)***

*Memory write: R2 ←* ***R1 or M [AR]*** $_{\textbf{\textit{destination}}}$ ***(address = 101) ←DR(address = 010)***

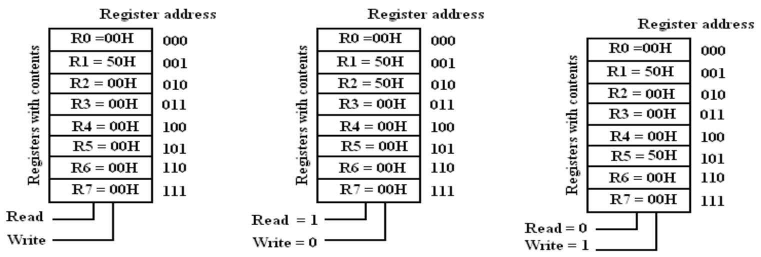


Fig.7 (a) Initial stage (b) Memory read (c) Memory write

There is an array of 8 registers in which the data is written. The regarray entity is used to model the set of registers within the CPU that are used to store intermediate values during instruction processing. These registers are read from and written to during the execution of instructions. The array of registers is modeled as a RAM of eight 64 bit words. The symbol for the regarray entity is shown in figure 8. To write a location in the regarray, set input *sel* to the location to be written, input data with the data to be written, and put a rising edge on input *clk*. To read a location from regarray, set input *sel* to the location to be read and set input en to a '1'; the data is output on port y. Each register is addressed by its 3 bits address as shown in table 4. The first process of the RAM data is to process the data in temporary resisters based on the rising edge of *clk* and enable input, *write =1*. When *read =1* the data is read from a particular register based on the selection logic.

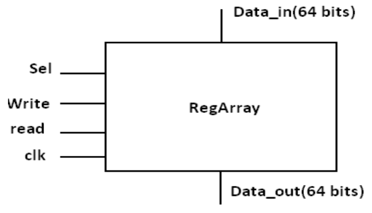


Fig.8 Register Array (RAM)

Table 4 Register selection

| Register Address | Register |
|---|---|
| 000 | Register $R_0$ is selected |
| 001 | Register $R_1$ is selected |
| 010 | Register $R_2$ is selected |
| 011 | Register $R_3$ is selected |
| 100 | Register $R_4$ is selected |
| 101 | Register $R_5$ is selected |
| 110 | Register $R_6$ is selected |
| 111 | Register $R_7$ is selected |

## 2.5 Bi register

The Bi**reg** entity is used for the address register and the instruction register. These registers need to be able to capture the input data on a rising edge of the ***clk*** input and drive output *Y* with the captured data. The value of input ***A*** is assigned to output ***Y*** when a rising edge occurs on input ***clk***. A symbol for the Bireg entity is shown in figure 9. The Bireg symbol contains three ports. In bi register *A* is the input and *Y* is the output of 64 bits, and port ***clk*** controls when the data is stored in the Bireg entity.

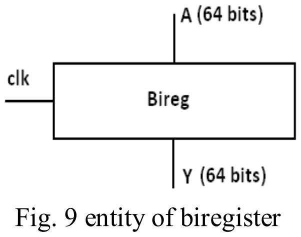


Fig. 9 entity of biregister

## 2.6 Tri-Register

The tristate register is connected to the main data bus and can store information from the data bus as well as drive information to the data bus. The **trireg** entity has four ports as shown in figure 10. Input **A** is the data input to the register, and port Y is the data output from the register. Input ***rd*** and ***wr*** is used to store a new value into the register. It can be used as program counter.

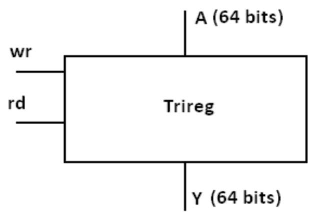


Fig.10 entity of Triregister

## 2.7 Control Unit

The **control unit** entity provides the necessary signal interactions to make the data flow properly through the CPU and perform the expected functions. Architecture **RTL** contains a state machine [3] that causes all appropriate signal values to update based on the current state and input signals and produces a next state for the state machine. The entity control unit is shown in Figure 11. The control symbol has only a few inputs, but a lot of outputs. The control block provides all of the control signals to regulate data traffic for the CPU. Control unit has reset, clk, ready, instreg, compout as the inputs, progcntwr,proncntrd, Addrregwr,Addregrd, Outregwr, Outregrd, shiftsel, Alusel, compsel, Oprregre, opregwr,instwr, regwr, regrd, RW and VMA.

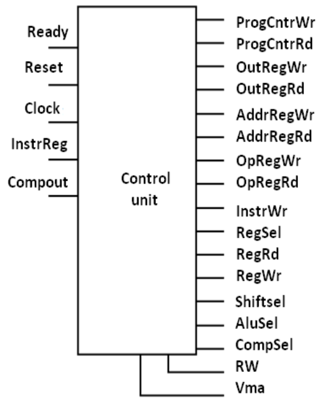


Fig.11 Control unit [3]

## 2.8 Flag register design

Flagsel activates the input flag based on a particular flag is set, the data is checked from the data bus which is inout to the entity.

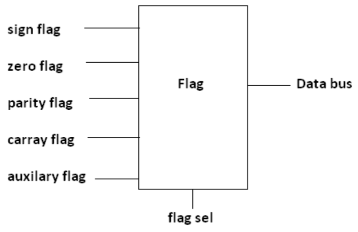


Fig.12 Flag design

## III. Instruction Decoding

The opcode of all instructions is presented by five most significant bits of the instruction. Single-word instructions [3] also contain two 3-bit register fields in the lowest 6 bits of the instruction. Some instructions, such as DEC (Decrement), only use one of the fields, but other instructions, such as MOV (Move), use both register fields to specify the source register and target register. In double-word instructions, the first word contains the opcode and destination register address, and the second word contains the immediate instruction location or data value to be loaded. When the control unit decodes the opcode of the first word, it determines that the instruction is two words long and loads the second word to complete the instruction. The instructions implemented in the processor and their opcodes are listed in table 5. The possible instructions have been implemented in this processor example to limit the complexity for ease of publication. RISC processors are much more complicated and have pipelined instruction streams for faster execution. To reduce complexity, this example is not pipelined. The instruction format to execute the instructions is shown in the figure 13. Five bits are 5 bits are for opcode (63 down to 59), three bits are for source register (0 to 2), and three bits are for destination register (3 to 5).

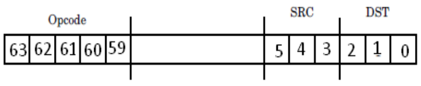


Fig. 13 Instruction Opcode format [3]

Table 5 Instructions [3]

| OPCODE | INSTRUCTION | FUNCTION |
|---|---|---|
| 00000 | NOP | Used for No operation |
| 00001 | LOAD | It load the register |
| 00010 | STORE | Save the result in a register |
| 00011 | MOVE | Move the contents form one register to another register |
| 00100 | LOADI | Load register with immediate value |
| 00101 | BRANCHI | Branch to immediate address |
| 00110 | BRANCHGTI | Branch greater than to immediate address |
| 00111 | INC | Increment the contents of register by 1 |
| 01000 | DEC | Decrement the contents of register by 1 |
| 01001 | AND | Perform AND operation of two registers |
| 01010 | OR | Perform OR operation of two registers |
| 01011 | XOR | Perform XOR operation of two registers |
| 01100 | NOT | Perform NOT operation of a register |
| 01101 | ADD | Perform addition of two registers |
| 01110 | SUB | Perform Subtraction of two registers |
| 01111 | ZERO | Perform Zero operation of a register |
| 10000 | BRANCHLTI | Branch less than to immediate address |
| 10001 | BRANCHLT | Branch less than |
| 10010 | BRANCHNEQ | Branch not equal |
| 10011 | BRANCHNEQI | Branch not equal to immediate address |
| 10100 | BRANCHGT | Branch greater than |
| 10101 | BRANCH | Branch all the time |
| 10110 | BRANCHEQ | Branch if equal |
| 10111 | BRANCHEQI | Branch if equal to immediate address |
| 11000 | BRANCHLTEI | Branch if less or equal to immediate address |
| 11001 | BRANCHLTE | Branch if less or equal |
| 11010 | SHL | Shift left, Perform shift left by one bit operation of a register contents |
| 11011 | SHR | Shift right, Perform shift right by one bit operation of a register contents |
| 11100 | ROTR | Rotate right, Perform rotate right by one bit operation of a register contents |
| 11101 | ROTL | Rotate left, Perform rotate left by one bit operation of a register contents |

## IV. Simulation Results

The Register Transfer Level (RTL) schematic of RISC is shown in figure 14 and internal schematic is shown in figure 15. In the RTL view *Clk* is the input of IC to synchronize the processor. *Reset* is the input to synchronize the processor with *clk* and used to *reset* the memory contents. *Add_bus(63:0)* is the address bus of 64 bits *data_bus (63:0)* is the data bus. *Vma* is valid memory address and *wrb* is read write signal. *Vma* and *wrb* both are control signal. *Ready* is the input signal that processor is ready to transfer the data over data bus. If **Ready = 1**, only then processor passes the data over data bus.

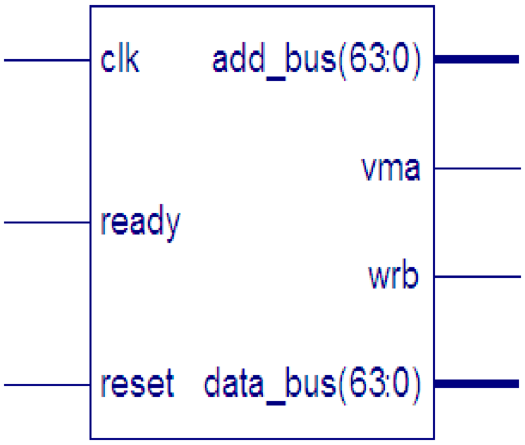


Fig. 14 RTL view of 64 bit RISC

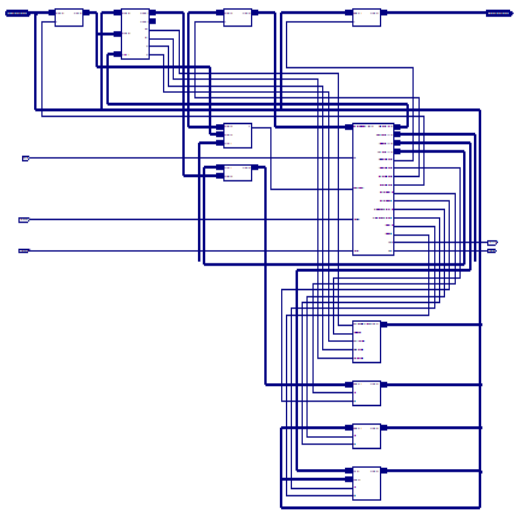
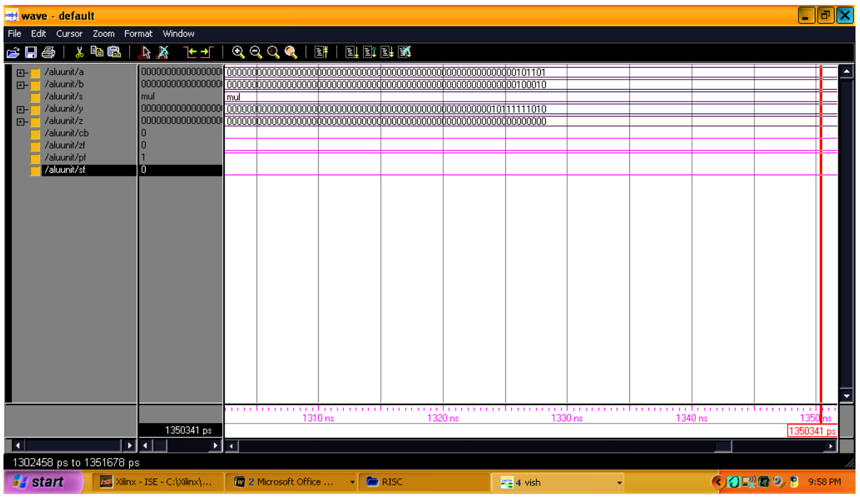


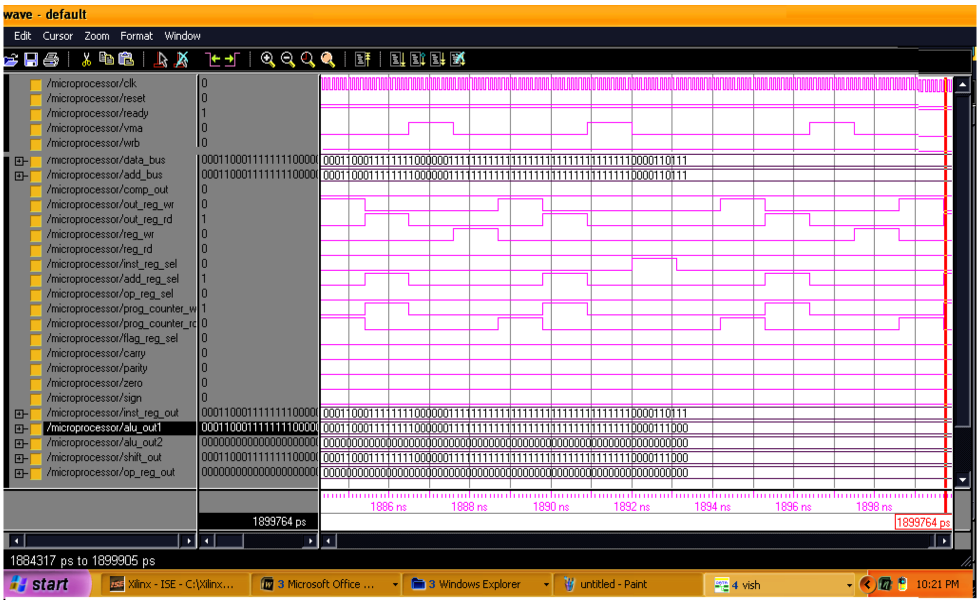
Fig. 15 internal schematic of RISC (level-1)



```
/aluunit/a    0000000000000000  0000000000000000000000000000000000000000000000000000000000101101
/aluunit/b    0000000000000000  0000000000000000000000000000000000000000000000000000000000100010
/aluunit/s    mul               mul
/aluunit/y    0000000000000000  0000000000000000000000000000000000000000000000000000010111111010
/aluunit/z    0000000000000000  0000000000000000000000000000000000000000000000000000000000000000
/aluunit/cb   0
/aluunit/zf   0
/aluunit/pf   1
/aluunit/sf   0

1350341 ps
1302458 ps to 1351678 ps
```

Fig. 16 Modelsim waveform of 64 bit ALU unit



```
/microprocessor/clk           0
/microprocessor/reset         0
/microprocessor/ready         1
/microprocessor/vma           0
/microprocessor/wrb           0
/microprocessor/data_bus      000110001111111100000  0001100011111111000001111111111111111111111111111111111110000110111
/microprocessor/add_bus       000110001111111100000  0001100011111111000001111111111111111111111111111111111110000110111
/microprocessor/comp_out      0
/microprocessor/out_reg_wr    0
/microprocessor/out_reg_rd    1
/microprocessor/reg_wr        0
/microprocessor/reg_rd        0
/microprocessor/inst_reg_sel  0
/microprocessor/add_reg_sel   1
/microprocessor/op_reg_sel    0
/microprocessor/prog_counter_w 1
/microprocessor/prog_counter_rc 0
/microprocessor/flag_reg_sel  0
/microprocessor/carry         0
/microprocessor/parity        0
/microprocessor/zero          0
/microprocessor/sign          0
/microprocessor/inst_reg_out  000110001111111100000  0001100011111111000001111111111111111111111111111111111110000110111
/microprocessor/alu_out1      000110001111111100000  0001100011111111000001111111111111111111111111111111111110000111000
/microprocessor/alu_out2      000000000000000000000  0000000000000000000000000000000000000000000000000000000000000000000
/microprocessor/shift_out     000110001111111100000  0001100011111111000001111111111111111111111111111111111110000111000
/microprocessor/op_reg_out    000000000000000000000  0000000000000000000000000000000000000000000000000000000000000000000

1899764 ps
1884317 ps to 1899905 ps
```

Fig. 17 Modelsim waveform of RISC processor for MOV instructioN

The functional simulation of the ALU and RISC processor is shown in the figure 16 and figure 17 respectively. ALU has 64 bit data *A* and *B* and output is presented by *Y.* The waveform is shown for the multiplication operation. The input A= 0000000000000000000000000000000000000000000000000000000000101101 and B= 0000000000000000000000000000000000000000000000000000000000100010, the result is Y = 000000000000000000000000000000000000000000000000000010111111010 and Z = 0000000000000000000000000000000000000000000000000000000000000000, carry flag, Cb =0, Zero flag Zf =0, parity flag Pf =1 and sign flag Sf = 0. The processor has the clk and reset input. First, reset = ‘1’, then the contents of all units of microprocessor are zero. When reset = ‘0’, applying clk pulse directly with the help of clk signal and forcing to 50 % duty cycle. As Valid Memory Address VMA = ‘1’, force the data on data bus according to the instruction format discussed. The data will pass by the sub components of processor and will show in output register.

## V. Synthesis Report

Synthesis report is the report presents the details of device hardware utilization and timing information. Device utilization report is the report of used device hardware in the implementation of the chip and timing report is the minimum and maximum time to reach the output. Timing parameters are synchronized with the clock signal. Timing details provides the information of net delay, minimum period, minimum input arrival time before clock and maximum output required time after clock. Device utilization is the summary of number of slices, registers, and memory flip flops etc used to optimize the design as a part of combinational and sequential logic. The detailed utilization as macro statistics is discussed below.

*Macro Statistics*

| | |
|---|---|
| # FSMs | : 1 |
| # Multipliers | : 1 |
| No. of 32x32-bit multiplier | : 1 |
| # Adders/Subtractors | : 291 |
| No. of 1-bit adder carry out | : 1 |
| No. of 10-bit adder | : 2 |
| No. of 11-bit adder | : 2 |
| No. of 12-bit adder | : 2 |
| No. of 13-bit adder | : 2 |
| No. of 14-bit adder | : 2 |
| No. of 15-bit adder | : 2 |
| No. of 16-bit adder | : 2 |
| No. of 17-bit adder | : 2 |
| No. of 18-bit adder | : 2 |
| No. of 19-bit adder | : 2 |
| No. of 2-bit adder | : 3 |
| No. of 2-bit adder carry out | : 1 |
| No. of 20-bit adder | : 2 |
| No. of 21-bit adder | : 2 |
| No. of 22-bit adder | : 2 |
| No. of 23-bit adder | : 2 |
| No. of 24-bit adder | : 2 |
| No. of 25-bit adder | : 2 |
| No. of 26-bit adder | : 2 |
| No. of 27-bit adder | : 2 |
| No. of 28-bit adder | : 2 |
| No. of 29-bit adder | : 2 |
| No. of 3-bit adder | : 5 |
| No. of 3-bit adder carry out | : 1 |
| No. of 30-bit adder | : 2 |
| No. of 31-bit adder | : 2 |
| No. of 32-bit adder | : 193 |
| No. of 32-bit addsub | : 1 |
| No. of 33-bit adder | : 3 |
| No. of 33-bit subtractor | : 3 |
| No. of 4-bit adder | : 9 |

| | |
|---|---|
| No. of 4-bit adder carry out | : 1 |
| No. of 5-bit adder | : 17 |
| No. of 5-bit adder carry out | : 1 |
| No. of 6-bit adder | : 2 |
| No. of 7-bit adder | : 2 |
| No. of 8-bit adder | : 2 |
| No. of 9-bit adder | : 2 |
| No. of Registers | : 321 |
| No. of Flip-Flops | : 321 |
| # Latches | : 8 |
| No. of 1-bit latch | : 3 |
| No. of 32-bit latch | : 4 |
| No. of 5-bit latch | : 1 |
| # Comparators | : 6 |
| No. of 32-bit comparator equal | : 1 |
| No. of 32-bit comparator greater equal | : 1 |
| No. of 32-bit comparator greater | : 1 |
| No. of 32-bit comparator less | : 1 |
| No. of 32-bit comparator less equal | : 1 |
| No. of 32-bit comparator not equal | : 1 |
| # Multiplexers | : 6 |
| No. of 1-bit 6-to-1 multiplexer | : 1 |
| No. of 2-bit 4-to-1 multiplexer | : 3 |
| No. of 32-bit 7-to-1 multiplexer | : 1 |
| No. of 32-bit 8-to-1 multiplexer | : 1 |
| # Xors | : 1 |
| No. of 32-bit xor2 | : 1 |
| *Design Statistics* | |
| No. of IOs | : 69 |
| ***Cell Usage:*** | |
| # BELS | : 8358 |
| #     GND | : 1 |
| #     INV | : 1025 |
| #     LUT1 | : 127 |
| #     LUT2 | : 118 |
| #     LUT3 | : 1528 |
| #     LUT3_D | : 1 |
| #     LUT3_L | : 13 |
| #     LUT4 | : 1188 |
| #     LUT4_L | : 2 |
| #     MUXCY | : 2145 |
| #     MUXF5 | : 64 |
| #     MUXF6 | : 32 |
| #     VCC | : 1 |
| #     XORCY | : 2113 |
| # FlipFlops/Latches | : 433 |
| #     FD | : 32 |
| #     FDC | : 32 |
| #     FDE | : 256 |
| #     FDP | : 1 |
| #     FDS | : 32 |
| #     LD | : 75 |
| #     LD_1 | : 5 |
| # Clock Buffers | : 6 |
| #     BUFG | : 5 |
| #     BUFGP | : 1 |
| # IO Buffers | : 68 |
| #     IBUF | : 2 |
| #     IOBUF | : 32 |

```
#     OBUF                    : 34
#     DSPs                     : 3
#     8DSPs                    : 3
```

***Device utilization summary***

```
Selected Device: 4vlx15sf363-12
Number of Slices:         2125 out of  6144   34%
Number of Slice Flip Flops: 401 out of 12288    3%
Number of 4 input LUTs: 2977 out of 12288   24%
Number of bonded IOBs:  69 out of   240   28%
IOB Flip Flops                    32
Number of GCLKs:               6  out of    32   18%  Number of DSP48s:              3  out of    32    9%
```

***Timing Summary***

```
Speed Grade: -12
Minimum period: 1.862ns (Maximum Frequency: 536.913MHz)
Minimum input arrival time before clock: 111.930ns
Maximum output required time after clock: 9.408ns
Maximum combinational path delay: 9.413ns
Total memory usage is 278656 kilobytes
```

## VI. Conclusion

The simulation of 64 bit RISC processor is done on Modelsim 10.1 b successfully. Different test benches are tested to simulate the design and 33 instructions are functionally verified. In device utilization report the number of slices are 34%, Number of Slice Flip Flops 3%, Number of 4 input LUTs 24%, Number of bonded IOBs 28%, IOB Flip Flops 32, Number of GCLKs 18% and Number of DSP48s 9% for the device 4vlx15sf363-12 on Spartan-3E FPGA. Minimum period is found 1.862ns, maximum frequency 536.913MHz, minimum input arrival time before clock 111.930ns, maximum output required time after clock 9.408ns and maximum combinational path delay is found 9.413ns. In the design data bus and address bus both are of 64 bits and optimized parameters are achieved in the design.

## Acknowledgments

The authors would like to express their sincere thanks to the support of Chitkara University, H.P, India to carry out this research work.

## References

[1] Brunelli Claudio, Cinelli Federico, Rossi Davide, NurmiJari, "A VHDL model And implementation of a coarse grain reconfigurable coprocessor for a RISC core", 2nd Conference on Ph.D. Research in Microelectronics and Electronics Proceedings, PRIME, 2006, p 229232.

[2] Bighneswar Panda. B. Vijay Bhaskar R. Surya Prakash, "32-Bit Risc Processor For Computer Architecture" International Journal of Engineering Research & Technology (IJERT) Vol. 1 Issue 8, Oct – 2012 page 1-6.

[3] Douglas L Perry "VHDL by examples" chapter-12 Top Level System Design, TMH New Delhi page 289-326

[4] H. Elaarag, "A complete design of a RISC processor for pedagogical purposes," Journal of Computing Sciences in Colleges, vol. 25, Issue 2, pp. 205-213, 2009.

[5] Imran Mohammad, Ramananjaneyulu K "FPGA Implementation of a 64-Bit RISC Processor Using VHDL" International Journal of Engineering Research and Applications (IJERA) ISSN: 2248-9622 Vol. 2, Issue 3, May-Jun 2012, pp.2544-2549

[6] Jiang, Hongtu, "FPGA implementation of controller data path pair in custom Image Processor design", IEEE International Symposium on Circuits and Systems Proceedings,2004, p V141V144.

[7] K. Vlachos, T. Orphanoudakis, Y. Papaeftathiou, N. Nikolaou, D. Pnevmatikatos, G. Konstantoulakis, J.A. Sanchez P., "Design and performance evaluation of a Programmable Packet Processing Engine (PPE) suitable for high speed network Processors units", Microprocessors and Microsystems 31, 2007, p 188–199.

[8] Rohit Sharma, Vivek Kumar Sehgal, Nitin Nitin1, PranavBhasker, IshitaVerma "Design and Implementation of a 64-bit RISC Processor using VHDL"UKSim 2009: 11th International Conference on Computer Modelling and Simulation, 978-0-7695-3593-7/09 $25.00 © 2009 IEEE DOI 10.1109/UKSIM.2009.30 page (568-574)

[9] R. Uma "Design and Performance Analysis of 8-bit RISC Processor using Xilinx Tool" International Journal of Engineering Research and Applications (IJERA) ISSN: 2248-9622 Vol. 2, Issue 2, Mar-Apr 2012, pp.053-058

[10] T. Moreo, P. N. Lorente, F. S. Valles, J. S. Muro, C. F. Andrés, Experiences on developing computer vision hardware algorithms using Xilinx system generators, Elsevier Journal of Microprocessors and Microsystems 29, 2005 pp 411- 419.