Four Stage Pipelined 16 bit RISC on Xilinx Spartan 3AN FPGA

Aboobacker Sidheeq.V.M
Department of Electrical & Computer Engineering,
Institute of Technology,
Hawassa University,
P. O. Box 05, Hawassa, Ethiopia

ABSTRACT
This paper describes the design and implementation of a 16 bit 4 stage pipelined Reduced Instruction Set Computer (RISC) processor on a Xilinx Spartan 3AN Field programmable gate array (FPGA). The processor implements the Harvard memory architecture, so the instruction and data memory spaces are both physically and logically separate. The RISC processor architecture presented in this paper is designed by six units, they are instruction cache, instruction unit, decode unit, execute unit, data cache unit and register file unit. The processor has been described using Verilog HDL, simulated using ModelSim 6.5-SE simulator and synthesized using Xilinx ISE 11.1i.The proposed processor has been implemented and physically tested Xilinx FPGA Spartan 3AN development board, It uses ChipScope Pro 9.2i embedded logic analyzer to monitor any or all of the signals in the design.

Keywords
FPGA, VERILOG HDL, RISC

1. INTRODUCTION
Reduced Instruction Set Computers (RISCs) are now in widespread use for all type of computational tasks [4]. In the area of scientific computing, RISC workstation are being increasingly used for compute intensive task such as digital signal and image processing. Pipelined RISC is an evolution in computer architecture, it emphasizes on speed and cost effectiveness over the ease of hardware description language programming and conservation of memory and RISC based designs will continue to grow in speed and ability, more rapidly than CISC design [1]. Pipelining, a standard feature in RISC processors, is much like an assembly line. Because the processor works on different steps of the instruction at the same time, more instructions can be executed in a shorter period of time [5].

FPGA based computing architectures offer a unique opportunity for the design of custom instruction processor matched to user specified application. One or several loop bodies can be synthesized on each of the FPGAs [8]. Internal FPGA register resources and direct wires can be used to establish high performance inter stage communication, avoiding excessive buffer read/write and speed, space, facilities and feature inclusion in their design. The designer may even specify the pin number and timing constraints to suit their requirements [4].

This paper presents a very simple 16 bit general purpose 4 stage pipelined processor on FPGA. The instruction cycle of pipeline stages are namely instruction fetch, instruction decode, ALU operation and register file write. After every instruction fetch, the program counter (PC) pointed to next the selected Instruction. The architecture in this paper supports 16 instructions, which are described in the figure 2.3 (Section II). They can be broadly classified into Arithmetic, Logical, Shifting and Rotational Instructions. In this paper all components of processor design have been designed in Verilog code, implemented and tested on Xilinx FPGA Spartan 3AN development board. ModelSim have been used for simulation. Before mapping the entire processor, all the six units in the entire processor architecture is separately mapped to the FPGA and verified by executing a random testbench which is also synthesized to the FPGA [4].

This paper is organized as follows. There are 6 sections in this paper. The introduction is given in section I; Section II describes about the system architecture. It also explains the each instruction format and addressing modes used in the processor. The design flow of 16 bit RISC processor architecture is given in section II; Section IV describes the simulation and synthesis results of the pipelined design by selecting best and suitable Xilinx Spartan 3an FPGA. It also gives the detailed synthesis report and other parameters, which is required for the optimization of the design. The simulation results consists of all the six modules separately and the combined top module waveforms in the system architecture ; The conclusion at the end in section V.

2. INSTRUCTION SET AND PROCESSOR ARCHITECTURE
A pipelined RISC improves CPU speed and system throughput because several instruction can be processed in parallel [1]. The CPU can begin processing the next instruction before the current instruction is completed, that is the system can overlap the execution of contiguous instructions [5].

Fetching the instruction from main memory or cache is a major bottleneck due to the relatively slow access time. The slow access time can be alleviated by prefetching instruction before they are required by processing unit. The prefetched instructions are loaded into a prefetch buffer where they are retained until needed by the processor. Prefetching can be overlapped with normal processing and can be accomplished on unused memory bus cycle [1] [7].

There are many different stages in a pipeline, with each stage performing one unique operation in the instruction processing. When the pipeline is full, a result is obtained every clock cycle [1]. A 4 stage pipeline is shown in Figure 1. The Instruction fetch stage fetches the instruction from memory, The Decode stage decode the instruction and fetches the operands, The Execute stage perform the operation specified in the instruction, The store stage store the result in the destination location. Four instructions are in progress at any

given clock cycle. Each stage of the pipeline performs its task in parallel with all other stages.

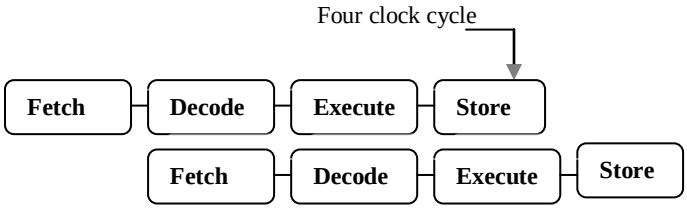

Figure 1: Example of 4 stage pipeline

If the instruction required is not available in cache, then a cache miss occur, necessitating a fetch from main memory [7]. This is referred to as a pipeline stall and delays processing the next instruction. Information is passed from one stage to the next by means of storage buffer as shown in Figure 2. There must be a register in the input of each stage (or between stages) to store information that is transmitted from the preceding stage. This prevents data being processed by one stage from interfering with the following stage during the same clock period.

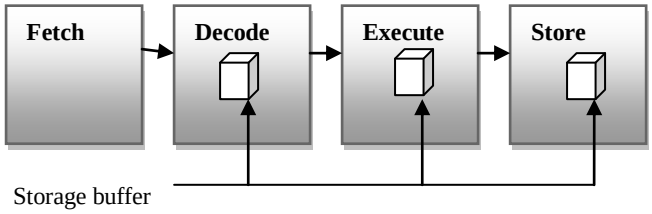

Figure 2: Four stage of pipeline showing the Inter stage storage buffer

RISC processors have only a small number of instructions compared to complex instruction set computer (CISC). The instructions are also smaller in size with a smaller number of field and usually of fixed length. Most instructions have same format with limited number of addressing modes and executed by hardware. RISC processors have an instruction cache and data cache, only load and store instructions reference memory [5].

Opcode	Opnd A	Opnd B	Dst
nnnn	RRRR	RRRR	RRRR

Load Operation

Opcode		Opnd A		Dst
1110 (Load)		Memory Address		RRR
15 12	1110	9 5	4	3 0

Store Operation

Opcode		Opnd A	Dst
1111 (Load)		RRR	Memory Address
15 12	11 9	8 5	4 0

PROGRAM			
LD	Regfile[0:15]=dcache[0:15]	NEG	See Instruction Cache
LD		SHR	
LD		SHL	
LD		ROR	
LD		ROL	
LD		ST	Dcache[8:15]=regfile[0:15]
LD		ST	
LD		ST	
LD		ST	
LD		ST	
LD		ST	
LD		ST	
LD		ST	
LD		ST	
LD		ST	
ADD	See Instruction Cache	ST	
SUB		ST	
AND		ST	
OR		ST	
XOR		ST	
INC		NOP	Clear
DEC		NOP	
NOT		NOP	

Figure 3: Instructions & Instructions set format for the pipelined RISC processor

The operands are loaded from register file and stored in the register file. There are large register files that provide fast access, thereby eliminating many slow memory accesses [1]. The control unit is hardwired, not micro program controlled, for increased speed [7].

Design module presents the Verilog design of pipelined RISC processor with no prefetch buffer. Although this is a small RISC processor, the same technique can be applied to any size processor. There are 16 instructions and the various fields for each instruction as shown in Figure3.All instructions, except load and store, have the same format.

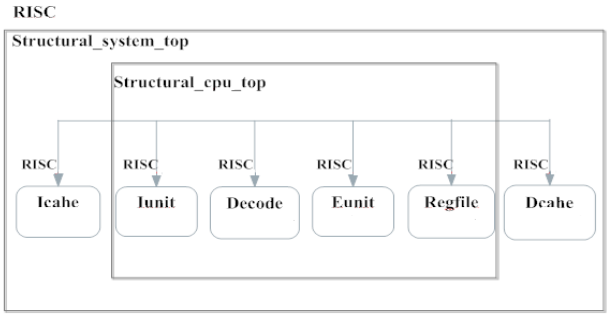

Figure 4: Structural block diagram for the pipelined RISC processor

There are six units (modules) in the processor architecture of 16 bit pipelined RISC. Instruction cache (icache), instruction unit (iunit), decode unit (decode), execution unit (eunit), data cache (dcache) and register file (regfile). Each unit will be behavioral oXr mixed design module and will be compiled

and simulated using a test bench to show binary output and waveforms. The entire processor will then configure using a structural module and will display the waveforms. The structural block diagram of Figure 4 shows the hierarchy. The system architecture of Figure 5 shows the six units and their interconnections.

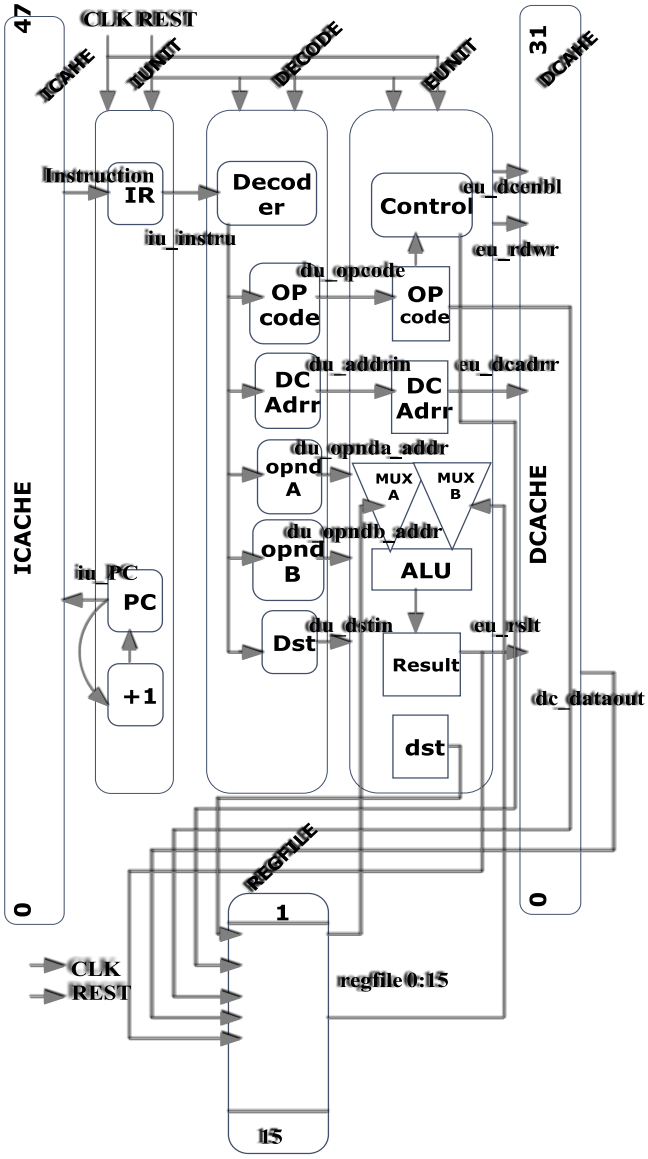

Figure 5: System Architecture for the pipelined RISC processor

The instruction cache and data cache will be preloaded in their respective modules with the instruction and data. The instruction cache content shows all the instruction and the application program. A predefined program is used so that the result can be easily verified. The operand size is sixteen bits.

RISC instruction typically has a fixed length instruction format [6]. The instruction length for this design is 16 bits. Only load and store instruction can access memory [1]. The addressing mode is register that means operand is in the register file. In this design there are 16 registers, which are initialized by 16 load instruction that load certain data cache contents into the register file. The initial contents of register file shown in Table 1. The final contents of register file are shown in Table 2.

Table 1. Register file contents before execution

Address	Data	Address	Data
0000	0000 0000 0000 0000	1000	0010 0010 0000 0000
0001	0000 0000 0100 0100	1001	0100 0100 0000 0000
0010	0000 0000 1000 1000	1010	1000 1000 0000 0000
0011	0000 0000 1011 1011	1011	1010 1010 0000 0000
0100	0000 0000 1111 1111	1100	1011 1011 0000 0000
0101	0100 0100 0000 0000	1101	1100 1100 0000 0000
0110	1000 1000 0000 0000	1110	1101 1101 0000 0000
0111	1011 1011 0000 0000	1111	1111 1111 0000 0000

Table 2. Register file contents after execution

Address	Data	Address	Data
0000	0000 0000 0100 0100	1000	1000 0111 1111 1111
0001	0000 0000 0100 0100	1001	0100 0100 1111 1111
0010	0000 0000 1000 1000	1010	0000 0000 0000 0000
0011	0100 0100 0000 0001	1011	0000 0000 0010 0010
0100	0100 0100 1111 1111	1100	0000 0001 0001 0000
0101	0000 0000 1111 1111	1101	0000 0000 1111 1111
0110	0000 0000 1000 1000	1110	0000 0001 1111 1110
0111	1111 1111 1011 1100	1111	1111 1111 0000 0000

2.1 Instruction Cache

The small dark squares represent the ports of module. There are two ports in the instruction cache. Instruction, which sends 16 bit instruction to the iunit, and a port for 6 bit program counter **pc** that receives the necxt instruction cache address from iunit. The instruction cache is as shown in the Figure 6.

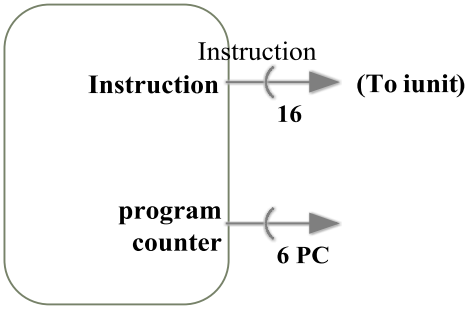

Figure 6: Instruction cache for the pipelined RISC processor

2.2 Instruction Unit

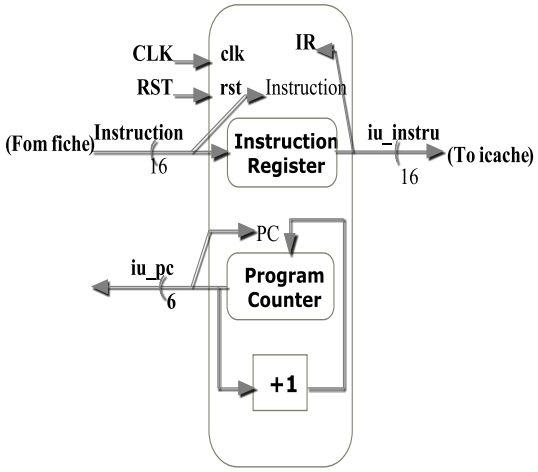

Figure 7: Instruction unit for the pipelined RISC processor

The instruction unit has five ports. One each for the clock clk and reset rst inputs, a port to receive a 16 bit instruction from the instruction cache called instruction[15:0] that is loaded into the instruction register ir, the net name from the instruction cache is labeled instruction[15:0], a port the output of the program counter pc[5:0] that sends the program counter to the instruction cache on a net labeled iu_pc[5:0] and a port called ir[15:0] that passes the 16 bit instruction to the decode unit on a net called iu_instr[15:0]. The instruction unit is as shown in the Figure 7.

2.3 Decode Unit

The decode unit has three input ports and five output ports. The input ports are clk and rst, plus a port instr to receive the instruction from the instruction unit. There is an output port opcode[3:0] that sends the operation code to the execution unit on a net labeled du_opcode[3:0]. A port dcaddr[4:0] that sends the data cache address to the execution unit on net labeled du_dcaddr4:0]; a port opnda[3:0] that sends the address of operand A to the execution unit on a net labeled du_opnd_addr[3:0], a net opndb[3:0] that sends the address of operand B to the execution unit on a net labeled du_opndb_addr[3:0], and a port dst that sends the destination address to the execution unit on a net labeled du_dst[3:0] to be used by the register file. The decode unit is as shown in the Figure 8.

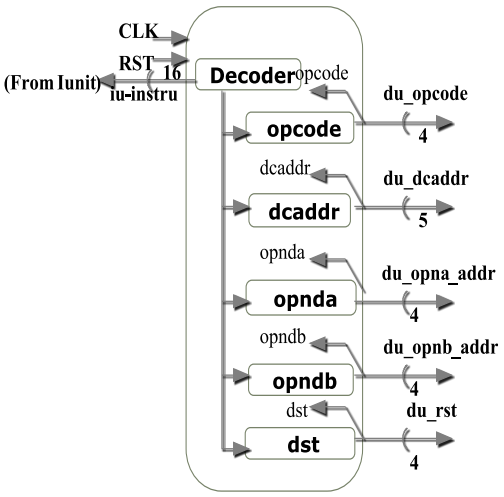

Figure 8: Decode unit for pipelined RISC processor

2.4 Execution Unit

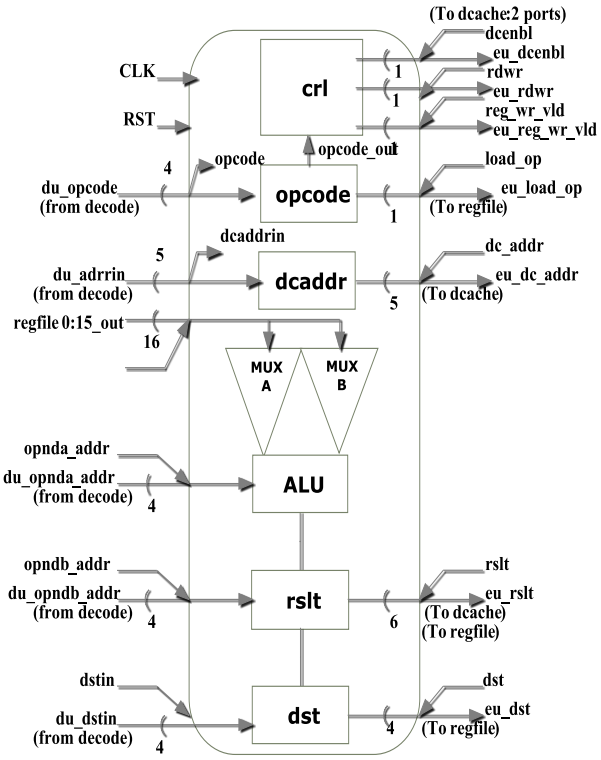

Figure 9: Execution unit for the pipelined RISC

The execution unit is the dominant unit in the processor all operations are performed by execution unit. It obtains the opcode and operand address from the decode unit. There are sixteen 16 bit input ports that connect the sixteen registers of the register file to the ALU by means of multiplexers. The multiplexers inputs are selected by signals from the ctrl block.

The operand addresses then select the appropriate operands to be utilized in the execution of the instruction. The result the operation is placed in an rslt register also controlled by the ctrl block then sent to the register file and data cache. This will become more apparent when the execution unit module is presented. The execution unit is as shown in the Figure 9.

2.5 Register File Unit

The reg_wr_vld port is an input from the execution unit that enables a write operation to the register file. The signal at the load_op port then loads the register file with the result of an instruction execution or data from the data cache. The result of an instruction execution is available at the input port rslt. The data from the data cache is available at the input port dcdataout. There are sixteen 16 bit output ports that provide data to the execution unit. A logic diagram for the register file will be shown when the register file module is presented. The register file unit is as shown in the Figure 10.

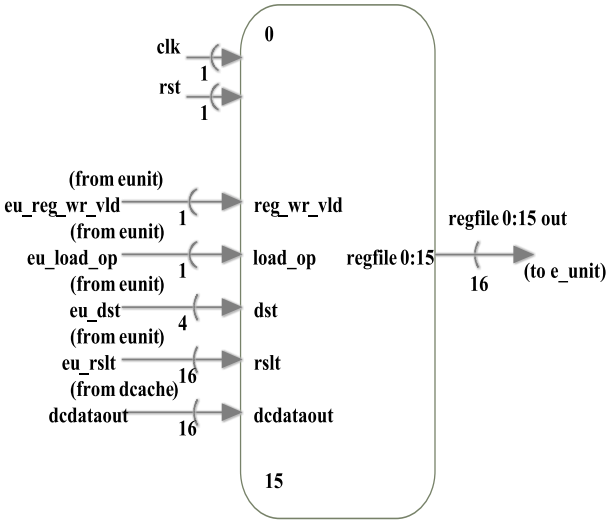

Figure 10: Register file for the pipelined RISC processor

2.6 Data Cache Unit

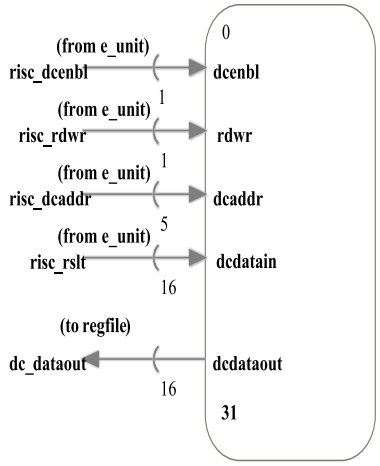

Figure 11: The data cache unit for the pipelined RISC processor

The data cache input port dcenbl enables the data cache for a read or write operation. The rdwr port provides a read enable (rdwr=1) or a write enable (rdwr=0) for the data cache at the address specified by the address at port dcaddr. The data to be written to the data cache is available at the dcdatain port. The data to be read from the data cache is available at the dcdataout port. The data cache unit is as shown in the Figure 11.

3. HARDWARE AND SOFTWARE DESIGN FLOW

The design phases of the 16 bit pipelined RISC processor architecture are Instruction cache (icache), instruction unit (iunit), decode unit (decode), execution unit (eunit), data cache (dcache) and register file (regfile). Each unit will be behavioral or mixed design module and will be compiled and simulated using a test bench to show binary output and waveforms. The entire processor will then configure using a structural module and will display the waveforms.

There are several differences between the traditional software design flow and the established Verilog design flow for FPGAs [2]. After designing and implementing hardware design there is a multistage process to go through before the design can be used in an FPGA [4]. The first stage is Synthesis, which takes HDL code and translates it into a netlist. A netlist is a textual description of a circuit diagram or schematic. Next, simulation is used to verify that the design specified in the netlist functions correctly. Once verified, the netlist is converted into a binary format (Translate), the components and connections that it defines are mapped to CLBs (Map) and the design is placed and routed to fit onto the target FPGA (Place and route)[8][2]. A second simulation (post place and route simulation) is performed to help establish how well the design has been placed and routed. Finally, a .bit file is generated to load the design onto an FPGA [2]. A .bit file is a configuration file that is used to program all of the resources within the FPGA. Using tools such as Xilinx ChipScope it is then possible to verify and debug the design while it is running on the FPGA [4]. The software design flow has no requirement for a pre implementation simulation step.

Compile times for software are much shorter than implementation times for hardware designs [2]. So it is practical to recompile code and perform debugging as an iterative process. In hardware it is very important to establish that a design is functionally correct prior to implementation as a broken design FPGA [8].

4. EXPERIMENTS AND RESULTS

4.1 Simulation Results

Simulation output is generated in the form of a waveform for visual inspection or data files for machine readability. The simulation results of all six modules in the design phase as explained in the following sections.

4.1.1 Instruction Cache

Instructions are assigned to the the system task ***$readmemb***,which reads and load data from a specified text file into the instruction cache. The text file is saved in the project file as *icache.instr* with no *.v extesin.* The outputs are shown in Table 3

Table 3 .Outputs for instruction cache

# address	0=e000	# address	0=e000
# address	1=e021	# address	1=e021
# address	2=e042	# address	2=e042
# address	3=e063	# address	3=e063
# address	4=e084	# address	4=e084
# address	5=e0a5	# address	5=e0a5
# address	6=e0c6	# address	6=e0c6
# address	7=e0e7	# address	7=e0e7
# address	8=e108	# address	8=e108
# address	9=e129	# address	9=e129
# address	10=e14a	# address	10=e14a
# address	11=e16b	# address	11=e16b
# address	12=e18c	# address	12=e18c
# address	13=e1ad	# address	13=e1ad
# address	14=e1ce	# address	14=e1ce
# address	15=e1ef	# address	15=e1ef
# address	16=1010	# address	16=1010
# address	17=2127	# address	17=2127
# address	18=3236	# address	18=3236
# address	19=4345	# address	19=4345
# address	20=5454	# address	20=5454
# address	21=6563	# address	21=6563
# address	22=7678	# address	22=7678
# address	23=8709	# address	23=8709

4.1.2 Instruction Unit

The outputs are shown in Table 4.The program counter places the instruction on *icdataout* and the next load the instruction into instruction register.

Table 4. Outputs for the instruction unit

pc=xx,instuction=e000,ir=xxxx
pc=01,instuction=e021,ir=e000
pc=02,instuction=e042,ir=e021
pc=03,instuction=e063,ir=e042
pc=04,instuction=e084,ir=e063
pc=05,instuction=e0a5,ir=e084
pc=06,instuction=e0c6,ir=e0a5
pc=07,instuction=e0e7,ir=e0c6
pc=08,instuction=e108,ir=e0e7
pc=09,instuction=e129,ir=e108
pc=0a,instuction=e14a,ir=e129
pc=0b,instuction=e16b,ir=e14a
pc=0c,instuction=e18c,ir=e16b
pc=0d,instuction=e1ad,ir=e18c
pc=0e,instuction=e1ce,ir=e1ad
pc=0f,instuction=e1ef,ir=e1ce
pc=10,instuction=2127,ir=e1ef
pc=11,instuction=3236,ir=2127
pc=12,instuction=4345,ir=3236
pc=13,instuction=5454,ir=4345
pc=14,instuction=6563,ir=5454
pc=15,instuction=7678,ir=6563
pc=16,instuction=8709,ir=7678
pc=17,instuction=901a,ir=8709
pc=18,instuction=a12b,ir=901a
pc=19,instuction=b23c,ir=a12b
pc=1a,instuction=c34d,ir=b23c
pc=1b,instuction=d45e,ir=c34d
pc=1c,instuction=f010,ir=d45e
pc=1d,instuction=f031,ir=f010
pc=1e,instuction=f052,ir=f031
pc=1f,instuction=f073,ir=f052
pc=20,instuction=f094,ir=f073
pc=21,instuction=f0b5,ir=f094
pc=22,instuction=f0d6,ir=f0b5
pc=23,instuction=f0f7,ir=f0d6
pc=24,instuction=f118,ir=f0f7
pc=25,instuction=f139,ir=f118
pc=26,instuction=f15a,ir=f139
pc=27,instuction=f17b,ir=f15a
pc=28,instuction=f19c,ir=f17b
pc=29,instuction=f1bd,ir=f19c
pc=2a,instuction=f1de,ir=f1bd
pc=2b,instuction=f1ff,ir=f1de
pc=2c,instuction=0000,ir=f1ff
pc=2d,instuction=0000,ir=0000
pc=2e,instuction=0000,ir=0000

4.1.3 Decode Unit

The 16 bit instruction received from the instruction unit is decoded into its constituent parts. A 4 bit operation code, a 4

bit operand *A*, a 4 bit operand *B* and a 4 bit designation address in the register file.

If the operation is a load instruction (load register file from data cache), then data cache address ins *instr[9:5]* and the destination address in the register file is *instr[3:0]*.If the operation is store instruction (store register file to data cache),then register file address is *instr[8:5]* and data cache address is *instr[4:0]*. The outputs are shown in Table 5.

Table 5: Outputs for decode unit

# opcode=1110,dcaddr=00000,dst=0000
# opcode=1110,dcaddr=00001,dst=0001
# opcode=1110,dcaddr=00010,dst=0010
# opcode=1110,dcaddr=00011,dst=0011
# opcode=1110,dcaddr=00100,dst=0100
# opcode=1110,dcaddr=00101,dst=0101
# opcode=1110,dcaddr=00110,dst=0110
# opcode=1110,dcaddr=00111,dst=0111
# opcode=1110,dcaddr=01000,dst=1000
# opcode=1110,dcaddr=01001,dst=1001
# opcode=1110,dcaddr=01010,dst=1010
# opcode=1110,dcaddr=01011,dst=1011
# opcode=1110,dcaddr=01100,dst=1100
# opcode=1110,dcaddr=01101,dst=1101
# opcode=1110,dcaddr=01110,dst=1110
# opcode=1110,dcaddr=01111,dst=1111
opcode=0001,opnda=0000,opndb=0001,dst=0000
opcode=0010,opnda=0001,opndb=0010,dst=0111
opcode=0011,opnda=0010,opndb=0011,dst=0110
opcode=0100,opnda=0011,opndb=0100,dst=0101
opcode=0101,opnda=0100,opndb=0101,dst=0100
opcode=0110,opnda=0101,opndb=0110,dst=0011
opcode=0111,opnda=0110,opndb=0111,dst=0010
opcode=1000,opnda=0111,opndb=0000,dst=0001
opcode=1001,opnda=0000,opndb=0001,dst=0000
opcode=1010,opnda=0001,opndb=0010,dst=0001
opcode=1011,opnda=0010,opndb=0011,dst=0010
opcode=1100,opnda=0011,opndb=0100,dst=0011
opcode=1101,opnda=0100,opndb=0101,dst=0100
# opcode=1111,dcaddr=10000,opnda=0000
# opcode=1111,dcaddr=10001,opnda=0001
# opcode=1111,dcaddr=10010,opnda=0010
# opcode=1111,dcaddr=10011,opnda=0011
# opcode=1111,dcaddr=10100,opnda=0100
# opcode=1111,dcaddr=10101,opnda=0101
# opcode=1111,dcaddr=10110,opnda=0110
# opcode=1111,dcaddr=10111,opnda=0111
# opcode=1111,dcaddr=11000,opnda=1000
# opcode=1111,dcaddr=11001,opnda=1001
# opcode=1111,dcaddr=11010,opnda=1010
# opcode=1111,dcaddr=11011,opnda=1011
# opcode=1111,dcaddr=11100,opnda=1100
# opcode=1111,dcaddr=11101,opnda=1101
# opcode=1111,dcaddr=11110,opnda=1110
# opcode=1111,dcaddr=11111,opnda=1111
opcode=0000,opnda=1111,opndb=0000,dst=0000
opcode=0000,opnda=0000,opndb=0000,dst=0000
opcode=0000,opnda=0000,opndb=0000,dst=0000

4.1.4 Execution Unit

The multiplexer logic that guides the operands to the ALU. The instruction and their functions are listed in Table 6.There are two sets of sixteen 16:1 multiplexer, one set for operand *A* and one set of operand *B*. Each multiplexer selects bit 0 through 15 of register file 0 through register file 7 for the appropriate operand, as a function of the multiplexer select inputs.

Table 6. Instruction for execution

Instruction	Function
Nop	No operation
Add	Operand *A* plus operand *B*
Sub	Operand *A* minus operand *B*
And	Operand *A* AND operand *B*
Or	Operand *A* OR operand *B*
Xor	Operand *A* EXCLUSIVE-OR operand *B*
Inc	Increment Operand *A* by 1
Dec	Decrement Operand *A* by 1
Not	Form the 1s complement of Increment of Operand *A*
Neg	Form the 2s complement of Increment of
Shr	

Shl	Operand *A* Shift right logical Operand *A* Shift left logical Operand *A*
Ror Rol Id st	Rotate right logical Operand *A* Rotate left logical Operand *A* Load register file from memory Store register file to memory

Operation code 0001 is an ADD operation. The operand *A* address is register file0, which contains the value 0000_0000_0000_ 0000.The operand *B* address is register file 7,which contains the value 0000_0000_0100_0100.After adding the two operands,the result is 0000_0000_0100_0100.

Now consider an *exclusive_OR* instruction with an operation code of 0101.Operand *A* is in register file 4, which contains the value 0000_0000_1111_1111.

The *negate* instruction has an operation code of 1001 and obtain the 2's complement of operand *A*,which is located in register file 0 with a value of 0000_0000_0000_0000.When the 2's complement of 0000_0000_0000_0000 is obtained the result will still be a value of 0000_0000_0000_0000.

The *shift right* instruction has an operation code of 1010.The operation is performed on operand *A* only,which is loaded in he register file 1 containing a value of 0000_0000_0100_0100.After shifting one bit position ,The result is 0000_0000_0010_0010.

The *rotatae left* instruction has an operation code of 1101 and uses operand *A* only,which is contained in the register file 4and has a value of 0000_0000_1111 _1111.After the execution of the instruction,the result is 0000_0001_1111_1110.

4.1.5 Register File

A register file is selected by means of 5:8 decoder with input that are destination address *dst[3:0]*.The output of decoder in conjunction with a valid write signal(*reg_wr_vld*) selects the appropreiate register file.The data that is to be written to the register file comes from one of two sources,from the result of an operation rslt[15:0] or from the data cache *dataout[15:0]*.

4.1.6 Data Cache

The data cache is a memory of 32,16 bit word that is preloaded with specific operands so that the result of the program execution will known.The data cache has five ports, they are *dcenbl* to enable cache for a read or write operation.rdwr to allow reading*(rdwr=1)*or writing(*rdwr=0*),*dacaddr[4:0]* selects an address in the cache to either read or write data,*dcdataout[15:0]* provides data for a write operation and *dcdataout[15:0]* provide cache contents for read operation.

The test bench reads the content of the cache addresses dcache[00000] through dcache[10000],then writes 0000_0000_0000_0000 to cache address dcache[10001] through dcache[11111].The ouputs are shown in Table 9.

Table 9. Outputs for the data cache

```
# dcaddr=00000,dcdatain=0000,dcataout=0000
# dcaddr=00001,dcdatain=0000,dcataout=0044
# dcaddr=00010,dcdatain=0000,dcataout=0088
# dcaddr=00011,dcdatain=0000,dcataout=00bb
# dcaddr=00100,dcdatain=0000,dcataout=00ff
# dcaddr=00101,dcdatain=0000,dcataout=4400
# dcaddr=00110,dcdatain=0000,dcataout=8800
# dcaddr=00111,dcdatain=0000,dcataout=bb00
# dcaddr=01000,dcdatain=0000,dcataout=2200
# dcaddr=01001,dcdatain=0000,dcataout=4400
# dcaddr=01010,dcdatain=0000,dcataout=8800
# dcaddr=01011,dcdatain=0000,dcataout=aa00
# dcaddr=01100,dcdatain=0000,dcataout=bb00
# dcaddr=01101,dcdatain=0000,dcataout=cc00
# dcaddr=01110,dcdatain=0000,dcataout=dd00
# dcaddr=01111,dcdatain=0000,dcataout=ff00
# dcaddr=10000,dcdatain=0000,dcataout=ff00
# dcaddr=10001,dcdatain=1111,dcataout=ff00
# dcaddr=10010,dcdatain=2222,dcataout=ff00
# dcaddr=10011,dcdatain=3333,dcataout=ff00
# dcaddr=10100,dcdatain=4444,dcataout=ff00
# dcaddr=10101,dcdatain=5555,dcataout=ff00
# dcaddr=10110,dcdatain=6666,dcataout=ff00
# dcaddr=10111,dcdatain=7777,dcataout=ff00
# dcaddr=11000,dcdatain=8888,dcataout=ff00
# dcaddr=11001,dcdatain=9999,dcataout=ff00
# dcaddr=11010,dcdatain=aaaa,dcataout=ff00
# dcaddr=11011,dcdatain=bbbb,dcataout=ff00
# dcaddr=11100,dcdatain=cccc,dcataout=ff00
# dcaddr=11101,dcdatain=dddd,dcataout=ff00
# dcaddr=11110,dcdatain=eeee,dcataout=ff00
#dcaddr=11111,dcdatain=ffff,dcataout=ff00
```

4.1.7 System Top

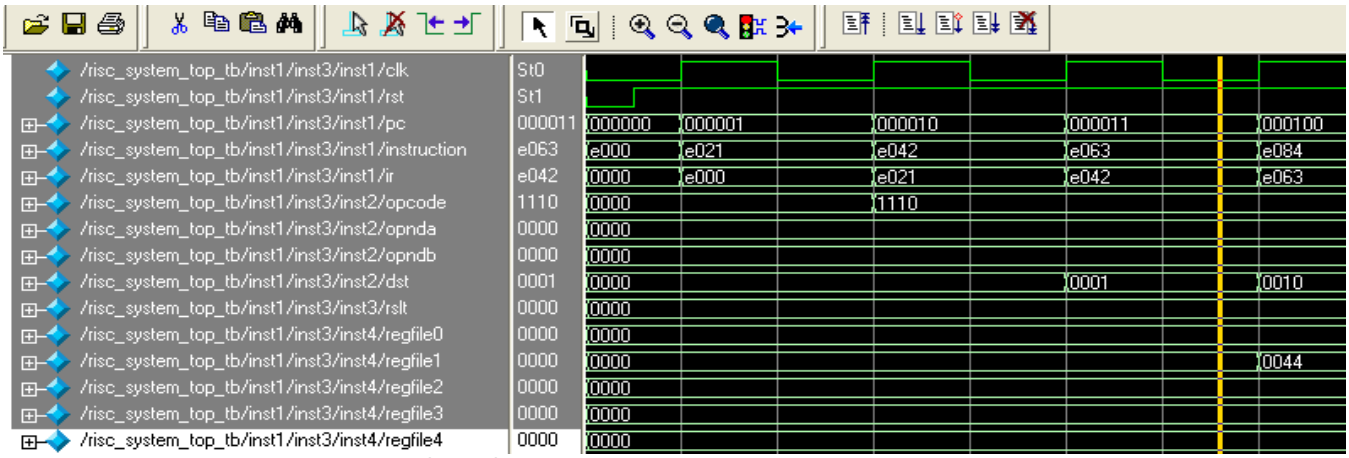

Figure 12:Simulation waveforms for pipelined RISC processor

The system top structural module instatiates the *risc_cpu_top* module together with the instruction cache and data cache.The only inputs are reset and clock signal.

The register file contents at the end of the waveform correctly depicts the result of program and based upon the original register file contents.Although this was a comparatively simple processor design,the same rationale applies equaly well to a more complex pielined RISC processor.

The simulation waveforms are displayed in the Figure 12 and show the complete execution of the application program together with the values of the register and wires for the modules.

4.2 Synthesis Results on Spartan 3AN FPGA

A bitstream file needs to be prepared for each design and downloaded onto the Pegasus prototyping board. The design step as follows

1. In order to test the design in the Pegasus board, the inputs need to be connected to the switches/buttons on the board and the outputs need to be connected to the onboard LED's.

2. Assign pin numbers to the input and output pins in the Verilog design file using a User Constraint File (ucf file). The pin numbers can be assigned by looking at Figure 13.

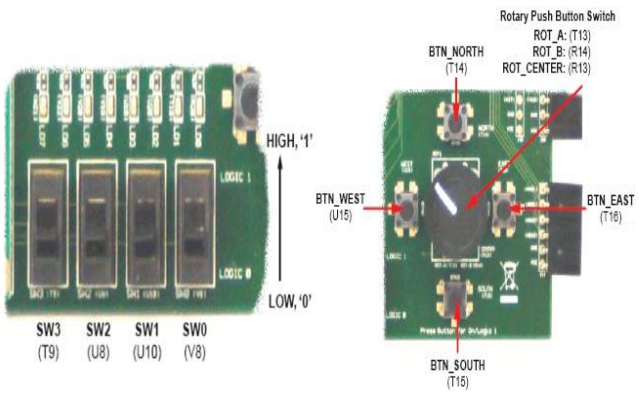

Figure 13: Slide Switches and Push Button Switches

Ensure that the Programmed Successfully message appears in the message window. In order to use the respective input/output device on the board, the pin number of the device must be connected properly to the designs input/output. If the Programmed Successfully message appears in the message window and can start testing the design in the FPGA board using the input and output devices on the board. The Figure 14 shows downloaded the design to target device FPGA.

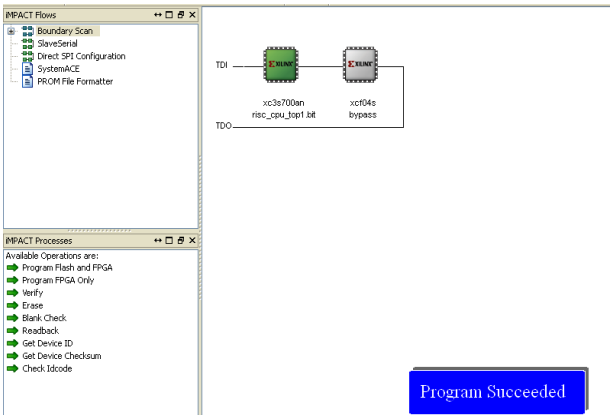

Figure 14: Downloading to target device FPGA

If this does not appear, it could be due to the following reasons.

1. The JTAG cable is not connected between the FPGA board and the PC parallel port.

2. You did not select the proper device for download in the JTAG chain.

3. The bitstream generated was not for the device: XC3S700AN -FGG484.

Once design is on the board and can actually use ChipScope to debug the design. Start the ***ChipScope Pro Analyzer***. It can run it either run it directly from the Start Menu or can use the ***Analyze Design Using ChipScope*** option from Processes window in Project Navigator.

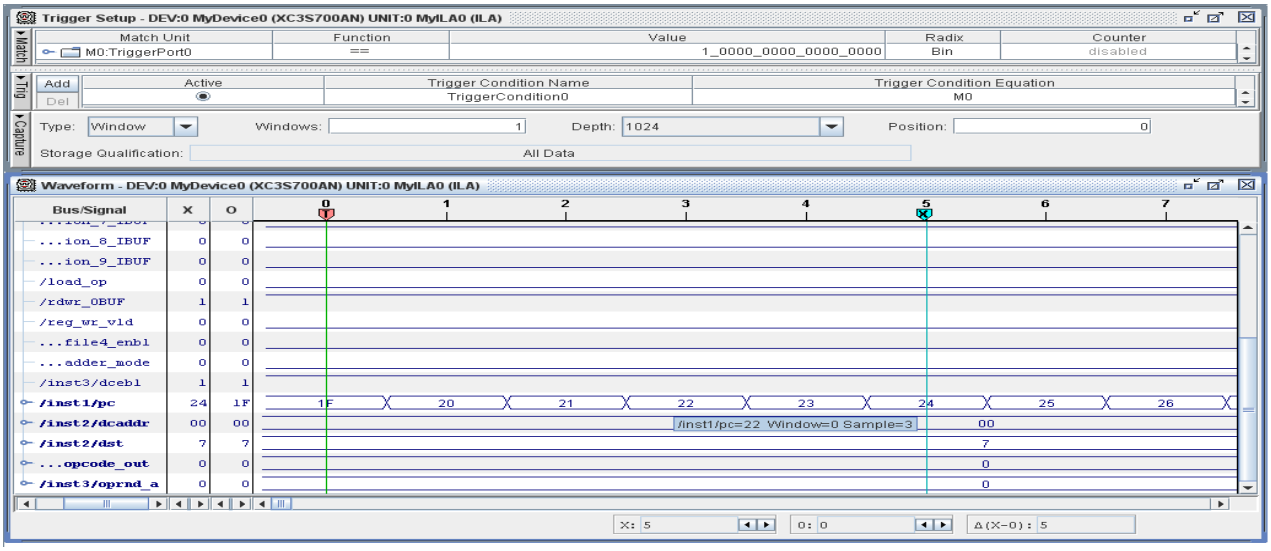

Figure 15: RISC Signals in ChipScope Pro Logic Analyzer

To view the waveform for a particular ILA or IBA core, select ***Window → New Unit Windows*** and the core desired. A dialog box appears for that ChipScope Pro Unit and the user can select the ***Trigger Setup***, ***Waveform***, ***Listing*** and/or ***Bus Plot*** window or any combination. The RISC signals displays on the waveform window as shown in the Figure 15. The Waveform window displays the sample buffer as a waveform display, similar to many modern simulators and logic analyzers.

5. CONCLUSION

In this paper we have presented 16 bit pipelined RISC Processor on Xilinx Spartan3AN FPGA. The design has been done in Verilog hardware description language and tested on Xilinx Spartan3AN development broad using ChipScope logic analyzer. All the blocks of processor were individually simulated using ModelSim 6.5-SE simulator and synthesized using Xilinx ISE 11.1i. Spartan3AN FPGA was sufficient for implementing the whole design into a real hardware, since the total available logic gate in Spartan3AN is 700K Logic Gate, which was more enough for implementing the whole processor. A simple sequential blocks performance was observed under the constraints clock frequency 50 MHz

6. REFERENCES

[1] J.L.Hennessy, D.A.Patterson.2003,'Computer Organization and Design: The Hardware/Software Interface', 2nd Edition, Morgan Kaufmann.

[2] D. J. Smith. (2010), 'HDL Chip Design', International Edition, Doone Publications,

[3] A. S. Tanenbaum. 2000, 'Structured Computer Organization', 4th Edition, Prentice-Hall

[4] Luker, Jarrod D., Prasad, Vinod B.2001, 'RISC system design in an FPGA', MWSCAS 2001, v2, , p532536.

[5] David A. Patterson, and John L Hennessy. 2006, 'Computer Architecture A Quantitative Approach', 4th Edition;.

[6] Vincent P. Heuring, and Harry F. Jordan. 2003, 'Computer Systems Design and Architecture', 2nd Edition..

[7] W. Stallings. 2003 'Computer Organization & Architecture. Designing for Performance'. Prentice Hall, 6th edition.

[8] Wayne Wolf. 2005, 'FPGA Based System Design', Prentice Hall.

[9] Samir Palnitkar.1996,'Verilog HDL A guide to Digital Design and Synthesis', SunSoft Pre.

[10] ChipScope pro logic anlyser.2010, 'Support, Documentation and Software Manual. Available at: http://www.xilinx.com/support/documentation/sw_manuals/chipscope_pro_sw_cores_10_1_ug029.pdf'

[11] Brunelli Claudio. 2006, Cinelli Federico, Rossi Davide, Nurmi Jari, "A VHDL model and implementation of a coarsegrain reconfigurable coprocessor for a RISC core", 2nd Conference on Ph.D. Research in MicroElectronics andElectronics Proceedings,PRIME, , p 229232.

7. ABOUT AUTHOR'S

Aboobacker Sidheeq.V.M is a Lecturer in Electrical and Computer Engineering at Hawassa University, Ethiopia. He received M-tech in VLSI Design and Embedded System from Visvesvaraya Technological University, and B-tech in Electronics and communication from Calicut University. His current research interests include high-speed parallel and VLSI computer architecture, computer-aided design, and reconfigurable computing.