Volume 2, No. 9, September 2011
Journal of Global Research in Computer Science
ISSN-2229-371X
RESEARCH PAPER
Available Online at www.jgrcs.info

DESIGN OF 32-BIT RISC CPU BASED ON MIPS

N.Alekya*[1] ,P.Ganesh Kumar[2]
[1]Department of ECE, Kakinada Institute of Engineering & Technology, Kakinada, AP, INDIA
alekhya.nakka@gmail.com [1]
[2]Assistant Professor ,Department of ECE, Kakinada Institute of Engineering & Technology, Kakinada, AP, INDIA

Abstract: The main aim of the project is simulation and synthesis of the 32-bit RISC CPU based on MIPS. The project involves design of a simple RISC processor and simulating it. A Reduced Instruction Set compiler (RISC) is a microprocessor that had been designed to perform a small set of instructions, with the aim of increasing the overall speed of the processor .In this work, we analyze MIPS instruction format, instruction data path, decoder module function and design theory based on RISC (Reduced Instruction Set Computer) CPU instruction set. Furthermore, we use pipeline design process to simulate successfully, which involves instruction fetch (IF), instruction decoder (ID), execution (EXE), data memory (MEM), write back (WB) modules of 32-bit CPU based on RISC CPU instruction set. Function of IF module is fetches the instruction from memory. The function of ID stage is sends control commands i.e., instructions are sending to control unit and decoded here. The EXE stage executes arithmetic. Main component of the EXE stage is ALU. The MEM stage is to fetch data from memory and store data to memory, if instruction is not memory/IO instruction, result is sent to WB stage. At last WB stage charges of writing the results, stores data and input data to register file. The purpose of WB stage is to write data to destination register.The idea of this project was to create a RISC processor as a building block in VHDL than later easily can be included in a larger design. It will be useful in systems where a problem is easy to solve in software but hard to solve with control logic. However at a high level of complexity it is easier to implement the function in software. In this project for simulation we use Modelsim for logical verification, and further synthesizing it on Xilinx-ISE tool using target technology and performing placing & routing operation for system verification. The language we used here is VHDL, and tools required here are MODELSIM III SE 6.4b – Simulation XILINX-ISE 10.1 – Synthesis. The applications are automatic robot control, bottling plant.

Keywords: *RISC, MIPS,Simulation Synthesis ,Instruction Set,MODELSIM.s*

INTRODUCTION

Risc Mips Features:

Processors are much faster than memories. For example, a processor clocked at 100 MHz would like to access memory in 10 nanoseconds, the period of its 100 MHz clock. Unfortunately, the memory interfaced to the processor might require 60 nanoseconds for an access. So, the processor ends up waiting during each memory access, wasting execution cycles.

To reduce the number of accesses to main memory, designers added instruction and data cache to the processors. A cache is a special type of high speed RAM where data and the address of the data are stored. Whenever the processor tries to read data from main memory, the cache is examined first. If one of the addresses stored in the cache matches the address being used for the memory read (called a hit), the cache will supply the data instead. Cache is commonly ten times faster than main memory, so you can see the advantage of getting data in 10 nanoseconds instead of 60 nanoseconds. Only when we miss (i.e., do not find the required data in the cache), does it take the full access time of 60 nanoseconds. But this can only happen once. Since a copy of the new data is written into the cache after a miss. The data will be there the next time we need it. Instruction cache is used to store frequently used instructions. Data cache is used to store frequently used data. Implementing fewer instructions and addressing modes on silicon reduces the complexity of the instruction decoder, the addressing logic, and the execution unit. This allows the machine to be clocked at a faster speed, since less work needs to be done each clock period.

RISC typically has large set of registers. The number of registers available in a processor can affect performance the same way a memory access does. A complex calculation may require the use of several data values. If the data values all reside in memory during the calculations, many memory accesses must be used to utilize them. If the data values are stored in the internal registers of the processor instead, their access during calculations will be much faster. It is good then to have lot of internal registers.

PREVIOUS WORK

The MIPS single-cycle processor performs the tasks of instruction fetch, instruction decode, execution, memory access and write-back all in one clock cycle. First the PC value is used as an address to index the instruction memory which supplies a 32-bit value of the next instruction to be executed. This instruction is then divided into the different fields shown in Table 2.1. The instructions opcode field bits [31-26] are sent to a control unit to determine the type of instruction to execute. The type of instruction then determines which control signals are to be asserted and what function the ALU is to perform, thus decoding the instruction. The instruction register address fields $rs bits [25 - 21], $rt bits [20 - 16], and $rd bits [15-11] are used to address the register file. The register file supports two independent register reads and one register write in one clock cycle. The register file reads in the requested addresses and outputs the data values contained in these registers. These data values can then be operated on by the ALU whose operation is determined by the control unit to either compute a memory address (e.g. load or store), compute an arithmetic result (e.g. add, and or slt), or perform a compare (e.g. branch). If the instruction decoded

is arithmetic, the ALU result must be written to a register. If the instruction decoded is a load or a store, the ALU result is then used to address the data memory. The final step writes the ALU result or memory value back to the register file.

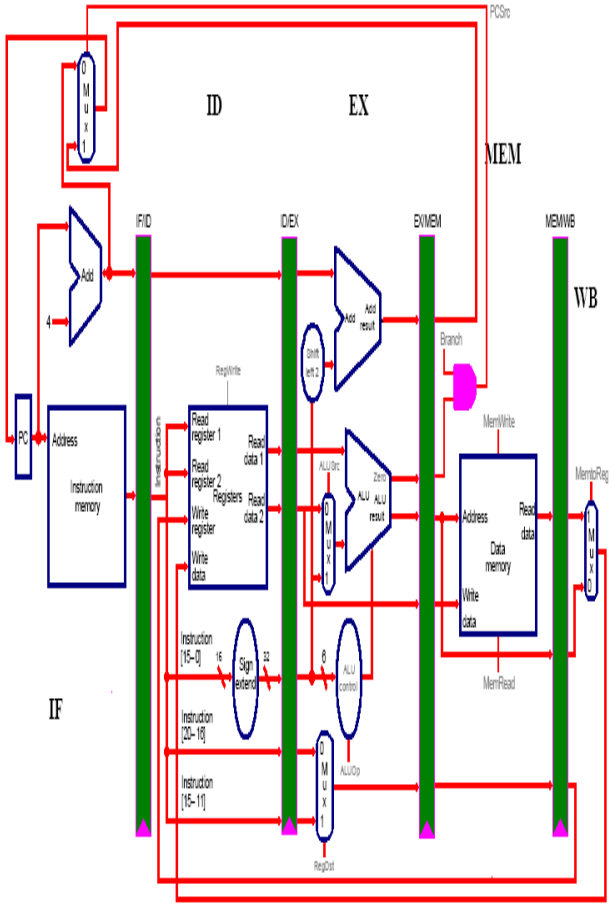

Figure 1. MIPS Single-cycle Processor

SYSTEM OVERVIEW

Mips Pipelined Processor Vhdl Implementation:

Once the MIPS single-cycle VHDL implementation was completed, our next task was to pipeline the MIPS processor. Pipelining, a standard feature in RISC processors, is a technique used to improve both clock speed and overall performance. Pipelining allows a processor to work on different steps of the instruction at the same time, thus more instruction can be executed in a shorter period of time. For example in the VHDL MIPS single-cycle implementation above, the datapath is divided into different modules, where each module must wait for the previous one to finish before it can execute, thereby completing one instruction in one long clock cycle. When the MIPS processor is pipelined, during a single clock cycle each one of those modules or stages is in use at exactly the same time executing on different instructions in parallel. Figure 2 shows an example of a MIPS single-cycle non-pipelined (a.) versus a MIPS pipelined implementation (b.). The pipelined implementation executes faster, keep in mind that both implementations use the same hardware components.

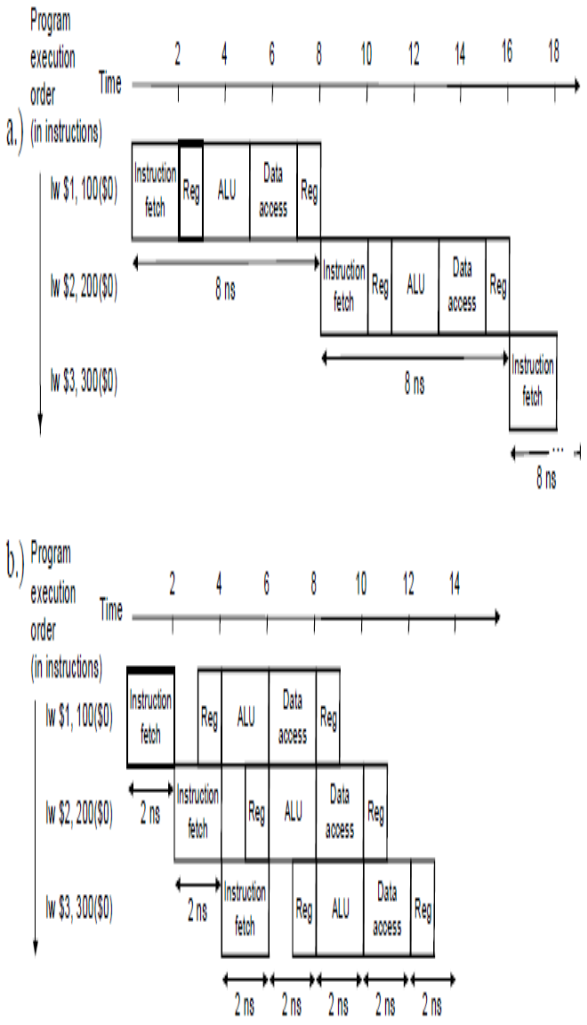

Figure 2. Single-cycle non-pipelined (a) vs. pipelined execution (b)

The MIPS pipelined processor involves five steps; the division of an instruction into five stages implies a five-stage pipeline:

a. Instruction Fetch (IF): fetching the instruction from the memory
b. Instruction Decode (ID): reading the registers and decoding the instruction
c. Execution (EX): executing an operation or calculating an address
d. Data Memory (MEM): accessing the data memory
e. Write Back (WB): writing the result into a register.

The key to pipelining the single-cycle implementation of the MIPS processor is the introduction of pipeline registers that are used to separate the datapath into the five sections IF, ID, EX, MEM and WB. Pipeline registers are used to store the values used by an instruction as it proceeds through the subsequent stages. The MIPS pipelined registers are labeled according to the stages they separate. (e.g. IF/ID, ID/EX, EX/MEM, MEM/WB) Figure 3 shows and example of a pipelined datapath excluding the control unit and control signal lines.

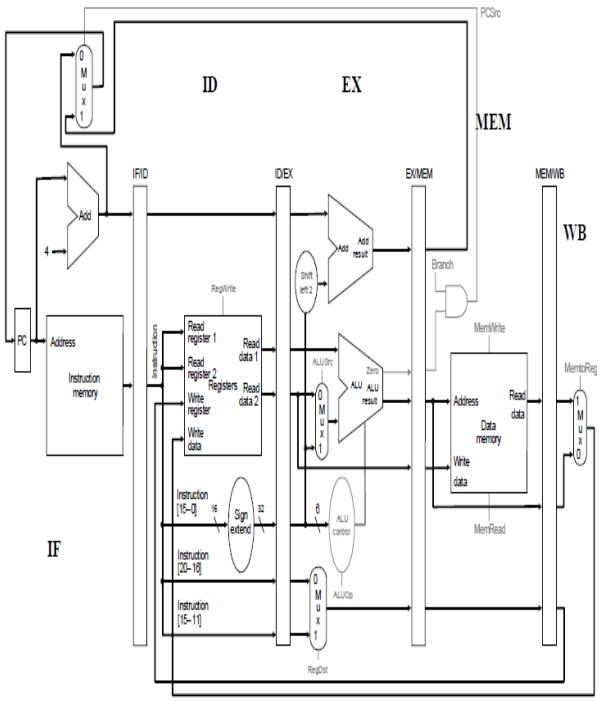

Figure 3. MIPS Pipelined Processor Datapath

To implement the MIPS pipelined processor, pipeline registers are placed into the corresponding VHDL modules that generate the input to the particular pipeline register. For example, the Instruction Fetch component will generate the 32-bit instruction and the PC+4 value and store them into the IF/ID pipeline register. When that instruction moves to the Instruction Decode stages it extracts those saved values from the IF/ID pipeline register. Appendix F contains the complete VHDL code used to implement the MIPS pipelined processor data path. Appendices G shows an example of MIPS processor pipelined being simulated.

DESIGN & IMPLEMENTATION

Introduction to Model Simulator:

Project Flow: A project is a collection mechanism for an HDL design under specification or test. Even though you don't have to use projects in ModelSim, they may ease interaction with the tool and are useful for organizing files and specifying simulation settings. The following diagram shows the basic steps for simulating a design within a ModelSim project.

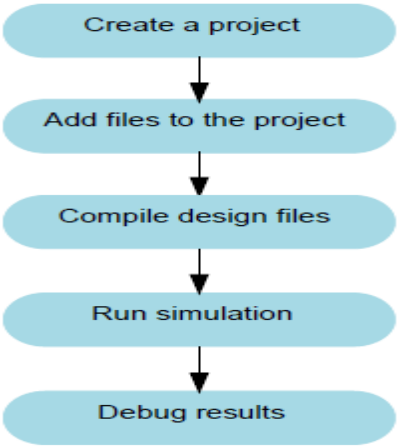

Figure 4. Project design flow

As you can see, the flow is similar to the basic simulation flow. However, there are two important differences:

You do not have to create a working library in the project flow; it is done for you automatically. Projects are persistent. In other words, they will open every time you invoke ModelSimunless you specifically close them.

Design Files for this Lesson: The sample design for this lesson is a simple 8-bit, binary up-counter with an associated testbench. The pathnames are as follows:

Verilog:

<install_dir>/examples/tutorials/verilog/basicSimulation/counter.v and t*counter.v*

VHDL:

<install_dir>/examples/tutorials/vhdl/basicSimulation/counter.vhd and *tcounter.vhd*

This lesson uses the Verilog files *counter.v* and *tcounter.v*. If you have a VHDL license, use *counter.vhd* and *tcounter.vhd* instead. Or, if you have a mixed license, feel free to use the Verilog testbench with the VHDL counter or vice versa.

Create the Working Design Library: Before you can simulate a design, you must first create a library and compile the source code into that library.

1. Create a new directory and copy the design files for this lesson into it.

Start by creating a new directory for this exercise (in case other users will be working with these lessons).

Verilog: Copy *counter.v* and *tcounter.v* files from */<install_dir>/examples/tutorials/verilog/basicSimulation* to the new directory.

VHDL: Copy *counter.vhd* and *tcounter.vhd* files from */<install_dir>/examples/tutorials/vhdl/basicSimulation* to the new directory.

2. Start ModelSim *if necessary*.

a. Type **vsim** at a UNIX shell prompt or use the ModelSim icon in Windows. Upon opening ModelSim for the first time, you will see the Welcome to ModelSim dialog. Click **Close**.

b. Select **File > Change Directory** and change to the directory you created in step 1.

3. Create the working library.

a. Select **File > New > Library**.

This opens a dialog where you specify physical and logical names for the library (Figure 5). You can create a new library or map to an existing library. We'll be doing the former.

Run the Simulation:

Now you will open the Wave window, add signals to it, then run the simulation.

1. Open the Wave debugging window.

a. Enter **view wave** at the command line

You can also use the **View > Wave** menu selection to open a Wave window.

The Wave window is one of several windows available for debugging. To see a list of the other debugging windows, select the **View** menu. You may need to move or resize the windows to your liking. Window panes within the Main window can be zoomed to occupy the entire Main window

or undocked to stand alone. For details, see Navigating the Interface.
2. Add signals to the Wave window.
a. In the Workspace pane, select the **sim** tab.
b. Right-click *test_counter* to open a popup context menu.
c. Select **Add > To Wave > All items in region** (Figure 5). All signals in the design are added to the Wave window.

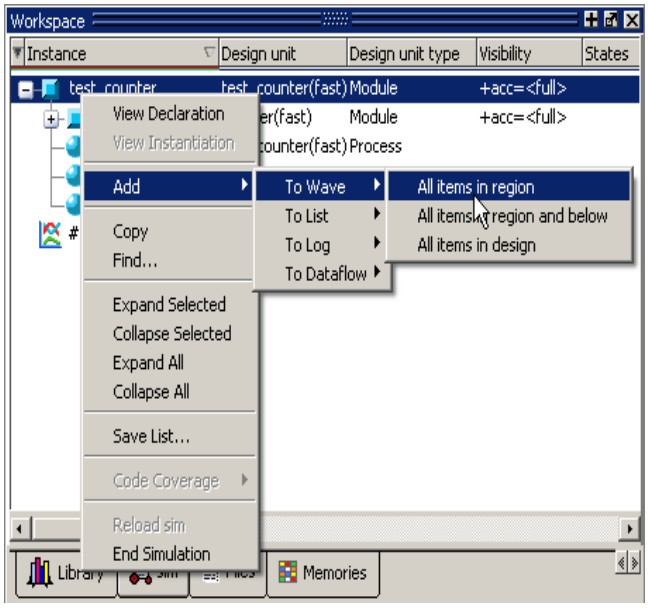

Figure 5. Using the Popup Menu to Add Signals to Wave Window

3. Run the simulation.
a. Click the Run icon in the Main or Wave window toolbar. The simulation runs for 100 ns (the default simulation length) and waves are drawn in the Wave window.
b. Enter **run 500** at the VSIM> prompt in the Main window. The simulation advances another 500 ns for a total of 600 ns (Figure 6).

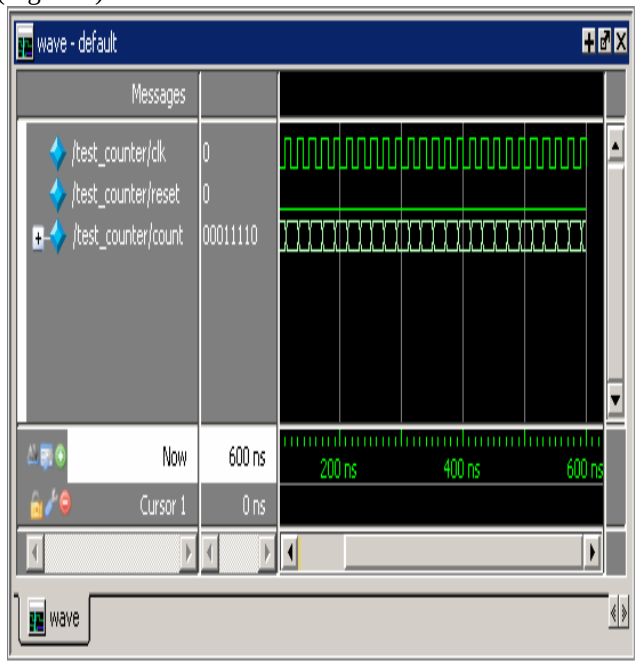

Figure 6. Waves Drawn in Wave Window

c. Click the **Run -All** icon on the Main or Wave window toolbar.The simulation continues running until you execute a break command or it hits a statement in your code (e.g., a Verilog $stop statement) that halts the simulation.
d. Click the Break icon. The simulation stops running.

RESULTS

Simulation Results:

The work presented in this Thesis describes a functional FPGA implementation design of a MIPS single-cycle and pipelined processor designed using VHDL. The VHDL designs of the MIPS processor were all simulated to ensure that the processors were functional and operated just as described by Patterson and Hennessy. The results show first the instruction memory initialization, which is used to fill the instruction memory with the instructions to be executed, which are indexed by the program counter (PC). The second is the actual 32-bit instruction represented using hexadecimal numbers. The third is the PC value used to index the instruction memory to retrieve an instruction. The next four columns are the MIPS instruction's mnemonic description. Finally last columns are the pseudo instructions using the actual values used during the simulation.

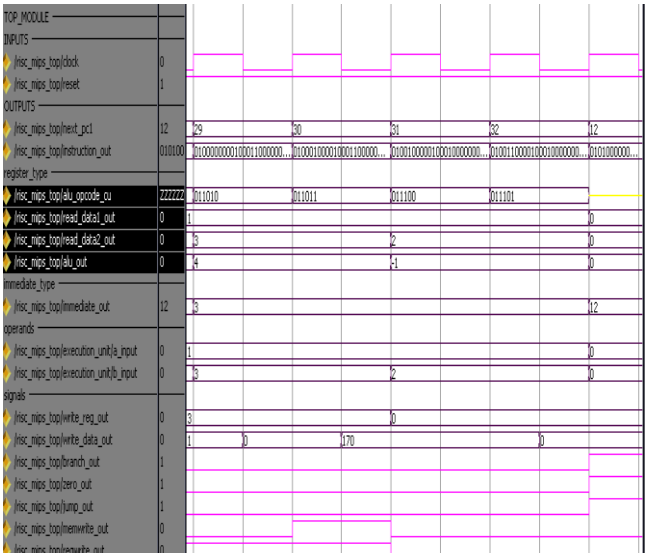

Figure 7. Result of simulation

Synthesis Result:

The developed convolution project is simulated and verified their functionality. Once the functional verification is done, the RTL model is taken to the synthesis process using the Xilinx ISE tool. In synthesis process, the RTL model will be converted to the gate level net-list mapped to a specific technology library. Here in this Spartan 3E family, many different devices were available in the Xilinx ISE tool. The target device is **SPARTAN 2 FPGA** kit. In order to synthesis this design the device named as "**XC3S100E**" has been chosen and the package as **"TQ144"** with the device speed such as "5".

RTL Schematic:

The RTL (Register Transfer Logic) can be viewed as black box after synthesize of design is made. It shows the inputs and outputs of the system. By double-clicking on the diagram we can see gates, flip-flops and MUX.

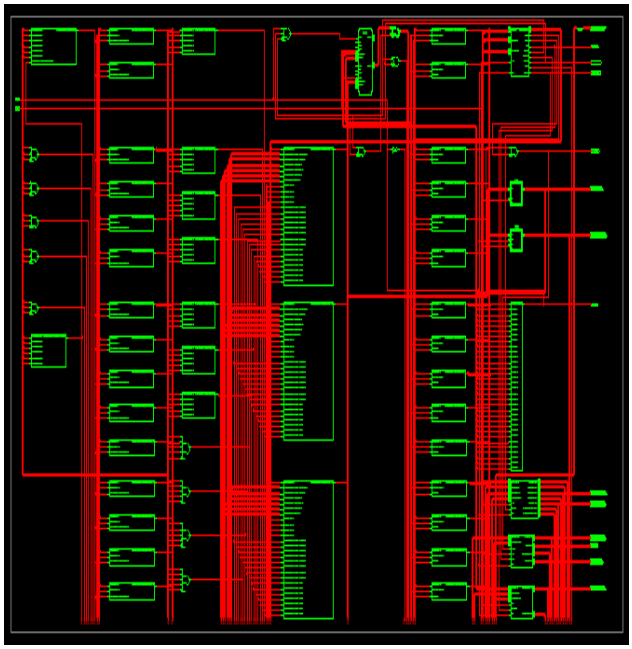

Figure 8. RTL Schematic internal view

The above figure 8. Shows the top level block diagram that contains the primary inputs and outputs of the design.

Device Utilization Summary:

This device utilization includes the following.

a. Logic Utilization
b. Logic Distribution
c. Total Gate count for the Design

nm Project Status			
Project File:	nm.ise	Current State:	Synthesized
Module Name:	Top2	• Errors:	No Errors
Target Device:	xc5vlx30t-3ff665	• Warnings:	76 Warnings
Product Version:	ISE 10.1 - Foundation Simulator	• Routing Results:	
Design Goal:	Balanced	• Timing Constraints:	
Design Strategy:	Xilinx Default (unlocked)	• Final Timing Score:	

nm Partition Summary
No partition information was found.

Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slice Registers	4283	19200	22%
Number of Slice LUTs	23545	19200	122%
Number of fully used LUT-FF pairs	275	27553	0%
Number of bonded IOBs	388	360	107%
Number of BUFG/BUFGCTRLs	17	32	53%
Number of DSP48Es	6	32	18%

Figure 9. Summary for the Device utilization time

The device utilization summery is shown above in which its gives the details of number of devices used from the available devices and also represented in %. Hence as the result of the synthesis process, the device utilization in the used device and package is shown above.

Timing Summary:

Speed Grade: -3 Minimum period: 3.203ns (Maximum Frequency: 312.173MHz) Minimum input arrival time before clock: 145.587ns Maximum output required time after clock: 6.156ns Maximum combinational path delay: 6.662ns

In timing summery, details regarding time period and frequency is shown are approximate while synthesize. After place and routing is over, we get the exact timing summery. Hence the maximum operating frequency of this synthesized design is given as 18.970 MHz and the minimum period as 52.719 ns. OFFSET IN is the minimum input arrival time before clock and OFFSET OUT is maximum output required time after clock.

CONCLUSIONS

In this project it is observed that the RISC MIPS based system is simulated using VHDL. The overall system is simulated and synthesized, after synthesizing the system we could get a statistical data about the number of input-output buffers, the number of registers, number of flip-flops and latches were used in the usage of FPGA tool. The modules simulated are Accumulator, Buffer, Clock Generator, Instruction Register, Multiplexer, Program Counter, Reset, Control Logic Decoder, Arithmetic Logic Unit and the overall system. Few instructions were executed and their timing sequences were analyzed. It is found that an each instruction taken 100ps.It shows that the different operations of the instruction including the decoding and execution comes to 40ns in the overall system. Therefore we conclude that the behavior shows, the system is working as RISC as instruction will be executed within a single clock cycle.

REFERENCES

[1]. Bai-ZhongYing, Computer Organization, Science Press, 2000.11.

[2]. Wang-AiYing, Organization and Structure of Computer, Tsinghua University Press, 2006.

[3]. Wang-YuanZhen, IBM-PC Macro Asm Program, Huazhong University of Science and Technology Press, 1996.9.

[4]. MIPS Technologies, Inc. MIPS32™ Architecture For ProgrammersVolume II: The MIPS32™ Instruction Set, June 9, 2003.

[5]. Zheng-WeiMin, Tang-ZhiZhong. Computer System Structure (The second edition), Tsinghua University Press, 2006.

[6]. Pan-Song, Huang-JiYe, SOPC Technology Utility Tutorial, Tsinghua University Press, 2006.

[7]. MIPS32 4KTMProcessor Core Family Software User's Manual, MIPS Technologies Inc.

[8]. Mo-JianKun, Gao-JianSheng,Computer Organization, Huazhong University of Science and Technology Press, 1996.

[9]. Zhang-XiuJuan, Chen-XinHua, EDA Design and emulation Practice [M]. BeiJing, Engine Industry Press. 2003.

[10]. "IEEE Standard of Binary Floating-Point Arithmetic" IEEE Standard754, IEEE Computer Society, 1985.

[11]. Yi-Kui, Ding-YueHua, Application of AMCCS5933 Controller in PCI BUS, DCABES2007, 20077.