Design of 16 Bit RISC Controller using VHDL

Patel Nilam S[1] Prof.J.H.Patil[2]
[1,2]Department of Electronics and Telecommunication Engineering
[1,2]D. N. Patel College of Engineering, Shahada, Nandurbar, Maharashtra, India

Abstract— This paper introduced design and implementation of some of the internal hardware component of controller. This controller is design using VHDL. VHDL is a very high speed integrated circuit hardware description language. In addition to these it is having three ports for communication with other I/O devices. The proposed controller is pipelined. In this paper, design methodology of such a multipurpose microcontroller has been provided along with the functional HDL code, simulation and synthesis results. Also the sequence has been highlighted in which this microcontroller can be made a general purpose controller unit that can be used by other system designs.

Key words: 16 bit Controller, RISC, VHDL, Xilinx

I. INTRODUCTION

When the controller design become more complex in CISC and the performance was also not up to expectations, people started looking on some other alternatives. It had been found that when a processor talks to the memory the speed gets killed. So the one improvement on CPI was to keep the instruction set very simple. Simple in not the way it works but the way it looks. That's why there are very few instructions in any typical RISC architecture where processor asks data from memory probably not other than Load and Store. At the end the pipelining added a new dimension in the speed just with the help of some Additional registers, which increases throughput by reducing CPI. Hence the instruction can be executed effectively in one clock cycle[1].

A common misunderstanding of the phrase "Reduced Instruction Set Computer" is the mistaken idea that instructions are simply eliminated, resulting in a smaller set of instructions. In fact, over the years, RISC instruction sets have grown in size and today many of them have a larger set of instructions than many CISC CPUs. The term "Reduced" in that phrase was intended to describe the fact that the amount of work any single instruction accomplishes is reduced at most a single data memory cycle compared to the "complex instructions" of CISC CPUs that may require number of data memory cycles in order to execute a single instruction[2]. Most controller in today's market are based on either RISC or CISC architectures. Research has shown that RISC architecture greatly boosts computer speed by using simplified machine instructions for frequently used functions. The following features typically found in RISC based systems.

1) Pre-fetching: The process of fetching next instruction or instructions into an event queue before the current instruction is complete is called pre-fetching.
2) Pipelining: Pipelining allows issuing an instruction prior to the completion of the currently executing one.
3) Superscalar operation: Superscalar operation refers to a processor that can issue more than one instruction simultaneously.

II. ARCHITECTURE

The objective of the project is to design a 16-bit RISC controller which utilizes minimum functional units. The architecture of proposed16-bit Controller is shown in Fig.1. The controller incorporates 16-bit ALU capable of performing 11arithmetical and logical operations, program counter, Instruction register, Sixteen 16-bit general purpose registers, flag register to indicate carry, zero and parity. The processor has four states idle, fetch, decode and execute. The control unit provides necessary signal interaction to perform expected function in all the states. The main objective of this project is to design a RISC microcontroller using VHDL and implement it in an FPGA. The microcontroller instruction set and features are based on Atmel AVR AT90S1200 RISC microcontroller.

Figure shows the top-level block diagram of the design, the bus structure has been simplified, but every block represents a module to be designed. At first glace, there are 11 modules in the top-level, with the 3 ports sharing the same module. These 11modules are to be design separately using the top down design approach. Some modules like the instruction register and status register are easy to design, but modules like ALU and the control unit require a lot of understanding.

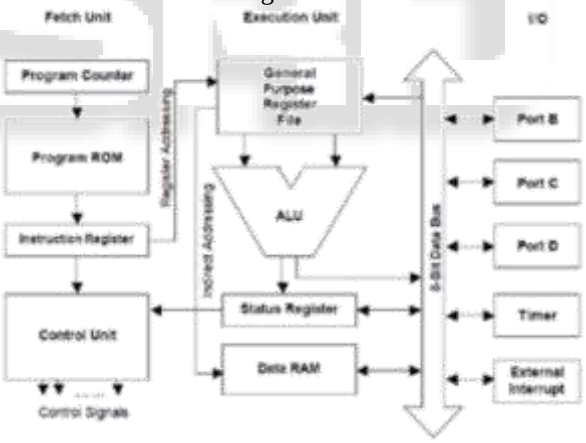

Fig. 1: Architecture Overview

The overall dataflow and bus structure between all the modules must be understand before designing the modules individually. There are basically two kinds of buses, direct bus and common bus. Direct bus connects two modules directly and is used specifically by the connected modules. There are many direct buses, such as the connection between program counter and program ROM, between program ROM and IR, between register file and ALU, etc. No control signals are required for direct buses. The data bus is the only common bus in this design. The data bus provides connection between the general purpose register file, ALU, status register, SRAM and all the I/O features. The register file can only receive data from the data bus. All others modules can receive and send data to the data bus. Since there are so many possible data flows, control signals are required to control the correct flow

direction. Only one source to the data bus is allowedat a time. If not, logic contentions will happen and the value of the data bus will be invalid. Tri-state bus is used to implement the common data bus. The impedance is so high that it can be seen as unconnected to the bus system. If the ALU is the data source, the data bus will be flooded with the result of the ALU and is available to all the connected modules. Control logic will generate an enable signal for the real destination to receive the data. The system can be divided into3 units, the fetch unit, execute unit and I/O unit. Fetch unit is in charge of fetching the next instruction and the execute unit is in charge of executing the current instruction. I/O unit provide a connection with the outside world. The fetch unit and execute unit form the CPU of the controller. The first module of the fetch unit is the program counter (PC). The PC contains the address of the next instruction to be executed. It points to the program ROM to locate the instruction. The instruction from the ROM is then latched into the instruction register (IR). The control unit takes the content of the IR and decodes it. It then assert the appropriate control signals to execute the instruction. All modules are connected with direct buses.

The execute unit in charge of executing most instructions. Normally, to execute an instruction, 2 operands are output from the register file to the ALU. The ALU then perform the operation and send the result to the data bus. Contents of the data bus (the result) is then stored back to the register file. The ALU also evaluate the status register flags and send them directly to the status register (SR). The whole execution process is done in a single cycle. The ALU perform many operations - include passing the contents of a general register to the data bus. SR also has a direct bus connection to the control unit required for branch evaluation. The register file is addressed directly by some bits in IR.

III. INSTRUCTION SET

The operation of the CPU is determined by the instruction it executes, referred to as machine instructions or computer instructions. The collection of different instructions that the CPU can execute is referred to as the CPU's instruction set. Since the instruction set defines the datapath and everything else in a processor, it is necessary to study it first.

Mnemonic	Operation	Flags	# Clocks
ARITHMETIC AND LOGIC INSTRUCTIONS			
ADD	Add Two Registers	S,Z,C,N,V,H	1
ADC	Add with Carry Two Registers	S,Z,C,N,V,H	1
SUB	Subtract Two Registers	S,Z,C,N,V,H	1
SUBI	Subtract Constant from Register	S,Z,C,N,V,H	1
SBC	Subtract with Carry Two Registers	S,Z,C,N,V,H	1
SBCI	Subtract with Carry Constant from Register	S,Z,C,N,V,H	1
AND	Logical AND Registers	S,Z,N,V	1
ANDI	Logical AND Register and Constant	S,Z,N,V	1
OR	Logical OR Registers	S,Z,N,V	1
ORI	Logical OR Register and Constant	S,Z,N,V	1
EOR	Exclusive OR Registers	S,Z,N,V	1
COM	One's Complement Register	S,C,Z,N,V	1
NEG	Negate (2's Complement) Register	S,C,Z,N,V,H	1
SBR	Set Bit(s) in Register	S,Z,N,V	1
CBR	Clear Bit(s) in Register	S,Z,N,V	1
INC	Increment	S,Z,N,V	1
DEC	Decrement	S,Z,N,V	1
TST	Test for Zero or Minus	S,Z,N,V	1
CLR	Clear Register	S,Z,N,V	1
SER	Set Register	None	1

Table 1: Basic Instruction set

Table 1 shows the instruction set summary of the designed controller, while the instruction set summary of the original AT90S1200 is shown. There are 92 instructions grouped into 4 categories: arithmetic and logic instructions, branch instructions, data transfer instructions and the bit and bit-test instructions. As mentioned earlier, instruction set of the design is based on Atmel AVR AT90S1200instruction set. In this way, the design can use the same assembler and simulator provided by Atmel since the final design is actually an AT90S1200 compatible controller.

IV. ADDRESSING MODES

There are 7 addressing modes in the microcontroller. Rd and Rr are devoted to the destination register and soure register.

1) Direct Single Register Addressing The operand is in Rd.
2) Direct Double Register Addressing The operands are in Rd and Rr. Result is stored back to Rd.
3) I/O Direct Addressing First operand is one of the I/O registers. The address is contained in 6 bits of the instruction word. The second operand is either Rd or Rr. Used by IN and OUT instructions to read from or write to the I/O registers.
4) Data Indirect Addressing Operand address is the contents of the Z-register. Used when accessing the SRAM with LD and ST instructions.
5) Data Indirect Addressing with Pre-Decrement Z-pointer is decremented by 1 before the operation. Operand address is the decremented contents of the Z-register. Used when accessing the SRAM with LD and ST instructions.
6) Data Indirect Addressing with Post-Increment The Z-register is incremented by 1 after the operation. Operand address is the original content of the Z-register before increment. Used when accessing the SRAM with LD and ST instructions.
7) Relative Program Memory Addressing Program execution continue at address PC + offset. The offset is contains in the instruction word. Unconditional branch instructions (RJMP, RCALL) can reach the entire program memory from every location.

V. VHDL

VHDL is acronym for very high speed integrated circuit Hardware Description Language. It is designed to fill a number of needs in the design process. First it allows description of the structure of system that is, how it is decomposed into subsystems and how they are interconnected. Second, it allows the specification of the function of a system using familiar programming language forms. Third, as a result, it allows the design of a system to be simulated before being manufactured, so that designers can quickly compare alternatives and test for correctness without the delay and expense of hardware prototyping. Fourth, it allows the detailed structure of a design to be synthesized from a more abstract specification, allowing designers to concentrate on more strategic design decisions and reducing time to market VHDL was established as the IEEE 1076 standard in 1987. In 1993, the IEEE 1076 standard was updated and an additional standard, IEEE 1164 was adopted. In 1996, IEEE 1076.3 became the VHDL synthesis standard.

VI. RESULTS

This project introduces a scheme of implementation of RISC controller using VHDL coding. VHDL code is developed module wise successfully run on Xilinx software. The fig. 2 and figure 3 shows the synthesis and simulation results of controller.

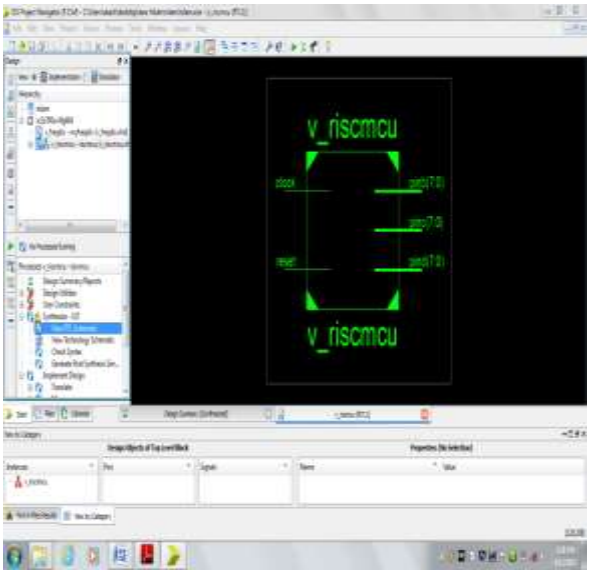

Fig. 2: Top block of RISC controller

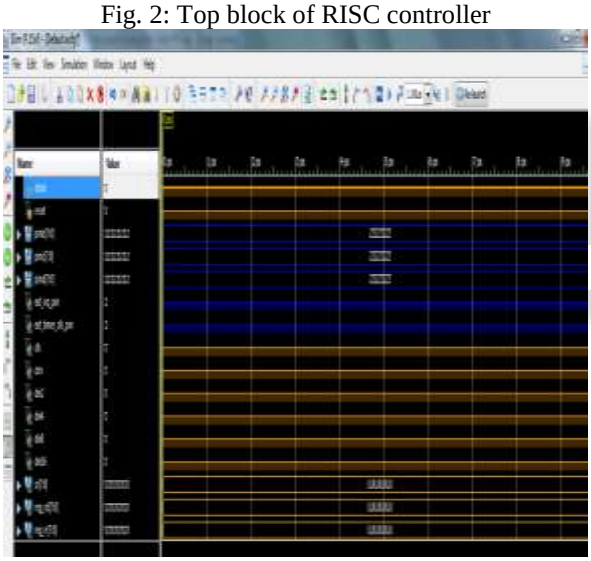

Fig. 3: Simulation Result

Running one application of RISC controller using VHDL code verifies the operation of the RISC Controller. Figure 4 shows the LED blinking application

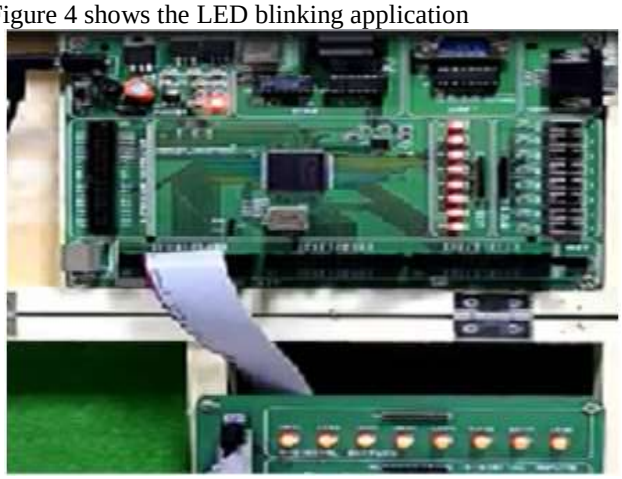

Fig. 4: LED blinking

VII. CONCLUSION

As a conclusion, this project has been completed successfully fulfilling the objectives specified. The data RAM that is also included in this project. Hardware stack is enlarged to 4-level instead of 3 and a total of 24 I/O lines are available. This Project describes the design of 16-bit RISC controller. The controller has been designed using Very high speed integrated circuit Hardware Description Language (VHDL). This controller is efficient for specific applications and suitable for small applications. It has an in built program and data memory. Also it has ports for communicating with other I/O devices. The design has been implemented using FPGA.

REFERENCES

[1] Mamun B, Shabiul I. and Sulaiman S,"A SingleClock Cycle MIPS RISC Processor Designusing VHDL"
[2] Hamblen J." Synthesis, Simulation, andHardware Emulation to Prototype a PipelinedRISC Computer System"
[3] Samiappa Sakthikumaran, S.Salivahanan and V.S.Kaanchana Bhaaskaran, June 2011, "16-Bit RISC Processor Design For Convolution Application", IEEE International Conference on Recent Trends In Information Technology, pp.394-397.
[4] "Implementation of RISC Processor on FPGA" Pravin S. Mane, Indra Gupta, M. K. Vasantha Computer Science & Engineering Department, Mody Institute Of Technology & Science, Lakshmangar, 2006. "Electrical Engineering Department, Indian Institute of Technology Roorkee, Roorkee-247667
[5] Tanaji M. Dudhane and Charudatta V Kulkarni "VLSI design of 8 bit microcontroller using reconfigurable hardware "MITCOE, Pune ,India . International Journal For Technical Research And Development ,Jan 2012 .
[6] Patterson A. and Hennessy J,"ComputerOrganization & Design", Morgan KaufmannPublishers, 1999
[7] Peter J Ashenden, "Digital Design, An embedded systems approach using Verilog", MorganKaufmann Publishers,2010
[8] Ramesh Gaonkar, "Microprocessor architecture, programming and applications with the 8085", Penram International Publishing,1989

