2021 IEEE 1st International Maghreb Meeting of the Conference on Sciences and Techniques of Automatic Control and Computer Engineering MI-STA, 25-27 May 2021, Tripoli-Libya

Reduced Instruction Set Computer Design on FPGA

Mohamed M. Eljhani
Department of Computer Engineering
University of Tripoli
Tripoli, Libya
M.Eljhani@uot.edu.ly

Veton Z. Kepuska
Department of Computer Engineering and Sciences
Florida Institute of Technology
vkepuska@fit.edu

***Abstract*—The main purpose of this paper is to design, verify and implement 16_bit RISC (Reduced Instruction Set Computer) processor that can be used for many embedded applications. The basic modules of this processor are programmed and simulated using Verilog HDL (Hardware Description Language), and implemented on Cyclone IV FPGA (Field Programmable Gate Arrays). Compared with general CPU it is not merely simplified the instruction set system but also make the computer structure simpler and more rational through simplifying the instruction system. Thus, the operating speed is highly improved. RISC adopts hardwire logic instead of micro-program control to realize its sequential control signals. The speed of control sequence generated is much faster than using micro-program control because it has saved the time of fetching microinstruction. The philosophy of RISC design presented here favors a smaller and simpler set of instructions. Those instructions take the same amount of time to execute. The philosophy of our design architecture was to keep the instruction set very simple. This in turn implies that addressing modes supported by instruction set a further streamlined compared to CISC (Complex Instruction Set Computer) architectures. Avoiding such addressing modes must be kept to minimum, which leads to the instructions that can be executed effectively in eight clock cycles.**

***Index Terms*—System-On-Programmable-Chip, RISC CPU, Processor Design, FPGA Design, Verilog Hardware Description Language.**

I. Introduction

RISC architectures are now used in many platforms, from cellular telephones to some of the world's fastest supercomputers. RISC based architectures have been used in both low level applications and mobile systems by the beginning of the 21st century [1]. The low power and low cost embedded market is dominated by the RISC based ARM [2] architectures. Most of the android based devices, Apple iPhone an iPad and most hand-held devices uses the ARM architecture. The MIPS [3] line can currently be found in games like PlayStation Portable game consoles, Nintendo 64 and personal residential gateways like Linksys WRT54G series. SuperH (SH) is another 32-bit RISC ISA developed by Hitachi. As many of the patents for SuperH are expiring, SuperH2 is being reimplemented as an open source hardware under the name J2 [4]. OpenRISC [5] is aimed at developing an

open source ISA based on RISC principles. OpenRISC implements architecture with 16 or 32 general purpose registers (32/64-bit) and a fixed 32-bit instruction length. Two mainline processor core implementations for OpenRSIC are OR1200 and mor1kx. The main idea of design a RISC processor was inspired by the concept that there are many features that were included in CISC designs were being ignored by the programs that were running on them, which these complex features took several processor cycles to be performed. Also the performance gap between the processor and main memory is increasing. This led to design a CPU with less total number of memory accesses. When a processor talks to the memory the speed gets slow. So the main advantage of RISC was to keep the instruction set very simple. Not only the way it works but also the way it looks. That is the reason there are only eight instructions in the RISC architecture. To improve the instruction access speed, addressing modes are avoided. The instruction can be executed effectively in eight clock cycles. Another difference between RISC CPU's and general CPU's lies in hardwired logic vs. micro-program implementation in realization of sequential control signals. Hardwired logic means that flip-flops and logic gates are linked directly to and from state machine and combinatorial logic that controls the whole system. On the other hand, microprogramming logic in general is slower in execution of command logic due to the need for additional cycles that are required in implementation of command logic. The new designed RISC philosophy uses smaller and simpler set of instructions. Those instructions take the same amount of time to execute. The goal of our effort is to design high performance RISC processor and implement it on FPGA board, that is designed specifically to host and test the design.

II. Methodology

The state machine module is the control core of the RISC, it acts as a brain of the RISC and its main task is to generate a series of control signals to control and operate other modules. Also, state machine is responsible of fetching instructions and read/write from I/O and memory.

A. Top Module View

As shown in Figure 1, the proposed RISC architecture is consisting of twelve sub-modules.

B. System Sub-modules

- Clock Generator
- Instruction Register
- Accumulator
- Arithmetic Logic Unit

978-1-6654-1855-3/21/$31.00 ©2021 IEEE

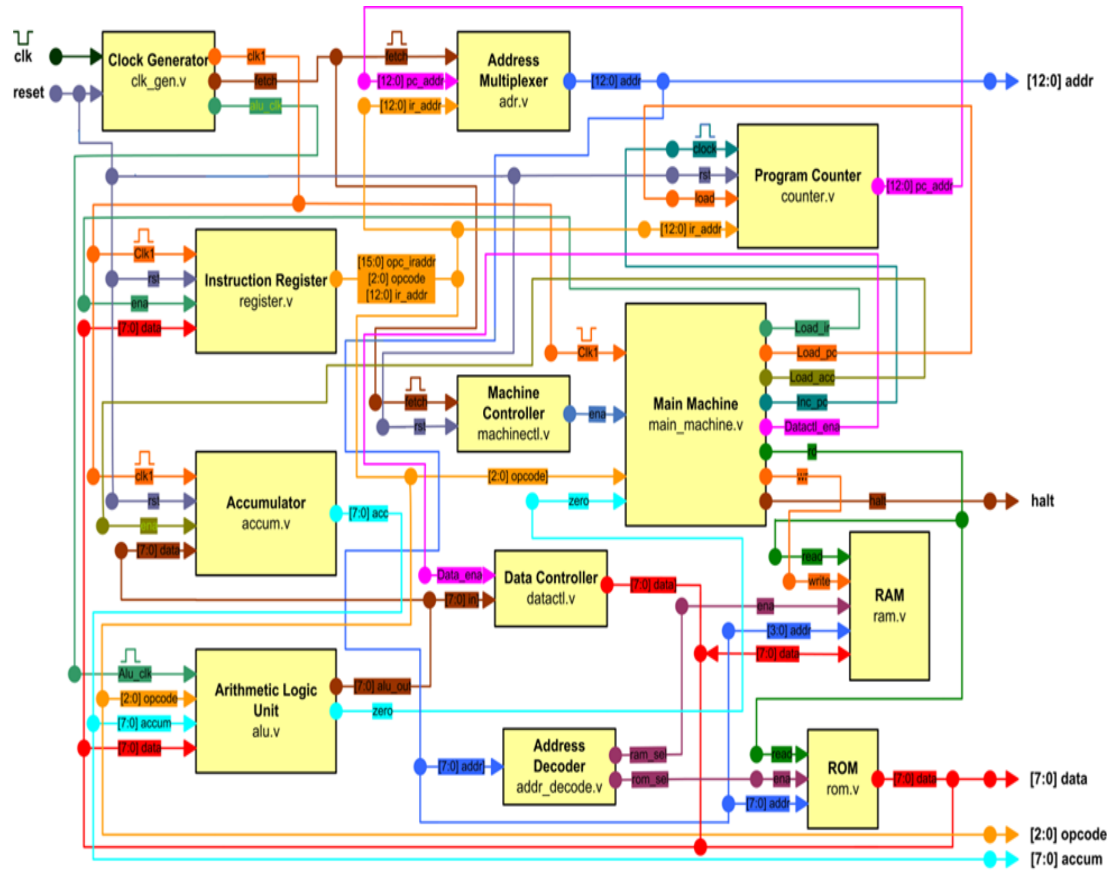

Fig. 1. System Architecture.

- Data Controller
- State Machine Controller
- State Machine
- Program Counter
- Address Multiplexer
- Address Decoder
- Memory RAM (16- bytes)
- Memory ROM (256- bytes)

C. Architecture and Implementation

The instruction cycle is made from up to eight clock cycles, and certain operation should be carried out in each clock cycle:
Zero clock cycle: Set read and load instruction register signals to 1, the other outputs of the state controller are zeros, so the higher 8-bits of the instruction located in ROM is stored in the Instruction Register.
First clock cycle: Compared with the former clock cycle, only instruction register has been changed from 0 to 1, so program counter is increased by 1, the lower 8-bits of the instruction from ROM memory is stored in the Instruction Register.
Second clock cycle: No operation.
Third clock cycle: The Program Counter is increased by 1, pointing to the next instruction in ROM.
Fourth clock cycle: Check if the instruction is ADD, AND, XOR or LDA, read data from the corresponding address; if the instruction is JMP, send the destination address to the Program Counter; if the instruction is STO, put the data in the Accumulator.
Fifth clock cycle: Check if the instruction is AND, ADD or XOR, the Arithmetic Logic Unit will carry out corresponding operation; if the instruction is LDA, send the data to the Accumulator through Arithmetic Logic Unit; if the instruction is SKZ, first judge whether the value of Accumulator is 0, if so, increase PC by 1, otherwise do not change; if the instruction is JMP, latch the destination address; if the instruction is STO, write the data in the RAM with specified address.
Sixth clock cycle: No operation.
Seventh clock cycle: If the instruction is SKZ and the value of the Accumulator is 0, the value of program counter will be increased by 1, and an instruction will be jumped over. Otherwise, the value remains in the same value. External clock is used as the timing mechanism for the Control and Datapath units. The input clock is divided into a series of different clock signals using frequency division operation of the clock generator module. These clock signals are used as input clock of the other parts of the RISC, and the operating sequence is controlled by the main state machine controller. The frequency of fetch clock is equal 18 of the system clock. When rising edge of fetch is active the main controller machine begins to carry out an instruction, in the meantime the fetch

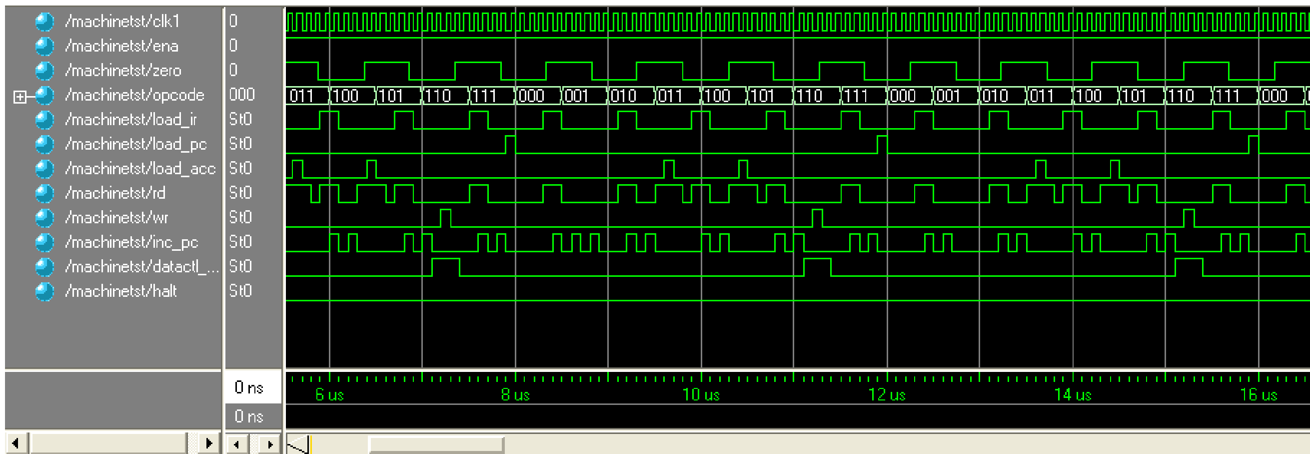

Fig. 2. State Machine Control Signals.

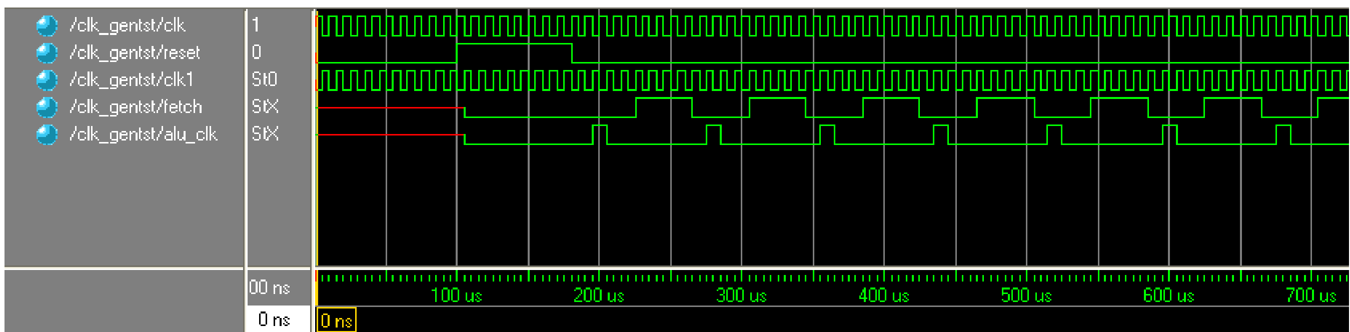

Fig. 3. System Clock.

signal controls the Address Multiplexer to point out instruction and data address. The Instruction Register, Accumulator and State Controller use clk1 as their clock signal, and Arithmetic Logic Unit uses alu_clk.

D. Instructions Manipulation

Our main design philosophy is favoring a smaller and simpler set of instructions. Those instructions take the same amount of time to execute. The idea of the RISC design is inspired by the discovery that many of the features that were included in traditional CPU designs are ignored by the programs. These features use many clock cycles to be performed. To avoid the addressing modes, we kept the minimum instructions, which leads to the instructions that can be executed effectively in less clock cycles. According to the different kinds of operation codes received, the Arithmetic Logic Unit (ALU) carries out the corresponding operation, as shown in Table I. The RISC processor performs various instructions here, we'll briefly detail how it performs its eight instructions. In these instruction sets; the system uses 3-bit opcode to provide instructions on how to perform various operations. Table below shows the instruction sets for RISC processor.

Each instruction consists of 16-bits. The higher 3-bits is operation code (opcode), and the lower 13-bits is address.

TABLE I
INSTRUCTION SET FOR RISC PROCESSOR.

ALU Operand Select Code	Operation	Description
000	HLT	Halt, Do nothing-operation
001	SKZ	Skip if zero, If the result is zero then jump over the next statement
010	ADD	Add the value of the accumulator to the value of data in memory , of giving address
011	AND	ANDing The value of the accumulator and the data in the memory of giving address
100	XOR	XORing The value of the accumulator and the data in the memory of giving address
101	LDA	Load the accumulator, put the data of giving address in the accumulator
110	STO	Store data, store the data of giving address in the memory RAM
111	JMP	Unconditional jump , jump to the destination address, and continue executing the program

The address bus is 13-bits, with addressing space up to 8-K bytes. The data bus is 8-bits, the system fetches every instruction twice, first the higher 8-bits and then the lower 8-bits, controlled by the state, when state is 0, the higher 8-bits of the instruction sent to the responsive register, and the state is set to 1. For the next operation, the lower 8-bits of the instruction is sent to the responsive register because the state is 1.

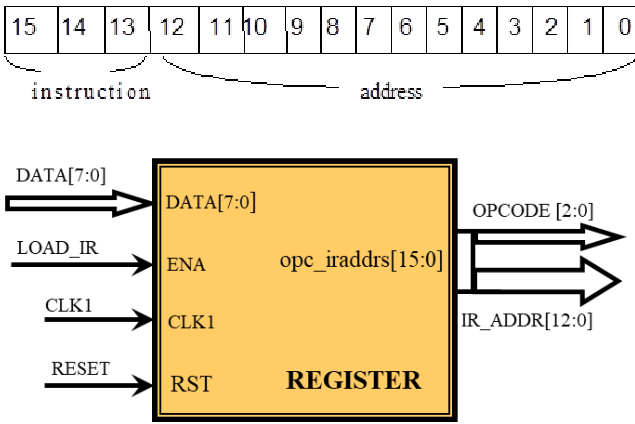

Fig. 4. Instruction register module.

E. System Reset and Read/Write Operations

Reset is triggered by signal rst. When rst is set to 1, the system will end the current operation and hold on in reset state as long as rst high, all registers are initialized to 0. The data bus hold in high-impedance state and the address bus is set to 0. All control signals are unactive. Once rst is set to 0 again, the system will start up at the rising edge of fetch, fetching the instruction from 00H address of ROM and then execute the corresponding operation.

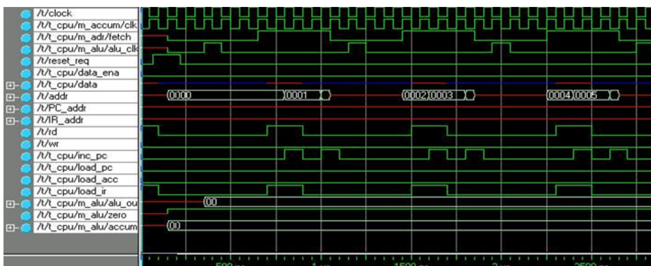

Fig. 5. System Reset.

The first three clock cycles are used to fetch new instruction. In the fourth to sixth clock cycle, read signal rd becomes active, and the data are sent to the data bus. In the 7th clock cycle, rd signal becomes inactive, and Program Counter set new address, preparing for the next instruction.

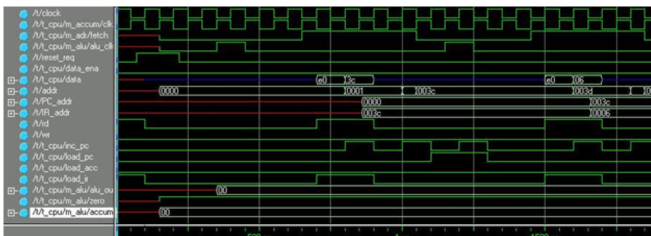

Fig. 6. Reading data from memory.

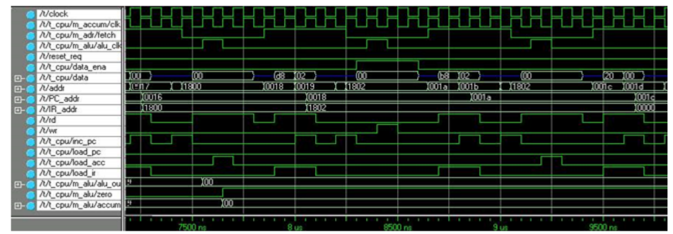

Fig. 7. Writing data to memory.

III. Simulation and Results

The Electronic Design Automation (EDA) design flow typically follows a path from Verilog/VHDL hardware description language, [6], or schematic design entry through synthesis and place and route tools to the programming of the FPGA, as shown in Figure 8. There are several software tools such as Altera's Quartus II and Xilinx's ISE, which are quite famous, used in both industry and academic areas.

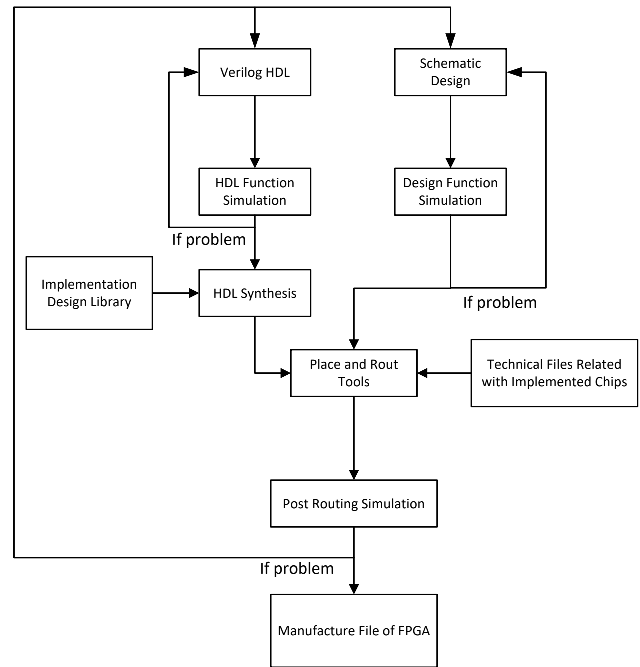

Fig. 8. EDA flowchart.

A. Top Module Instantiation and Simulation

After verifying RISC sub-modules individually, the sub-modules are instantiated, simulated and verified the RISC as top-level module. The system was implemented and tested using Cyclone EP1C6Q240C8 FPGA evaluation platform, that designed specially to test the functionality of the RISC processor in hardware.

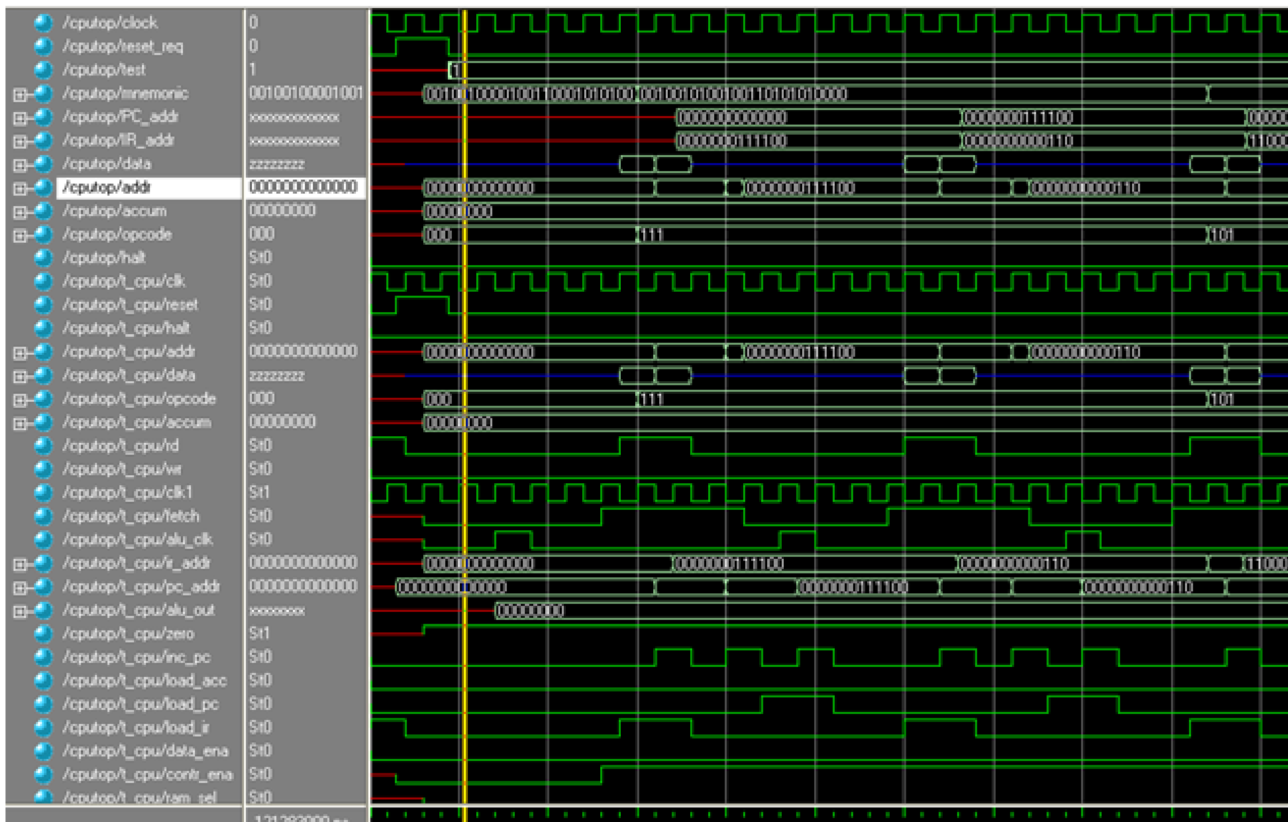

Fig. 9. Top module waveforms.

B. Machine Code Assembley Language

Test.pro File :

```
@00
101_11111      // 00    BEGIN:  LDA DATA_2
0000_0001
011_11111      // 02    AND DATA_3
0000_0010
100_11111      // 04    XOR DATA_2
0000_0001
001_00000      // 06    SKZ
0000_0000
000_00000      // 08    HLT
0000_0000
010_11111      // 0A    ADD DATA_1
0000_0000
001_00000      // 0C    SKZ
0000_0000
111_00000      // 0E    JMP
0001_0010
000_00000      // 10    HLT
0000_0000
100_11111      // 12    XOR DATA_3
0000_0010
010_11111      // 14    ADD DATA_1
0000_0000
110_11111      // 16    STO TEMP
0000_0011
```

Test.dat File :

```
@00
00000001   // 1F00     DATA_1:    // 01 Hex
10101010   // 1F01     DATA_2:    // AA Hex
11111111   // 1F02     DATA_3:    // FF Hex
00000000   // 1F03     TEMP:
```

C. RISC Debugging

```
Run
# ROM loaded successfully!
# RAM loaded successfully!
#
# *** RUNNING RISC - RISC Diagnostic Program ***
#
#   TIME          PC      INSTR   ADDR    DATA
# ----------   ----------  -----------  -----------  ----------
#  16200.0 ns    0000    LDA     1F01    AA
#  17000.0 ns    0002    AND     1F02    FF
#  17800.0 ns    0004    XOR     1F01    AA
#  18600.0 ns    0006    SKZ     0000    ZZ
#  19400.0 ns    000A    ADD     1F00    01
#  20200.0 ns    000C    SKZ     0000    ZZ
#  21000.0 ns    000E    JMP     0012    ZZ
#  21800.0 ns    0012    XOR     1F02    FF
#  22600.0 ns    0014    ADD     1F00    01
#  23400.0 ns    0016    STO     1F03    FF
#
# ***********************************************
```

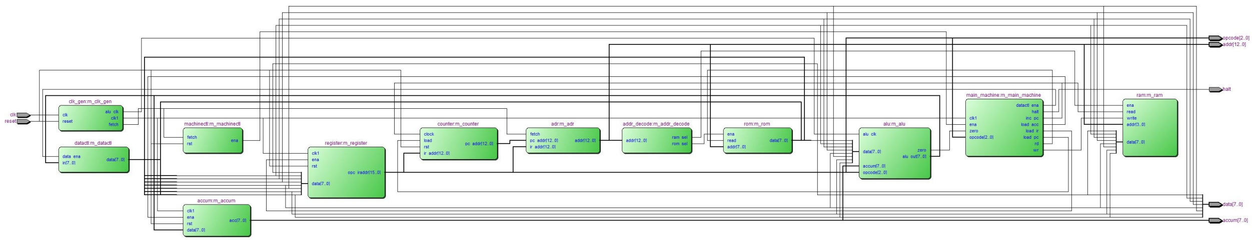

Fig. 10. Synthesis Register-Transfer-level.

Fig. 11. PCB with Cyclone FPGA.

D. Hardware Test Environment

For further advanced FPGA development tools, a complete EDA Printed Circuit Board system is designed to test and verify the RISC in hardware, see Figure 11.

IV. Conclusion

The 16-bit RISC Processor with 8 instructions set has been designed and implemented on Cyclone FPGA. The design is simulated and verified using ModelSim 10.1 and Quartus II 12.1 Altera 10.1b simulator, and programmed by Verilog Hardware Description Language. ALU is analyzed and an exhaustive set of debugging and testing assembly language code patterns is developed with several instructions to verify the operation of the RISC. Future work will be added by increasing the number of instructions with less clock cycles per instruction and more improvement can be added in the future work.

References

[1] S. P. Dandamudi, Guide to RISC Processors: For Programmers and Engineers. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005.

[2] Wikipedia, "Arm architecture — wikipedia, the free encyclopedia," 2017, [Online; accessed 26 February-2017]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=ARM Architecture & oldid=766746198

[3] Primefac, "Reduced instruction set computing wikipedia, the free encyclopedia," 2017, [Online; accessed 26-February- 2017]. [Online]. Available: https://en.wikipedia.org/w/index.php?title= Reduced instruction set computing&oldid=765887154

[4] J. core Organisation, "J2 open processor," 2014, [Online; accessed 28-February-2017]. [Online]. Available: http://j-core.org/

[5] J. Tandon, "The openrisc processor: Open hardware and linux," Linux J., vol. 2011, no. 212, Dec. 2011. [Online]. Available: http://dl.acm.org/citation.cfm?id=2123870.2123876

[6] The IEEE Standard Hardware Description Language based on the Verilog Hardware Description Language (IEEE Std 1364-2001).