REALIZATION OF HIGH PERFORMANCE RUN-TIME LOADABLE MIPS SOFT-CORE PROCESSOR

Prathibha S R[1], M Z Kurian[2]

[1]*M.Tech Student, Department of E&C, Sri Siddhartha Institute of Technology, Tumkur, Karnataka, India*
[2]*Head, Department of E&C, Sri Siddhartha Institute of Technology, Tumkur, Karnataka, India*

Abstract

Soft-core processors on Field Programmable Gate Array (FPGA) chips are becoming a solution for application-specific customization. Soft-core processors provide several advantages for designer. Currently using microprocessor doesn't support real-time loading. If, any change in the assembly code of the implemented processor, it requires re-implementation and downloads the soft-core on FPGA. This project proposes realization of run-time loading technique of a MIPS (Microprocessor without Interlocked Pipeline Stages) soft- core processor on FPGA. The proposed system consists of mainly three components: microprocessor soft-core, software tool and universal asynchronous Receiver/Transmitter (UART). Since, here MIPS code is updating instantly there is no need of resynthesize, place, route, and reload the soft-core. The software tool communicates between the user and MIPS soft-core processor through UART. Five stage pipelining is included to improve the overall processor performance. In this paper 32-bit MIPS soft-core processor, RAM module, APB Interface is designed and simulated using Modelsim. Design architecture will be doing in Verilog, simulate using Modelsim 10.3 simulator and realize in Spartan-6 FPGA using Xilinx ISE 14.2.

Keywords: *FPGA, MIPS, PERL, RISC, UART etc…*

----------***----------

1. INTRODUCTION

Soft-core processor is a microprocessor their architecture and behavior are defined in Hardware Description Language (HDL), which can be synthesized in programmable hardware, like FPGAs or ASICs. MIPS is a RISC processor, which is a 32-bit processor designed with single cycle implementation.[2] For the realization of MIPS soft-core processor using Field Programmable Gate Array (FPGA) many techniques have been reported and documented in open literature. However, the run time loading for CPU on FPGA and its hardware realization using FPGA has not been reported as of our knowledge.

This project is to propose run-time loading technique of machine code for MIPS-32 soft-core processor on FPGA. Some logic has to be introduced in the program memory made it rewritable in runtime. ROM (Read Only Memory) logic has to be changed to RAM. So bipartite the RAM (Random Access Memory) into two sections, one section is for instruction memory and the other one is used as data memory.[9] The software tool is build by using Perl programming language. It is used to write the MIPS assembly code, debug the code, generates the machine code and sends it to the FPGA.

The objective of this project is to design and implementation of run-time loading of machine code for MIPS-32 soft-core processor on FPGA. Write the PERL (Practical Extract and Report Language) script to generate the opcode of the MIPS processor and that should be stored in text file and after storing the opcode information that sent to the instruction decoder through UART/RS232 bus protocols and that will fetch the correct opcode from that instruction decoder and send to the MIPS logic through Advance Peripheral Bus protocol (APB) interface and it can change in the run time through programming the FPGA. Implementing the required MIPS using FPGA that controls the processor at run time and it debug easily which gives the complete functional design of the MIPS processor. Five stage pipelining is included to improve the overall processor performance. [1]

The organization of this paper is as follows. Section II presents proposed work, Section III discussed on system design of proposed work, Section IV shows the results. Section V concludes this work and future enhancement.

2. PROPOSED WORK

The fig.1 shows the block diagram of the proposed system. Writing the perl script to generates the opcode and operand code fields in a text file format.

Text file: It stores the opcode and operand information in machine level language.

UART: From computer side it is the media to send and receive the data serially to/from external world. The UART is used to connect the FPGA and computer for data and

instruction transfer. This UART information will be stored on the FPGA RAM for further processing.

MIPS Logic: The architecture is standard processor architecture with fetch, decode and execute blocks with run time instruction execution. The internal blocks of MIPS logic are data and instruction buffers, ALU controller and ALU. The data and instructions will be feed from RAM.

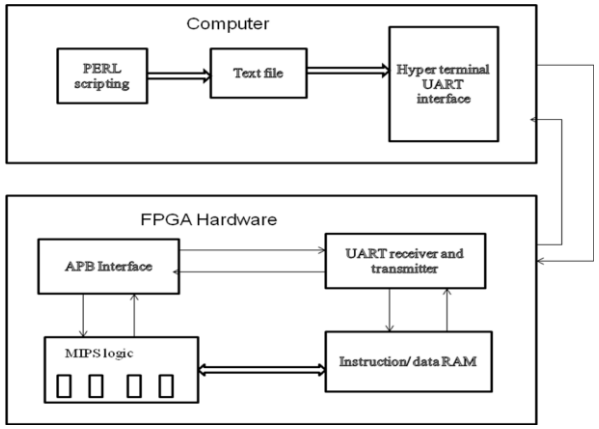

Fig -1: Block Diagram of a system

APB: This is one of the standard bus protocol defined by ARM Company which is used to interconnect the modules.

3. SYSTEM DESIGN

3.1 Design of MIPS Processor

MIPS processor is 32-bit processor designed with single cycle implementation operate with frequency of 100MHz.

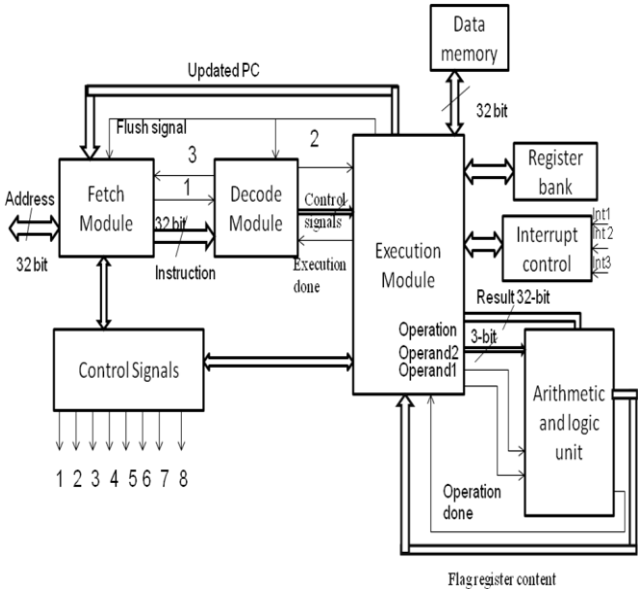

Fig -2: Architecture of proposed MIPS Processor

As shown in fig.2 the numerical 1, 2, 3 indicate as below;

1. Waiting_For_Decoder: Wait for the fetched instruction if it is not fetched any instruction in decoder module.
2. Waiting_For_Execution: After completion of decoding it will wait for next execution done.
3. Decoder_Fetch: Fetch the valid instruction from the fetch module through advance peripheral bus.

The Fig.2 shows the architecture of the proposed processor. It mainly consists of Fetch module, Decode module, execution module, register bank and updated PC.

Fetch module: In fetch unit instruction is fetched from memory. The address of the instruction will be fetched from program counter. The fetched instruction is again moved to the instruction register through data bus.

Decode module: Here transfer the instruction from instruction register to decode unit. In this unit the fetched instruction is decoded by the processor. It will get the operands and opcode given in the instruction.

Execute module: In this unit executes the instruction. ALU performs arithmetic and logical operation depending on requirement of the instruction and result is stored back in register/memory.

Register Bank: It contains array of registers. It has 32 registers each of 32bit wide.

PC Update: Update the value of program counter normally incremented by 1. If any jump or branch instructions PC updated to other address.

Interrupt Control: It allows interrupts on one or more CPU lines, based on priority assigned. When interrupt occurs, disturbs the execution of current code in processor and needs immediate attention.

Flush Signal: As soon as flush signal received, it cancels instruction queues cannot performing any other operations. Program counter jumps to specific address, start fetching in normal way.

Control Signals are

1. Fetch instruction: This signal from fetch unit to RAM
2. Valid fetch: This signal from RAM to fetch unit valid instruction is fetched.
3. Data memory request: This signal from execution unit to the Data memory requesting for access memory.
4. Valid read data: This signal from memory to the execution unit after read the data.
5. Interrupt: This signal from external device to the interrupt control.

6. Data memory operation: This signal form data memory to the execution unit.
7. Reset: Global signal.
8. Clock: Global signal.

3.1.1 Design of Fetch Unit

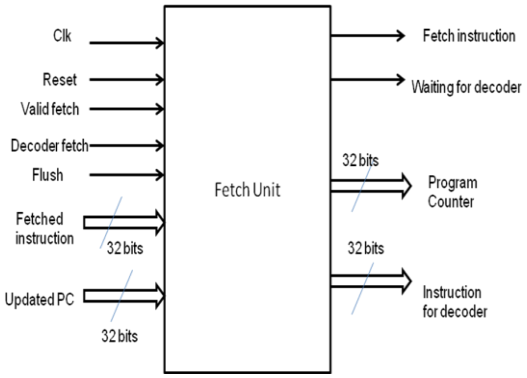

Fig -3: Fetch unit of MIPS Processor

Above fig.3 shows Fetch unit of the MIPS processor. Simulation result of the Fetch unit is shown in fig 8.

Clk, Reset: Global signals.
Valid fetch: It will fetch the valid instruction from the ram module through APB.

Decoder fetch: Decoder sends this signal to fetch unit to send next fetched instruction.

Flush: It will flush the entire module when jump or call instruction comes.

Fetched Instruction: It will fetch the instruction (data and address) from the RAM.

Updated PC: It is used to update the instruction address in the fetch module.

Fetch Instruction (IF): The instruction is fetched from memory and placed in the instruction register (IR).

Waiting for decoder: Fetched instruction is waiting for decoder to decode the operands and opcode.

Program counter: It is a register that contains the address (location) of the instruction being executed at the current time. After each instruction gets fetched, the program counter increases its value by 1 and points to the next instruction in the execution sequence. When the processor restarts or reset, the program counter normally back to 0.

3.1.2 Design of Decode Unit

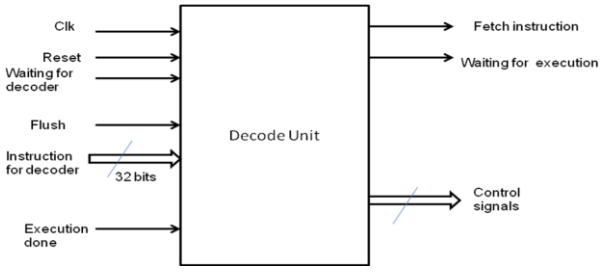

Fig -4: Decode unit of the MIPS Processor

Above fig.4 shows Decode unit of the MIPS processor. Simulation result of the Decode unit is shown in fig 9.

Clk, Reset: Global signals.
Waiting for decoder: when this signal is high, indicates that the fetch module is waiting for decoder to decode next instruction.

Flush: It will flush the entire module when jump or call instruction.

Instruction for decoder: Fetch unit sends 32-bit instruction to the decoder module.

Execution done: After execution completes, execution module sends this signal to the decoder unit.

Decoder fetch: Decoder sends this signal to fetch unit to send next fetched instruction.

Waiting for execution: After completion of decoding it will wait for next execution.

Control signals: which control all instruction like opcode, fetch, program counter etc,.

3.1.3 Design of Execution Unit

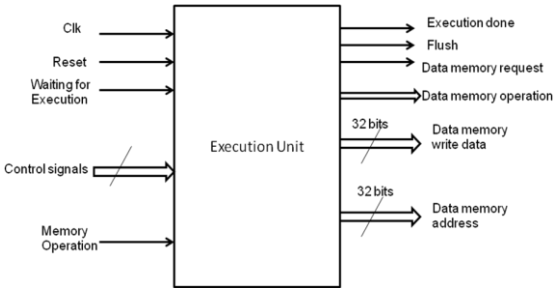

Fig -5: Execution unit of MIPS Processor

The fig.5 shows Execution unit of the MIPS processor. The fig.10 shows the simulation result of Execution unit.

Clk, Reset: Global signals.
Waiting for execution: After completion of decoding it will wait for next execution.

Control signals: It controls all instruction like opcode, fetch, program counter etc.

Memory operation done: After complete execution from the execution unit it will send the acknowledgement to memory that it completes operation.

Execution done: when it will high indicates that the execution of the instruction is complete.

Data memory request: Requesting for the data memory to store the data.

Data memory operation: This stores required data in the data memory.

Data memory address: This stores required address in the data memory.

3.2 RAM Module

The soft-core processor has RAM and it is 32 bit aligned. Some of the internal signals of RAM are Address, Data_out, Read, Ready, Clock, Reset.

Fig -6: State Diagram of RAM

Above fig.6 shows state diagram of RAM. To support real time loading and ROM logic has to be changed to RAM. ROM (Read Only Memory) is called instruction memory can't be rewritable. Program memory made it rewritable in runtime, so bipartite the RAM (Random Access Memory) into nearly two equal sections. One section is for instruction memory and the other one is used as data memory approximately 2KB each.

When WE (Write enable) =1, Write the data in to RAM.

When WE (Write enable) =0, Read the data from the RAM.

The fig.12 shows the simulation result of RAM unit.

3.3 Advance Peripheral Bus Protocol

APB is a part of the AMBA 3 (Advanced Microcontroller Bus Architecture) protocol family. It gives a low-cost interface that is optimized for minimal power consumption and reduced interface complexity. APB interfaces to any peripherals having low-bandwidth and do not require the pipelined bus interface. It has non-pipelined bus and synchronous. All signal transitions are only related to the rising edge of the clock to enable the integration of APB peripherals. Each transfer takes minimum two cycles. It is used for write and read transfers. The below tabel.1shows the list of APB signals with its description.

Table -1: APB Signal Descriptions

Sl No	Signals	Description
1	PREADY	A ready signal used by slaves to extend a APB transfer
2	PSLVERR	An error signal indicate the failure of a transfer (Valid only PSEL,PENABLE,PREADY are high)
3	PCLK	The rising edge of PCLK times all transfers on APB
4	PADDR	APB address bus can be extend up to 32 bits and driven by peripheral bus bridge
5	PWRITE	This signal indicates an APB write access when HIGH and APB read access when LOW
6	PSEL	APB bridge unit generates this signal to each slave devise and select the desired one and transfer the data
7	PENABLE	Enable signal indicated the second and further cycles used for an APB transfer
8	PWDATA	The write data, it is 32 bits wide and operates when PWRITE = 1
9	PRDATA	The read data, it is 32 bits wide and operates when PWRITE = 0
10	PRESET	APB reset signal is active LOW, its normally connected to the system bus reset signal

Operating States

The operation of APB is as shown in fig.7. The state machine consists of three states:

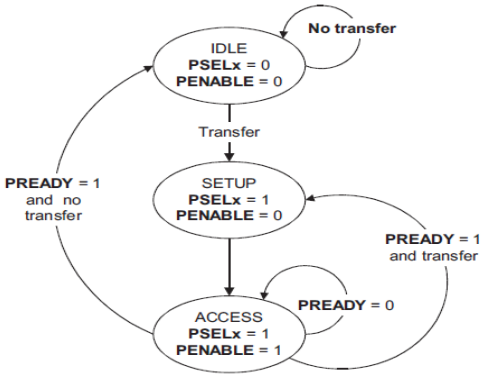

Fig -7: State diagram of operating states of APB

IDLE: It is a default state of APB.

SETUP: When a transfer is required this bus moves to the SETUP state. An appropriate select signal **PSEL** is chosen and it remains in this state for one clock cycle and next jump to ACCESS state on next rising edge of the clock.

ACCESS: When **PENABLE** is HIGH, ACCESS state is activated. The address, write, select, and write data signals stable during transition from SETUP to ACCESS state. This state is controlled by the **PREADY** signal from the slave:

PREADY =0, Bus will remains in the ACCESS state.

PREADY=1, Bus returns to IDLE state, if no other transfers are required and Bus directly moves to SETUP state still transfers are there.

The fig.13 shows the simulation result of APB.

4. RESULTS

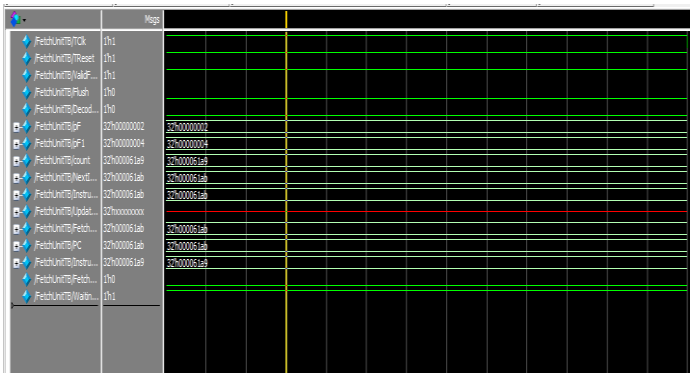

Fig -8: Simulation Result of Fetch Unit

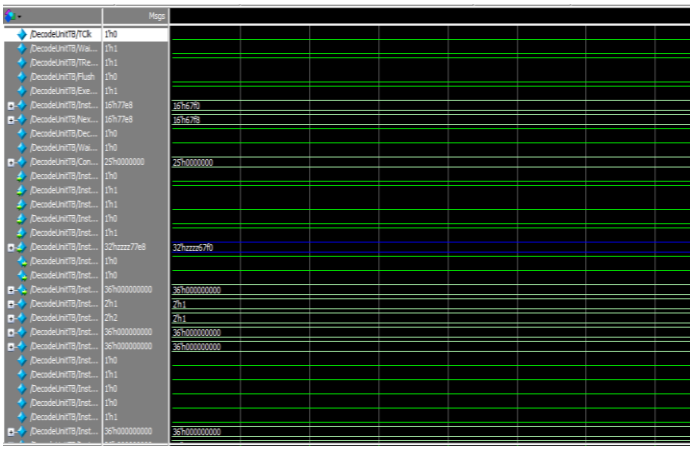

Fig -9: Simulation Result of Decode Unit

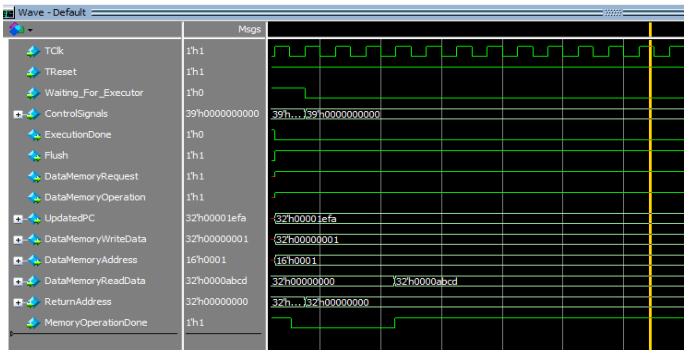

Fig -10: Simulation Result of Execution Unit

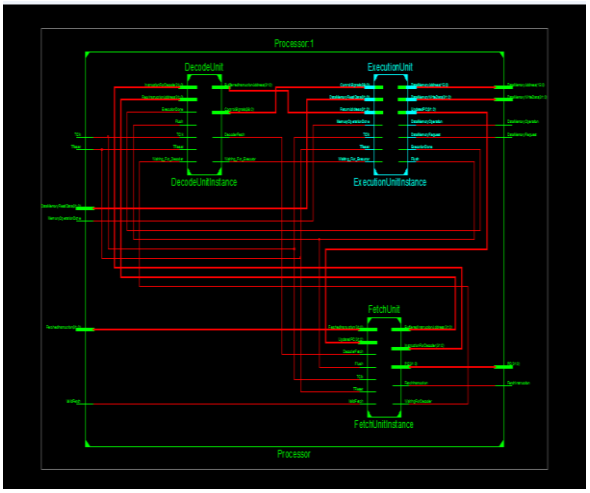

Fig -11: Top Level RTL Schematic of MIPS Processor

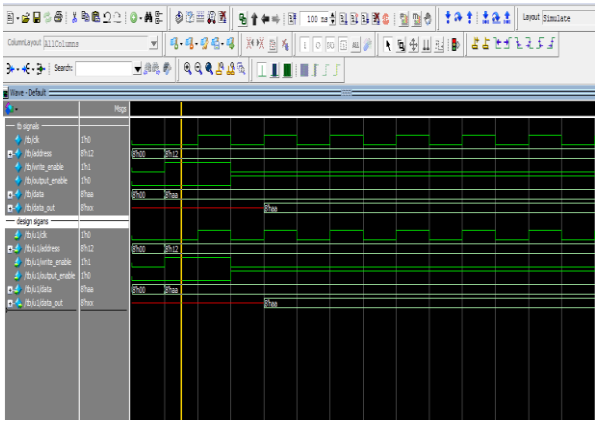

Fig -12: Simulation Result of RAM

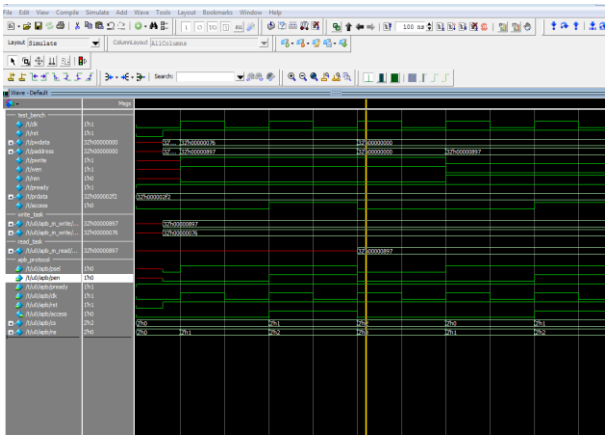

Fig -13: Simulation Result of APB

5. CONCLUSIONS AND FUTURE WORK

This paper proposes design and simulation result of fetch unit, decode unit, execution unit, MIPS processor and RAM module of 32-bit Run-Time Loadable MIPS soft-core processor and also includes APB Interface Block. The proposed system is trying to utilize less FPGA resources and reduce area consumption to consume small area out of the entire FPGA leaving plenty of FPGA resources for implementing other parallel processors on the same device. It focus on remaining blocks of system design, different optimization techniques such as pipelining will be applied in order to obtain high performance considering the parameters like speed and area.

REFERENCES

[1] Balaji valli, A. Uday Kumar, B.Vijay Bhaskar, "FPGA Implementation and Functional Verification of a Pipelined MIPS Processor" International Journal Of Computational Engineering Research (ijceronline.com) Vol. 2 Issue. 5, September-2012.

[2] Jason G. Tong, Ian D. L. Anderson and Mohammed A. S. Khalid, "Soft-Core Processors for Embedded Systems" 18th International Confernece on Microelectronics (ICM) 2006

[3] Ortega-Sanchez C., " MiniMIPS: An 8-Bit MIPS in an FPGA for Educational Purposes," in Proc. of International Conference on Reconfigurable Computing and FPGAs,pp. 152-157, 2011.

[4] Wael M ElMedany, Khalid A AlKooheji "Design and Implementation of a 32bit RISC Processor on Xilinx FPGA".

[5] V.N.Sireesha, D.Hari Hara Santosh, " FPGA Implementation of A MIPS RISC Processor" Int.J.Computer Technology & Applications,Vol 3 (3), 1251-1253, MAY-JUNE 2012.

[6] Galani Tina, R.D.Daruwala "Performance Improvement of MIPS Architecture by Adding New Features" International Journal of Advanced Research in Computer Science and Software Engineering 3(2), February - 2013, pp. 1-6.

[7] Prashant D Bhirange, V. G. Nasre, M. A. Gaikwad3, "Processor Realization for Application of Convolution" International Journal of Engineering Research and Development, Volume 2, Issue 6 (August 2012), PP. 51-55

[8] Rupali S. Balpande, Rashmi S. Keote "Design of FPGA Based Instruction Fetch & Decode Module of 32-Bit RISC (MIPS) Processor", in Proc. of International Conference on Communication and Network Technologies, Pp 409-413, 2011.

[9] Mazen Bahaidarah, Hesham Al-Obaisi "A Novel Technique for Run- Time Loading for MIPS Soft-Core Processor"Saudi International Electronics, Communications and Photonics Conference (SIECPC) 2013.