LOADING OF MACHINE CODE AT RUN TIME FOR SOFT-CORE PROCESSOR ON FPGA

1R. AROKIA PRIYA, 2PRIYANKA BALU BHOR
Assistant Professor, D.Y Patil College Of Engineering,Akurdi
Email- pbpriyankabhor@gmail.com, Email-Prinnu@yahoo.com

Abstract- Application specific customization nowadays can be very well achieved by implementing soft-core processor on FPGA’s. But if any changes are to be made to the assembly codes of the implemented processor it is required to re-implement and again download the soft-core. Here is a technique to implement a run time loading technique for a MIPS processor. This proposed design consists of three main blocks: a UART soft-core, a software tool, soft-core processor. UART soft core used in the proposed design will be superior to the conventional one, because it will be customizable as well as will also indicate the error type if any error occurs during the transmission of data. The assembly code generated and compiled by the software will be sent through UART to the CPU implemented on the FPGA at run time.

Keywords- soft-core, UART, MIPS processor

I. INTRODUCTION

A microprocessor which is defined in HDL and can be synthesized in programmable hardware like FPGA is called soft core processor. Soft-core processors provide a lot of advantages over hardcore processors. Soft-core processors are platform independent, immune to obsolescence, flexible, and are less costly. There exist 2 types of processors - (1) Complex instruction set computers (2) Reduced instruction set computers (RISC). One example of RISC processor is MIPS (Microprocessor without interlocked pipeline stages), which is a 32 bit processor designed with single cycle implementation. Three types of instructions in MIPS are: (1) immediate (I-type) (2) Jump (J-type) (3) Register (R-type).

II. PAST WORK

In the past there have been many implementations of MIPS soft-core processors as mentioned in the open literature. Some of those realizations are presented here.

1) Economic CPU design:-

In this realization of MIPS processor the focus is mainly on the resource utilization. Resource utilization is done economically on the modern chips.

Also this architecture is made customizable and reusable. Parts that are customizable are (1) memory module (2) ALU. The resource usage will be reduced if the ALU data width is reduced or if the size of memory is reduced. FPGA chips like Spartan 3 family support and give a lot many specialized resources as well as they also give ever increasing gate numbers.

ASIC applications are moving towards a much more flexible FPGA based design. In order to establish a complete application system it is needed to implement a co-processor in order to achieve some input output operations and control sensitive tasks.

Modern and powerful FPGA chips are being produced which have the capacity of migrating a general DSP or microprocessor into a single FPGA chip. Both the ASIC and CPU should co-exist in a single chip. This type of chip will not only reduce the cost but will also increase the systems speed. Also by embedding a fast CPU module in FPGA new possibilities will open up of redistributing processing tasks between the CPU and the ASIC. This will reduce system complexity and ease the design of both.

Some obstacles are yet to be resolved. FPGA has limited resources and a general purpose CPU will consume a lot of resources. Commercial CPU’s IP cores are usually less customizable than the self-designed CPU soft cores. The system consumes very little resources and leaves abundant room for implementing other customizable special control and processing modules. It would be easy to reuse the CPU core in many applications.

2) Design of 32 bit RISC processor’s instruction fetch and decode module:-

Instruction fetch (IF) module of 32 bit CPU based on RISC has been designed. IF module function is to mainly fetch instruction. Instruction decoder module is designed by pipeline theory. Instruction decoder includes write back to register file, register file, relativity check, sign extend.

Because memory was expensive in old days, designers enhanced the complication of instructions to reduce program length. This brought up a new design style called the CISC (computer instruction set computer). But long run time cost and low universal property resulted in great disparity of instructions.

RISC CPU has more advantages over CISC. It is faster, simple, easier to implement. It is extensively used in embedded systems. MIPS stand for

LOADING OF MACHINE CODE AT RUN TIME FOR SOFT-CORE PROCESSOR ON FPGA

63
microcomputer without interlocked pipeline stages. MIPS instruction format - 3 different instruction formats used for instruction are - Register format (R-type), Immediate format (I-type), Branch type format (J-type).

Data flow - R-format data path, R-I format data path, Load word data path, Memory word data path.

Pipeline design- Pipeline decomposition is as follows (a) IF instruction fetch (b) instruction decodes (ID) (c) Execution (EXE) (d) Memory input/output (MEM) (e) Write back (WB). This method uses a top down approach and uses VHDL to describe the system.

3) FPGA implementation of an integer MIPS processor

An FPGA implementation has been presented here of integer based MIPS processor using Handel-C hardware design language. The processor is a bit modified to meet the system requirements. It is implemented on the RC200 development board. Here a face detection module is implemented a part of which is MIPS processor. The processor is used for noise reduction; by means of density map regulation. The other 2 parts are video output and the video input and color detection. These 3 subsystems work together to complete the system. The system works without the MIPS stage as well but the result is unsatisfactory. After color detection noise removal is done using MIPS processor.

4) A different approach to realize UART.

Here is a presentation of a realization of UART with CPLD, FPGA and other PLD’s is done. The whole design is made very compact, stable and reliable. Use of PLD’s to implement UART can be changed in accordance with the actual needs, which is why the design is very flexible.

5) UART design

This design has an auto tuning baud rate generator. It takes asynchronous first in first out buffers to realize data exchange between UART and external devices. It can produce baud rate data according to the I/P serial data. Also FIFO has been used to speed up data transfer between peripherals and the CPU. UART is divided into the following five sub modules: transmitter logic, receiver logic, baud generator, transmitter FIFO and receiver FIFO.

III. PROPOSED WORK

As it has been mentioned in the past literature there are different implementations of both MIPS processor and UART soft core.

In this proposed architecture a technique is proposed where both the soft core UART and soft core MIPS processor can be used together to build a system.

This system will be able to make changes to the MIPS soft core processor at run time using the UART soft core.

The system will be using UART soft core to send the code at run time to the FPGA. Hence the overhead of re-implementing and reloading the whole code in the FPGA is reduced. This architecture has many advantages:

The literature mentioned in [1]-[3] implementations does not support run time loading of code. Also the UART that will be implemented here will have different properties than the ones presented in the literature [4]-[5].The UART implementation will be both customizable and reliable.

At the time of data transfer if any error occurs the type of error that has occurred will be notified. Error status register will be responsible to notify the error that has occurred. One more 8 bit register is present which will make the design customizable. Its name is LCR that is Line Control Register. LCR’s structure is as follows:

![Figure 2-LCR structure](image)

Figure 2-LCR structure

The different types of errors that would be notified are (a) Data error, (b) Parity error, (c) Framing error, (d) Overrun error (e) Break error. Basically the idea is to convert the ROM in RAM. And hence run time loading will be possible.
Some MIPS instructions are shown in table I.

<table>
<thead>
<tr>
<th>Instructions</th>
<th>Symbol</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add immediate</td>
<td>Addi</td>
<td>I</td>
</tr>
<tr>
<td>Store word</td>
<td>Sw</td>
<td>I</td>
</tr>
<tr>
<td>Load word</td>
<td>Lw</td>
<td>I</td>
</tr>
<tr>
<td>Branch on equal</td>
<td>Beq</td>
<td>I</td>
</tr>
<tr>
<td>Branch on not equal</td>
<td>Bne</td>
<td>I</td>
</tr>
<tr>
<td>Add</td>
<td>Add</td>
<td>R</td>
</tr>
<tr>
<td>Subtract</td>
<td>Sub</td>
<td>R</td>
</tr>
<tr>
<td>And</td>
<td>And</td>
<td>R</td>
</tr>
<tr>
<td>Or</td>
<td>Or</td>
<td>R</td>
</tr>
<tr>
<td>Shift left logical</td>
<td>Sll</td>
<td>R</td>
</tr>
<tr>
<td>Shift right logical</td>
<td>Srl</td>
<td>R</td>
</tr>
<tr>
<td>Shift less than</td>
<td>Slt</td>
<td>R</td>
</tr>
<tr>
<td>Jump</td>
<td>J</td>
<td>J</td>
</tr>
</tbody>
</table>

CONCLUSION

The systems that are mentioned in the literature above [1]-[3] do not have run time loading. The proposed system will be able to load the assembly code at run time through UART and the software. Hence if any modification in the CPU design is required at run time, it is possible in this proposed architecture. Now in the [4]-[5] papers the UART implementation do not have the advantage of being customizable. In the proposed design the UART will be customizable. The data can be sent at different baud rates. Also the number of stop bits can be chosen. The no of data bits to be sent can also be selected. This way the data transmission will be very flexible. Also during data transfer if any type of error occurs it will be detected and notified. In this way the overall system is made very flexible and reliable.

REFERENCES