Available online at http://www.csjonline.org/

The Caspian Sea Journal
ISSN: 1578-7899

Volume 10, Issue 1, Supplement 4 (2016) 590-596

Design and Implementation of the Mixed Single Cycle Processor on FPGA

Aghakhani Amin,
Department of Electrical Engineering, Hamedan University of Technology, Hamedan, Iran
Doost Mohammadi Mohammad Hossien
Department of Electrical Engineering, Hamedan University of Technology, Hamedan, Iran
Corresponding Author Email: Doost.mohammadi@hut.ac.ir

ABSTRACT— In this paper, a single cycle processor from MIPs family was designed and implemented on FPGA. This processor includes some features such as floating point unit (FPU), communication protocol USART, LCD driver, etc. together with specific instructions to set and use all of them. All of the proposed processor's blocks was designed using VHDL and then implemented on XC3S400 chip from Xilinx Company. Since no internal processor core has been used to design the proposed processor, so it can be implemented on any FPGA chips from any company taking into account the basic requirements.
KEYWORDS: MIPS processor, Single cycle, VHDL, FPU, FPGA

Introduction

Nowadays, there is many demands to use Field Programmable Gate Array chips (FPGAs) for high speed and parallel processing in the various industrial applications such as image processing, signal processing, neural networks, etc. FPGA are the chips in which logic elements such as logic gates AND, OR, etc. have been put together in an array form. By using hardware description languages (HDL) digital system designer can connect these elements to each other to implement digital systems. In most systems, there is a demand for a processor to execute some commands. That's why some company have placed two different kinds of processor core inside the chip alongside the configurable logic blocks (CLBs), so there would be no need to external processors and the whole system could be implemented in a single chip. However, the cost of the chips containing such a processor core are much higher than common chips with no processor core. In spite of the cost importance, there is still needs for an internal processor. In such a situation and considering the fact that the processor's main blocks are formed by logic gates, an internal soft processor can be made by designing and HDL programming of the main blocks inside a chip. In each processor, one of the main design parameters is its interior architecture. The architecture of processor has a significant effect on its structure and speed, and each designer should first determine what type of architecture is more efficient for that particular purpose. Two types of Important and famous architectures are RISC architecture and CISC architecture which respectively stand for "Reduced Instruction Set Computer" and "Complex Instruction Set Computer" [1]. Nowadays the RISC architecture is of particular significance in computer architecture and processor design. This kind of architecture has been used widely in modern processor design. Some of the specifications of this architecture can be noted such as the commands decrement, the ability of being implemented in Pipeline mode, designed to retrieve / store, less addressing mode and more internal registers. In contrast, there is the CISC architecture with the specifications such as numerous instructions set, various addressing mode, high-speed and complex instructions [1]. The purpose of this paper is to design and implement a processor based on the RISC architecture together with some useful specifications of the CISC architecture. The presented processor can be considered as a subset of the MIPS processor. The MIPS processor - Millions Instructions Per Second – is capable of running millions of instructions per second. This processor has been designed based on the RISC architecture [2]. The proposed processor was designed using hardware description language VHDL and implemented on XC3S400 chip from Xilinx Company. VHDL was the first hardware description language that has been developed by the Defense Department of the United State. This hardware description language (HDL) was designed based on VHSIC - Very High Speed Integrated Circuit [3]. The rest of this paper is organized as follow: In Section 2, the methodology and design flow of the proposed processor is presented. In this section, all of the internal blocks forming the proposed processor are introduced and their functionalities are explained. The simulation and experimental results of the proposed processor is presented in Section 3, as well as accuracy comparison with various algorithms, together with a discussion of the advantages of the proposed algorithm. Finally in Section 4, a brief conclusion of this study is presented.

Methodology and Design flow

In this section, all of the internal blocks forming the proposed processor are going to be introduced and their functionalities will be explained.

Copyright © 2016 by author and Scientific Research Publishing Inc. All Rights Reserved.

Architecture and structural units of the processor

In spite of the fact that the proposed processor is a subset of the MIPS processor, but there is still fundamental differences between them. Although 32-bit instruction format has been used to design the MIPS processor, but in this work, a 64-bit instruction format is considered to design the proposed processor. Opcode is defined by six most significant bits, thus, many commands can be added to the instructions set. One of the advantages of 64-bit format is the ability of loading registers directly with 32-bit numbers. In 32-bit format, a 32-bit number must be sign extended or zero filled to be loaded [2,4]. But thanks to 64-bit format, this is possible to directly load any 32-bit number. In a Single Cycle processor, each arithmetic operation has a specific opcode and control signals must have a certain value to execute that command. Most of these signals receive the same value in various arithmetic operations and the only thing that varies according to the desired command is the desired arithmetic operations. In the case of the proposed processor, all the expected arithmetic operations are performed using an opcode and three additional bits. By using such a method, some of the FPGA's resources consumed can be saved. The instructions set considered for the proposed processor is presented in Table 1.

Table 1. Instructions Set for the proposed processor

	Instructions	Operations
1	Add $R0,$R1,$R2	$R0<=$R1+$R2 (ALU)
2	Addi $R0,$R1,imm	$R0<=$R1+imm (ALU)
3	Sub $R0,$R1,$R2	$R0<=$R1-$R2 (ALU)
4	Subi $R0,$R1,imm	$R0<=$R1-imm (ALU)
5	Mul $R0,$R1,$R2	$R0<=$R1*$R2 (ALU)
6	Muli $R0,$R1,imm	$R0<=$R1*imm (ALU)
7	FAdd $R0,$R1,$R2	$R0<=$R1+$R2 (FPU)
8	FAddi $R0,$R1,imm	$R0<=$R1+imm (FPU)
9	FSub $R0,$R1,$R2	$R0<=$R1-$R2 (FPU)
10	FSubi $R0,$R1,imm	$R0<=$R1-imm (FPU)
11	FMul $R0,$R1,$R2	$R0<=$R1*$R2 (FPU)
12	FMuli $R0,$R1,imm	$R0<=$R1*imm (FPU)
13	F2I $R0,$R1	Convert $R1 to integer number and store it in $R0
14	I2F $R0,$R1	Convert $R1 to float number and store it in $R0
15	Set $R0,imm	$R0<=imm
16	SetR $reg,$R0	$reg<=$R0
17	LW $R0,addr	$R0<=mem(addr)
18	LWR $R0,$R1	$R0<=mem($R1)
19	SW addr,$R0	Mem(addr)<=$R0
20	SWR $R0,$R1	Mem($R0)<=$R1
21	JMP addr	PC<=addr
22	JZ addr	If(Z=1) then PC<=addr Else PC<=PC+1
23	JGR addr	
24	JLW addr	
25	JNZ addr	If(Z=0) then PC<=addr Else PC<=PC+1
26	FJZ addr	If(FEQ=1)then PC<=addr Else PC<=PC+1
27	FJGR addr	If(FGR=1)then PC<=addr Else PC<=PC+1
28	FJLW addr	If(FLW=1)then PC<=addr Else PC<=PC+1
29	Call addr	Push PC to stack and then PC<=addr
30	Return	PC<=Pop from Stack
31	IOWR $R0	IO<=$R0
32	IORD $R0	$R0<=IO
33	INT num,imm	Send imm to module
34	INT num,$R0	Send $R0 to module

Similar to MIPS, there is a Register File unit in the proposed processor that contains 32 internal register with addresses from 0 to 31 which can be read or written. The instructions set of the proposed processor is presented in Table1. The internal registers of the Register File can be initialized by using command 15 from Table 1. Two arithmetic units have been design in the proposed processor which are named ALU and FPU. ALU unit is designed to calculate the integer and fixed point values, and the FPU unit for calculation of floating point ones. Thus, the arithmetic instructions set have been provided separately for FPU and ALU units, so they can be used by user according to the certain demands. The output status pointers of the arithmetic units are saved in a 32-bits register. The status of each flag in the Status Register is shown in Figure 1.

31	30	29	28	27	26	25	24
0							

23	22	21	20	19	18	17	16
0							

15	14	13	12	11	10	9	8
0	0	FPU LW	FPU GR	FPU EQU	0	Rx Ready	Tx Ready

7	6	5	4	3	2	1	0
LCD Ready	F2I Ready	I2F Ready	FPU Ready	Sing	OV	Z	C

Figure 1. The proposed status register which includes desired status flags

Both of the main memory and the Instruction file memory are inside the chip, but they can be accessed from an external memory by some changes in the structure of the CPU.

Internal modules for the proposed processor

The proposed processor has several internal modules such as USART communication protocol, LCD driver and 7-segment driver. In this regard, the processor will be like a microcontroller. To access these modules, a command called INT is considered. Immediate data or data that is stored in the internal registers can be sent for each module just by sending the module' number. These internal modules have to be set and activated before being used. To do that, a 32-bits register in considered which is called "Config". The structure of this register is shown in Figure 2. In continue, the proposed internal modules will be introduced and explained.

31	30	29	28	27	26	25	24
Active 7-segments			-	-	-	-	-

23	22	21	20	19	18	17	16
7-segment refresh freq							

15	14	13	12	11	10	9	8
7-segment refresh freq				-	-	-	-

7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	7-EN

Figure 2. The "Config" register and its internal structure

USART Module -Universal Synchronous Asynchronous Receiver Transmitter

Both sender and receiver are designed in this module. A queue with the depth of 32 bytes is also considered for each section. Data that is sent can be an immediate data or a data from internal registers of the processor. This module can be controlled with the help of commands 33 and 34 from Table 1.

LCD Driver Module

For the proposed processor, the LCD module has been driven in 8-bit mode. The driver of LCD module has a queue with the depth of 32 that hold the data for display or execution. When data is sent and the queue is not empty, this module automatically displays the data. The main data is formed of 8-bits. But, an extra bit is also sent together with the main data to determine whether the information sent to the display, are commands to be executed or data that should be displayed on the screen. This operation can be performed on the module with the help of commands 33 and 34 from Table 1.

Segment Driver Module

This module is so designed to be able to control eight 7-segments with its outputs. Desired values are placed in two 32-bits registers called "SEG_MSB" and "SEG_LSB". The number of active displays and the updating frequency can also be set in the "Config" register. The commands 33 and 34 from Table 1 can be used to initialize registers "SEG_MSB" and "SEG_LSB" as well as specific settings for 7-segment driver.

Arithmetic Units ALU & FPU

One of the most important parts of the processor is arithmetic unit. Since one of the task of every processor is to perform arithmetic operations, therefore, the speed, accuracy and efficiency of this unit is far more substantial than other units. Two different arithmetic units were considered for the proposed processor which has been individually designed and implemented. Unlike the common processors that have a specific code for each arithmetic instruction, for the proposed processor, a specific code is intended to perform arithmetic operations. The type of arithmetic operation is specified by 3-bits of 64-bits instruction format. In addition, by using these three bits, it can be specified that which unit's output should be saved. The ALU - Arithmetic Logic Unit – is much simpler than FPU - Floating Point Unit. This unit requires no clock to do desired operation, while FPU may require several clock cycles depending on the input numbers. FPU unit is designed and implemented using IEEE 32-bit standard. All the operations considered for the ALU are also available for the FPU, such as arithmetic operations and conditional jumps. The FPU has an extra input and output signal more than ALU. The incoming signal to this unit commands the start of the calculations, and the output signal indicates that the operation has been completed and desired data can be achieved from the output of this unit. To complete the process and activate the output signal, the processor will not execute another command. Input numbers to these units can be the value of two registers of the CPU's internal registers or a register with immediate data. Description and function of each command related to these two units are given in Table 1. Arithmetic instructions are 1 to 14. Instructions 1 to 6 and 7 to 12 are related to the ALU and FPU respectively. Commands 13 and 14 provide conversion of floating-point value to integer value and vice versa.

Memory organization

The memory is composed of two parts: “Instruction File” and “Memory” similar to MIPS processor [2, 4]. The instructions are loaded from the instruction file and if required, data would be stored or retrieved in the memory. Since the two units are designed inside the same processor, thus only one clock is required to access and operate them. Instruction file has an input address and a single output data. Input address is determined by the jump instruction (conditional and unconditional) or calling a subroutine. In the case of unconditional jump instruction, the address will change to the designated location. Speaking of the conditional jump instructions, if desired condition are met then the jump to the desired address will be done, otherwise, as a normal procedures, one unit is added to the current address and instructions associated with that address will be executed. Another change in address can be made with calling subroutine. In this case, first the current address is stored in the stack consists of 32 room and then, the jump will be done to the desired address. After the subroutine procedure is completed, with a specific command, the last address on the stack is loaded on the memory address and operations will return to where they were stopped. In the case of memory unit’s management, we are able to read and write the data within. The input address of this unit can be an immediate data (direct access) or an address from one of the registers inside the processor (indirect access). Input signals named “Read” and “Write” show whether the central controller needs to read or write. Instructions 17 to 20 of Table 1 are related to the memory access operations.

Conditional and unconditional jumps

Several jump instructions have been considered for the proposed processor which also can be found in the 8086 processor [6]. These jumps are divided into three categories: unconditional jump, conditional jump and jump to a subroutine. In the case of unconditional jump, execution of instructions will goes on from a specified address. Instruction 21 of Table 1 is assigned to this jump. The conditional jumps are also divided into two categories: conditional jumps related to the ALU, and conditional jumps related to the FPU. Conditional jumps related to the ALU are done according to the output signals of this unit. After calculations are completed, signals named “SIGN, Z, OV and COUT” will be set with special values depends on the outputs of the ALU. According to these signals, various conditional jumps can be proposed such as JGR, JZ and so on. Instructions 22 to 25 of Table 1 are assigned to the conditional jumps for the ALU. Similar jump instructions have been considered for the FPU. Instructions 26 to 28 of Table 1 are assigned to the conditional jumps for the FPU. If the condition required for the jump is not met, instructions will continue from the next line, otherwise, the instructions will be executed from a specified address. An instruction for calling subroutine has been also considered for the proposed processor. When this instruction is executed, first, current address is stored in the stack and then, there will be a jump to the address of subroutine. When “Return” instruction is met, the address saved in the stack will be loaded into program counter (PC) and the instructions will be executed from where they were left. In the proposed processor, there could be 32 times of jump from one subroutine to another. Instructions 29 and 30 of Table 1 are assigned to the subroutine call.

Simulation and experimental results

In this section, the performance of the proposed processor will be evaluated using experimental tests. It should be mentioned that due to the time consumption simulations, related delays have been changed from millisecond to microsecond, to perform the operations faster.

Performance evaluation of FPU and LCD driver

A typical test program to do add, subtract and multiply operations is shown in Figure 3. These operations have been executed with the help of FPU and the output result is shown on the LCD display. The experimental result is shown in Figure 4. In this program, the result is converted to an integer value before being sent to the output. Then the result is converted to its equivalent ASCI code to be shown on display.

```
SET  $0,0,6
SET  $1,4,4
FADD $2,$0,$1
FSUB $3,$1,$0
FMUL $4,$1,$0
F2I  $5,$2
F2I  $6,$3
F2I  $7,$4
ADDI $5,$5,304
ADDI $6,$6,304
ADDI $7,$7,304
INT  9,$5
INT  1,0x12C
INT  9,$6
INT  1,0x12C
INT  9,$7
JMP  16
```

Figure 3. Typical program for performance evaluation of FPU and LCD driver

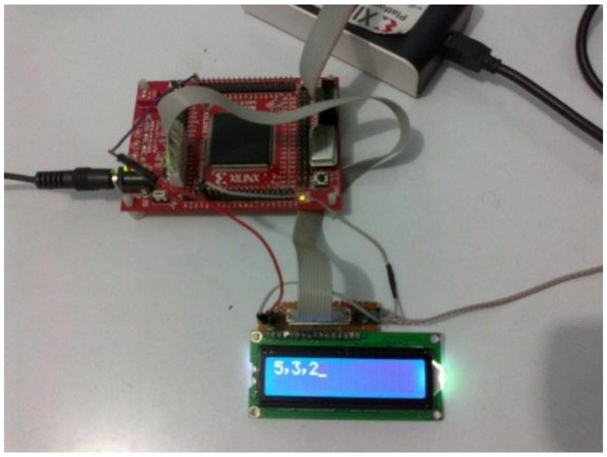


Figure 4. The experimental result of executing the test program (Figure 3)

Counter and its result on LCD
A typical program for counting from 0 to 9 and displaying result on LCD is shown in Figure 5. At first, the character zero, the value one and the value 250000000 are stored in registers 0, 1 and 2, respectively. Then, a command is sent to the LCD to clear its screen. The character saved in register 0 is sent to the LCD first and then, one unit is added to this character. At the next instruction, a delay operation of 0.5 second is performed (considering the working frequency of 50 MHz). In continue, one unit is subtracted from register 2. If the result is not equal to zero, this operation will be repeated again, otherwise, the character 9 is subtracted from register 0 and stored in register 2. If the result is equal to zero, it means that the processor is trapped in an infinite loop.
Data communication using USART protocol
A typical program is presented to test this protocol. When this program is executed, the information will be transmitted from USART protocol. This program is shown in Figure 6.

```
SET  $0,0x130
SET  $1,1
SET  $2,25000000
INT  1,0x001
INT  9,$0
ADDI $0,$0,1
SUB  $2,$2,$1
JNZ  6
SUBI $2,$0,0x13B
JNZ  2
JMP  10
```

Figure 5. Typical program for counting from 0 to 9

```
SET     $1,1
SET     $2,0x61
SET     $0,25000000
INT     2,$2
SUB     $0,$0,$1
JNZ     4
ADD     $2,$2,$1
SUBI    $7,$2,0x71
JNZ     2
JMP     9
```

Figure 6. Typical program to test the USART protocol

Simulation results related to this program is shown in Figure 7. The "CLK" signal shown in Figure 7 is the system clock needed for operations. Signal "tx_line" is the output signal related to the USART module. As a practical test, a Logic Analyzer device has been used to test the output signals. The results obtained is presented in Table 2.
Segment Driver
A typical program to test the 7-segment driver is shown in Figure 8. When executing this program, four 7-segments are activated and the number 3156 will be displayed. One to eight displays can be activated or deactivated using "Config" register. At first, the number 3165 is sent into the register 0 in a coded form, and then is stored in the "SEG_LSB". Four lower parts out of the whole eight parts of the LCD display is activated and the working frequency is specified and stored in the "Config" register. Various numbers and values can be displayed by updating "SEG_LSB" register. At the end, there is an infinite loop. The experimental result of the presented program (Figure 8) is shown in Figure 9.

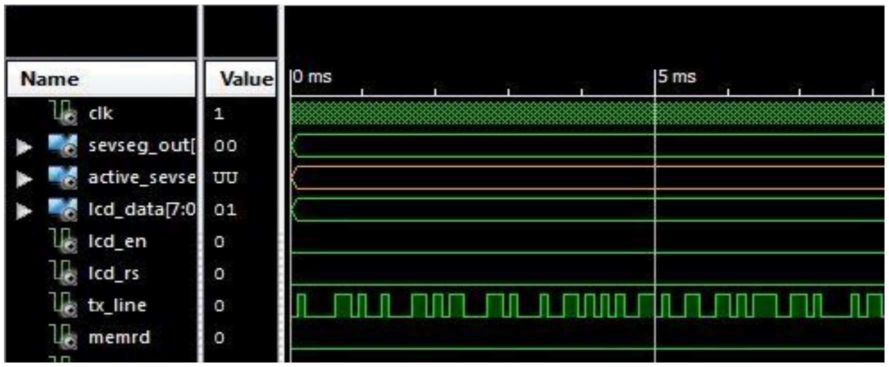


Figure 7. Simulation result of USART evaluation program

```
SET  $0,0xF40AB6BE
SETR SEG_LSB,$0
SET  $0,0x60FFF001
SETR CONFIG,$0
JMP  4
```

Figure 8. Typical program to test the 7-segment driver

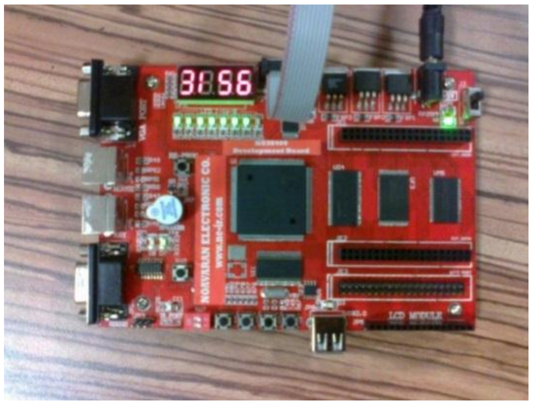


Figure 9. The result of executing program shown in Figure 8

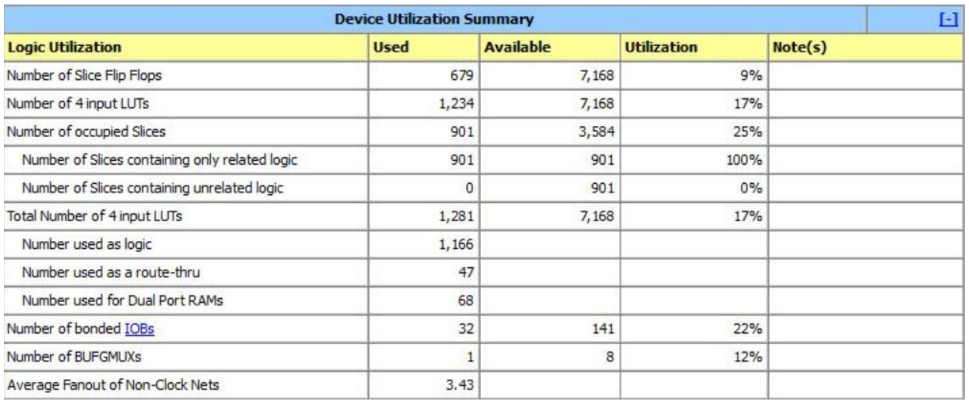
Table 2. Registered data using Logic Analyzer

Data	Time
A	2.575539000000000
B	3.575664500000000
C	4.575790000000000
D	5.575915500000000
E	6.576041250000000
F	7.576166750000000
G	8.576292250000000
H	9.576417749999999
I	10.576543250000000
J	11.576668750000000
K	12.576794250000001
L	13.576919999999999
M	13.576919999999999
N	15.577171000000000
O	16.577296499999999
P	17.577421999999999

The summary of implementing proposed processor on FPGA and the source consumed is shown in Figure 10. The FPGA's sources which have been used, can be reduced by removing some unnecessary parts. For example, FPU occupied many sources and it can be removed if there is no demand for it.

Conclusion

In this paper, a 32-bits processor was designed and implemented. Some features of RISC and CISC architectures were used for designing the proposed processor to improve the performance and make the programing easy for programmer. Various modules were considered and designed for displaying the results and communicating with out world. The performance of the proposed processor was evaluated using some practical program. Some advantages of this processor can be mentioned such as FPU unit, various applicable modules, direct and indirect access to memory. The presented processor can be used for special propose by improving its performance. This processor was implemented on XC3S400 chip from Xilinx Company. The working frequency achieved after implementation was 50MHz.



Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	679	7,168	9%	
Number of 4 input LUTs	1,234	7,168	17%	
Number of occupied Slices	901	3,584	25%	
Number of Slices containing only related logic	901	901	100%	
Number of Slices containing unrelated logic	0	901	0%	
Total Number of 4 input LUTs	1,281	7,168	17%	
Number used as logic	1,166			
Number used as a route-thru	47			
Number used for Dual Port RAMs	68			
Number of bonded IOBs	32	141	22%	
Number of BUFGMUXs	1	8	12%	
Average Fanout of Non-Clock Nets	3.43			

Figure 10. Summary of processor implementation on FPGA (source consumption)

References

1. Kumar, J.V., Swapna, C., Nagaraju, B., Ramanjappa, T. (2014). FPGA Based Implementation of Pipelined 32-bit RISC Processor with Floating Point unit, Int. Journal of Engineering Research and Applications, 4(4), pp. 0-7.
2. MIPS32™ Architecture for Programmers Volume I: Introduction to the MIPS32™ Architecture, Document Number: MD00082, Cornel University, 2001.
3. Douglas, L., Perry, L. (2002). VHDL Programming by Example, 4th edition, ISBN-13: 978-0071400701, McGraw-Hill.
4. Micro Blaze Processor Reference Guide Embedded Development Kit EDK 10.1i, Xilinx, Inc, 2008.
5. ANSI/IEEE Standard 754-1985, Standard for Binary Floating Point Arithmetic.
6. 8086 16-BIT HMOS MICROPROCESSOR 8086/8086-2/8086-1 datasheet, 1990.