FPGA Implementation of MIPS RISC Processor for Educational Purposes

Safaa S. Omran
College of Electrical and Electronic Technique
Baghdad/Iraq
omran_safaa@ymail.com

Ali J. Ibada
College of Electrical and Electronic Techniques
Baghdad/Iraq
ali.alshukri@yahoo.com

Abstract

The aim of this research is to design a 32-bit MIPS (Microprocessor without Interlocked Pipeline Stages) for RISC (Reduced Instruction Set Computer) processor. This MIPS can be used for teaching computer structure. This design defines MIPS ISA (Instruction Set Architecture), and divides the processor into two parts: the datapath unit, and the control unit. Next, a top level is implemented by connecting data and instruction memories to the processor. The VHDL (Very High Speed integrated circuit Hardware Description Language) to design hardware modling is used. The single cycle and top level is designed by using (Xilinx ISE Design Suite 13.4) software. This design is conducted on FPGA (Field Programmable Gate Array) Spartan-3AN starter kit and results from the kit are obtained.
Keywords: Single cycle, MIPS, RISC, ISA, datapath, VHDL, FPGA.

الخلاصة

الهدف من هذا البحث هو بناء معالج بدون خط انابيب مشابك (MIPS) ذو 32- بت لجهاز حاسوب ذي مجموعة الايعازات المختصره (RISC) الذي يمكن استخدامه لتدريس بنية الحاسوب. يقوم هذا التصميم بتعريف معمارية مجموعة الايعازات (ISA), ثم يقوم بتقسيم المعالج الى جزئين: مسار البيانات و وحدة التحكم. ثم بناء المستوى الاعلى بواسطة توصيل ذاكرتي الايعازات والبيانات الى المعالج. استعملت لغة وصف الكيان المادي للدوائر المتكاملة ذات السرع العالية (VHDL) في تصميم الكيان المادي. صُمم المعالج والمستوى الاعلى بواسطة برنامج (Xilinx ISE Design Suite 13.4). هذا التصميم طبق على جهاز حقل مصفوفة البوابات القابلة للبرمجة (FPGA) نوع (Spartan-3AN starter kit) وتم الحصول على النتائج.
الكلمات المفتاحية:- معالج بدون خط انابيب مشابك, حاسوب ذي مجموعة الايعازات المختصرة, لغة وصف الكيان المادي للدوائر المتكاملة ذات السرع العالية, حقل مصفوفة البوابات القابلة للبرمجة.

1. Introduction

RISC is a processor that is designed to perform a fewer number of types of computer instructions so that it can operate at a higher speed. It is also called load-store architecture because of the only operations that affect memory are load and store operations. They move data from the memory to a register or from a register to the memory, respectively. So the instruction formats are few in number with all instructions typically being one size. (John and David, 2007).

Single cycle MIPS is a RISC processor that can execute an entires instruction in one cycle. The cycle time is limited by the slowest instructions.

Many previous researches have implemented the simple design of a single cycle RISC processor in VHDL. Anjana and Krunal (2012) and Reaz, and Rahman (2012) perform the simple design of MIPS processor which can execute basic instructions (less than 12 instructions). Victor (2004) implements a single cycle and pipeline MIPS processor which can execute only 15 instructions. Sharda & Jain (2012) and Kishore and Shabeena (2011) design the 5-stage pipeline architecture of the 32-bit MIPS processor. While Ignatius Edmond Anthony (2008) made a VHDL model of pipelined DLX processor. Safaa and Hadeel (2013) design a single cycle RISC processor that can execute almost all instruction of MIPS processor. In this research a single cycle processor is designed and implemented for whole MIPS

instructions (49 instructions), hardware size is less than the last research, also another one instruction is added (hlt instruction).

VHDL is a VHSIC Hardware description language. VHSIC is an abbreviation for Very High Speed Integrated Circuit. It describes the behavior of an electronic circuit or system, such as ASICs (Application Specific Integrated Circuit) and FPGAs as well as conventional digital circuits. A fundamental motivation is used VHDL. It is a standard, technology/vendor independent language. It is therefore portable and reusable. (Volnei, 2004).

VHDL has a feature to allow the synthesis of a circuit or system in a programmable device. This paper studies the designing and prototyping of a complete single cycle MPIS RISC processor in VHDL.

FPGA (Field Programmable Gate Array) are digital integrated circuits (ICs) that contain blocks of logic along with programmable interconnects between these blocks. Design engineers can program such devices to perform a tremendous variety of tasks (Clive Maxfield, 2004).

2. Instruction Set Architecture

The first step in understanding any computer architecture is to learn its language. The words in a computer's language are called instructions. The computer's vocabulary is called the instruction set. All programs running on a computer use the same instruction set.

All instructions in RISC processor have the same length and have a single instruction format. MIPS processor uses 32-bit instructions and defines three instruction formats as shown in table 1:

1) R-type instructions: are shorts forms for register-type. They use three registers as operands: two as sources, and one as destination.
2) I-type instructions: are shorts forms for immediate-type. They use two registers operand and one 16-bit immediate operand.
3) J-type instructions: are shorts forms for jump-type. They are used only with jump instructions and use a single 26-bit address operand.

Table 1. MIPS instructions format

Field size	6-Bit	5-Bit	5-Bit	5-Bit	5-Bit	6-Bit
R-Type	op	rs	rt	rd	shamt	funct
I-Type	op	rs	rt	imm		
J-Type	op	addr				

Where:
op: basic operation of the instruction, traditionally called the opcode.
rs: the first register source operand.
rt: the second register source operand.
rd: the register destination operand, it gets the result of the operation.
shamt: shift amount, it is used in shift instruction to hold shift amount.
funct: function, it selects the specific variant of the operation in the op field.
imm: the 16-bit address which is used in data transfer instructions.
addr: the 26-bit address which is used in jump instructions.

3. Single cycle MIPS Processor Design

The complete design of a 32-bit single cycle MIPS processor consists of two parts: 32-bit datapath and control unit.

Single cycle MIPS processor has ability to execute each instruction in a single clock cycle; therefore the clock cycle period is limited by the slowest instruction's

execution time. A 32-bit MIPS requires a 32-bit datapath. The datapath can deal with 32-bit words of data, 16-bit half words of data or 8-bit bytes of data. The datapath is constructed by connecting the state elements with combinational logic that can execute the various instructions. Fig. 1 shows the state elements of single cycle MIPS processor. Control signals determine which specific instruction is carried out by the datapath at any given time. The controller contains combinational logic that generates the appropriate control signals based on the current instruction.

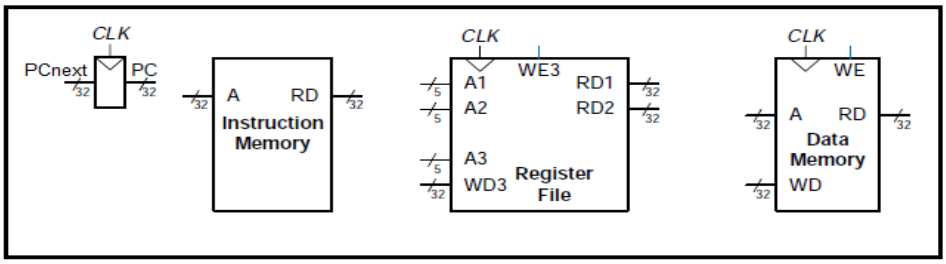


Fig. 1. The state elements of Single cycle MIPS processor

The program counter (PC) register contains the address of the instruction to be executed. To read this instruction from instruction memory, PC is simply connected to the address input (A) of the instruction memory. The instruction memory fetches the 32-bit instruction on read port (RD). The processor's actions depend on the specific instruction that is fetched. First the datapath is connected for the load (lw) instruction. For a lw instruction, the source register contains the base address. This register is specified in the first register source operand (rs) field of the instruction. These bits of the instruction connected to the address input A1 read ports of the register file. The lw instruction also requires an offset. The offset is stored in the immediate field of the instruction. The 16-bit immediate might be either positive or negative; it must be sign-extended to 32 bits. The 32-bit sign-extended value is called SignImm. The processor must add the base address to the offset to find the address to read from memory. The ALU receives two operands, SrcA comes from the register file, and SrcB comes from the sign-extended immediately. The ALU can perform many operations; 6-bit ALU control signal specifies the operation, the ALU generates a 32-bit ALU result and a zero flag. For a lw instruction, the ALU control signal should be set to "000010" to add the base address and offset. ALU result is sent to the data memory as the address for the load instruction. The data is read from the data memory onto the read data (RD) bus, then written back to the destination register in the register file at the end of the cycle. Port 3 of the register file is the write port. The destination register for the lw instruction is specified in the second register source operand *(*rt*)* field, which is connected to the port 3 address input, A3, of the register file. The ReadData bus is connected to the port 3 writing data input (WD3), of the register file. A control signal called RegWrite is connected to the port 3 writing enable input (WE3), and is asserted during a lw instruction so that the data value is written into the register file. The writing takes place on the rising edge of the clock at the end of the cycle. While the instruction is being executed, the processor must compute the address of the next instruction (PCnext). Instructions are 32 bits, the next instruction is at PC+ 4. Another adder is used to increment the PC by 4. The new address is written into the program counter on the next rising edge of the clock. This completes the datapath for the lw instruction as shown in fig. 2.

Then datapath is extended to handle the *sw* instruction. The sw instruction reads a base address from port 1 of the register and sign-extends immediately. The ALU adds the base address to the immediate to find the memory address. All of these functions are already supported by the datapath. The sw instruction also reads a second register from the register file and writes it to the data memory. The register is specified in the *rt* field. These bits of the instruction are connected to the second register file read port, A2.

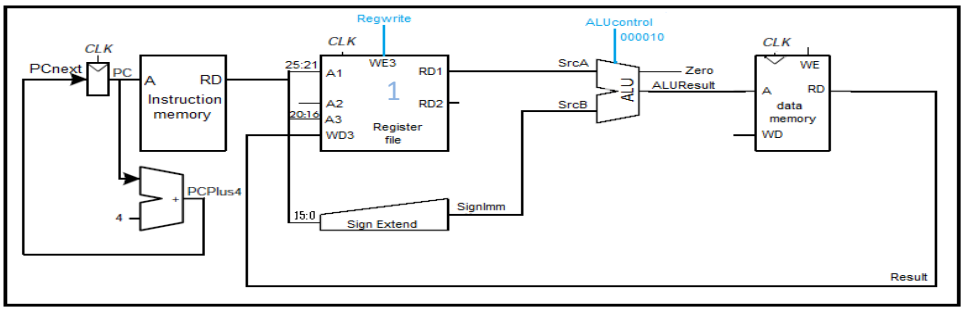


Fig. 2. Complete datapath for lw instruction

The register value is read onto the RD2 port. It is connected to the write data port of the data memory. The writing enable port of the data memory, WE, is controlled by MemWrite. For a sw instruction, MemWrite equals 1, to write the data to memory; ALU control equals "0000010", to add the base address and offset; and RegWrite equals 0, because nothing should be written to the register file. Fig. 3 shows the new connections for this function.

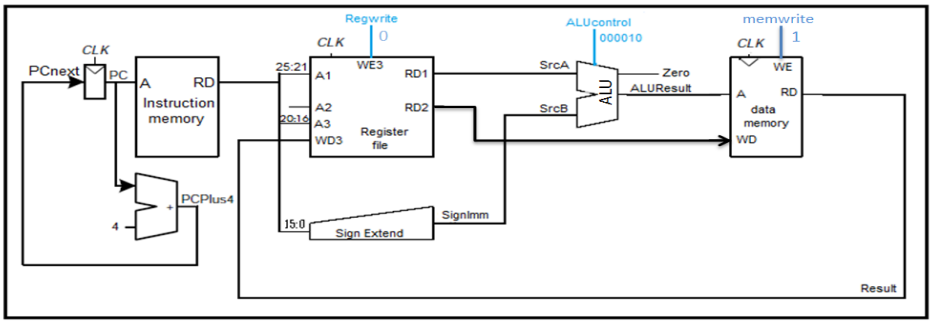


Fig. 3. Write data to memory for sw instruction

Each instruction was designed separately and added extra hardware to datapath and control unit. For all instructions design the datapath and control unit were extended.

3.1 Datapath

32-bit datapath contains elements such as memories, registers, multiplexers, sign and zero extenders. A description of each datapath element is given below:

1. Program counter (PC): is a 32-bit register. Its output (PC) represents the address of the current instruction (instr) to be executed while its input (PC next) represents the address of the next instruction.
2. Instruction memory takes a 32-bit address from PC register and read a 32-bit data at the output port.
3. Register file consists of 32 registers each of 32-bit in size. It has two reading ports (RD1 and RD2) and one writing port (WD3). Reading ports (RD1 and RD2) take 5-bit address inputs(A1 and A2) which in turn select one of the 32 registers to be

read on the reading output ports (RD1 and RD2). Writing port (WD3) takes 5-bit address input (A3) which in turn select one of the 32 registers to which the 32-bit at (WD3) input port will be written if the WE signal is 1 on the raising edge of the clock signal.

4. Data memory has one output reading port (RD) and one input writing port (WR). 32-bit data at input (WR) port is written to the memory location specified by the address (A) if (WE) signal is 1 at the raising edge of the clock signal. Signal (size_in) limits the size of data to be written either to byte (8-bit) or half word (16-bit). The content of memory location selected by (A) input is always available at (RD) output port.
5. Multiplexers are used to select one input from several inputs and pass it to the output. Mux (multiplexer) select line is controlled by a signal provided by the control unit.
6. Sign extender simply copies the sign bit (most significant bit) of half word or byte to all of the upper word output.
7. Zero extender takes a half word or byte and simply puts zeros in all of the upper word output.
8. ALU (Arithmetic Logic Unit) has been designed in order to execute all the arithmetic-logical instructions. Fig. 4 shows the new architecture of the extended ALU. ALU takes ALU control (5:0) as inputs and generates the ALU functions according to it. Table 2 illustrates functions that can be executed by the ALU.

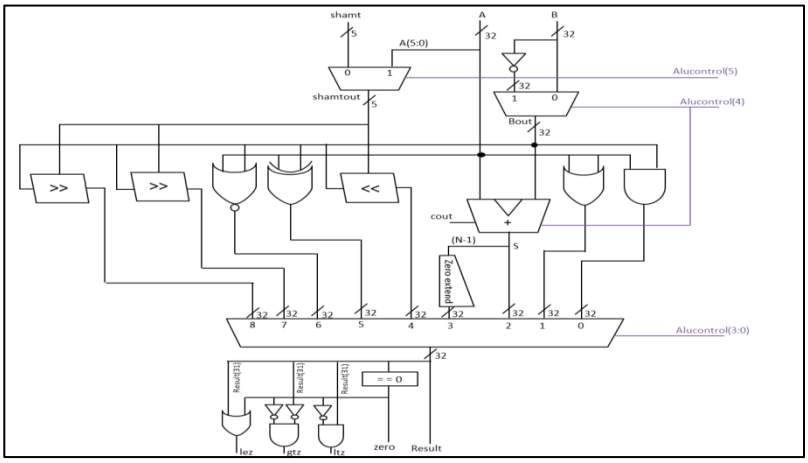


Fig. 4. ALU

Table 2. ALU functions

Alucontrol (5:0)	function
000000	A and B
000001	A or B
000010	A + B
000011	Not used
000100	Sll
000101	A xor B
000110	A nor B
000111	Srl
001000	Sra
010000	A & B’
010001	A or B’
010010	A – B
010011	Slt
010100	Not used
010101	A xor B’
010110	A nor B’
100100	Sllv
100111	Srlv
101000	Srav

9. Mul/div unit performs signed/unsigned multiplication and division. It takes two inputs of 32 bits (A and B) and produce y output of 64 bits, if the sign is 1 then signed operation will be performed, otherwise it will perform unsigned operation. When mult input is 1 and div is 0, Y(63:0)=A*B and when mult is 0 and div is 1, Y(31:0)=a/b and Y(63:32)= remainder. Then Y(63:32) is stored at hi register while Y (31:0) is stored at low (lo) register.Fig. 5 shows the complete design of the single cycle MIPS processor with its datapath and the control unit. The design can perform the integer arithmetic-logical instructions, the memory reference instructions and the branch instructions.

3.2. Control unit

The Control unit receives opcode (instr 31:26) and function (instr5:0) fields of the current instruction from the datapath to provide multiplexers select control signals, memory writing signal, register writing signal and control signals of ALU and mul/div unit. It consists of two parts: main control and R-type control.

Main control uses opcode (instr 31:26) field as inputs and produce multiplexers select control signal, memory writing, register writing and 3-bit ALUop signals as shown in table 3. The meanings of ALUop signals are giving in table 4.

Table 3. main control truth table

Instruction	Opcode	Sh_B	S_zext	Regwrite	Regdst	Alusrc	Branch	Brchne	Blez	Bltz	Bgtz	Memwrite	Memtoreg	Jump	Jal	ALUop	hlt
R_type	000000	xx	x	1	01	00	0	0	0	0	0	0	00	0	0	110	0
lw	100011	xx	x	1	00	01	0	0	0	0	0	0	01	0	0	000	0
sw	101011	11	x	0	xx	01	0	0	0	0	0	1	xx	0	0	000	0
beq	000100	xx	x	0	xx	00	1	0	0	0	0	0	xx	0	0	001	0
bne	000101	xx	x	0	xx	00	0	1	0	0	0	0	xx	0	0	001	0
blez	000111	xx	x	0	xx	00	0	0	1	0	0	0	xx	0	0	001	0
bltz	000001	xx	x	0	xx	00	0	0	0	1	0	0	xx	0	0	001	0
bgtz	000110	xx	x	0	xx	00	0	0	0	0	1	0	xx	0	0	001	0
addi	001000	xx	x	1	00	01	0	0	0	0	0	0	00	0	0	000	0
addiu	001001	xx	x	1	00	01	0	0	0	0	0	0	00	0	0	000	0
j	000010	xx	x	0	xx	xx	x	x	x	x	x	0	xx	1	0	xxx	0
jal	000011	xx	x	1	10	xx	x	x	x	x	x	0	xx	1	1	xxx	0
andi	001100	xx	x	1	00	10	0	0	0	0	0	0	00	0	0	010	0
ori	001101	xx	x	1	00	10	0	0	0	0	0	0	00	0	0	011	0
xori	001110	xx	x	1	00	10	0	0	0	0	0	0	00	0	0	100	0
slti	001010	xx	x	1	00	01	0	0	0	0	0	0	00	0	0	101	0
sltiu	001011	xx	x	1	00	01	0	0	0	0	0	0	00	0	0	101	0
lui	001111	xx	x	1	00	11	0	0	0	0	0	0	00	0	0	000	0
lb	100000	xx	0	1	00	01	0	0	0	0	0	0	11	0	0	000	0
lbu	100100	xx	1	1	00	01	0	0	0	0	0	0	11	0	0	000	0
lh	100001	xx	0	1	00	01	0	0	0	0	0	0	10	0	0	000	0
lhu	100101	xx	1	1	00	01	0	0	0	0	0	0	10	0	0	000	0
sb	101000	00	x	0	xx	01	0	0	0	0	0	1	xx	0	0	000	0
sh	101001	01	x	0	xx	01	0	0	0	0	0	1	xx	0	0	000	0
hlt	111100	xx	x	0	xx	xx	x	x	x	X	x	0	xx	x	x	xxx	1

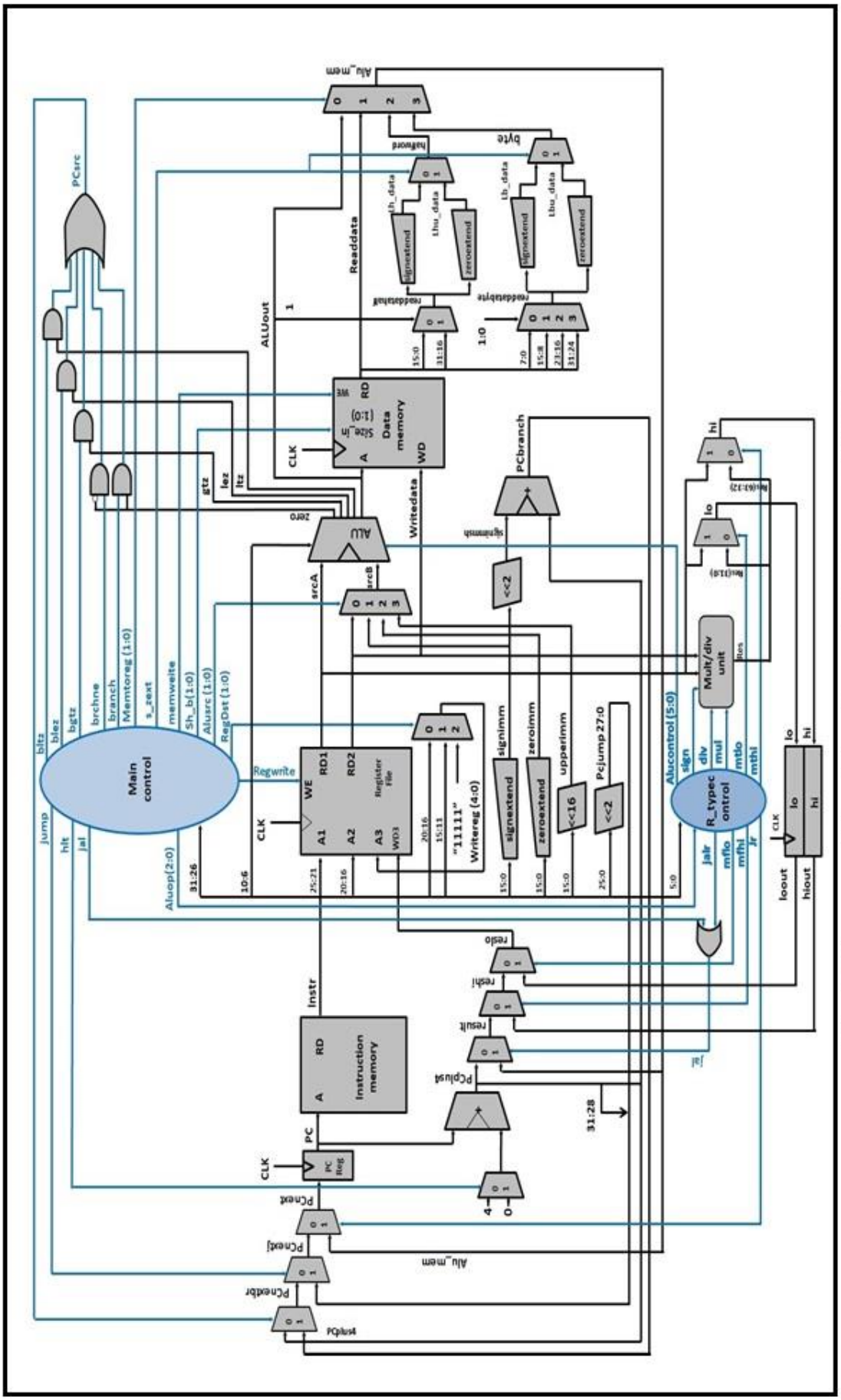


Fig.5 The complete datapath control scheme of MIPS

Table 4. ALUop meaning

Aluop	Meaning
000	Add
001	Sub
010	And
011	Or
100	Xor
101	Slt
110	Look at funct field
111	N/a

R-type control uses ALUop signals with function (instr 5:0) field of instruction to produces ALU control (5:0) signals and several signals used in the execution of R-type instructions. Table 5 shows the truth table of R-type control.

Table 5. R-type control the truth table.

Aluop	Funct	Alucontrol	Jr	Jalr	div	Mult	Sign	Mthi	Mtlo	Mfhi	Mflo
000	xxxxxx	000010 (add)	0	0	0	0	0	0	0	0	0
001	xxxxxx	010010 (sub)	0	0	0	0	0	0	0	0	0
010	xxxxxx	000000 (and)	0	0	0	0	0	0	0	0	0
011	xxxxxx	000001 (or)	0	0	0	0	0	0	0	0	0
100	xxxxxx	000101 (xor)	0	0	0	0	0	0	0	0	0
101	xxxxxx	010011 (slt)	0	0	0	0	0	0	0	0	0
11x	100000	000010 (add)	0	0	0	0	0	0	0	0	0
11x	100001	000010 (add)	0	0	0	0	0	0	0	0	0
11x	100010	010010 (sub)	0	0	0	0	0	0	0	0	0
11x	100011	010010 (sub)	0	0	0	0	0	0	0	0	0
11x	100100	000000 (and)	0	0	0	0	0	0	0	0	0
11x	100101	000001 (or)	0	0	0	0	0	0	0	0	0
11x	100110	000101 (xor)	0	0	0	0	0	0	0	0	0
11x	100111	000110 (nor)	0	0	0	0	0	0	0	0	0
11x	101010	010011 (slt)	0	0	0	0	0	0	0	0	0
11x	101011	010011 (slt)	0	0	0	0	0	0	0	0	0
11x	000000	000100 (sll)	0	0	0	0	0	0	0	0	0
11x	000010	000111 (srl)	0	0	0	0	0	0	0	0	0
11x	000011	001000 (sra)	0	0	0	0	0	0	0	0	0
11x	000100	100100 (sllv)	0	0	0	0	0	0	0	0	0
11x	000110	100111 (srlv)	0	0	0	0	0	0	0	0	0
11x	000111	101000 (srav)	0	0	0	0	0	0	0	0	0
11x	001000	000010 (jr)	1	0	0	0	0	0	0	0	0
11x	001001	000010 (jalr)	1	1	0	0	0	0	0	0	0
11x	011000	000000 (mult)	0	0	0	1	1	0	0	0	0
11x	011001	000000 (multu)	0	0	0	1	0	0	0	0	0
11x	011010	000000 (div)	0	0	1	0	1	0	0	0	0
11x	011011	000000 (divu)	0	0	1	0	0	0	0	0	0
11x	010001	000000 (mthi)	0	0	0	0	0	1	0	0	0
11x	010011	000000 (mtlo)	0	0	0	0	0	0	1	0	0
11x	010000	000000 (mfhi)	0	0	0	0	0	0	0	1	0
11x	010010	000000 (mflo)	0	0	0	0	0	0	0	0	1

3.3. VHDL top-level implementation

In VHDL each element in the datapath and control unit is composed as component then these components are combined to form the Single cycle MIPS processor. This processor is connected to an external separate instruction and data memories through the data and address busses.

Fig.6 demonstrates MIPS interfacing with data and instruction memories. In this work, data and instruction memories size is 256 byte hold 64 words of 32-bit, which are quite enough to load the designed programs. Test bench is written to test the single cycle MIPS processor and shows simulation of program's execution before configure FPGA.

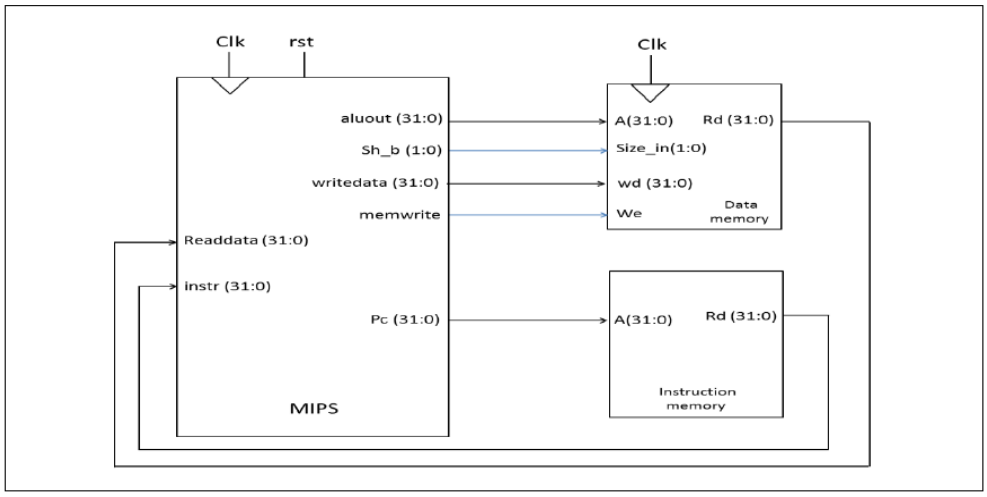


Fig.6. MIPS Single-cycle processor top level

When the test bench is written and checks simulation. The result shows that PC continues point to the next instruction after all instructions of program that are written in the instruction memory are executed, this causes an empty 32 bit is executed and gives zero to all signal and addresses. When program counter reaches the end of instruction memory, program counter points to the first instruction in the instruction memory and executes same program again. It is necessary to design new instruction that must write after last instruction in program and used to stop program.

Hlt instruction which has opcode (111100) is designed. Extra signal is added (hlt signal) to control multiplexer that select the value that should be added to program counter (4 when hlt = 0, and zero when hlt = 1) to give PC next as shown in Fig.5. Main control sends zero in hlt signal in all instruction except when it reads (opcode = 111100) (send one)as shown in Table 3, this makes program counter keep staying at the same instruction and never point to the next instruction.

4. Hardware Implementation

The final design is a hierarchical, VHDL-based design. This means that the top-level design file is a VHDL file that references to several other lower-level macros. The lower level macros are all VHDL modules. This design is synthesized by using Xilinx ISE (13.4) design suite.

Once the VHDL code is simulated and all operations were verified using Xilinx Isim simulator, the design would be implemented on the Spartan-3AN starter

kit with (XC3S700) device. Downloading the design to a Spartan-3AN board requirs setting on-board jumpers as indicated in fig. 7.

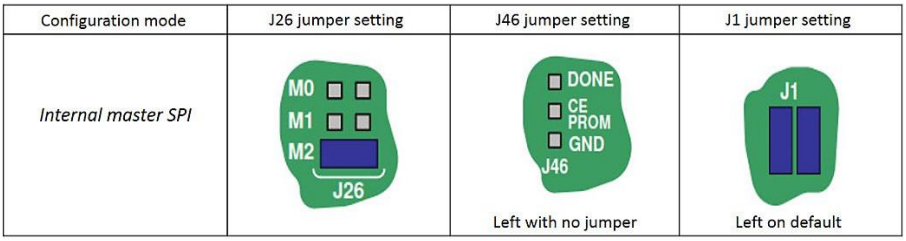


Fig. 7. Configuration mode jumper settings

These Jumper settings indicate that the Spartan-3AN FPGA is configured by using the internal In-system flash memory which called internal master SPI configuration mode.

The Spartan-3AN starter kit board has a 2-line by 16-character LCD (Liquid Crystal Display) display which has an internal Sitronix ST7066U graphics controller. The FPGA controls the LCD via the 8-bit data interface shown in fig. 8.

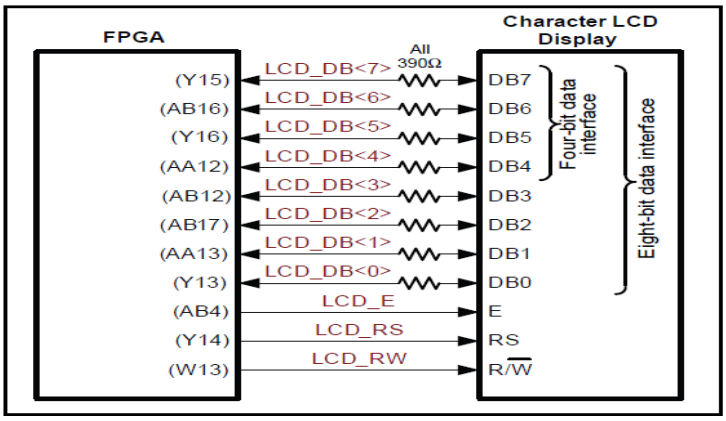


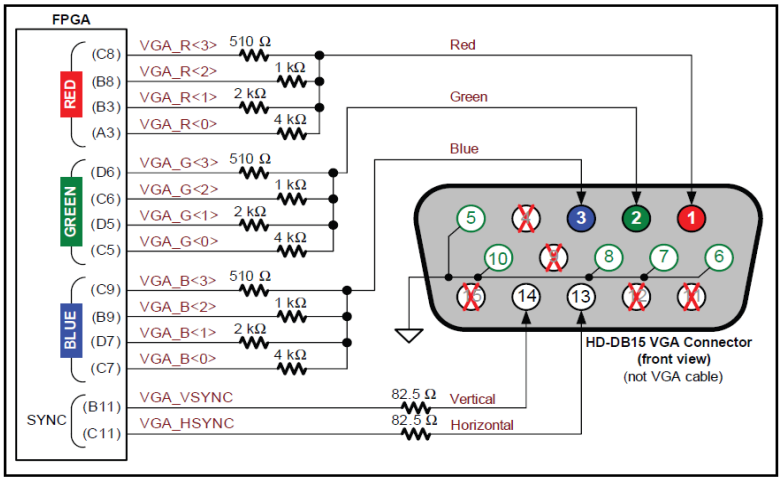
Fig. 8. Character LCD interface

By sending appropriate commands to the display controller, text can be displayed on the LCD. Therefore, a VHDL code is required for Power-On Initialization, Display Configuration and writing data to the LCD.

Since 2-line by 16-character LCD is not enough to show the results, FPGA is interfaced to PC monitor via a standard high-density HD-DB15 female connector VGA (Video Graphics Array) display port. The display device can be a CRT (Cathode-Ray Tube), a flat panel or even a projector. The FPGA directly drives the five VGA signals via resistors. Each red, green, and blue signal has four outputs from the FPGA that feed a resistor-divider tree. This approach provides 4-bit resolution per color, generating 12-bit color, or 4,096 possible colors. The series resistor, in combination with the 75Ω termination built into the VGA cable, ensures that the color signals remain in the VGA-specified 0V to 0.7V range. Fig. 9 shows VGA connections from the starter kit board.

Driving the VGA_R [3:0], VGA_G [3:0], and VGA_B [3:0] signals High (1) or Low (0) to generate the desired colors. The scaled analog output is generated by a

resistor-divider that converts the FPGA's digital outputs for an individual color. Each individual color output supports 16 possible values, as described by Equation 1. The three separate controls for red, green ,and blue support a maximum of 12-bit color, or 4,096 values.



ig. 9. VGA Connections from the Starter Kit Board

$$\text{Color out} = \frac{\text{VGA[3:0]}}{15} \; Color \quad \ldots\ldots\ldots\ldots\ldots (1)$$

The FPGA application can also treat the VGA port as a three-bit interface by driving all four color outputs with the same digital value. The corresponding eight basic color values are shown in Table 6.

Table 6. Example Display Color Codes

VGA_R[3:0]	VGA_G[3:0]	VGA_B[4:0]	Resulting Color
0000	0000	0000	Black
0000	0000	1111	Blue
0000	1111	0000	Green
0000	1111	1111	Cyan
1111	0000	0000	Red
1111	0000	1111	Magenta
1111	1111	0000	Yellow
1111	1111	1111	White

The FPGA may drive the VGA monitor in 640 by 480 mode. CRT-based VGA displays use amplitude-modulated and moving electron beams (or cathode rays) to display information on a phosphor-coated screen. LCDs use an array of switches that can impose a voltage across a small amount of liquid crystal, thereby changing light permittivity through the crystal on a pixel-by-pixel basis.

Displaying text is an important function of a video controller. To display text on VGA display, video controller organizes the 640x480 display area into larger tiles where each tile represents a character location. The size of each character in the font

is 8 pixels wide and 16 pixels high. Mapped onto a 640x480 display, this font displays 80 text characters in each line (640 pixels divided by 8 columns per character) and 30 lines (480 / 16). Each of the 640x480 pixels in the display are associated with one of the 80x30 character locations.

The appearance of the characters on the screen is determined by a font ROM. The font ROM contains the pattern of pixels that should be displayed on the screen when a particular character needs to be displayed. The bits within the font ROM indicate which pixels of a 8 x 16 bit tile should be displayed in the foreground and which pixels on the display should be displayed in the background. A '1' in the font ROM indicates the corresponding pixel should be displayed in the foreground while a '0' in the font ROM indicates that the corresponding pixel should be blanked or put in the background (black in our case). Fig. 10 demonstrates the contents of the font ROM for the upper-case character 'A'.

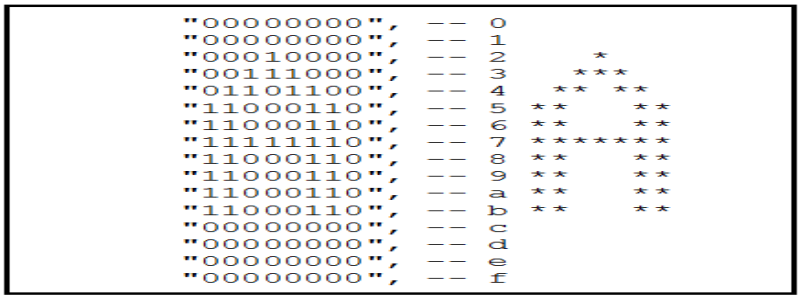


Fig. 10 Contents of the font ROM for the upper-case character 'A'

In addition to the font ROM, it is needed to use a character memory that stores the character value at each of the 80x30 character locations on the display. The minimum size of this memory is 80x30x8 bits to provide enough room to store characters (one byte each) for each of the 80 columns and 30 rows. This size, however, is not an even power of two. To simplify the circuit, the memory size will be 128x32x8 bits. Although this result in wasted memory, it is significantly simplify the circuit.

The character memory and font ROM are combined together as shown in fig. 11, to determine the pixel value for the current pixel value being displayed. There are three steps in the process of determining the current pixel value:

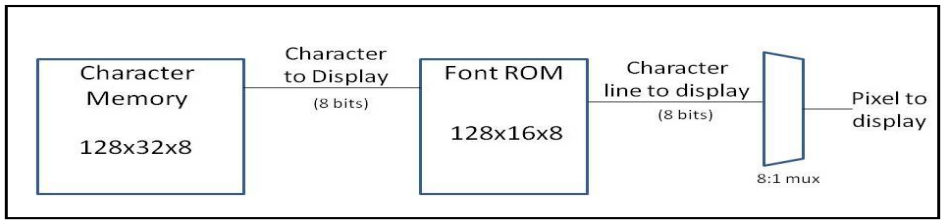


Fig. 11 character memory and font ROM for displaying text

First step in this process is to determine the current character based on the current pixel address. Second step is to use this character value to access the font ROM so the current character pixels can be drawn. The address for the font ROM is

based on the current character value (which was read from the character memory) and the current pixel row. The current pixel row is used to determine which line of the character needs to be displayed . Final step is to determine which of the 8 pixels from the current character row should be displayed. The multiplexer 8:1 is used to select the current pixel.

5. Results

The program shown in fig. 12 is stored in the instruction memory. It is used to calculate Fibonacci sequence {1, 1, 2, 3, 5, 8, 13, etc.} for first 40 number and save resulting sequence in data memory at address [0: 39]. Fig.12 shows the procedure to calculate Fibonacci sequence.

	Assembly	Address	Description	Machine
	addi $t6, $zero, 1	0	$t6 = 1	200E0001
	addi $t7, $zero, 4	4	$t7 = 4	200F0004
	sw $t6, 0($zero)	8	mem[$zero + 0] = $t6	AC0E0000
	sw $t6, 0($t7)	c	mem[$t7 + 0] = $t6	ADEE0000
	addi $t9, $zero, 9C	10	$t9 = 9C	2019009C
	addi $t0, $zero, 8	14	$t0 = 8	20080008
loop:	addi $t3, $t0, -8	18	$t3 = $t0 + -8	210BFFF8
	addi $t4, $t0, -4	1c	$t4 = $t0 + -4	210CFFFC
	lw $t1, 0($t3)	20	$t1 = mem[$t3 + 0]	8D690000
	lw $t2, 0($t4)	24	$t2 = mem[$t4 + 0]	8D8A0000
	add $t5, $t1, $t2	28	$t5 = $t1 + $t2	012A6820
	addi $t0, $t0, 4	2c	$t0 = $t0 + 4	21080004
	sw $t5, -4($t0)	30	mem[$t0 + -4] = $t5	AD0DFFFC
	slt $t8, $t9, $t0	34	if ($t9 < $t0) $t8 = 1 else $t8 = 0	0328C02A
	beq $zero, $t8, loop	38	if $zero = t8 then go to -9	1018FFF7
	hlt	3c	stop program	F0000000

Fig.12. Fibonacci sequence test program

Fibonacci sequence program shown in Fig. 12 is executed. The results are shown in Fig. 13. As long as memwrite signal is 1, the results are stored at data memory. For first 40 number of Fibonacci sequence the last value is "06197ecb"$_h$ stored at address $(9c)_h$ in data memory. Fig.14 shows all contents of data memory after executing Fibonacci sequence test program. It shows data for 64 words; data is saved in first 40 locations in data memory, while other data memory values are staying zeros.

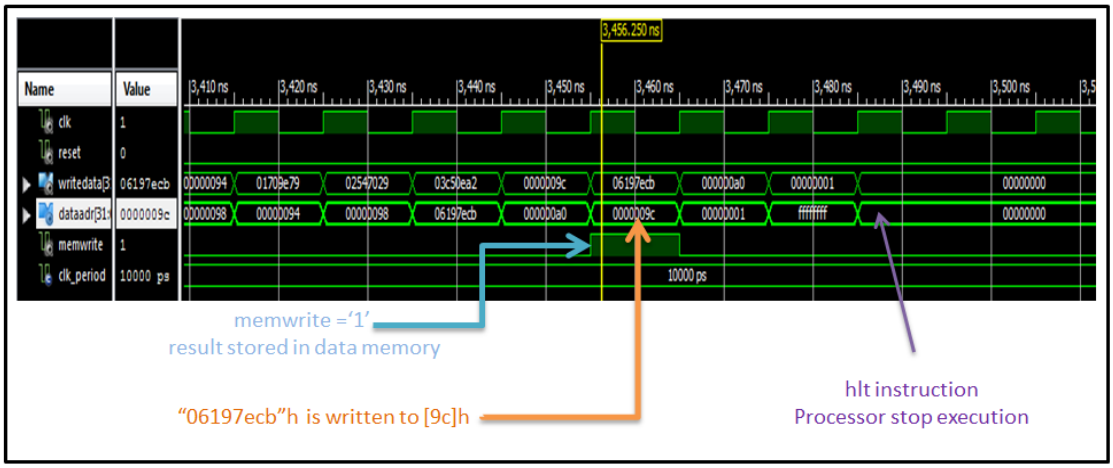


Fig.13. Single cycle MIPS simulation waveform of Fibonacci sequence.

0x3F	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
0x37	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
0x2F	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
0x27	06197ECB	03C50EA2	02547029	01709E79	00E3D1B0	008CCCC9	005704E7	0035C7E2
0x1F	00213D05	00148ADD	000CB228	0007D8B5	0004D973	0002FF42	0001DA31	00012511
0x17	0000B520	00006FF1	0000452F	00002AC2	00001A6D	00001055	00000A18	0000063D
0xF	000003DB	00000262	00000179	000000E9	00000090	00000059	00000037	00000022
0x7	00000015	0000000D	00000008	00000005	00000003	00000002	00000001	00000001

Fig.14. Contents of data memory after execute Fibonacci sequence program.

This design is configured on Xilinx Spartan-3AN starter kit FPGA and results shown in fig.15 and fig.16.

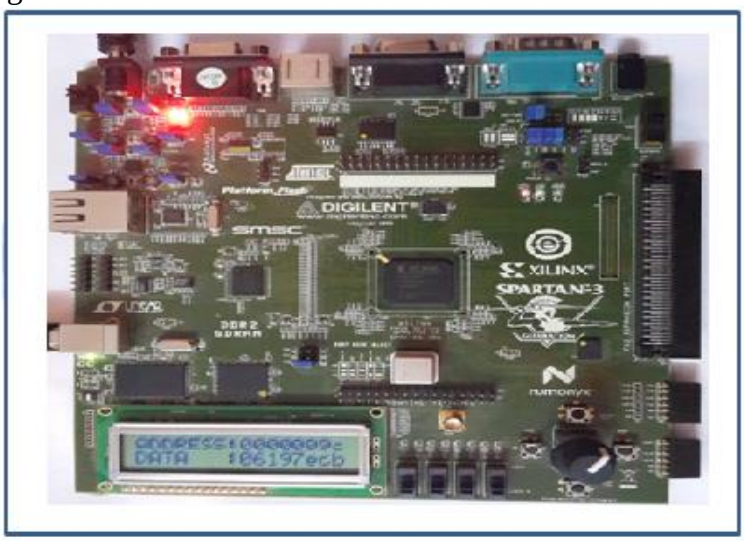


Fig.15. Fibonacci sequence program Results on Xilinx Spartan-3AN starter kit which in last value shows on LCD

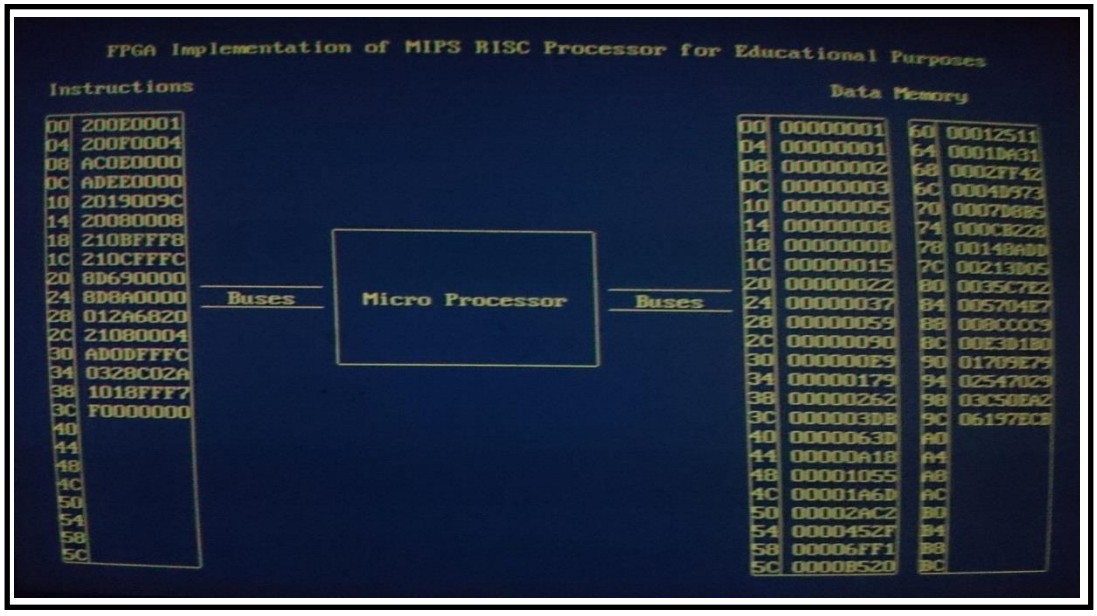


Fig.16. Fibonacci sequence program Results on Xilinx Spartan-3AN starter kit that showed on PC monitor

6. Conclusions

VHDL design of MIPS RISC processor for single cycle processor has been implemented for whole instructions which consist of 49 instructions. Also hlt instruction is added to processor to stop program execution. After complete this design, various programs are simulated by using (Xilinx ISE Design Suite 13.4). Xilinx Spartan-3AN starter kit FPGA configured by using internal master SPI mode and results from the kit are obtained. It is meaning that the design works properly.

7. References

Anjana R. and Krunal Gandhi, 2012, " VHDL Implementation of a MIPS RISC Processor", International Journal of Advanced Research in Computer Science and Software Engineering, Vol. 2, No.8, pp.83-88.

Clive Maxfield, 2004, "The Design Warrior's Guide to FPGAs: Devices Tools and Flows", Elsevier, USA.

Ignatius Edmond Anthony, 2008, "VHDL Implementation of Pipelined DLX Microprocessor", MSc. Thesis, University Technology Malaysia (UTM), Malaysia.

John L. Hennessy and David A. Patterson, 2007, "Computer Architecture: A Quantitative Approach", Morgan Kaufmann, San Francisco, USA.

M. B. I. Reaz, J. Jalil and L.F. Rahman, 2012, "Single Core Hardware Modeling Of 32-Bit MIPS RISC Processor With A Single Clock ", Research Journal Of Applied Sciences, Engineering And Technology, Vol.4, No.7 , pp.825-832.

M. Kishore Kumar and MD. Shabeena Begum, 2011, "FPGA based implementation of 32 bit RISC processor", International Journal of Engineering Research and Applications, Vol. 1, NO 3, pp.1148-1151.

Safaa S. Omran and Hadeel S. Mahmood, 2013, "Hardware modelling of a 32-bit, single cycle RISC processor using VHDL", ICIT 2013 The 6th International Conference on Information Technology, ISSN 2306-6105, ISBN 978-9957-8583-1-5.

Sharda P. Katke, G.P. Jain, 2012, "Design and Implementation of 5 Stages Pipeline Architecture in 32 Bit RISC Processor", International Journal of Emerging Technology and Advance Engineering, Vol. 2, NO. 4, pp. 340-346, Apr.

Victor P. Rubio, 2004, "A FPGA Implementation of A MIPS RISC Processor for Computer Architecture Education", MSc. Thesis, New Mexico State University, Las Cruses, New Mexico, America.

Volnei A. Pedroni, 2004, "circuit design with VHDL", MIT Press, London, England.