Lab 8 Single Cycle CPU

CSC343 Fall 2017

December, 6 2017

Jeter Gutierrez

TABLE OF CONTENTS

1.OBJECTIVE: In this lab I will be implementing a single cycle cpu that can perform several operations. It will also support I type and R type instructions. I will use 3 ported memory. I will be able to read from two addresses at a time and write to one at a time. I will design a instruction interpretation component that will be used in other to understand what the operation code for each operation is responsible for and how it is performed. I will also implement an ALU that will take a 32 bit instruction and perform the operations necessary according to the operation instruction that is sent to it. In order to make it more advanced it will also be used to store the dates into the 3 ported memory that has 32 registers each 32 bits wide. Not only will my single cycle cpu be capable of performing two different types of reading operations (i- type and r-type) but it will also be able to calculate the dot product, read it's instruction from the Synchronous static ram of the DE2-115 Programmable FPGA board but it will also run branch if equal operation. I will implement all these interactions on the finalized product. The finalized product will use all of the components together, it will be capable of storing instructions into memory, values into registers and reading instructions from memory to then use the data stored in registers to perform the different operations. After I have brought everything together on the SINGLE CYCLE CPU component I will demonstrate the final product on the PROGRAMMABLE FPGA DE2-115 board.

The circuits that will be designed in this project are:

- RAM 3 PORT
- INSTRUCTION INTERPRETATION UNIT

- ALU
- SINGLE CYCLE CPU

BITS OF THE GIVEN INSTRUCTION							
31-26		25-21	20-16	15-11	"10-7"	"6-3"	"2-0"
"000000"	and	registre a	register b	write registe	N/A	N/A	N/A
"000001"	or	registre a	register b	write registe	N/A	N/A	N/A
"000010"	xor	registre a	register b	write registe	N/A	N/A	N/A
"000011"	not	registre a	register b	write registe	N/A	N/A	N/A
"000100"	sll	registre a	register b	write registe	N/A	N/A	N/A
"000101"	srl	registre a	register b	write registe	N/A	N/A	N/A
"000110"	rol	registre a	register b	write registe	N/A	N/A	N/A
"000111"	ror	registre a	register b	write registe	N/A	N/A	N/A
"001000"	slt	registre a	register b	write registe	N/A	N/A	N/A
"001001"	addition	registre a	register b	write registe	N/A	N/A	N/A
"001010"	subtraction	registre a	register b	write registe	N/A	N/A	N/A
"001011"	multiplicatio	registre a	register b	write registe	N/A	N/A	N/A
"001100"	ori	register a	write registe	immediate register			
"001101"	dot product	top array 1	top array 2	write registe	#elements in 1	#elements in 2	N/A
"001110"	branch	register a	register b	address of instruction to jump to if beq is 1			

Figure: Truth table for the implementation of the single cycle cpu that I will be designing in this project.

2. RAM 3 PORT

2.1 FUNCTIONALITY AND SPECIFICATIONS FOR RAM 3 PORT MEMORY

The purpose of this component is to make a 3 ported register memory. There will be 32 registers in this design and each one will be 32 bits wide. As the inputs we will have a clock signal that will be used to write to the specified register. We will also have 3 addresses that will each be 5 bits wide in order to specify which registers are being used. The first two will be the two addresses we will read from and the third one will be the address that we write to when the clock signal received a rising edge and write enable is on for the input. It will also have 2 32 bit output signals that will return the data that is stored in the first two addresses when they are specified. There will also be an input data that is 32 bits wide and will be used to store the data into the

register specified by the write address. This will eventually be used as a component in the ALU UNIT both to store values into the register and also to perform the operations using the INSTRUCTION INTERPRETATION UNIT.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
LIBRARY ALTERA_MF;
USE ALTERA_MF.ALL;
ENTITY GUTI_3_PORT_MEMORY IS
PORT
(       CLOCK: IN STD_LOGIC ;
        DATA: IN STD_LOGIC_VECTOR (31 DOWNTO 0);
        RDADDRESS_A: IN STD_LOGIC_VECTOR (4 DOWNTO 0);
        RDADDRESS_B: IN STD_LOGIC_VECTOR (4 DOWNTO 0);
        WRADDRESS: IN STD_LOGIC_VECTOR (4 DOWNTO 0);
        WREN: IN STD_LOGIC := '1';
        QA: OUT STD_LOGIC_VECTOR (31 DOWNTO 0);
        QB: OUT STD_LOGIC_VECTOR (31 DOWNTO 0));
END GUTI_3_PORT_MEMORY;
ARCHITECTURE SYN OF GUTI_3_PORT_MEMORY IS
SIGNAL SUB_WIRE0: STD_LOGIC_VECTOR (31 DOWNTO 0);
SIGNAL SUB_WIRE1: STD_LOGIC_VECTOR (31 DOWNTO 0);
COMPONENT ALT3PRAM
GENERIC (INDATA_ACLR: STRING;
                        INDATA_REG: STRING;
                        INTENDED_DEVICE_FAMILY: STRING;
                        LPM_TYPE: STRING;
                        OUTDATA_ACLR_A: STRING;
                        OUTDATA_ACLR_B: STRING;
                        OUTDATA_REG_A: STRING;
                        OUTDATA_REG_B: STRING;
                        RDADDRESS_ACLR_A: STRING;
                        RDADDRESS_ACLR_B: STRING;
                        RDADDRESS_REG_A: STRING;
                        RDADDRESS_REG_B: STRING;
                        RDCONTROL_ACLR_A: STRING;
                        RDCONTROL_ACLR_B: STRING;
                        RDCONTROL_REG_A: STRING;
                        RDCONTROL_REG_B: STRING;
                        WIDTH: NATURAL;
                        WIDTHAD: NATURAL;
                        WRITE_ACLR: STRING;
                        WRITE_REG: STRING);
PORT (QA : OUT STD_LOGIC_VECTOR (31 DOWNTO 0);
                OUTCLOCK: IN STD_LOGIC ;
                QB : OUT STD_LOGIC_VECTOR (31 DOWNTO 0);
                WREN,INCLOCK : IN STD_LOGIC ;
                DATA: IN STD_LOGIC_VECTOR (31 DOWNTO 0);
                RDADDRESS_A : IN STD_LOGIC_VECTOR (4 DOWNTO 0);
                WRADDRESS: IN STD_LOGIC_VECTOR (4 DOWNTO 0);
                RDADDRESS_B : IN STD_LOGIC_VECTOR (4 DOWNTO 0)
);
```

Figure 1: VHDL code for 3 ported memory first half.

```
BEGIN
        QA<= SUB_WIRE0(31 DOWNTO 0);
        QB<= SUB_WIRE1(31 DOWNTO 0);
        ALT3PRAM_COMPONENT : ALT3PRAM
GENERIC MAP (INDATA_ACLR => "OFF",
                                INDATA_REG=> "INCLOCK",
                                INTENDED_DEVICE_FAMILY => "STRATIX II",
                                LPM_TYPE =>"ALT3PRAM",
                                OUTDATA_ACLR_A => "OFF",
                                OUTDATA_ACLR_B=> "OFF",
                                OUTDATA_REG_A => "OUTCLOCK",
                                OUTDATA_REG_B => "OUTCLOCK",
                                RDADDRESS_ACLR_A => "OFF",
                                RDADDRESS_ACLR_B => "OFF",
                                RDADDRESS_REG_A => "INCLOCK",
                                RDADDRESS_REG_B => "INCLOCK",
                                RDCONTROL_ACLR_A => "OFF",
                                RDCONTROL_ACLR_B => "OFF",
                                RDCONTROL_REG_A => "UNREGISTERED",
                                RDCONTROL_REG_B => "UNREGISTERED",
                                WIDTH=> 32,
                                WIDTHAD => 5,
                                WRITE_ACLR =>"OFF",
                                WRITE_REG =>"INCLOCK")
PORT MAP (OUTCLOCK => CLOCK,
                        WREN=> WREN,
                        INCLOCK =>CLOCK,
                        DATA => DATA,
                        RDADDRESS_A => RDADDRESS_A,
                        WRADDRESS => WRADDRESS,
                        RDADDRESS_B => RDADDRESS_B,
                        QA =>SUB_WIRE0,
                        QB => SUB_WIRE1);
END SYN;
```

Figure 2: VHDL code for 3 ported memory second half.

2.2 SIMULATION FOR RAM 3 PORT MEMORY

In this section I will be writing to one address that will be specified in simulation. I will also be reading from the same address in order to show that the 3 ported memory works. I will need to change the clock signal between 0 and 1 in order to know what the delay is for storing and I will need to have writing enabled to stored the data that will be stored at the specific register. Then I will analyze the output in order to determine if I was successful in designing a 3 ported memory.

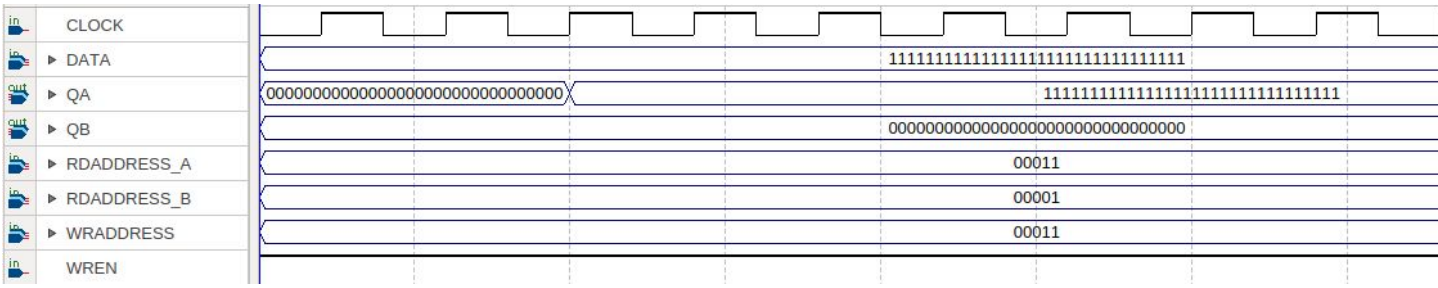

Figure 3: Vector Waveform simulation for 3 ported memory storing -1 value into 3 address and reading it when there are 3 rising edges. As we can see the expected output at the read address 3 is the same as the value that was stored when writing was enabled. The output from what is stored in 1 is 0 because we have not stored anything at that register yet. The fact that the output for address y hasn't changed means that the registers are working independent of each other, I was able to successfully design a 3 ported memory that I can read from two addresses from and write to one simultaneously. There were no errors in my design, it works as designed and as expected. The three registers could be set to the same values and it would still continue to work properly.

3. OPERATION INTERPRETATION UNIT

3.1 FUNCTIONALITY AND SPECIFICATIONS FOR OPERATION INTERPRETATION UNIT

The purpose of this unit is to function like an opcode component would. This component will implement different operations that are bitwise. It will compute bitwise and, or, xor, not, shift right, shift left, rotate right, rotate left, addition, subtraction, set less than, and multiplication. It will perform each of these operations on 32 bit words, however multiplication will only use the lower 16 bits of 2 32 bit worse in order to perform multiplication considering that the result will be a 32 bit word. Each operation will give a 32 bit word as a result after it has finished

processing. It is dependent on a clock. This will also be used as a component for the single ALU which will also implement the dot product, or immediate and branch if equal.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
ENTITY GUTI_INSTRUCTION_INTERPRETATION IS
PORT( PERFORMANCECLOCK: IN STD_LOGIC;
                OPERATION: IN STD_LOGIC_VECTOR(5 DOWNTO 0);
                WORDONE, WORDTWO: IN STD_LOGIC_VECTOR(31 DOWNTO 0);
                COMPUTATION: OUT STD_LOGIC_VECTOR(31 DOWNTO 0));
END GUTI_INSTRUCTION_INTERPRETATION;
ARCHITECTURE ARCH OF GUTI_INSTRUCTION_INTERPRETATION IS
        BEGIN
        PROCESS(PERFORMANCECLOCK,OPERATION)
                BEGIN
                        IF(PERFORMANCECLOCK = '1') THEN
                                CASE OPERATION IS
                                WHEN "000000" =>
                                        COMPUTATION <= WORDONE AND WORDTWO;
                                WHEN "000001" =>
                                        COMPUTATION <= WORDONE OR WORDTWO;
                                WHEN "000010" =>
                                        COMPUTATION <= WORDONE XOR WORDTWO;
                                WHEN "000011" =>
                                        COMPUTATION <= NOT WORDONE;
                                WHEN "000100" =>
                                        COMPUTATION <= TO_STDLOGICVECTOR(TO_BITVECTOR(WORDONE)SLL 1);
                                WHEN "000101" =>
                                        COMPUTATION <= TO_STDLOGICVECTOR(TO_BITVECTOR(WORDONE)SRL 1);
                                WHEN "000110" =>
                                        COMPUTATION <= TO_STDLOGICVECTOR(TO_BITVECTOR(WORDONE)ROL 1);
                                WHEN "000111" =>
                                        COMPUTATION <= TO_STDLOGICVECTOR(TO_BITVECTOR(WORDONE)ROR 1);
                                WHEN "001000"=> IF (WORDONE<WORDTWO) THEN COMPUTATION<=WORDONE; END IF;
                                                                IF (WORDONE>WORDTWO) THEN COMPUTATION<=WORDTWO; END IF;
                                WHEN "001001"=> COMPUTATION<=WORDONE+WORDTWO;
                                WHEN "001010"=> COMPUTATION<=WORDONE-WORDTWO;
                                WHEN "001011"=> COMPUTATION<=WORDONE(15 DOWNTO 0)*WORDTWO(15 DOWNTO 0);
                                WHEN OTHERS =>
                                NULL;
                                END CASE;
                        END IF;
                END PROCESS;
END ARCH;
```

Figure 4: VHDL code for INSTRUCTION INTERPRETATION unit.

3.2 SIMULATION FOR OPERATION INTERPRETATION UNIT

In this simulation I will give the operation code different values to show that they perform correctly on two different 32 bit words. I will be sending a signal to the clock in order for the operation to perform, I will be analyzing the result and the simulation using quartus in order to determine whether my design was successfully created.

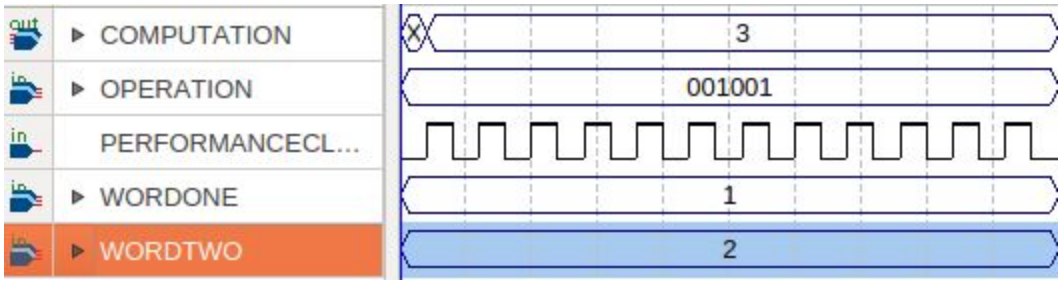

Figure 5: Vector Waveform simulation for INSTRUCTION INTERPRETATION unit. As we can see from the simulation I am giving different values to the clock. In order for it to allow the operation to be performed. I am using values 1 and 3 for the inputs in order for it to perform the operations specified by the operation code which is addition and sending the result to the 32 bit word result. We can go see that the results that we have here are the results that we expected we have no errors. I was successfully able to design the INSTRUCTION INTERPRETATION unit that can perform each of the aforementioned operations. Considering that the results from the simulation are correct according to what was expected and from the design of the component it means that I designed the INSTRUCTION INTERPRETATION unit and I can now use it as a component in the ALU unit in the next section.

4. ALU UNIT

4.1 FUNCTIONALITY AND SPECIFICATIONS FOR ALU UNIT

The purpose of this section is to design a component that will combine my 3 ported memory with my INSTRUCTION INTERPRETATION unit. Additionally I will implement dot product, or immediate and branch if equal here in this component. I will be able to specify the data that will be stored into the 32 registers that I am using in the 3 ported memory. The INSTRUCTION INTERPRETATION unit with take as its inputs for data the values read from the registers of the 3 ported memory. There will also be a 32 bit input that will be the instruction to the entire UNIT,

and a 33 bit data for storing. The first 6 bits will be the instruction, the next 15 will be 3 registers the ones to read from and to write to after an operation is performed and the data that will be stored into the write register from the instruction is stored after the operation is completed. If the instruction is or immediate or branch if equal then the last 16 bits will either be the address of the instruction to jump to after sign extension will be performed. Or the last 16 bits will be the immediate value that will be sign extended and become the address of the instruction if the values at each of the registers that are being checked are equal. There will be two different clocks as well, one for storing values to the registers using manual data and another that will be for the operation to run based on the instruction, which will also store the result for the operations that have that requirement. There will also be a signal for writing to the 3 ported memory and another signal for choosing whether or not data is going to be stored or an operation read from an instruction is going to be performed.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY GUTI_ALU IS
        PORT(STORING_CLOCK,
                  RUNNING_CLOCK,
                  STORE_SIGNAL,SAVINGEVALUATING :IN STD_LOGIC;
                REGISTER_ONE_ADDRESS,REGISTER_TWO_ADDRESS,COMPUTATION_RESULT:IN STD_LOGIC_VECTOR(4 DOWNTO 0);
        COMPLETEINSTRUCTION,RAMTEMPORARY_DATA: IN STD_LOGIC_VECTOR(31 DOWNTO 0);
                DATA_IN_REGISTER_ONE,DATA_IN_REGISTER_TWO,COMPLETERESULT:OUT STD_LOGIC_VECTOR(31 DOWNTO 0));
END GUTI_ALU;

ARCHITECTURE DESIGN OF GUTI_ALU IS

        COMPONENT GUTI_INSTRUCTION_INTERPRETATION IS
                PORT( PERFORMANCECLOCK: IN STD_LOGIC;
                OPERATION: IN STD_LOGIC_VECTOR(5 DOWNTO 0);
                WORDONE, WORDTWO: IN STD_LOGIC_VECTOR(31 DOWNTO 0);
                COMPUTATION: OUT STD_LOGIC_VECTOR(31 DOWNTO 0));
        END COMPONENT;

        COMPONENT GUTI_3_PORT_MEMORY IS
                PORT
                (       CLOCK: IN STD_LOGIC ;
                        DATA: IN STD_LOGIC_VECTOR (31 DOWNTO 0);
                        RDADDRESS_A: IN STD_LOGIC_VECTOR (4 DOWNTO 0);
                        RDADDRESS_B: IN STD_LOGIC_VECTOR (4 DOWNTO 0);
                        WRADDRESS: IN STD_LOGIC_VECTOR (4 DOWNTO 0);
                        WREN: IN STD_LOGIC := '1';
                        QA: OUT STD_LOGIC_VECTOR (31 DOWNTO 0);
                        QB: OUT STD_LOGIC_VECTOR (31 DOWNTO 0));
        END COMPONENT;

        SIGNAL  TEMPORARY_DATA,FINAL_OUTPUT,
                                TEMPORARY_FINAL_OUTPUT,
                                TEMPORARY_A,TEMPORARY_B,
                                FIRST_TEMPORARY_A,
                                FIRST_TEMPORARY_B,
                                EXTENDEDVALUE,
                                DOT_TEMP,
                                DOT_COMPLETE,
                                BRANCH:STD_LOGIC_VECTOR(31 DOWNTO 0);
        SIGNAL OPCODE,TEMPORARY_OPCODE : STD_LOGIC_VECTOR(5 DOWNTO 0);
        SIGNAL TEMPORARY_REGISTER1,
                        TEMPORARY_REGISTER2,
                        TEMPORARY_WRITE,
                        HOLD_A,
                        HOLDB,
                        HOLDWRITE,
                        WRITE_HOLD: STD_LOGIC_VECTOR(4 DOWNTO 0);
        SIGNAL IMMEDIATE16: STD_LOGIC_VECTOR(15 DOWNTO 0);
        SIGNAL SIGNEXTENSION,HOLD_CLOCK: STD_LOGIC;
        SIGNAL ARRAY1SIZE,ARRAY2SIZE:STD_LOGIC_VECTOR(3 DOWNTO 0);

        BEGIN
                ARRAY1SIZE<=COMPLETEINSTRUCTION(6 DOWNTO 3);
                ARRAY2SIZE<=COMPLETEINSTRUCTION(10 DOWNTO 7);
                BRANCH<=EXTENDEDVALUE;
```

Figure 6: VHDL code for ALU.

```
OPCODE <= COMPLETEINSTRUCTION(31 DOWNTO 26);
HOLD_A <= REGISTER_ONE_ADDRESS WHEN SAVINGEVALUATING='0' ELSE TEMPORARY_REGISTER1;
HOLDB <= REGISTER_TWO_ADDRESS WHEN SAVINGEVALUATING='0' ELSE TEMPORARY_REGISTER2;
HOLDWRITE <= COMPUTATION_RESULT WHEN SAVINGEVALUATING='0' ELSE TEMPORARY_WRITE;
IMMEDIATE16 <= COMPLETEINSTRUCTION(15 DOWNTO 0);
EXTENDEDVALUE(15 DOWNTO 0) <= IMMEDIATE16;
EXTENDEDVALUE(31 DOWNTO 16) <= (OTHERS => IMMEDIATE16(15));
DOT_TEMP<=EXTENDEDVALUE;

PROCESS(OPCODE) BEGIN
        CASE OPCODE IS
                WHEN "001100" => SIGNEXTENSION<='1';
                WHEN "001110" => BRANCH<=EXTENDEDVALUE;
                WHEN "001101" =>ARRAY1SIZE<=COMPLETEINSTRUCTION(6 DOWNTO 3);
                                                ARRAY2SIZE<=COMPLETEINSTRUCTION(10 DOWNTO 7);
                                                DOT_COMPLETE<=DOT_TEMP;
                WHEN OTHERS=> SIGNEXTENSION<='0';
        END CASE;
END PROCESS;

TEMPORARY_REGISTER1 <= COMPLETEINSTRUCTION(25 DOWNTO 21);
TEMPORARY_REGISTER2 <= COMPLETEINSTRUCTION(20 DOWNTO 16);
TEMPORARY_WRITE <= COMPLETEINSTRUCTION(15 DOWNTO 11);

HOLD_CLOCK <= STORING_CLOCK;
TEMPORARY_DATA <= RAMTEMPORARY_DATA WHEN SAVINGEVALUATING = '0' ELSE FINAL_OUTPUT;


WRITE_HOLD<=HOLDWRITE WHEN SIGNEXTENSION='0' ELSE HOLDB;
SAVINGTO3PORTEDMEMORY:GUTI_3_PORT_MEMORY PORT MAP(HOLD_CLOCK,
                                                  TEMPORARY_DATA,
                                                  HOLD_A, HOLDB, WRITE_HOLD,
                                                  STORE_SIGNAL,
                                                  TEMPORARY_A, TEMPORARY_B);

FIRST_TEMPORARY_A <= TEMPORARY_A;
FIRST_TEMPORARY_B <= TEMPORARY_B WHEN SIGNEXTENSION='0' ELSE EXTENDEDVALUE;

TEMPORARY_OPCODE <= OPCODE WHEN SIGNEXTENSION='0' ELSE "000001";

INTERPRETINGTHEINSTRUCTION : GUTI_INSTRUCTION_INTERPRETATION PORT MAP(RUNNING_CLOCK,
                                                                      TEMPORARY_OPCODE,
                                                                      FIRST_TEMPORARY_A,
                                                                      FIRST_TEMPORARY_B,
                                                                      TEMPORARY_FINAL_OUTPUT);
FINAL_OUTPUT <= TEMPORARY_FINAL_OUTPUT;
DATA_IN_REGISTER_ONE <= TEMPORARY_A;
DATA_IN_REGISTER_TWO <= TEMPORARY_B;
COMPLETERESULT <=TEMPORARY_FINAL_OUTPUT;
END DESIGN;
```

Figure 7: VHDL code for ALU continued.

4.2 SIMULATION FOR ALU UNIT

In this simulation I will specify one operation that will be performed after I have stored two values into registers in the 3 ported memory using the ALU. Then I will assure that the result was stored into the register by reading from it. This way I will then analyze and determine the success I have made in designing the ALU.

I will also need to specify the 32 bit instruction and data in this simulation, I will only use one example for simplicity, if it works it will work for each of the other specifications but I will not display those because there are too many.

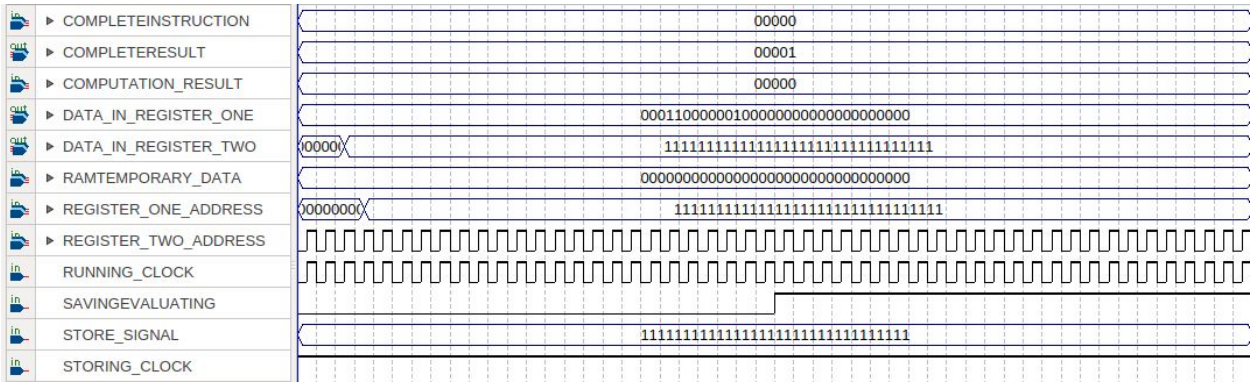

Figure 8: Vector Waveform simulation of the ALU showing that I am storing -1 value in register 0 and reading from register 1 and 0. I also stored data x and I am setting the instruction to use those registers and perform the or operation. I alternate between storing values and performing an operation. It does it correctly because the expected result is 0 which is correct and that means I was able to successfully design an ALU unit that can use 3 ported memory and INSTRUCTION INTERPRETATION unit to operate using registers. This is roughly a single CYCLE CPU the missing component to it is that it should use SSRAM to read it's instruction like most compilers do in a real world scenario.

5. SINGLE CYCLE CPU

5.1 FUNCTIONALITY AND SPECIFICATIONS FOR SINGLE CYCLE CPU

The purpose of this component is to connect the ALU to the SSRAM. Once the two are connected, I can use the switches from the board in order to specify the address I will store the instruction in. The actual instruction that will be stored will be stored into the SSRAM. I will also specify the 32 bit data that will be stored into the registers on the 3 port memory. I will also specify the register that I am storing to on the 3 port memory and the program counter will increment by 4 and be responsible for changing the address of the instruction that will be

performed. I will need to have several clock signals and 5 bit wide selector in order to specify each of the values that I will need to specify independently of the other information. I will need to send a writing signal for the 3 ported memory, for the SSRAM. Everything will be displayed on the 7 segment display using hexadecimal format in order to be as efficient as possible. I will not test this in simulation, but i will demonstrate it on the DE2-115 board because it isn't possible to run this on simulation considering that I can't store values to SRAM unless its onto the board.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY GUTI_SINGLE_CYCLE_CPU IS
    PORT(
        UPPER_HALF,INPUT_WE,CLOCKFORSAVING,CLOCKFORRUNNING,RAMWRITE,PERFORMENABLE,ZORR,INSTRUCTIONSTORECLOCK,CLOCKFORREGISTGERS:STD_LOGIC;
                VALUES: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
                S: IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        SRAM_ADDR: OUT STD_LOGIC_VECTOR(19 DOWNTO 0);
        SRAM_DQ: INOUT STD_LOGIC_VECTOR(15 DOWNTO 0);
        SRAM_CE_N,SRAM_OE_N,SRAM_WE_N,SRAM_UB_N,SRAM_LB_N: OUT STD_LOGIC;
                FIRSTHEX,SECONDHEX,THIRDHEX,FOURTHHEX,FIFTHHEX,SIXHEX,SEVENHEX,EIGHTHEX: OUT STD_LOGIC_VECTOR(6 DOWNTO 0));
END GUTI_SINGLE_CYCLE_CPU;
ARCHITECTURE ARCH OF GUTI_SINGLE_CYCLE_CPU IS
    SIGNAL INPUT_TEMP: STD_LOGIC_VECTOR(7 DOWNTO 0);
    SIGNAL WENABLE: STD_LOGIC;
    SIGNAL LAST4DISPLAY,DISPLAY,FIRST4DISPLAY: STD_LOGIC_VECTOR(15 DOWNTO 0);
         SIGNAL DATASTOREDFORRAM,INSTRUCTIONCODE,DQA,DQB,RESULT:STD_LOGIC_VECTOR(31 DOWNTO 0);
         SIGNAL DATA:STD_LOGIC_VECTOR(15 DOWNTO 0);
         SIGNAL LOWERHALFADDRESS,UPPERHALFADDRESS,UPPERHALFTEMP1,LOWERHALFTEMP1: STD_LOGIC_VECTOR (19 DOWNTO 0);
         SIGNAL PROGRAMCOUNTER: STD_LOGIC_VECTOR(19 DOWNTO 0) :="00000000000000000000";
         SIGNAL R1HOLD,R2HOLD,R3HOLD,ADD1TEMP,ADD2TEMP,ADDWTEMP:STD_LOGIC_VECTOR(4 DOWNTO 0);

COMPONENT GUTI_HEX IS
        PORT(
                BINARY : IN STD_LOGIC_VECTOR(3 DOWNTO 0) := X"0";
                HEXADECIMAL : OUT STD_LOGIC_VECTOR(6 DOWNTO 0) := "0000000"
        );
END COMPONENT;

COMPONENT GUTI_ALU IS
        PORT(STORING_CLOCK,
                  RUNNING_CLOCK,
                  STORE_SIGNAL,SAVINGEVALUATING :IN STD_LOGIC;
                REGISTER_ONE_ADDRESS,REGISTER_TWO_ADDRESS,COMPUTATION_RESULT:IN STD_LOGIC_VECTOR(4 DOWNTO 0);
        COMPLETEINSTRUCTION,RAMTEMPORARY_DATA: IN STD_LOGIC_VECTOR(31 DOWNTO 0);
                DATA_IN_REGISTER_ONE,DATA_IN_REGISTER_TWO,COMPLETERESULT:OUT STD_LOGIC_VECTOR(31 DOWNTO 0));
END COMPONENT;
BEGIN
         WENABLE<=INPUT_WE;
    SRAM_WE_N <= NOT WENABLE;
    SRAM_CE_N <= '0';
    SRAM_OE_N <= '0';
    SRAM_UB_N <= '0';
    SRAM_LB_N <= '0';
         LAST4DISPLAY<=DISPLAY WHEN ZORR='0' ELSE RESULT(15 DOWNTO 0);

        UPPERHALFADDRESS<=(LOWERHALFADDRESS+"00000000000000000001");
        PROCESS(INSTRUCTIONSTORECLOCK)
        BEGIN
        IF (INSTRUCTIONSTORECLOCK='0') THEN
        IF (UPPER_HALF='0')THEN
        UPPERHALFTEMP1<= LOWERHALFADDRESS;
        ELSE
        LOWERHALFTEMP1<=UPPERHALFADDRESS;
```

Figure 9: VHDL code for SINGLE CYCLE CPU.

```
        END IF;
        END IF;
        END PROCESS;
        PROCESS(CLOCKFORRUNNING)
        BEGIN
        IF(VALUES(1)='0') THEN
        FIRST4DISPLAY<=PROGRAMCOUNTER(15 DOWNTO 0)+"0000000000000100";
        ELSE
        FIRST4DISPLAY<=PROGRAMCOUNTER(15 DOWNTO 0);
        END IF;

        END PROCESS;

        SRAM_ADDR<=UPPERHALFTEMP1 WHEN UPPER_HALF='0' ELSE LOWERHALFTEMP1;
        SRAM_DQ<= DATA WHEN WENABLE='1' ELSE (OTHERS=>'Z');
        DISPLAY(15 DOWNTO 0)<="0000000000000000"WHEN VALUES(0)='0' ELSE RESULT(15 DOWNTO 0);
        INSTRUCTIONCODE(15 DOWNTO 0)<=SRAM_DQ WHEN UPPER_HALF='0' ELSE DATA(15 DOWNTO 0);
        INSTRUCTIONCODE(31 DOWNTO 16)<=SRAM_DQ WHEN UPPER_HALF='1' ELSE DATA(15 DOWNTO 0);
RUNNING:GUTI_ALU PORT MAP (NOT CLOCKFORSAVING,
                           NOT CLOCKFORRUNNING,
                           RAMWRITE,
                           PERFORMENABLE,
                           ADD1TEMP,
                           ADD2TEMP,
                                                                ADDWTEMP,
                                                                INSTRUCTIONCODE,
                           DATASTOREDFORRAM,
                           DQA,
                           DQB,
                           RESULT
        );

        PROCESS(S)
BEGIN
CASE S IS
      WHEN"0000"=>LOWERHALFADDRESS(7 DOWNTO 0)<=VALUES+"00000100";
      WHEN"0001"=>LOWERHALFADDRESS(15 DOWNTO 8)<=VALUES;
      WHEN"0010"=>LOWERHALFADDRESS(19 DOWNTO 16)<=VALUES(3 DOWNTO 0);

      WHEN"0011"=>DATA(7 DOWNTO 0)<=VALUES;
      WHEN"0100"=>DATA(15 DOWNTO 8)<=VALUES;

      WHEN"0101"=>R1HOLD<=VALUES(4 DOWNTO 0);
      WHEN"0110"=>R2HOLD<=VALUES(4 DOWNTO 0);
      WHEN"0111"=>R3HOLD<=VALUES(4 DOWNTO 0);

      WHEN"1000"=>DATASTOREDFORRAM(7 DOWNTO 0)<=VALUES;
      WHEN"1001"=>DATASTOREDFORRAM(15 DOWNTO 8)<=VALUES;
      WHEN"1010"=>DATASTOREDFORRAM(23 DOWNTO 16)<=VALUES;
      WHEN"1011"=>DATASTOREDFORRAM(31 DOWNTO 24)<=VALUES;
      WHEN OTHERS=>NULL;
    END CASE;
END PROCESS;
```

Figure 10: VHDL code for SINGLE CYCLE CPU CONT.

```
D0: GUTI_HEX PORT MAP (LAST4DISPLAY(3 DOWNTO 0), FIRSTHEX);
D1: GUTI_HEX PORT MAP (LAST4DISPLAY(7 DOWNTO 4), SECONDHEX);
D2: GUTI_HEX PORT MAP (LAST4DISPLAY(11 DOWNTO 8), THIRDHEX);
D3: GUTI_HEX PORT MAP (LAST4DISPLAY(15 DOWNTO 12), FOURTHHEX);
D4: GUTI_HEX PORT MAP (FIRST4DISPLAY(3 DOWNTO 0), FIFTHHEX);
D5: GUTI_HEX PORT MAP (FIRST4DISPLAY(7 DOWNTO 4), SIXHEX);
D6: GUTI_HEX PORT MAP (FIRST4DISPLAY(11 DOWNTO 8), SEVENHEX);
D7: GUTI_HEX PORT MAP (FIRST4DISPLAY(15 DOWNTO 12), EIGHTHEX);

END ARCH;
```

Figure 11: VHDL code for SINGLE CYCLE CPU CONT 2.

5.2 DEMONSTRATION OF SINGLE CYCLE CPU ON DE2-115 BOARD

In this section, I will give an example of me storing values to an address in the SSRAM, into the register and performing the operation. The video that is accompanied with this project will show more operations that are being used and make it clearer to understand how the SINGLE CYCLE CPU.

The pins that are assigned to the DE2-115 board are:

To,Location

VALUES[0],PIN_AB28

VALUES[1],PIN_AC28

VALUES[2],PIN_AC27

VALUES[3],PIN_AD27

VALUES[4],PIN_AB27

VALUES[5],PIN_AC26

VALUES[6],PIN_AD26

VALUES[7],PIN_AB26

UPPER_HALF,PIN_Y24

S[0],PIN_AC25

S[1],PIN_AB25

S[2],PIN_AC24

S[3],PIN_AB24

ZORR,PIN_AB23

CLOCKFORSAVING,PIN_M23

CLOCKFORRUNNING,PIN_M21

RAMWRITE,PIN_AA23

PERFORMENABLE,PIN_AA22

INPUT_WE,PIN_Y23

INSTRUCTIONSTORECLOCK,PIN_N21

CLOCKFORREGISTGERS,PIN_R24

SRAM_ADDR[0],PIN_AB7

SRAM_ADDR[1],PIN_AD7

SRAM_ADDR[2],PIN_AE7

SRAM_ADDR[3],PIN_AC7

SRAM_ADDR[4],PIN_AB6

SRAM_ADDR[5],PIN_AE6

SRAM_ADDR[6],PIN_AB5

SRAM_ADDR[7],PIN_AC5

SRAM_ADDR[8],PIN_AF5

SRAM_ADDR[9],PIN_T7

SRAM_ADDR[10],PIN_AF2

SRAM_ADDR[11],PIN_AD3

SRAM_ADDR[12],PIN_AB4

SRAM_ADDR[13],PIN_AC3

SRAM_ADDR[14],PIN_AA4

SRAM_ADDR[15],PIN_AB11

SRAM_ADDR[16],PIN_AC11

SRAM_ADDR[17],PIN_AB9

SRAM_ADDR[18],PIN_AB8

SRAM_ADDR[19],PIN_T8

SRAM_CE_N, PIN_AF8

SRAM_OE_N, PIN_AD5

SRAM_WE_N, PIN_AE8

SRAM_UB_N, PIN_AC4

SRAM_LB_N, PIN_AD4

SRAM_DQ[0],PIN_AH3

SRAM_DQ[1],PIN_AF4

SRAM_DQ[2],PIN_AG4

SRAM_DQ[3],PIN_AH4

SRAM_DQ[4],PIN_AF6

SRAM_DQ[5],PIN_AG6

SRAM_DQ[6],PIN_AH6

SRAM_DQ[7],PIN_AF7

SRAM_DQ[8],PIN_AD1

SRAM_DQ[9],PIN_AD2

SRAM_DQ[10],PIN_AE2

SRAM_DQ[11],PIN_AE1

SRAM_DQ[12],PIN_AE3

SRAM_DQ[13],PIN_AE4

SRAM_DQ[14],PIN_AF3

SRAM_DQ[15],PIN_AG3

FIRSTHEX[0],PIN_G18

FIRSTHEX[1],PIN_F22

FIRSTHEX[2],PIN_E17

FIRSTHEX[3],PIN_L26

FIRSTHEX[4],PIN_L25

FIRSTHEX[5],PIN_J22

FIRSTHEX[6],PIN_H22

SECONDHEX[0],PIN_M24

SECONDHEX[1],PIN_Y22

SECONDHEX[2],PIN_W21

SECONDHEX[3],PIN_W22

SECONDHEX[4],PIN_W25

SECONDHEX[5],PIN_U23

SECONDHEX[6],PIN_U24

THIRDHEX[0],PIN_AA25

THIRDHEX[1],PIN_AA26

THIRDHEX[2],PIN_Y25

THIRDHEX[3],PIN_W26

THIRDHEX[4],PIN_Y26

THIRDHEX[5],PIN_W27

THIRDHEX[6],PIN_W28

FOURTHHEX[0],PIN_V21

FOURTHHEX[1],PIN_U21

FOURTHHEX[2],PIN_AB20

FOURTHHEX[3],PIN_AA21

FOURTHHEX[4],PIN_AD24

FOURTHHEX[5],PIN_AF23

FOURTHHEX[6],PIN_Y19

FIFTHHEX[0],PIN_AB19

FIFTHHEX[1],PIN_AA19

FIFTHHEX[2],PIN_AG21

FIFTHHEX[3],PIN_AH21

FIFTHHEX[4],PIN_AE19

FIFTHHEX[5],PIN_AF19

FIFTHHEX[6],PIN_AE18

SIXHEX[0],PIN_AD18

SIXHEX[1],PIN_AC18

SIXHEX[2],PIN_AB18

SIXHEX[3],PIN_AH19

SIXHEX[4],PIN_AG19

SIXHEX[5],PIN_AF18

SIXHEX[6],PIN_AH18

SEVENHEX[0],PIN_AA17

SEVENHEX[1],PIN_AB16

SEVENHEX[2],PIN_AA16

SEVENHEX[3],PIN_AB17

SEVENHEX[4],PIN_AB15

SEVENHEX[5],PIN_AA15

SEVENHEX[6],PIN_AC17

EIGHTHEX[0],PIN_AD17

EIGHTHEX[1],PIN_AE17

EIGHTHEX[2],PIN_AG17

EIGHTHEX[3],PIN_AH17

EIGHTHEX[4],PIN_AF17

EIGHTHEX[5],PIN_AG18

EIGHTHEX[6],PIN_AA14

The demonstration is on the video.

6. Conclusion

As it turns out the hardest part in this project is the final one. It is not difficult it just takes a lot of time to perform any one instruction considering that I am using so many bits of information. In order to do one single operation, I would need to specify 2 SRAM address and 32 bits of data to store, 16 bits in each, this will be the instruction. I will need to specify at least 2 registers to store the data to them, that would be 32 bits of data twice in order have at least 2 values that can be

used from the instruction in the INSTRUCTION INTERPRETER unit. I also have to take into account the delay that exists in each of the clocks for storing values and for performing the specified operation based on the 32 bit instruction. The dot product was what made me have to think more in order to make the design because it takes much more time depending on how many elements are going to be specified for each array. It might have been easier to use only SSRAM data instead of registers because that way I could simply store data into higher addresses and instructions into lower addresses. That would have made me use one less component but I was still able to design the SINGLE CYCLE CPU during this course and better learn how compilers and computers are organized.