CPU- Architecture

LAB 5- Single cycle MIPS CPU

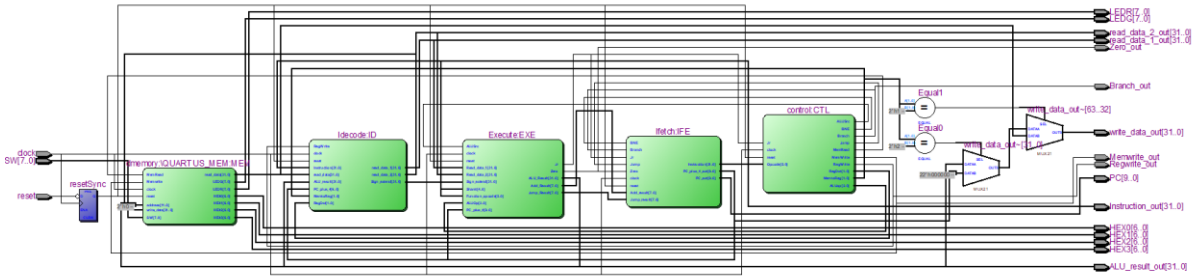
Presenting:

Elad Sofer 312124662

Tomer Shaked 315822221

1. RTL Design

The Top Design (MIPS):



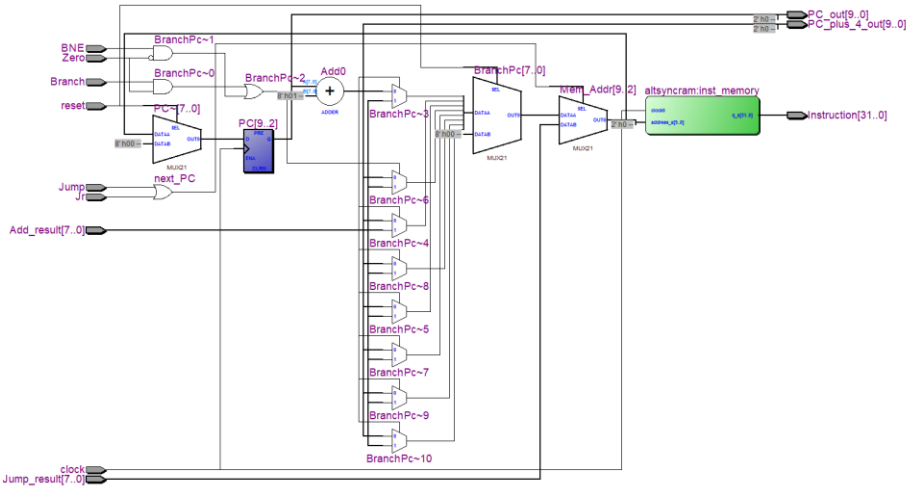
The Mips block serves as the top architecture of the system. It connects all the blocks together and passes the inputs outputs(IO) out of the design.

```
ENTITY MIPS IS
    GENERIC (BUS_W : INTEGER := 12; ADD_BUS: INTEGER :=10; QUARTUS : INTEGER := 1); -- QUARTUS MODE = 12; 10 | MODELSIM = 8; 8
        --GENERIC (BUS_W : INTEGER := 8; ADD_BUS: INTEGER :=8; QUARTUS : INTEGER := 0); -- QUARTUS MODE = 12; 10 | MODELSIM = 8; 8
    PORT( reset, clock                  : IN    STD_LOGIC;
        -- Output important signals to pins for easy display in Simulator
        PC                              : OUT STD_LOGIC_VECTOR( 9 DOWNTO 0 );
        ALU_result_out, read_data_1_out : OUT STD_LOGIC_VECTOR( 31 DOWNTO 0 );
        read_data_2_out, write_data_out : OUT STD_LOGIC_VECTOR( 31 DOWNTO 0 );
        Instruction_out                 : OUT STD_LOGIC_VECTOR( 31 DOWNTO 0 );
        Branch_out, Zero_out            : OUT STD_LOGIC;
        Memwrite_out,Regwrite_out       : OUT STD_LOGIC;
        LEDG, LEDR                      : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
        HEX0, HEX1, HEX2, HEX3          : OUT STD_LOGIC_VECTOR (6 DOWNTO 0));
        SW                              : IN  STD_LOGIC_VECTOR (7 DOWNTO 0);
END     MIPS;
```

The Fetch Design:

The fetch block is responsible for the Program counter and the instruction memory. It is the block that pulls a new instruction in the start of a new clock cycle. This is also the block that takes care of all the different jump commands, and the jumping part of the branch commands.

RTL viewer:



Ports:

```
ENTITY Ifetch IS
    GENERIC(BUS_W : INTEGER := 12; ADD_BUS: INTEGER :=10; QUARTUS : INTEGER := 1);
    PORT(   SIGNAL Instruction      : OUT   STD_LOGIC_VECTOR( 31 DOWNTO 0 );
            SIGNAL PC_plus_4_out    : OUT   STD_LOGIC_VECTOR( 9 DOWNTO 0 );
            SIGNAL Add_result       : IN    STD_LOGIC_VECTOR( 7 DOWNTO 0 );
            SIGNAL Jump_result      : IN    STD_LOGIC_VECTOR( 7 DOWNTO 0 );
            SIGNAL Branch           : IN    STD_LOGIC;
            SIGNAL BNE              : IN    STD_LOGIC;
            SIGNAL Jump             : IN    STD_LOGIC;
            SIGNAL Jr               : IN    STD_LOGIC;
            SIGNAL Zero             : IN    STD_LOGIC;
            SIGNAL PC_out           : OUT   STD_LOGIC_VECTOR( 9 DOWNTO 0 );
            SIGNAL clock, reset     : IN    STD_LOGIC);
END Ifetch;
```

The design has 1726 registers and 3146 combinational units.

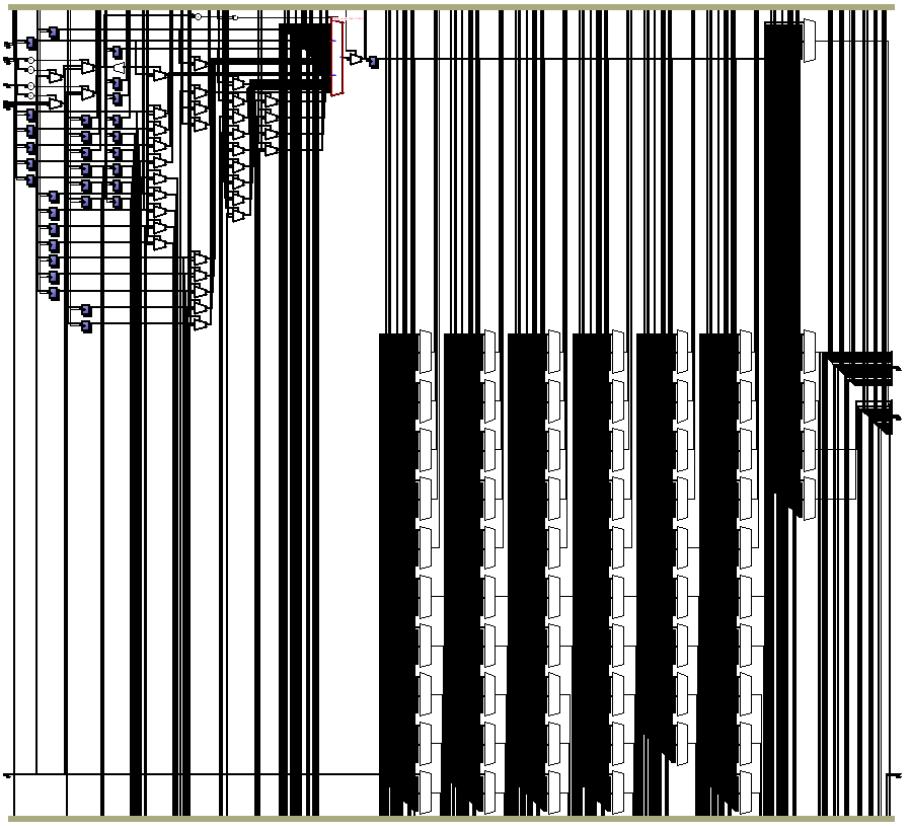
Analysis & Synthesis Summary

Analysis & Synthesis Status	Successful - Wed Jun 23 21:24:41 2021
Quartus II 32-bit Version	12.1 Build 177 11/07/2012 SJ Web Edition
Revision Name	Lab5Quart
Top-level Entity Name	MIPS
Family	Cyclone II
Total logic elements	4,524
Total combinational functions	3,146
Dedicated logic registers	1,726
Total registers	1726
Total pins	228
Total virtual pins	0
Total memory bits	159,744
Embedded Multiplier 9-bit elements	6
Total PLLs	0

The Decode Design:

The Decode block is responsible for the register file. It takes the already received instruction and according to it reads and writes to the necessary registers.

RTL viewer:



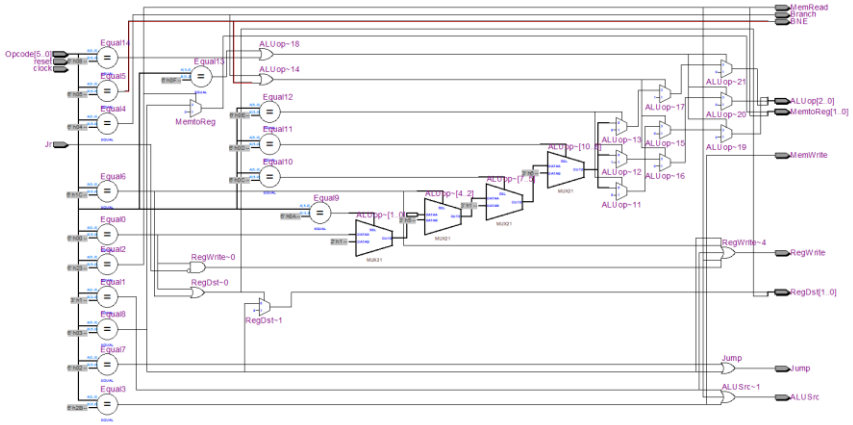
Ports:

```
ENTITY Idecode IS
      PORT( read_data_1 : OUT    STD_LOGIC_VECTOR( 31 DOWNTO 0 );
            read_data_2 : OUT    STD_LOGIC_VECTOR( 31 DOWNTO 0 );
            Instruction : IN     STD_LOGIC_VECTOR( 31 DOWNTO 0 );
            read_data   : IN     STD_LOGIC_VECTOR( 31 DOWNTO 0 );
            ALU_result  : IN     STD_LOGIC_VECTOR( 31 DOWNTO 0 );
            PC_plus_4   : IN     STD_LOGIC_VECTOR( 9 DOWNTO 0 );
            RegWrite    : IN     STD_LOGIC;
            MemtoReg    : IN     STD_LOGIC_VECTOR(1 DOWNTO 0);
            RegDst      : IN     STD_LOGIC_VECTOR(1 DOWNTO 0);
            Sign_extend : OUT    STD_LOGIC_VECTOR( 31 DOWNTO 0 );
            clock,reset : IN     STD_LOGIC );
END Idecode;
```

The Control Design:

The Control Receives the opcode of the command and sends out control signals used in the rest of the design.

RTL viewer:



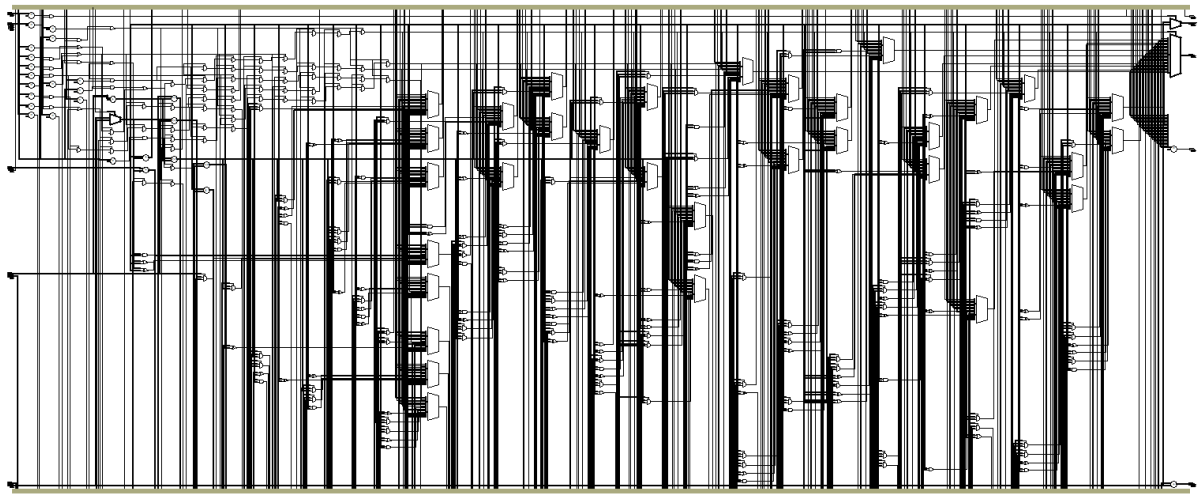
Ports:

```
ENTITY control IS
   PORT(
    Opcode       : IN    STD_LOGIC_VECTOR( 5 DOWNTO 0 );
    RegDst       : OUT   STD_LOGIC_VECTOR( 1 DOWNTO 0 );
    ALUSrc       : OUT   STD_LOGIC;
    MemtoReg     : OUT   STD_LOGIC_VECTOR( 1 DOWNTO 0 );
    RegWrite     : OUT   STD_LOGIC;
    MemRead      : OUT   STD_LOGIC;
    MemWrite     : OUT   STD_LOGIC;
    Branch       : OUT   STD_LOGIC;
    BNE          : OUT   STD_LOGIC;
    Jump         : OUT   STD_LOGIC;
    Jr           : IN    STD_LOGIC;
    ALUop        : OUT   STD_LOGIC_VECTOR( 2 DOWNTO 0 );
    clock        : IN    STD_LOGIC;
    reset        : IN    STD_LOGIC );

END control;
```

The Execute Design:

The Execute block is Responsible for most of the calculations done in the mips cpu. The Execute Receives two inputs which could be registers or immediate values a command from the control unit and a function field from the instruction and chooses which action to perform on the data.
RTL viewer:



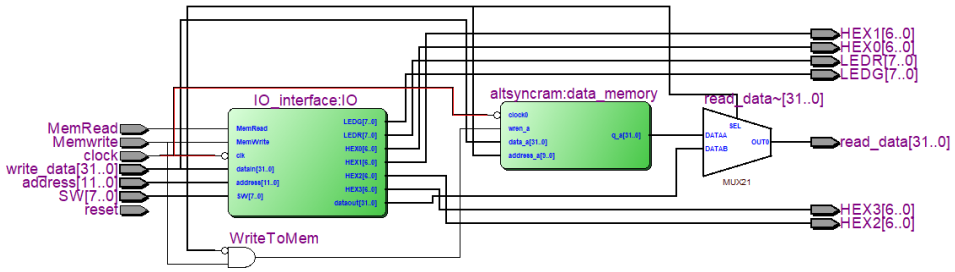
Ports:

```
ENTITY  Execute IS
    PORT(   Read_data_1      : IN    STD_LOGIC_VECTOR( 31 DOWNTO 0 );
            Read_data_2      : IN    STD_LOGIC_VECTOR( 31 DOWNTO 0 );
            Sign_extend      : IN    STD_LOGIC_VECTOR( 31 DOWNTO 0 );
            Shamt            : IN    STD_LOGIC_VECTOR( 4 DOWNTO 0 );
            Function_opcode  : IN    STD_LOGIC_VECTOR( 5 DOWNTO 0 );
            ALUOp            : IN    STD_LOGIC_VECTOR( 2 DOWNTO 0 );
            ALUSrc           : IN    STD_LOGIC;
            Jr               : OUT   STD_LOGIC;
            Zero             : OUT   STD_LOGIC;
            ALU_Result       : OUT   STD_LOGIC_VECTOR( 31 DOWNTO 0 );
            Add_Result       : OUT   STD_LOGIC_VECTOR( 7 DOWNTO 0 );
            Jump_Result      : OUT   STD_LOGIC_VECTOR( 7 DOWNTO 0 );
            PC_plus_4        : IN    STD_LOGIC_VECTOR( 9 DOWNTO 0 );
            clock, reset     : IN    STD_LOGIC );
END Execute;
```

The Memory Design:

The memory block is responsible for the memory… but it is also responsible for all the I/O device signals. It receives the address and IN/OUT Data and connects the data to the correct Mem/IO device.

RTL viewer:



Ports:

```
ENTITY dmemory IS
    GENERIC (BUS_W : INTEGER := 8; ADD_BUS: INTEGER :=8); -- QUARTUS MODE = 12; 10 | MODELSIM = 8; 8

    PORT(   read_data               : OUT    STD_LOGIC_VECTOR( 31 DOWNTO 0 );
            address                 : IN     STD_LOGIC_VECTOR( BUS_W-1 DOWNTO 0 );
            write_data              : IN     STD_LOGIC_VECTOR( 31 DOWNTO 0 );
            MemRead, Memwrite       : IN     STD_LOGIC;
            clock,reset             : IN     STD_LOGIC;
            SW                      : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
            LEDG, LEDR              : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
            HEX0, HEX1, HEX2, HEX3 : OUT STD_LOGIC_VECTOR (6 DOWNTO 0));
END dmemory;
```

2. Critical Path and timing analysis

The critical path is the path that passes through a command fetch, pulls out registers performs a shift operation and stores the data in an io element. This is practically a SW command.

To improve the critical path we can either reduce the number of elements in the decode sequence (by either having less registers or creating a more efficient pulling method). Another way to improve the critical path would be to turn the mips cpu into a multi-cycle cpu. In doing so we can cut the critical path into sections shortening the critical path. The maximal clock frequency is 22.27MHz

Slow Model Fmax Summary

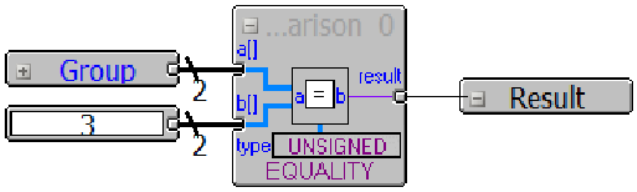
	Fmax	Restricted Fmax	Clock Name	Note
1	22.27 MHz	22.27 MHz	clock	

3. Signal Tap analysis

in order to prove the code is working we will try to catch one of the branches in the delay function:

```
65
66  delay:  move $t6,$zero
67  L:      addi $t6,$t6,1
68          slt  $t5,$t6,$t8
69          beq  $t5,$zero,back
70          j    L
71  back: jr $ra
```

Though when attempted to capture the result we found that we cant capture a jump state in the branch command since in 99% of the cases the branch does not jump(the delay counts down 12M). we needed the signal tap to trigger only when both the branch out and the zero out where equal 1. To capture that case we grouped both the zero and the branch command together into one signal and created an advance trigger to capture only when both of them are active(value=3 which is in binary 11):



And this is the result we got from the signal tap: