EASTERN MEDITERRANEAN UNIVERSITY
COMPUTER ENGINEERING DEPARTMENT

CMPE 324 - Computer Architecture and Organization

Lab 6: Single Clock Data Path for 16-bit R-type Instructions in ALTERA QUARTUS VHDL Environment

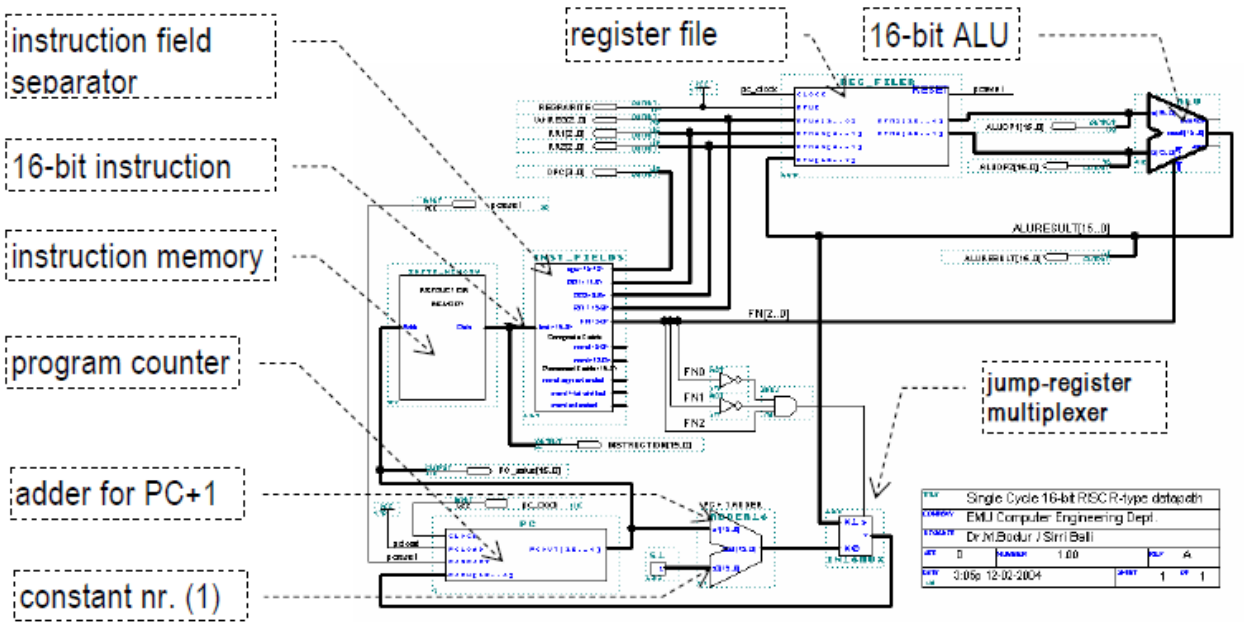
1. Objective:

To familiarize with the Single-Clock VHDL implementation for a set of 16-bit R-type instructions, and to measure the response time of several building components in a RISC datapath.

1. Introduction

You have already worked to construct a simple 16-bit instruction set for your project/homework. A typical simple 16-bit instruction set and a corresponding R-type datapath is provided in this experiment. You will be asked to take several measurements that can lead you to a conclusion about the typical speed constraints, and possible improvements of the similar circuits.

In this experiment, we will focus only on the R-type instructions, so that we can observe typical properties of some of the basic building blocks such as the Program-Counter, the Instruction-Memory, the Register-File, and the Adder for updating the Program-Counter.



The complete set of the 16-bit instructions are summarized in the following tables.

Opcode	Read1 reg	Read2 reg	Write reg	Function code
4-bits	3-bits	3-bits	3-bits	3-bits

Table 1-2 Function Codes for the R-type Instructions

FNCODE	.00	.01	.11	.10
0..	And	Or	Xor	Add
1..	Jr	ShrA	Slt	Sub

Table 1-3 Representation of R-type Instructions with opcodes

instruction	Opcode	Read-reg1	Read-reg2	Write-reg	Fn code
And	0000	XXX	XXX	XXX	000
Or	0000	XXX	XXX	XXX	001
Add	0000	XXX	XXX	XXX	010
Xor	0000	XXX	XXX	XXX	011
Jump-Register	0000	XXX	000	XXX	100
Shift-right-Arith	0000	XXX	XXX	XXX	101
Subtraction	0000	XXX	XXX	XXX	110
Set less than	0000	XXX	XXX	XXX	111

2. QUARTUS and VHDL Code

In this section you should learn how to implement a VHDL code for single cycle data path. Before we go any further, let's define some of the terms that we use throughout the book.

Entity: All designs are expressed in terms of entities. An entity is the most basic building block in a design. The uppermost level of the design is the top-level entity. If the design is hierarchical, then the top-level description will have lower-level descriptions contained in it. These lower-level descriptions will be lower-level entities contained in the top-level entity description.

Architecture: All entities that can be simulated have an architecture description. The architecture describes the behavior of the entity. A single entity can have multiple architectures. One architecture might be behavioral while another might be a structural description of the design.

Configuration: A configuration statement is used to bind a component instance to an entity-architecture pair. A configuration can be considered like a parts list for a

design. It describes which behavior to use for each entity, much like a parts list describes which part to use for each part in the design.

Package: A package is a collection of commonly used data types and subprograms used in a design. Think of a package as a toolbox that contains tools used to build designs.

Driver: This is a source on a signal. If a signal is driven by two sources, then when both sources are active, the signal will have two drivers.

Bus: The term "bus" usually brings to mind a group of signals or a particular method of communication used in the design of hardware. In VHDL, a bus is a special kind of signal that may have its drivers turned off.

Attribute: An attribute is data that are attached to VHDL objects or predefined data about VHDL objects. Examples are the current drive capability of a buffer or the maximum operating temperature of the device.

Generic: A generic is VHDL's term for a parameter that passes information to an entity. For instance, if an entity is a gate level model with a rise and a fall delay, values for the rise and fall delays could be passed into the entity with generics.

Process: A process is the basic unit of execution in VHDL. All operations that are performed in a simulation of a

Thus, let us get our feet wet and look at VHDL structure briefly as follow:

```
ENTITY entity_name IS
    GENERIC(
          generic_1_name      : generic_1_type;
          generic_2_name      : generic_2_type;
          generic_n_name      : generic_n_type
    );
    PORT(
          port_1_name   : port_1_dir  port_1_type;
          port_2_name   : port_2_dir  port_2_type;
          port_n_name   : port_n_dir  port_n_type
    );
END entity_name;
```

IN	Input port
OUT	Output port
INOUT	Bidirectional port
BUFFER	Buffered output port

```
ARCHITECTURE architecture_name OF entity_name IS
BEGIN
    -- Insert VHDL statements to assign outputs to
    -- each of the output signals defined in the
    -- entity declaration.
END architecture_name;
```

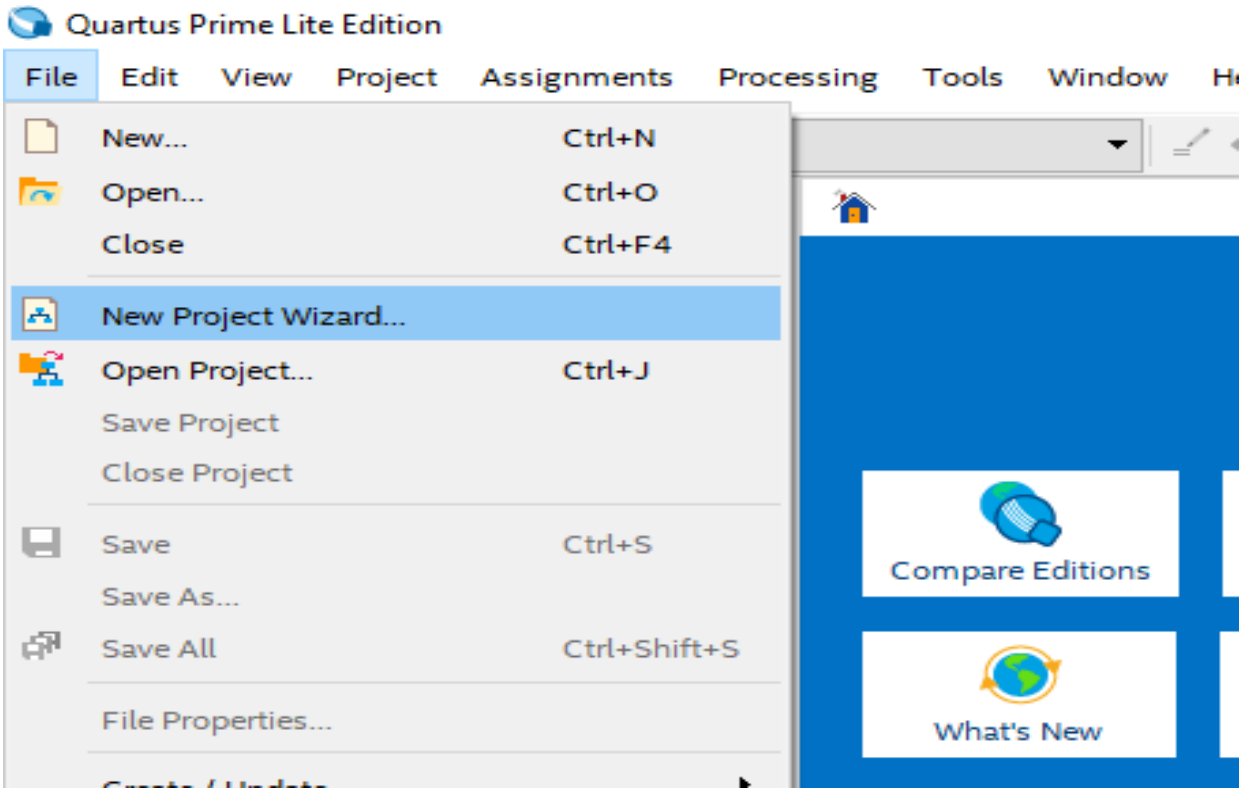
This is a simple example of VHDL code that just work as buffer and send input to output after 10 ns:

- Consider the following VHDL code:

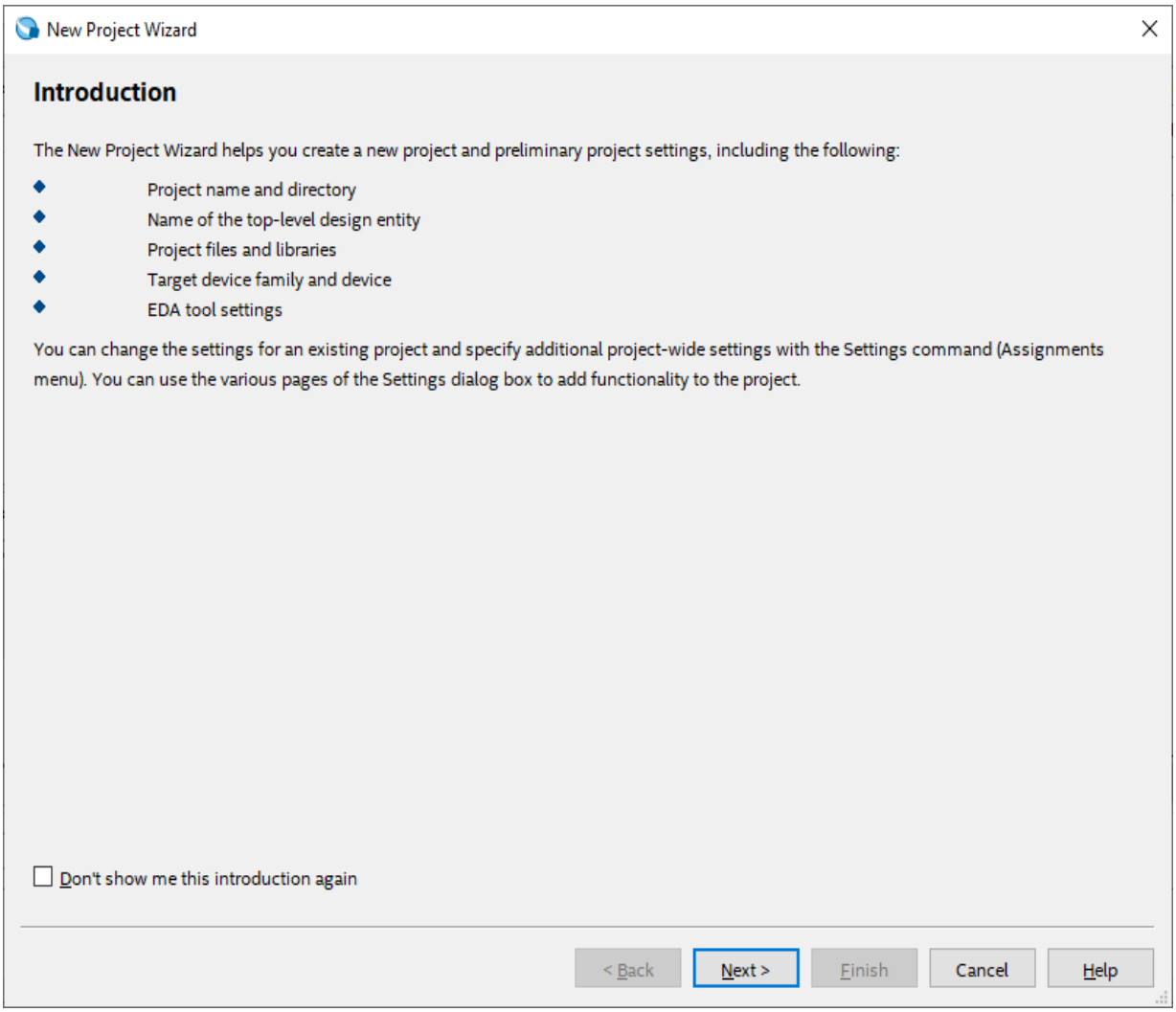
```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY simple_buffer IS
      PORT (   din      : IN     std_logic;
               dout     : OUT    std_logic );
END simple_buffer;

ARCHITECTURE behavioural1 OF simple_buffer IS
BEGIN
      dout <= din AFTER 10 ns;
END behavioural1;
```

First simple program using VHDL:

Step 1. Create a Project: Every new design you make in the Quartus program must be under a project name. After opening the program, to create a project, click on: [File] -> [New Project Wizard]



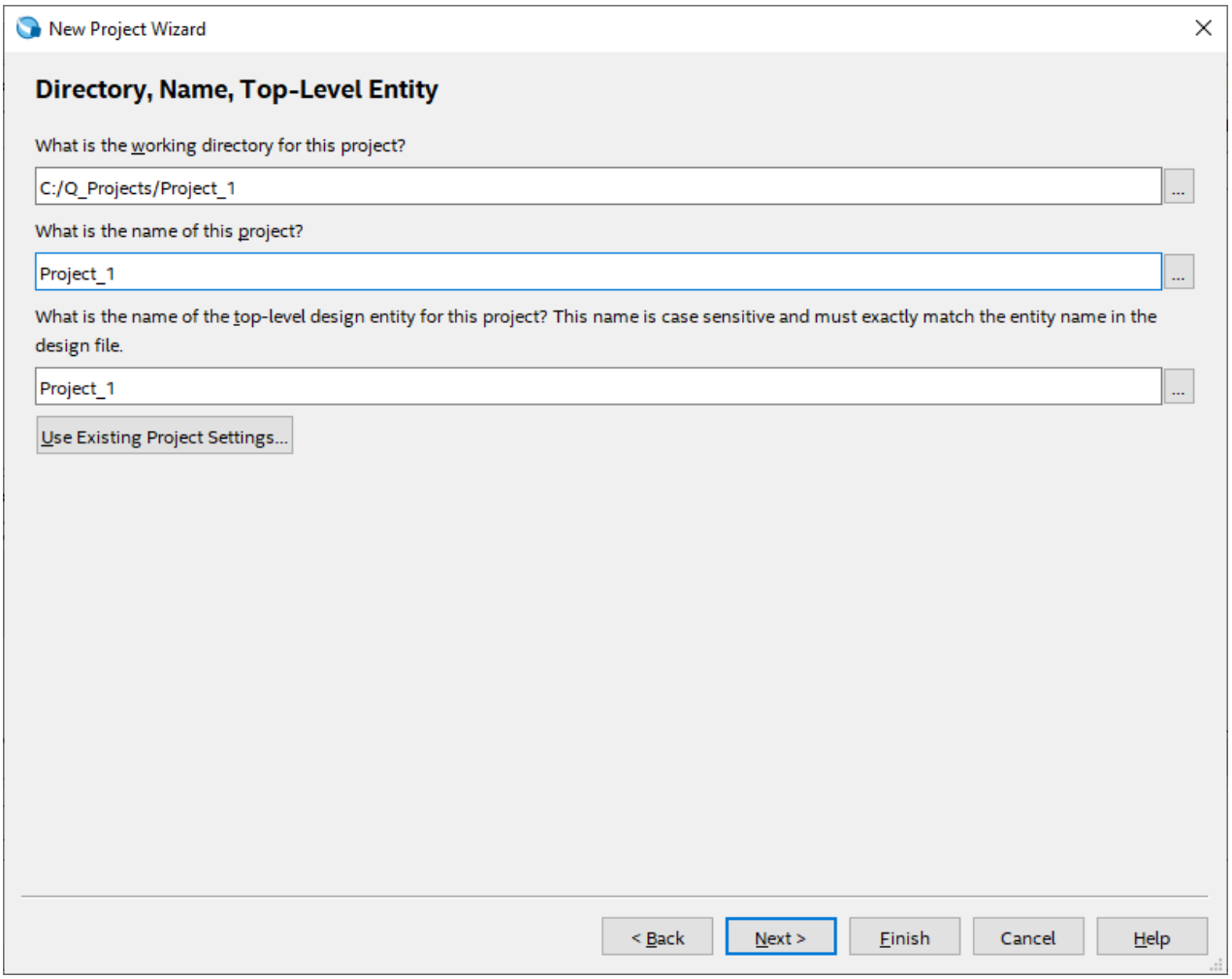
After this stage you will see the following project builder screen.



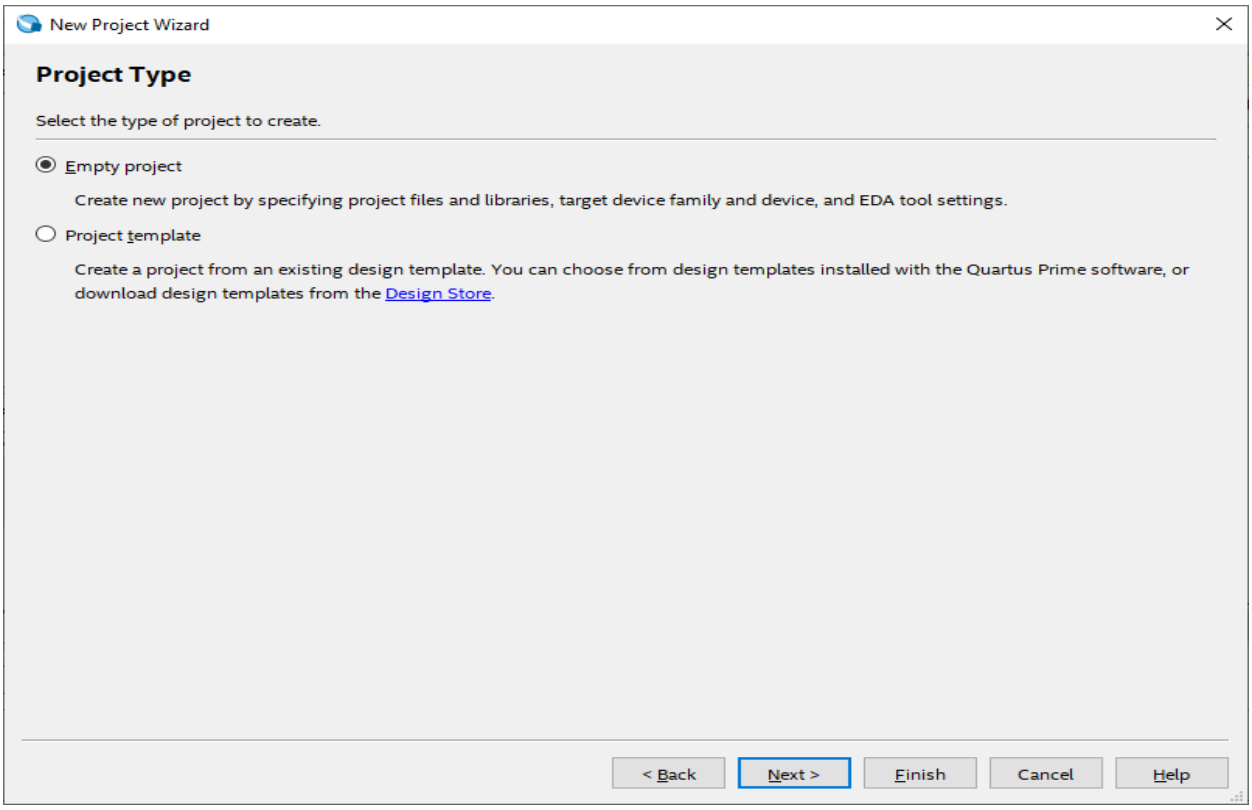
This screen is the home screen of the project builder. Here you will see which stages of your project are formed and what you can organize.

Here, press [Next >] button to continue.

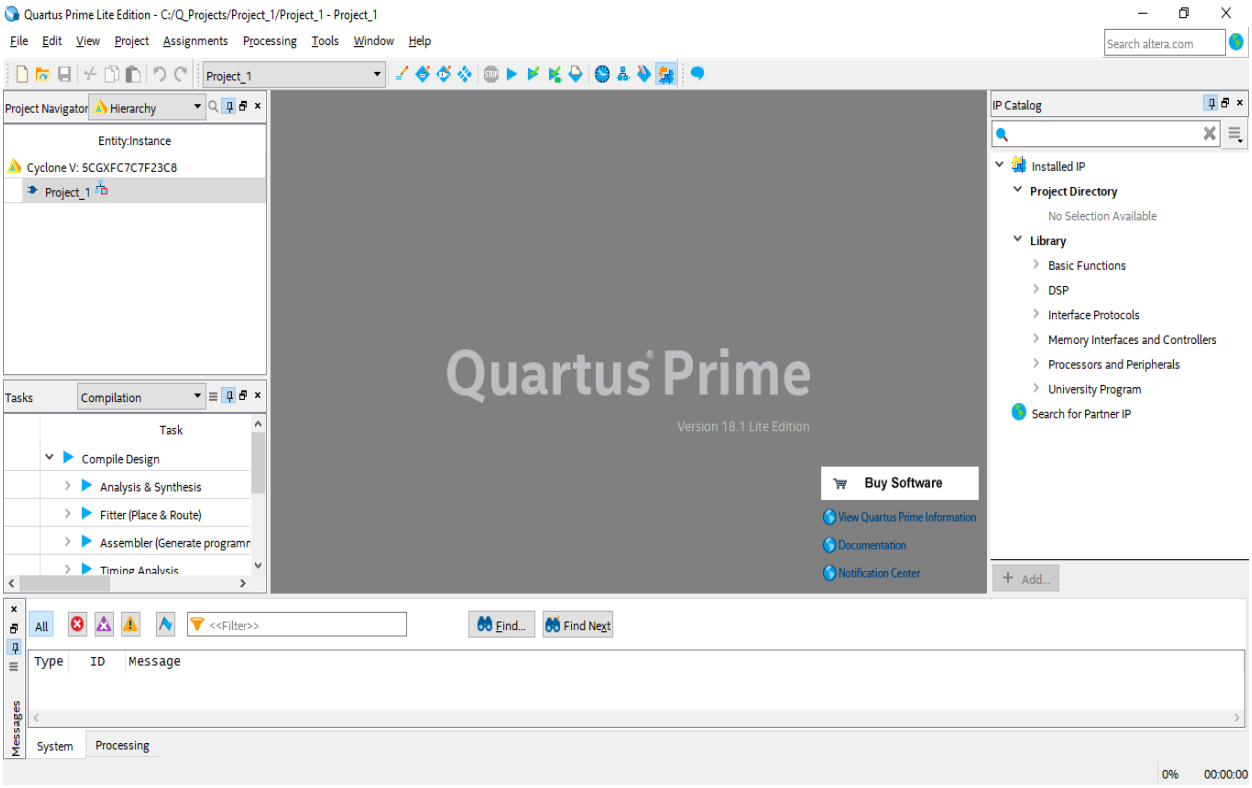
On the new screen, you will see the input sections that ask us to enter the directory and the name of the related project. Please fill in the appropriate information here. (Project names for the Quartus program is very important.)



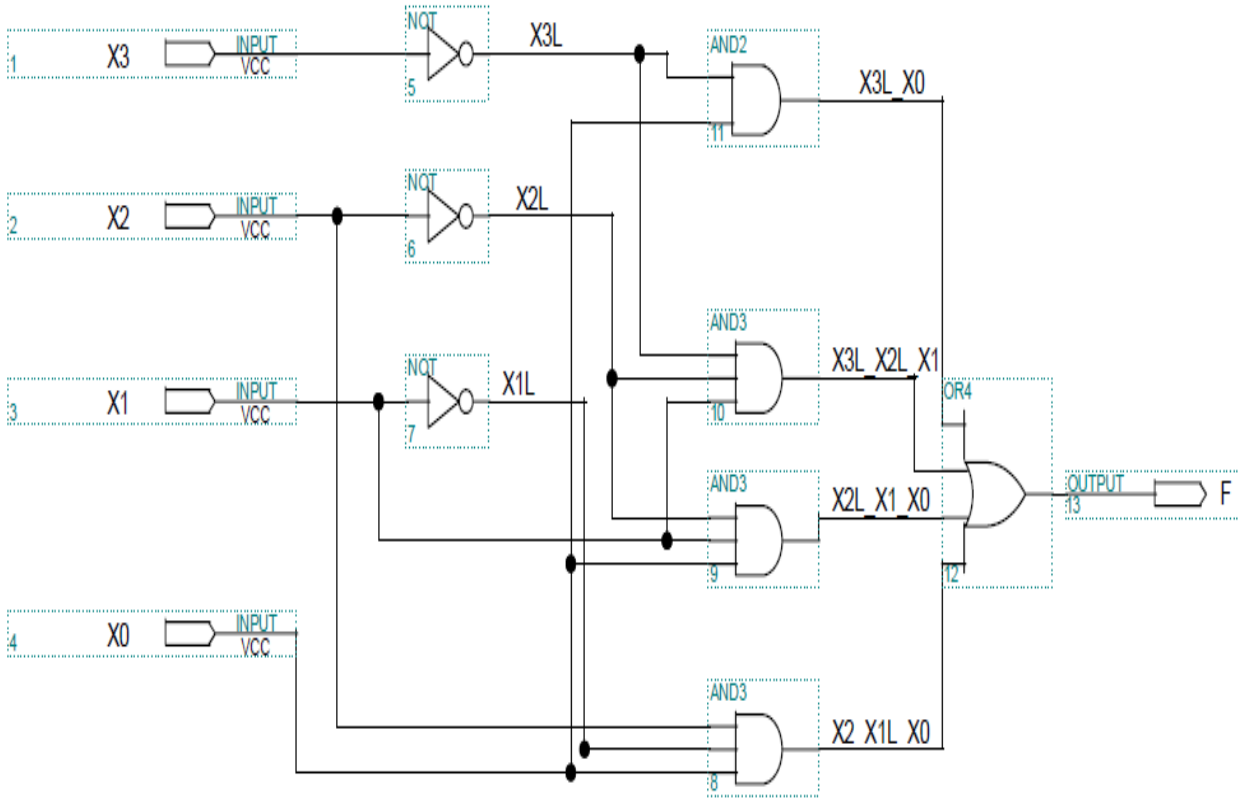
The project type display gives you additional setting options for what hardware the project will be compiled and implemented. Since, we'll simulate our projects on the program and we won't run on any real hardware, we'll choose the [Empty Project] option here, and then click the [Finish] button to create our project.



Once your project is created, you will see a screen like this:



Step2: Look at the program using VHDL code, here we have a practical circuit, implement it in Quartus (destine / compile/ simulate/show waveform), you can review pervious lab experiment and follow the steps.



Step3: In order to create a VHDL file follow the steps:

File→New→VHDL File

This is the VHDL code of above program:

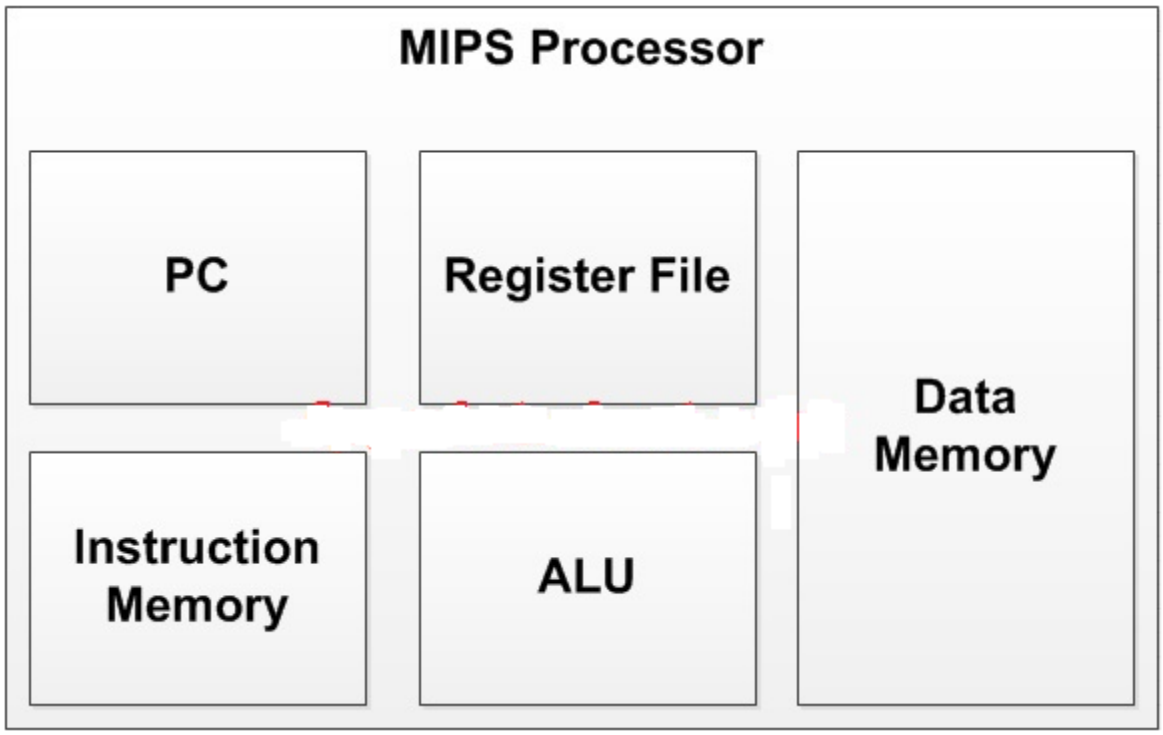
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY lab6 IS
PORT (x0, x1, x2,X3 : IN STD_LOGIC;
F : OUT STD_LOGIC);
END lab6;
ARCHITECTURE Arch_NelistStruct of lab6 is
SIGNAL X3L, X2L, X1L, X3L_X0, X3L_X2L_X1, X2L_X1_X0, X2_X1L_X0:STD_LOGIC;
BEGIN
X3L <= not X3;
X2L <= not X2;
X1L <= not X1;
X3L_X0 <= X3L and X0;
X3L_X2L_X1 <= X3L and X2L and X1;
X2L_X1_X0 <= X2L and X1 and X0;
X2_X1L_X0 <= X2 and X1 and X0;
F <= x3L_X0 or X3L_X2L_X1 or X2L_X1_X0 or x2_X1L_X0;
END Arch_NelistStruct;
```

Note: Trace the program line by line, now you are ready to learn single cycle data path using VHDL code.

1. Experimental Practice

Note that our instructions have no mnemonics at this stage, because an assembler has not been written yet for this brand new processor. We use the binary codes of the instructions to write them into the instruction memory.

The VHDL code for the MIPS Processor will be presented. A simple VHDL testbench for the MIPS processor will be also provided for simulation purposes.



Note: Be careful in your project name and VHDL file name and entity name must be same.

VHDL code for Data Memory of the MIPS processor:

```
-- fpga4student.com: FPGA projects, Verilog projects,
VHDL projects
-- VHDL project: VHDL code for single-cycle MIPS
Processor
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE IEEE.numeric_std.all;
-- VHDL code for the data Memory of the MIPS Processor
entity Data_Memory_VHDL is
port (
 clk: in std_logic;
 mem_access_addr: in std_logic_Vector(15 downto 0);
 mem_write_data: in std_logic_Vector(15 downto 0);
 mem_write_en,mem_read:in std_logic;
 mem_read_data: out std_logic_Vector(15 downto 0)
);
end Data_Memory_VHDL;

architecture Behavioral of Data_Memory_VHDL is
signal i: integer;
signal ram_addr: std_logic_vector(7 downto 0);
type data_mem is array (0 to 255 ) of std_logic_vector
(15 downto 0);
signal RAM: data_mem :=((others=> (others=>'0')));
begin

 ram_addr <= mem_access_addr(8 downto 1);
 process(clk)
 begin
  if(rising_edge(clk)) then
  if (mem_write_en='1') then
  ram(to_integer(unsigned(ram_addr))) <=
mem_write_data;
  end if;
  end if;
```

```
 end process;
   mem_read_data <= ram(to_integer(unsigned(ram_addr)))
when (mem_read='1') else x"0000";

end Behavioral;
```

VHDL code for ALU of the MIPS processor:

```
-- fpga4student.com: FPGA projects, Verilog projects,
VHDL projects
-- VHDL project: VHDL code for single-cycle MIPS
Processor
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_signed.all;
-- VHDL code for ALU of the MIPS Processor
entity ALU_VHDL is
port(
 a,b : in std_logic_vector(15 downto 0); -- src1, src2
 alu_control : in std_logic_vector(2 downto 0); --
function select
 alu_result: out std_logic_vector(15 downto 0); -- ALU
Output Result
 zero: out std_logic -- Zero Flag
 );
end ALU_VHDL;

architecture Behavioral of ALU_VHDL is
signal result: std_logic_vector(15 downto 0);
begin
process(alu_control,a,b)
begin
 case alu_control is
 when "000" =>
  result <= a + b; -- add
 when "001" =>
  result <= a - b; -- sub
```

```
 when "010" =>
  result <= a and b; -- and
 when "011" =>
  result <= a or b; -- or
 when "100" =>
  if (a<b) then
   result <= x"0001";
  else
   result <= x"0000";
  end if;
 when others => result <= a + b; -- add
 end case;
end process;
  zero <= '1' when result=x"0000" else '0';
  alu_result <= result;
end Behavioral;
```

VHDL code for ALU Control Unit of the MIPS processor:

```
-- fpga4student.com: FPGA projects, Verilog projects,
VHDL projects
-- VHDL project: VHDL code for single-cycle MIPS
Processor
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
-- VHDL code for ALU Control Unit of the MIPS Processor
entity ALU_Control_VHDL is
port(
  ALU_Control: out std_logic_vector(2 downto 0);
  ALUOp : in std_logic_vector(1 downto 0);
  ALU_Funct : in std_logic_vector(2 downto 0)
);
end ALU_Control_VHDL;

architecture Behavioral of ALU_Control_VHDL is
begin
```

```
process(ALUOp,ALU_Funct)
begin
case ALUOp is
when "00" =>
 ALU_Control <= ALU_Funct(2 downto 0);
when "01" =>
 ALU_Control <= "001";
when "10" =>
 ALU_Control <= "100";
when "11" =>
 ALU_Control <= "000";
when others => ALU_Control <= "000";
end case;
end process;
end Behavioral;
```

VHDL code for Register File of the MIPS processor:

```
-- fpga4student.com: FPGA projects, Verilog projects,
VHDL projects
-- VHDL project: VHDL code for single-cycle MIPS
Processor
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE IEEE.numeric_std.all;
-- VHDL code for the register file of the MIPS
Processor
entity register_file_VHDL is
port (
 clk,rst: in std_logic;
 reg_write_en: in std_logic;
 reg_write_dest: in std_logic_vector(2 downto 0);
 reg_write_data: in std_logic_vector(15 downto 0);
 reg_read_addr_1: in std_logic_vector(2 downto 0);
```

```
 reg_read_data_1: out std_logic_vector(15 downto 0);
 reg_read_addr_2: in std_logic_vector(2 downto 0);
 reg_read_data_2: out std_logic_vector(15 downto 0)
);
end register_file_VHDL;

architecture Behavioral of register_file_VHDL is
type reg_type is array (0 to 7 ) of std_logic_vector
(15 downto 0);
signal reg_array: reg_type;
begin
 process(clk,rst)
 begin
 if(rst='1') then
   reg_array(0) <= x"0001";
   reg_array(1) <= x"0002";
   reg_array(2) <= x"0003";
   reg_array(3) <= x"0004";
   reg_array(4) <= x"0005";
   reg_array(5) <= x"0006";
   reg_array(6) <= x"0007";
   reg_array(7) <= x"0008";
 elsif(rising_edge(clk)) then
   if(reg_write_en='1') then
    reg_array(to_integer(unsigned(reg_write_dest))) <=
reg_write_data;
   end if;
 end if;
 end process;

 reg_read_data_1 <= x"0000" when reg_read_addr_1 =
"000" else
reg_array(to_integer(unsigned(reg_read_addr_1)));
 reg_read_data_2 <= x"0000" when reg_read_addr_2 =
"000" else
reg_array(to_integer(unsigned(reg_read_addr_2)));
```

```
end Behavioral;
```

VHDL code for Control Unit of the MIPS processor:

```
- fpga4student.com: FPGA projects, Verilog projects,
VHDL projects
-- VHDL project: VHDL code for single-cycle MIPS
Processor
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
-- VHDL code for Control Unit of the MIPS Processor
entity control_unit_VHDL is
port (
  opcode: in std_logic_vector(2 downto 0);
  reset: in std_logic;
  reg_dst,mem_to_reg,alu_op: out std_logic_vector(1
downto 0);

jump,branch,mem_read,mem_write,alu_src,reg_write,sign_o
r_zero: out std_logic
 );
end control_unit_VHDL;

architecture Behavioral of control_unit_VHDL is

begin
process(reset,opcode)
begin
 if(reset = '1') then
   reg_dst <= "00";
   mem_to_reg <= "00";
   alu_op <= "00";
   jump <= '0';
   branch <= '0';
   mem_read <= '0';
   mem_write <= '0';
   alu_src <= '0';
```

```
    reg_write <= '0';
    sign_or_zero <= '1';
  else
  case opcode is
   when "000" => -- add
     reg_dst <= "01";
     mem_to_reg <= "00";
     alu_op <= "00";
     jump <= '0';
     branch <= '0';
     mem_read <= '0';
     mem_write <= '0';
     alu_src <= '0';
     reg_write <= '1';
     sign_or_zero <= '1';
   when "001" => -- sliu
    reg_dst <= "00";
    mem_to_reg <= "00";
    alu_op <= "10";
    jump <= '0';
    branch <= '0';
    mem_read <= '0';
    mem_write <= '0';
    alu_src <= '1';
    reg_write <= '1';
    sign_or_zero <= '0';
   when "010" => -- j
    reg_dst <= "00";
    mem_to_reg <= "00";
    alu_op <= "00";
    jump <= '1';
    branch <= '0';
    mem_read <= '0';
    mem_write <= '0';
    alu_src <= '0';
    reg_write <= '0';
    sign_or_zero <= '1';
```

```
when "011" =>-- jal
  reg_dst <= "10";
  mem_to_reg <= "10";
  alu_op <= "00";
  jump <= '1';
  branch <=  '0';
  mem_read <=  '0';
  mem_write <=  '0';
  alu_src <= '0';
  reg_write <=  '1';
  sign_or_zero <= '1';
when "100" =>-- lw
  reg_dst <= "00";
  mem_to_reg <= "01";
  alu_op <= "11";
  jump <= '0';
  branch <= '0';
  mem_read <= '1';
  mem_write <= '0';
  alu_src <= '1';
  reg_write <= '1';
  sign_or_zero <= '1';
when "101" => -- sw
  reg_dst <= "00";
  mem_to_reg <= "00";
  alu_op <= "11";
  jump <= '0';
  branch <= '0';
  mem_read <= '0';
  mem_write <= '1';
  alu_src <= '1';
  reg_write <= '0';
  sign_or_zero <= '1';
when "110" => -- beq
  reg_dst <= "00";
  mem_to_reg <= "00";
  alu_op <= "01";
```

```
    jump <= '0';
    branch <= '1';
    mem_read <= '0';
    mem_write <= '0';
    alu_src <= '0';
    reg_write <= '0';
    sign_or_zero <= '1';
  when "111" =>-- addi
    reg_dst <= "00";
    mem_to_reg <= "00";
    alu_op <= "11";
    jump <= '0';
    branch <= '0';
    mem_read <= '0';
    mem_write <= '0';
    alu_src <= '1';
    reg_write <= '1';
    sign_or_zero <= '1';
  when others =>
     reg_dst <= "01";
     mem_to_reg <= "00";
     alu_op <= "00";
     jump <= '0';
     branch <= '0';
     mem_read <= '0';
     mem_write <= '0';
     alu_src <= '0';
     reg_write <= '1';
     sign_or_zero <= '1';
  end case;
  end if;
end process;

end Behavioral;
```

VHDL code for Instruction Memory of the MIPS processor:

```
-- fpga4student.com: FPGA projects, Verilog projects,
VHDL projects
-- VHDL project: VHDL code for single-cycle MIPS
Processor
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE IEEE.numeric_std.all;
-- VHDL code for the Instruction Memory of the MIPS
Processor
entity Instruction_Memory_VHDL is
port (
 pc: in std_logic_vector(15 downto 0);
 instruction: out  std_logic_vector(15 downto 0)
);
end Instruction_Memory_VHDL;

architecture Behavioral of Instruction_Memory_VHDL is
signal rom_addr: std_logic_vector(3 downto 0);
 -- lw $3, 0($0) -- pc=0
 -- Loop: sltiu  $1, $3, 11= pc = 2
 -- beq $1, $0, Skip = 4 --
PCnext=PC_current+2+BranchAddr
 -- add $4, $4, $3 = 6
 -- addi $3, $3, 1 = 8
 -- beq $0, $0, Loop--a
 -- Skip c = 12 = 4 + 2 + br
 type ROM_type is array (0 to 15 ) of
std_logic_vector(15 downto 0);
 constant rom_data: ROM_type:=(
   "1000000110000000",
   "0010110010001011",
   "1100010000000011",
   "0001000111000000",
   "1110110110000001",
   "1100000001111011",
   "0000000000000000",
   "0000000000000000",
```

```
    "0000000000000000",
    "0000000000000000",
    "0000000000000000",
    "0000000000000000",
    "0000000000000000",
    "0000000000000000",
    "0000000000000000",
    "0000000000000000"
   );
begin

 rom_addr <= pc(4 downto 1);
  instruction <=
rom_data(to_integer(unsigned(rom_addr))) when pc <
x"0020" else x"0000";

end Behavioral;
```

2. Reporting

Before the Lab-time is over, show the simulation result to your lab assistant, in order to grading.

Grading:

Lab Performance:

Asst. Observations: