به نام خدا

موضوع پروژه:

طراحی و تست واحد پردازش مرکزی برای انجام عمل XOR دو عدد ۴بیتی

استاد درس:

دکتر پریا دربانی

اعضای گروه:

محمد اصولیان (۹۹۵۲۱۰۷۳)

نوید ابراهیمی (۹۹۵۲۱۰۰۱)

دی ماه ۱۴۰۱

توضیح قسمت‌های مختلف پروژه:

۱- ALU:

وظیفه این واحد محاسبه XOR دو عدد ۴بیتی‌ست. همانطور که در entity ALU دیده میشود ورودی این قطعه ۲ عدد ۴ بیتی به همراه op است که نوع عمل منطقی که در اینجا XOR است را نشان میدهد.

در قسمت architecture ALU_behave هم XOR ۲ عدد را محاسبه و آن را داخل result می‌ریزد.

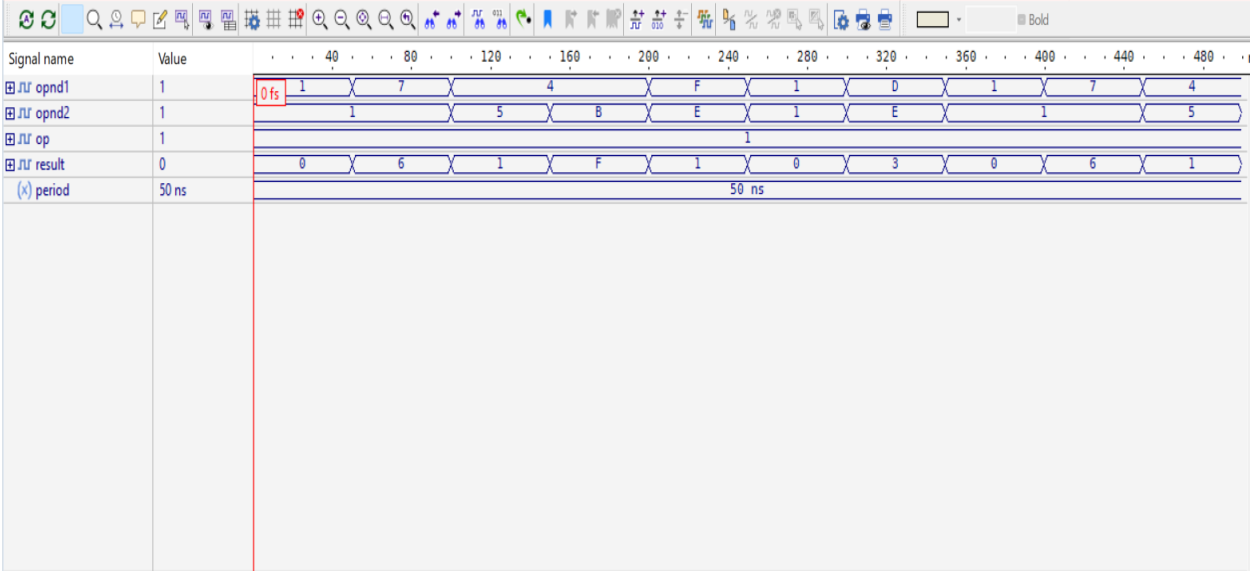
```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  -- This is ALU. Calculates only "XOR" operation as was mentions in project document.
6  entity ALU is
7      port(
8      opnd1: in std_logic_vector(3 downto 0);
9      opnd2: in std_logic_vector(3 downto 0);
10     op: in std_logic_vector(1 downto 0);
11     result: out std_logic_vector(3 downto 0)
12     );
13 end ALU;
14
15 architecture ALU_behave of ALU is
16 begin
17     process(opnd1, opnd2, op)
18     begin
19         if op = "01" then
20             result <= opnd1 xor opnd2;
21         end if;
22     end process;
23 end ALU_behave;
24
```

شکل۱ - قطعه کد ALU

```
process
begin

    opnd1 <= "0001";
    opnd2 <= "0001";
    op <= "01";
    wait for period;
    opnd1 <= "0111";
    opnd2 <= "0001";
    op <= "01";
    wait for period;
    opnd1 <= "0100";
    opnd2 <= "0101";
    op <= "01";
    wait for period;
    opnd1 <= "0100";
    opnd2 <= "1011";
    op <= "01";
    wait for period;
    opnd1 <= "1111";
    opnd2 <= "1110";
    op <= "01";
    wait for period;
    opnd1 <= "0001";
    opnd2 <= "0001";
    op <= "01";
    wait for period;
    opnd1 <= "1101";
    opnd2 <= "1110";
    op <= "01";
    wait for period;
end process;
```

شکل۲ – فایل تست ALU



شکل۳ – موج خروجی ALU

۲- Instruction Memory:

در این قطعه با استفاده از addr آن قسمت از حافظه که میخواهیم را استخراج کرده و writer, readr1, readr2 و op را از آن استخراج میکنیم.

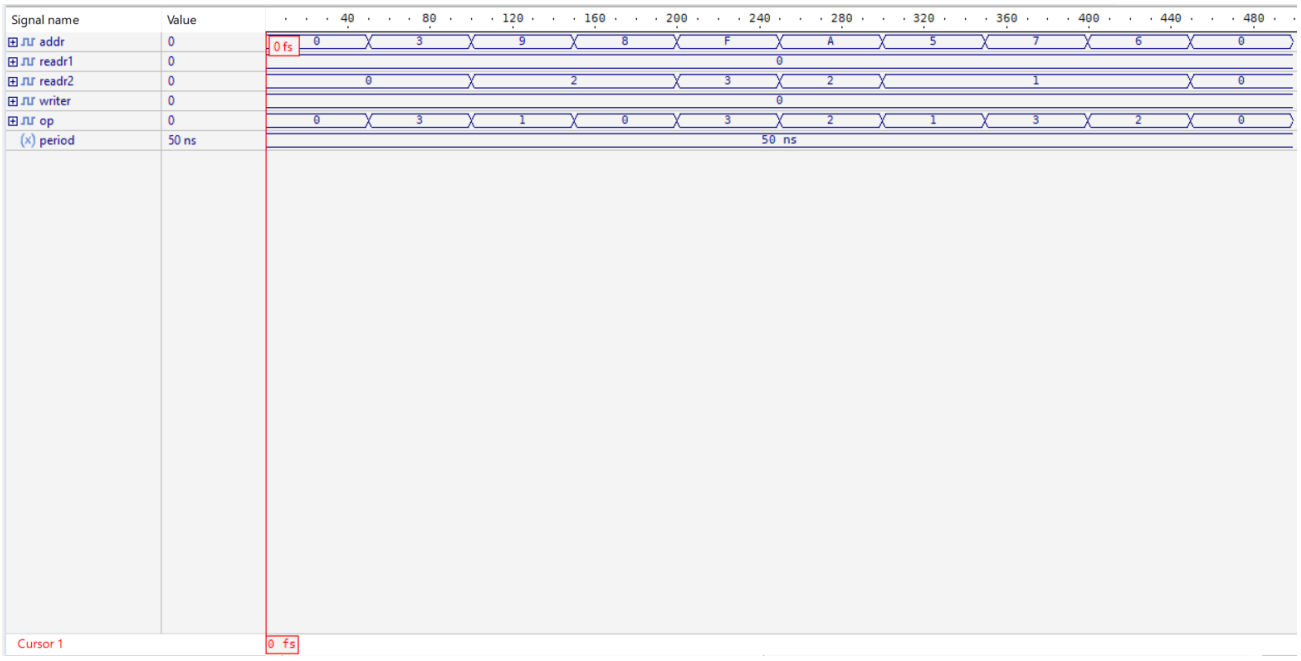
وظیفه هرکدام مشخص کردن آدرس هرکدام از registerهاست که در ادامه در قسمت Registers از آن استفاده میکنیم.

```
6  -- Instruction Memory saves Instructions that will be run in cpu.
7  -- For this project instructions are hard coded in a ROM type in architecture
8
9  entity InstructionMemory is
10     port(
11     addr: in std_logic_vector(3 downto 0);
12     readr1: out std_logic_vector(1 downto 0);
13     readr2: out std_logic_vector(1 downto 0);
14     writer: out std_logic_vector(1 downto 0);
15     op: out std_logic_vector(1 downto 0)
16     );
17 end InstructionMemory;
18
19 architecture InstructionMemory_behave of InstructionMemory is
20 type Mem_type is array(0 to 15) of std_logic_vector(7 downto 0);
21 signal mem: Mem_type :=    ("10000101",
22                             "11011001",
23                             "00101101",
24                             "01110001",
25                             "10000101",
26                             "11011001",
27                             "00000000",
28                             "00000000",
29                             "00000000",
30                             "00000000",
31                             "00000000",
32                             "00000000",
33                             "00000000",
34                             "00000000",
35                             "00000000",
36                             "00000000");
37
38 signal data:std_logic_vector(7 downto 0) := "00000000";
39
40 begin
41     data <= mem(to_integer(unsigned(addr)));
42     writer <= data(7 downto 6);
43     readr1 <= data(5 downto 4);
44     readr2 <= data(3 downto 2);
45     op <= data(1 downto 0);
46 end InstructionMemory_behave;
```

شکل۴ – قطعه کد Instruction Memory

```
process
begin
    addr<="0000";
    wait for period;
    addr<="0011";
    wait for period;
    addr<="1001";
    wait for period;
    addr<="1000";
    wait for period;
    addr<="1111";
    wait for period;
    addr<="1010";
    wait for period;
    addr<="0101";
    wait for period;
    addr<="0111";
    wait for period;
    addr<="0110";
    wait for period;
end process;
```

شکل۵ – فایل تست Instruction Memory



شکل۶ – موج خروجی Instruction Memory

۳- PC:

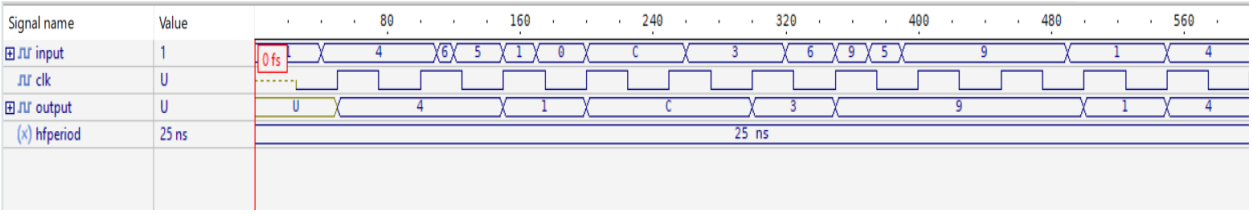
آدرس بخشی که قرار است execute شود را نگهداری میکند.

```
5   -- Program Counter only counts the instruction that is going to be run
6   entity PC is
7       port(
8       input: in std_logic_vector(3 downto 0) := "0000";
9       clk: in std_logic;
10      output: out std_logic_vector(3 downto 0) := "0000"
11      );
12  end PC;
13
14  architecture PC_behave of PC is
15  begin
16      process(clk, input)
17      begin
18          if rising_edge(clk) then
19              output<=input;
20          end if;
21      end process;
22  end PC_behave;
23
```

شکل۷ – قطعه کد PC

```
    process
    begin
        wait for hfperiod;
        clk <= '0';
        wait for hfperiod;
        clk <= '1';
    end process;
    process
    begin
        input <= "0001";
        wait for 40 ns;
        input <= "0100";
        wait for 70 ns;
        input <= "0110";
        wait for 10 ns;
        input <= "0101";
        wait for 30 ns;
        input <= "0001";
        wait for 20 ns;
        input <= "0000";
        wait for 30 ns;
        input <= "1100";
        wait for 40 ns;
        input <= "1100";
        wait for 20 ns;
        input <= "0011";
        wait for 30 ns;
        input <= "0011";
        wait for 30 ns;
        input <= "0110";
        wait for 30 ns;
        input <= "1001";
        wait for 20 ns;
        input <= "0101";
        wait for 20 ns;
        input <= "1001";
        wait for 60 ns;
        input <= "1001";
        wait for 40 ns;
        input <= "0001";
        wait for 20 ns;
```

شکل۸ – فایل تست PC



شکل۹ – موج خروجی PC

۴- Register File:

در این قطعه به این صورت عمل میکنیم که یک regFile داریم که آدرس Registerهایی که ما با آن کار داریم در آن ذخیره شده است. با استفاده از regFile و آدرس هر register، دیتای موردنظر را استخراج میکنیم.

از این قطعه ۲ استفاده میتوان کرد:

۱- استخراج دیتای مورد نیاز با readr و دادن دیتاها به ALU.

۲- در حالتی که بخواهیم عمل write انجام دهیم، مقداری که از مراحل بعدی به دست آمده را در register موردنظر write کنیم.

```
5   -- Register File has the data of registers.
6   entity RegisterFile is
7       port(
8       readr1: in std_logic_vector(1 downto 0);
9       readr2: in std_logic_vector(1 downto 0);
10      writer: in std_logic_vector(1 downto 0);
11      writed: in std_logic_vector(3 downto 0);
12      regwrite: in std_logic;
13      readd1: out std_logic_vector(3 downto 0);
14      readd2: out std_logic_vector(3 downto 0)
15      );
16  end RegisterFile;
17
18  architecture RegisterFile_behave of RegisterFile is
19  type RegFile_type is array(0 to 3) of std_logic_vector(3 downto 0);
20  signal regfile: RegFile_type := ("0011",
21                                   "0100",
22                                   "0000",
23                                   "0000");
24  begin
25      process(readr1, readr2, writer, writed, regwrite)
26      begin
27          readd1 <= regfile(to_integer(unsigned(readr1)));
28          readd2 <= regfile(to_integer(unsigned(readr2)));
29          if writed'event then
30              regfile(to_integer(unsigned(writer))) <= writed;
31          end if;
32      end process;
33
34  end RegisterFile_behave;
```

شکل ۱۰ – قطعه کد Register File

```
process
begin
    -- initialize regwrite
    regwrite <= '0';
    -- check read from registers
    readr1 <= "00";
    readr2 <= "01";
    wait for period;

    readr1 <= "10";
    readr2 <= "00";
    wait for period;

    readr1 <= "11";
    readr2 <= "10";
    wait for period;

    --check write to registers
    writer <= "00";
    writed <= "1000";
    regwrite <= '1';
    wait for pulseperiod;
    regwrite <= '0';
    wait for period;

    writer <= "01";
    writed <= "1001";
    regwrite <= '1';
    wait for pulseperiod;
    regwrite <= '0';
    wait for period;

    writer <= "10";
    writed <= "1010";
    regwrite <= '1';
    wait for pulseperiod;
    regwrite <= '0';
    wait for period;

end process;
```

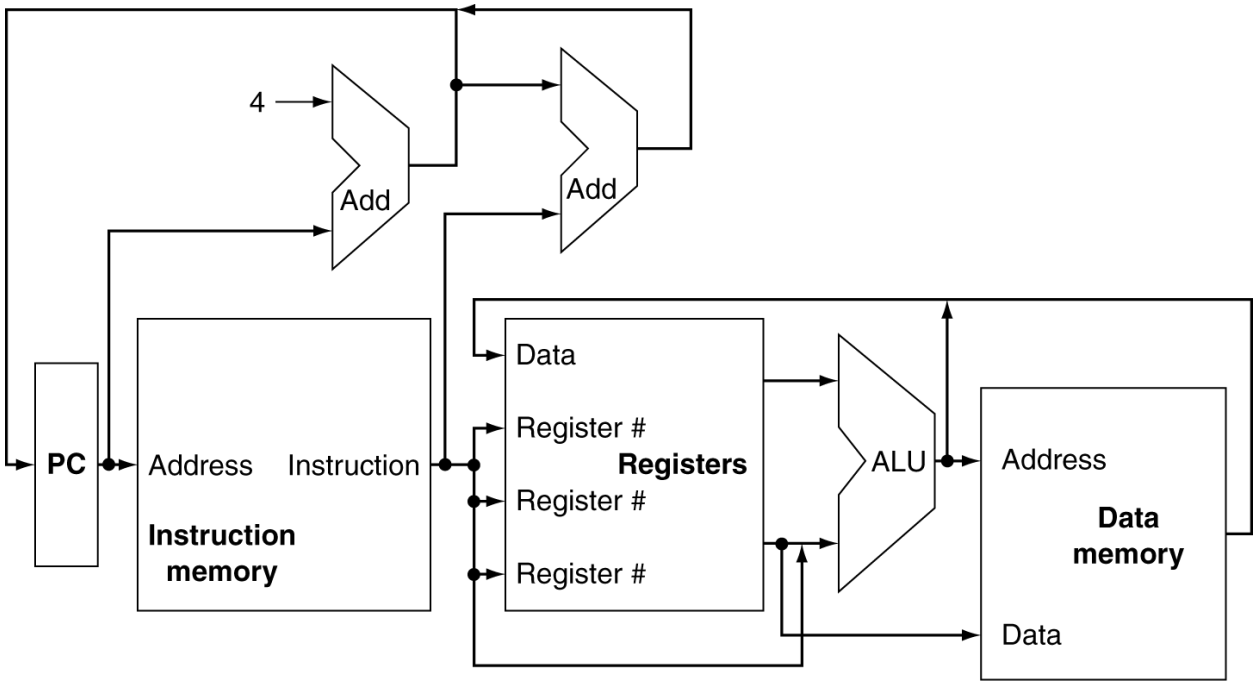
شکل۱۱ – فایل تست Regiater Files

شکل۱۲ – موج خروجی Register File

۵- Controller:

در کد ما نقش ارتباط دهنده بین قطعات مختلفی که در بالا تعریف کردیم را دارد. به این صورت که از همه قطعات در این فایل با component استفاده شده است و خروجی هر قطعه ورودی قطعه دیگر خواهد بود.

شکل زیر نحوه کار این فایل است:



شکل۱۳ – نحوه عملکرد Controller

در واقع تستی که برای این سوال نوشتیم روی این قطعه ران میشود.

تعدادی سیگنال دیباگ هم برای مشاهده جزئیات هر کامپوننت تعریف کرده‌ایم.

```
entity Controller is
    port(
    reset: in std_logic;
    clk: in std_logic;
    dbg_counter: out std_logic_vector(3 downto 0);
    dbg_im_readr1: out std_logic_vector(1 downto 0);
    dbg_im_readr2: out std_logic_vector(1 downto 0);
    dbg_im_writer: out std_logic_vector(1 downto 0);
    dbg_im_op: out std_logic_vector(1 downto 0);
    dbg_rf_readd1: out std_logic_vector(3 downto 0);
    dbg_rf_readd2: out std_logic_vector(3 downto 0);
    dbg_alu_result: out std_logic_vector(3 downto 0)
    );
end Controller;

architecture Controller_behave of Controller is
component ALU is
    port(
    opnd1: in std_logic_vector(3 downto 0);
    opnd2: in std_logic_vector(3 downto 0);
    op: in std_logic_vector(1 downto 0);
    result: out std_logic_vector(3 downto 0)
    );
end component;

component InstructionMemory is
    port(
    addr: in std_logic_vector(3 downto 0);
    readr1: out std_logic_vector(1 downto 0);
    readr2: out std_logic_vector(1 downto 0);
    writer: out std_logic_vector(1 downto 0);
    op: out std_logic_vector(1 downto 0)
    );
end component;


component PC is
    port(
    input: in std_logic_vector(3 downto 0);
    clk: in std_logic;
    output: out std_logic_vector(3 downto 0)
    );
end component;

component RegisterFile is
    port(
    readr1: in std_logic_vector(1 downto 0);
    readr2: in std_logic_vector(1 downto 0);
    writer: in std_logic_vector(1 downto 0);
    writed: in std_logic_vector(3 downto 0);
    regwrite: in std_logic;
    readd1: out std_logic_vector(3 downto 0);
    readd2: out std_logic_vector(3 downto 0)
    );
end component;

-- ALU signals
signal alu_opnd1:std_logic_vector(3 downto 0);
signal alu_opnd2:std_logic_vector(3 downto 0);
signal alu_op: std_logic_vector(1 downto 0);
signal alu_result: std_logic_vector(3 downto 0);

-- InstructionMemory signals
signal im_addr: std_logic_vector(3 downto 0);
signal im_readr1: std_logic_vector(1 downto 0);
signal im_readr2: std_logic_vector(1 downto 0);
signal im_writer: std_logic_vector(1 downto 0);
signal im_op: std_logic_vector(1 downto 0);

-- PC signals
signal pc_input: std_logic_vector(3 downto 0);
signal pc_clk: std_logic;
signal pc_output: std_logic_vector(3 downto 0);

-- RegisterFile signals
signal rf_readr1: std_logic_vector(1 downto 0);
signal rf_readr2: std_logic_vector(1 downto 0);
signal rf_writer: std_logic_vector(1 downto 0);
signal rf_writed: std_logic_vector(3 downto 0);
signal rf_regwrite: std_logic;
signal rf_readd1: std_logic_vector(3 downto 0);
signal rf_readd2: std_logic_vector(3 downto 0);
```

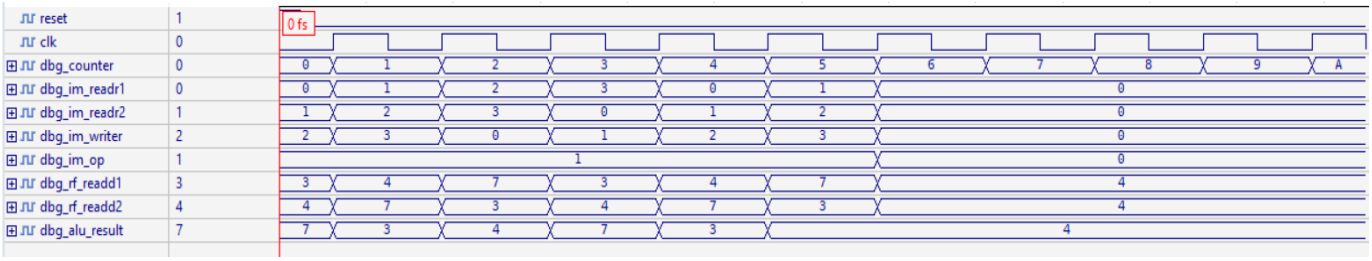
```
96  signal counter: std_logic_vector(3 downto 0) := "0000";
97
98
99  begin
100     pccom: PC port map(pc_input, pc_clk, pc_output);
101     alucom: ALU port map(alu_opnd1, alu_opnd2, alu_op, alu_result);
102     imcom: InstructionMemory port map(im_addr, im_readr1, im_readr2, im_writer, im_op);
103     rfcom: RegisterFile port map(rf_readr1, rf_readr2, rf_writer, rf_writed, rf_regwrite, rf_readd1, rf_readd2);
104     pc_clk <= clk;
105     im_addr <= counter;
106     rf_readr1 <= im_readr1;
107     rf_readr2 <= im_readr2;
108     rf_writer <= im_writer;
109     alu_opnd1 <= rf_readd1;
110     alu_opnd2 <= rf_readd2;
111     alu_op <= im_op;
112     rf_writed <= alu_result;
113
114
115     dbg_counter <= counter;
116     dbg_im_readr1 <= im_readr1;
117     dbg_im_readr2 <= im_readr2;
118     dbg_im_writer <= im_writer;
119     dbg_im_op <= im_op;
120     dbg_rf_readd1 <= rf_readd1;
121     dbg_rf_readd2 <= rf_readd2;
122     dbg_alu_result <= alu_result;
123     process(clk)
124     begin
125         if (rising_edge(clk)) then
126             if reset = '1' then
127                 counter <= "0000";
128             else
129                 counter <= std_logic_vector(unsigned(counter) + 1);
130
131             end if;
132         end if;
133     end process;
134
135
136 end Controller_behave;
```

شکل۱۴ – قطعه کدهای Controller

```
process
begin
    clk <= '0';
    wait for 25 ns;
    clk <= '1';
    wait for 25 ns;
end process;

process
begin
    reset <= '1';
    wait for 10 ns;
    reset <= '0';
    wait for 1000 ns;
end process;
```

شکل۱۵ – فایل تست Controller



شکل۱۶ – موج خروجی Controller