مستند پروژه درس CAD

فاطمه زهرا بخشنده 98522157

افشین زنگنه 98521243

ساختار پروژه:

فایل ALU_utility_functions.vhd:

این بخش شامل توابع کمکی برای محاسبه عملیات ریاضی مثل جذر، لگاریتم و ... است و امضای پکیج آن به صورت زیر است:

```
package ALU_utility_functions is
   function DIVIDE(x, y: INTEGER) return INTEGER;
   function LOG(number: INTEGER) return INTEGER;
   function SQRT(number: INTEGER) return INTEGER;
   function POW(x, y: INTEGER) return INTEGER;
end ALU_utility_functions;
```

فایل ALU_Procedure.vhd:

در این بخش منطق برنامه پیاده‌سازی شده‌است و با توجه به نوع عملگرها، عملیات مناسب انجام می‌گیرد و بدنه آن به این صورت است:

```
case(OP) is
   when "000" => -- Addition
      Output := A + B ;
   when "001" => -- Subtraction
      Output := A - B ;
   when "010" => -- Multiplication
      Output := std_logic_vector(to_signed((to_integer(signed(A)) * to_integer(signed(B))), 8)) ;
   when "011" => -- Division
      Output := std_logic_vector(to_signed(DIVIDE(to_integer(signed(A)), to_integer(signed(B))), 8)) ;
   when "100" => -- Power
      Output := std_logic_vector(to_signed(POW(to_integer(signed(A)), to_integer(signed(B))), 8));
   when "101" => -- logarithm
      Output := std_logic_vector(to_signed(LOG(to_integer(signed(A))), 8)) ;
   when "110" => -- Sqrt
      Output := std_logic_vector(to_signed(SQRT(to_integer(signed(A))), 8)) ;
   when others => Output := (others => 'X');
end case;
```

فایل ALU.vhd:

این بخش نقطه شروع برنامه است و با توجه به ورودی‌های داده شده، سه بار ماژول ALU فراخوانی می‌شود تا به ترتیب نتیجه پرانتز اول، پرانتز دوم و ترکیب دو پرانتز به دست آید.

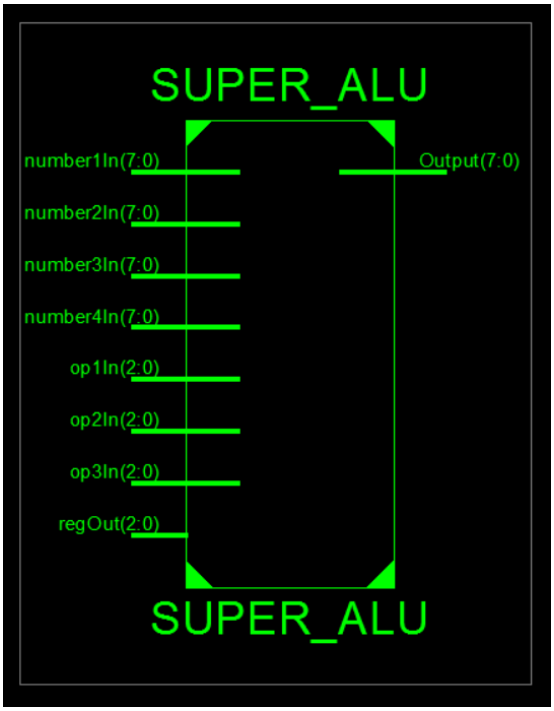
```
ALU(number1, number2, op1, out1);
ALU(number3, number4, op3, out2);
ALU(out1, out2, op2, out3);
```

فایل ALU_tb.vhd:

در این فایل به تست برنامه می‌پردازیم و حالت‌های مختلف را بررسی می‌کنیم.

بررسی قابل سنتز بودن:

* با زدن دکمه سنتز، در ابتدا خطاهایی مشاهده می‌شد که در ادامه این خطاها برطرف شد و در نتیجه کد قابل سنتز می‌باشد.



تست و بررسی:

در این قسمت به بررسی چند نمونه تست می‌پردازیم:

```
wait for 10 ns;
-- (12 + 6) * (2 ^ 4) = 288 => "00100000" after overflow
op1In <= "000"; --sum
op2In <= "010"; --multiply
op3In <= "100"; --power
number1In <= "00001100"; --12
number2In <= "00000110"; --6
number3In <= "00000010"; --2
number4In <= "00000100"; --4
regOut <= "000";
```

* این همان مثال مستند پروژه است که در نهایت جواب 288 می‌شود اما چون فقط 8 بیت برای نمایش داریم، جواب برابر 32 می‌شود و در نتیجه مدار به درستی کار می‌کند.

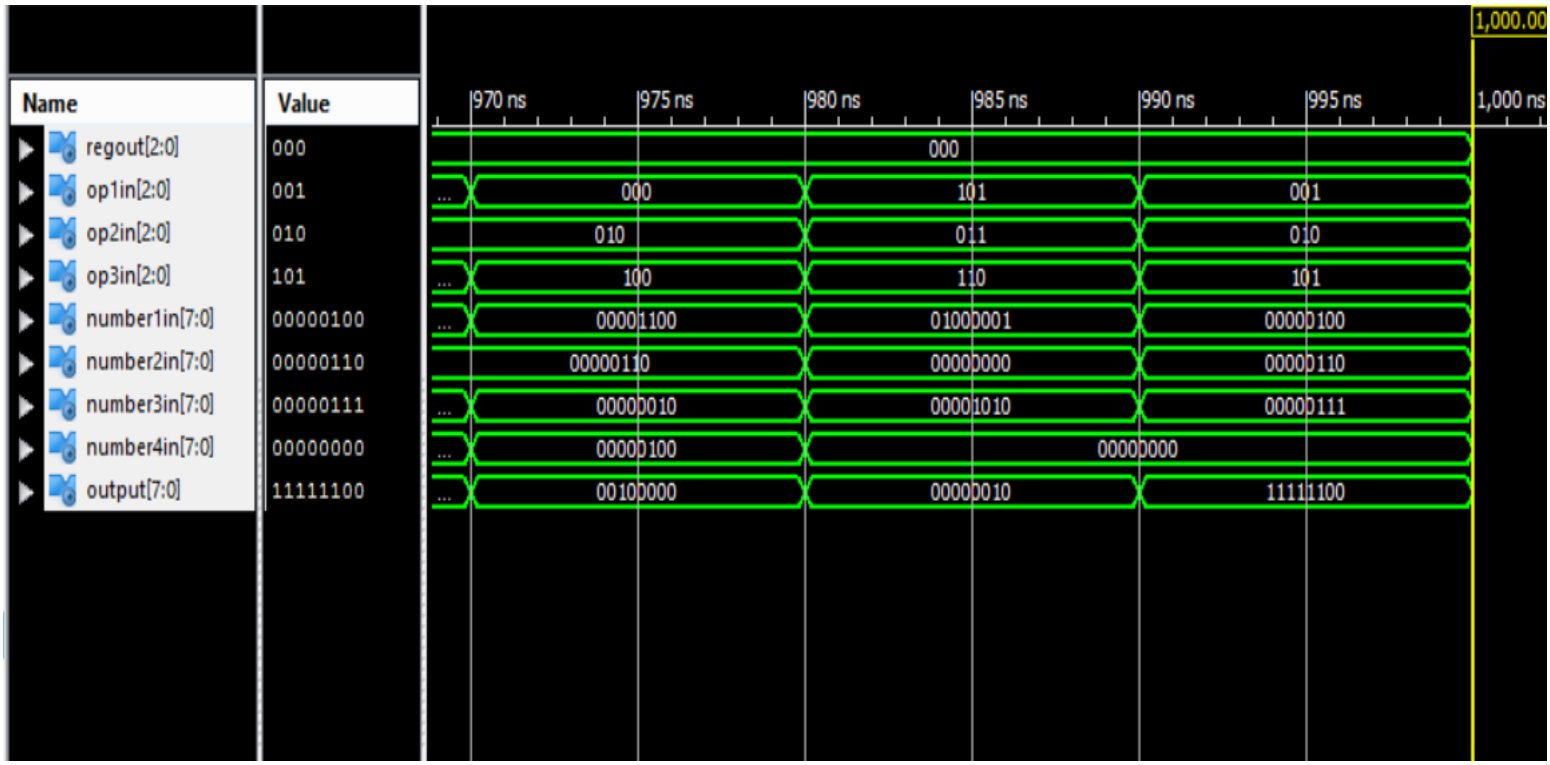
```
wait for 10ns;
-- (log 65) / (sqrt(10)) = 6 / 3 = 2 => "00000010"
op1In <= "101"; --log
op2In <= "011"; --division
op3In <= "110"; --sqrt
number1In <= "01000001"; --65
number2In <= "00000000"; --0
number3In <= "00001010"; --10
number4In <= "00000000"; --0
regOut <= "000";
```

* در این تست سایر عملگرها شامل جذر، لگاریتم و تقسیم تست شده‌اند و نتیجه به درستی محاسبه شده‌است.

```
wait for 10ns;
-- (4 - 6) * (log 7) = -2 * 2 = -4 => "11111100" 2's complement of -4
op1In <= "001"; --subtraction
op2In <= "010"; --multiply
op3In <= "101"; --log
number1In <= "00000100"; --4
number2In <= "00000110"; --6
number3In <= "00000111"; --7
number4In <= "00000000"; --0
regOut <= "000";
```

* در این تست نیز اعداد منفی بررسی شده‌اند و جواب به درستی محاسبه شده‌است.

نمودار شبیه‌سازی:



*بخش امتیازی اول:

برای پیاده سازی بخش امتیازی، ابتدا 8 رجیستر به ALU اضافه می کنیم. این آرایه از نوع inout است.

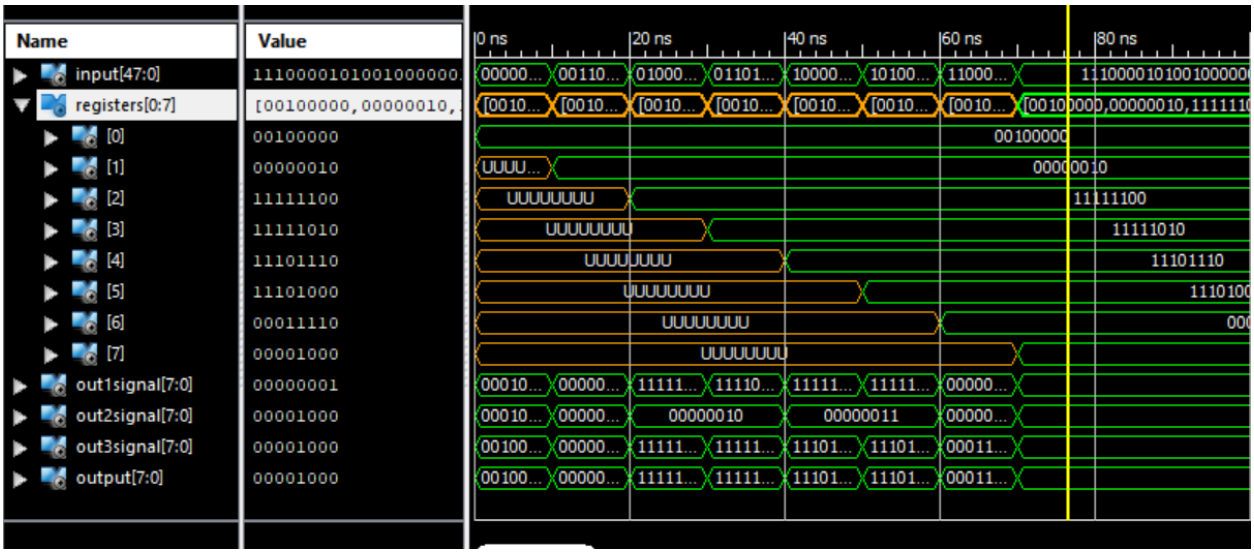
ورودی ما در این حالت 48 بیتی است. هر دفعه ابتدا بیت قبل هر عدد را چک می کنیم اگر صفر بود خود عدد را خوانده و اگر یک بود محتوای رجیستر متناظر با آن را می خوانیم.

در آخر مانند قسمت اول محاسبات را انجام داده و در شماره رجیستری که در سه بیت اول آمده بود ذخیره می کنیم.

```
    if number1In(8)='0' then
      number1 := number1In(7 downto 0);
    else
      number1 := registers(to_integer(unsigned(number1In(2 downto 0))));
    end if;

    if number2In(8)='0' then
      number2 := number2In(7 downto 0);
    else
      number2 := std_logic_vector(signed(registers(to_integer(unsigned(number2In(2 downto 0))))));
    end if;

    if number3In(8)='0' then
      number3 := number3In(7 downto 0);
    else
      number3 := registers(to_integer(unsigned(number3In(2 downto 0))));
    end if;

    if number4In(8)='0' then
      number4 := number4In(7 downto 0);
    else
      number4 := registers(to_integer(unsigned(number4In(2 downto 0))));
    end if;
```

برای تست آن یک تست بنچ نوشته که 8 تست دارد و در آخر تمام 8 رجیستر را پر می کند که دو تای آن ها در زیر نشان داده شده است:

```
stim_proc: process
begin
   -- (12 + 6) * (2 ^ 4) = 288 => "00100000" after overflow
   input <= "00000000101000000011000000001100000000010000000100";
   --op1In <= "000"; --sum
   --op2In <= "010"; --multiply
   --op3In <= "100"; --power
   --number1In <= "000001100"; --12
   --number2In <= "000000110"; --6
   --number3In <= "000000010"; --2
   --number4In <= "000000100"; --4
   --regOut <= "000";

   wait for 10ns;
   -- (log 65) / (sqrt(10)) = 6 / 3 = 2 => "00000010"
   input <= "00110101111100010000010000000000000001010000000000";
   --op1In <= "101"; --log
   --op2In <= "011"; --division
   --op3In <= "110"; --sqrt
   --number1In <= "001000001"; --65
   --number2In <= "000000000"; --0
   --number3In <= "000001010"; --10
   --number4In <= "000000000"; --0
   --regOut <= "001";
```



*بخش امتیازی دوم:

برای پیاده سازی بخش امتیازی دوم، کافیست از یک loop استفاده کنیم که به تعداد عدد n generic خط برنامه میگیرد و پشت سر هم خط هارا مانند بخش قبل اجرا کرده و مقدار رجیستر هارا آپدیت می کند.

```
entity SUPER_ALU is
   generic (n: integer := 3);
  Port (
   program: in lines(0 to n - 1);
   registers : inout regs;
   out1Signal, out2Signal, out3Signal : inout std_logic_vector (7 downto 0);
   Output: out STD_LOGIC_VECTOR(7 downto 0)
   );
end SUPER_ALU;

architecture Behavioral of SUPER_ALU is
signal number1signal, number2signal, number3signal, number4signal: std_logic_vector(7 downto 0);
shared variable regOut: std_logic_vector (2 downto 0);
begin

   process(program)
      variable input : std_logic_vector (47 downto 0);
      variable op1, op2, op3: std_logic_vector (2 downto 0);
      variable number1In, number2In, number3In, number4In: STD_LOGIC_VECTOR(8 downto 0);
      variable number1, number2, number3, number4 : std_logic_vector (7 downto 0);
      variable out1, out2, out3 : std_logic_vector (7 downto 0);
   begin

      for i in 0 to n - 1 loop

         input := program(i);
         regOut:= input(47 downto 45);
         op1 := input(44 downto 42);
         op2 := input(41 downto 39);
         op3 := input(38 downto 36);

         number1In := input(35 downto 27);
```

تست آن را به این صورت می نویسیم که یک برنامه با چند خط به آن می دهیم و انتظار داریم مقدار رجیستر هارا آپدیت کند. دقت کنید باید علاوه بر port map، generic map هم انجام دهیم.

```
BEGIN

   -- Instantiate the Unit Under Test (UUT)
   uut: SUPER_ALU
   GENERIC MAP (n => parameter)
   PORT MAP (
          program => program,
          registers => registers,
          out1Signal => out1Signal,
          out2Signal => out2Signal,
          out3Signal => out3Signal,
          Output => Output
        );


   -- Stimulus process
   stim_proc: process
   begin
      wait for 10ns;
      program <=
      (
      "00000001010000000110000000011000000001000000100",
      "00110101111000100000100000000000000101000000000",
      "11000101010100000010000000011000000011100000000"
      );
```