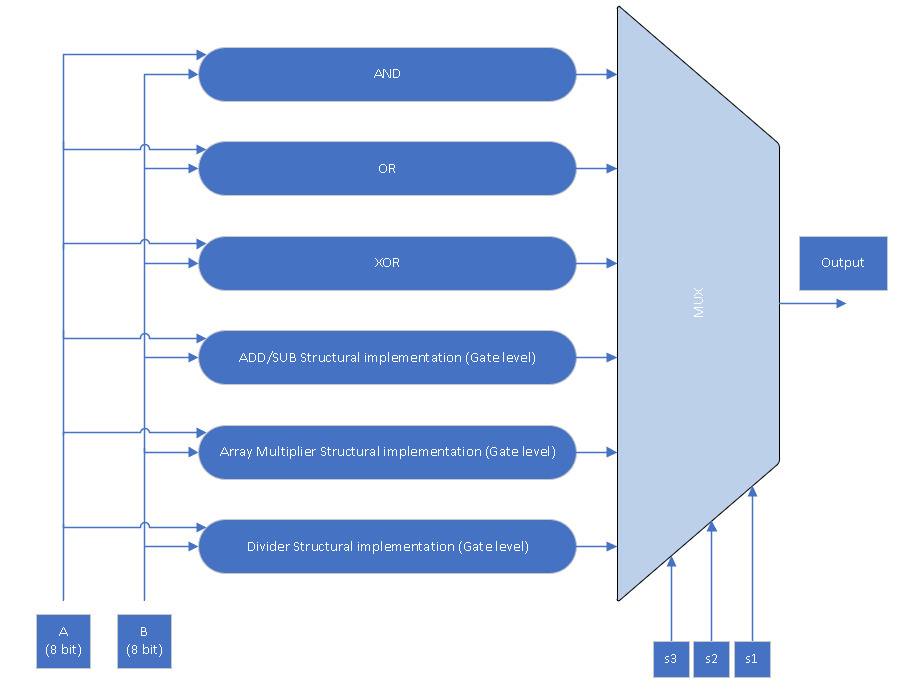


**آزمایش شماره ۹**



**Divider:**

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

entity division is

port (

A : in std\_logic\_vector(7 downto 0);

B : in std\_logic\_vector(7 downto 0);

clk : in std\_logic;

Y : out std\_logic\_vector(7 downto 0);

OverFlow : out std\_logic

);

end division;

architecture Behavioral of division is

type states is (S, D, R);

begin

process (A, B ,clk)

variable EAQ : std\_logic\_vector(8 downto 0);

variable state : states := S;

begin

if rising\_edge(clk) then

case state is

when S =>

EAQ(8) := '0';

EAQ(7 downto 0) := A;

EAQ(8 downto 4):= std\_logic\_vector(unsigned(EAQ(8 downto 4)) + not unsigned(B(3 downto 0)) + 1);

state := D;

when D =>

if (EAQ(8) = '1') then

EAQ(7 downto 4):= std\_logic\_vector(unsigned(EAQ(7 downto 4)) + unsigned(B(3 downto 0)));

Y <= (others => '0');

OverFlow <= '1';

else

EAQ(7 downto 4) := std\_logic\_vector(unsigned(EAQ(7 downto 4)) + unsigned(B(3 downto 0)));

for i in 0 to 3 loop

EAQ := std\_logic\_vector(shift\_left(unsigned(EAQ), 1));

if (EAQ(8) = '0') then

EAQ(8 downto 4):= std\_logic\_vector(unsigned(EAQ(8 downto 4)) + not unsigned(B(3 downto 0)) + 1);

if (EAQ(8) = '0') then

EAQ(0) := '0';

EAQ(7 downto 4):= std\_logic\_vector(unsigned(EAQ(7 downto 4)) + unsigned(B(3 downto 0)));

else

EAQ(0) := '1';

end if;

else

EAQ(8 downto 4):= std\_logic\_vector(unsigned(EAQ(8 downto 4)) + not unsigned(B(3 downto 0)) + 1);

EAQ(0):= '1';

end if;

end loop;

state := R;

end if;

when R =>

Y <= EAQ(7 downto 0);

OverFlow <= '0';

end case;

end if;

end process;

end Behavioral;

**ALU:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

entity ALU is

Port ( A : in STD\_LOGIC\_VECTOR (7 downto 0);

B : in STD\_LOGIC\_VECTOR (3 downto 0);

S : in STD\_LOGIC\_VECTOR (2 downto 0);

Cin : in std\_logic;

clk : in STD\_LOGIC;

Y : out STD\_LOGIC\_VECTOR (7 downto 0);

C : out STD\_LOGIC);

end ALU;

architecture Behavioral of ALU is

component AND\_8bit is

Port ( clk : in std\_logic;

A : in STD\_LOGIC\_VECTOR (7 downto 0);

B : in STD\_LOGIC\_VECTOR (7 downto 0);

Y : out STD\_LOGIC\_VECTOR (7 downto 0)

);

end component;

component XOR\_8bit is

Port ( clk : in std\_logic;

A : in STD\_LOGIC\_VECTOR (7 downto 0);

B : in STD\_LOGIC\_VECTOR (7 downto 0);

Y : out STD\_LOGIC\_VECTOR (7 downto 0));

end component;

component OR\_8bit is

Port ( clk : in std\_logic;

A : in STD\_LOGIC\_VECTOR (7 downto 0);

B : in STD\_LOGIC\_VECTOR (7 downto 0);

Y : out STD\_LOGIC\_VECTOR (7 downto 0));

end component;

component AddSub is

Port (

A : in STD\_LOGIC\_VECTOR (7 downto 0);

B : in STD\_LOGIC\_VECTOR (7 downto 0);

Cin : in STD\_LOGIC;

clk : in std\_logic;

Y : out STD\_LOGIC\_VECTOR (7 downto 0);

Cout : out STD\_LOGIC

);

end component;

component Multiplier\_8bit is

Port (

A : in STD\_LOGIC\_VECTOR (7 downto 0);

B : in STD\_LOGIC\_VECTOR (7 downto 0);

clk : in std\_logic;

Y : out STD\_LOGIC\_VECTOR (15 downto 0)

);

end component;

component division is

port (

A : in std\_logic\_vector(7 downto 0);

B : in std\_logic\_vector(7 downto 0);

clk : in std\_logic;

Y : out std\_logic\_vector(7 downto 0);

OverFlow : out std\_logic

);

end component;

signal XoRR : std\_logic\_vector(7 downto 0);

signal oRR : std\_logic\_vector(7 downto 0);

signal ANDD : std\_logic\_vector(7 downto 0);

signal Add\_Subb : std\_logic\_vector(7 downto 0);

signal Mult : std\_logic\_vector(15 downto 0);

signal divi : std\_logic\_vector(7 downto 0);

signal C\_temp : std\_logic\_vector(1 downto 0);

signal B\_temp : std\_logic\_vector(7 downto 0);

begin

B\_temp <= "0000" & B;

F1 : AND\_8bit port map(clk ,A,B\_temp,ANDD);

F2 : OR\_8bit port map(clk ,A,B\_temp,oRR);

F3 : XOR\_8bit port map(clk ,A,B\_temp,XoRR);

F4 : AddSub port map(A,B\_temp,Cin ,clk,Add\_Subb,C\_temp(0));

F5 : Multiplier\_8bit port map(A,B\_temp,clk,mult);

F6 : division port map(A,B\_temp,clk,divi,C\_temp(1));

process( clk , A, B , S) begin

if rising\_edge(clk)then

case S is

when "000" =>

Y<= ANDD;

C<= '0';

when "001" =>

Y<= oRR;

C <= '0';

when "010" =>

Y<= XoRR;

C<= '0';

when "011" =>

Y<= Add\_Subb;

C <= C\_temp(0);

when "100" =>

Y<= Mult(7 downto 0);

C<= '0';

when "101" =>

Y<= divi;

C<= C\_temp(1);

when others =>

Y <="00000000";

C <= '0';

end case;

end if;

end process;

end Behavioral;

**توضیحات:**

در این آزمایش ما یک ALU طراحی کردیم که یکسری از محاسبات را انجام میدهد.

برای ساخت این ALU ما از یک mux 8:1 استفاده کردیم که سه تا selector دارد به وسیله آن ما operand های مختلف را انتخاب میکنیم. در جدول زیر شماره operand ها مشخص شده است.

|  |  |
| --- | --- |
| Operand | S0 S1 S2 |
| AND | 000 |
| OR | 001 |
| XOR | 010 |
| if Cin = 0 => add / if Cin = 1 => sub | 011 |
| Array multiplier | 100 |
| Divider | 101 |
| Don’t Care | 110 |
| Don’t Care | 111 |

در این حالت با استفاده از selector ها ما هر بار یکی از operand ها رو انتخاب و دو عدد A و B رو که بترتیب ۸ بیتی و ۸ بیتی هستند رو مقدار دهی میکنیم(البته بدلیل کمبود کلید در fpga ما B را ۴ بیتی در نظر میگیریم). البته در حالت 011 ما به Cin نیز نیاز داریم که اگر آن را 0 کردیم عملیات جمع و اگر هم 1 دادیم عملیات تفریق را انجام میدهیم. خروجی حاصل هم برای بعضی از عملیات ها ۱۶ و برای بعضی ها هم ۸ بیتی در نظر میگیرم.

**Simulation**:

