Johannes joujo labb4

This is the labb4 it has 4-bit adder, 4-bit multiplier, 4-bit inverter, 4-bit and, 4-bit or operations.

The cpu uses a clock to work on the operations, it also uses ax_temp, BX, CX and DX as registers to store information.

```
Library IEEE;

use work.all;

use IEEE.STD_LOGIC_1164.all;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

use IEEE.STD_LOGIC_ARITH.all;


entity add4 is

Port (

in_1 : in STD_LOGIC_VECTOR (3 downto 0);

in_2 : in STD_LOGIC_VECTOR (3 downto 0);

s_out : out STD_LOGIC_VECTOR (4 downto 0));

end add4;


architecture Behavioral of add4 is


begin

s_out <= ( '0' & in_1 ) + ( '0' & in_2 );


end Behavioral;


Library IEEE;

use work.all;

use IEEE.STD_LOGIC_1164.all;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

use IEEE.STD_LOGIC_ARITH.all;


entity mul4 is
```

```
Port (
in_1 : in STD_LOGIC_VECTOR (3 downto 0);
in_2 : in STD_LOGIC_VECTOR (3 downto 0);
m_out : out STD_LOGIC_VECTOR (7 downto 0));
end mul4;

architecture behavioral of mul4 is

begin
m_out <= in_1 * in_2 ;

end Behavioral;

Library IEEE;
use work.all;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.all;

entity invrt4 is --
Port (
in_1 : in STD_LOGIC_VECTOR (3 downto 0);
i_out : out STD_LOGIC_VECTOR (3 downto 0));
end invrt4;

architecture behavioral of invrt4 is

begin
i_out <= NOT ( in_1 );
end Behavioral;
```

```
Library IEEE;
use work.all;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.all;

entity bit_and4 is --
Port (
in_1 : in STD_LOGIC_VECTOR (3 downto 0);
in_2 : in STD_LOGIC_VECTOR (3 downto 0);
a_out : out STD_LOGIC_VECTOR (3 downto 0));
end bit_and4;

architecture behavioral of bit_and4 is

begin
a_out <= ( in_1 AND in_2 );
end Behavioral;

Library IEEE;
use work.all;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.all;

entity bit_or4 is
Port (
in_1 : in STD_LOGIC_VECTOR (3 downto 0);
in_2 : in STD_LOGIC_VECTOR (3 downto 0);
o_out : out STD_LOGIC_VECTOR (3 downto 0));
end bit_or4;
```

```
architecture behavioral of bit_or4 is

begin
o_out <= (in_1 OR in_2);
end Behavioral;


Library IEEE;
use work.all;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.all;

entity cpu is
Port (
clk : in STD_LOGIC;
update : in STD_LOGIC;
ax : in STD_LOGIC_VECTOR(3 downto 0);
opcode : in STD_LOGIC_VECTOR(3 downto 0);
result_add : in STD_LOGIC_VECTOR(4 downto 0);
result_mult : in STD_LOGIC_VECTOR(7 downto 0);
result_not : in STD_LOGIC_VECTOR(3 downto 0);
result_and : in STD_LOGIC_VECTOR(3 downto 0);
result_or : in STD_LOGIC_VECTOR(3 downto 0);
A : out STD_LOGIC_VECTOR(3 downto 0);
B : out STD_LOGIC_VECTOR(3 downto 0);
C : out STD_LOGIC_VECTOR(3 downto 0);
D : out STD_LOGIC_VECTOR(3 downto 0);
op_in1 : out STD_LOGIC_VECTOR(3 downto 0);
op_in2 : out STD_LOGIC_VECTOR(3 downto 0) );
```

```
end cpu;
architecture Behave of cpu is
signal BX, CX, DX : STD_LOGIC_VECTOR(3 downto 0);
begin
A <= ax;
B <= BX;
C <= CX;
D <= DX;
clock: process(clk)
begin if
rising_edge(clk)
then if update = '1'
then case
opcode is when "0000" => BX <= AX;
when "0001" => CX <= AX;
when "0010" => DX <= AX;
when "0011" => BX <= "0000";
when "0100" => CX <= "0000";
when "0101" => DX <= "0000";
when "0110" => op_in1 <= "0010";
op_in2 <= "0011";
CX <= result_add(3 downto 0);
DX <= "0101";
when "0111" => op_in1 <= "0010";
op_in2 <= "0011";
CX <= result_mult(3 downto 0);
DX <= "0110";
when "1000" => op_in1 <= "0010";
CX <= result_not(3 downto 0);
DX <= "1101";
when "1001" => op_in1 <= "0010";
```

```
op_in2 <= "0011";
CX <= result_and(3 downto 0);
DX <= "0010";
when "1010" => op_in1 <= "0010";
op_in2 <= "0011";
CX <= result_or(3 downto 0);
DX <= "0011";
when "1011" => op_in1 <= AX;
op_in2 <= BX;
CX <= result_add(3 downto 0);
DX <= "000" & result_add(4);
when "1100" => op_in1 <= AX;
op_in2 <= BX;
CX <= result_mult(7 downto 4);
DX <= result_mult(3 downto 0);
when "1101" => op_in1 <= AX;
CX <= result_not(3 downto 0);
DX <= "0000";
when "1110" => op_in1 <= AX;
op_in2 <= BX;
CX <= result_and(3 downto 0);
DX <= "0000";
when "1111" => op_in1 <= AX;
op_in2 <= BX;
CX <= result_or(3 downto 0);
DX <= "0000";
when others => NULL;
end case;
end if;
end if;
end process clock;
```

```
end;
```

This is the labb4 testbench, this testbench simulates the behavior of the CPU under different conditions and monitors its outputs. The simulation process generates clock pulses and control signals to simulate the CPU's behavior under different conditions.

```
Library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.all;

entity cpu_tb is end cpu_tb;

architecture Behavioral of cpu_tb is

component add4
Port (
in_1 : in STD_LOGIC_VECTOR (3 downto 0);
in_2 : in STD_LOGIC_VECTOR (3 downto 0);
s_out : out STD_LOGIC_VECTOR (4 downto 0));
end component;

component mul4
Port (
in_1 : in STD_LOGIC_VECTOR (3 downto 0);
in_2 : in STD_LOGIC_VECTOR (3 downto 0);
m_out : out STD_LOGIC_VECTOR (7 downto 0));
end component;

component invrt4
```

```
Port (
in_1 : in STD_LOGIC_VECTOR (3 downto 0);
i_out : out STD_LOGIC_VECTOR (3 downto 0));
end component;

component bit_or4
Port (
in_1 : in STD_LOGIC_VECTOR (3 downto 0);
in_2 : in STD_LOGIC_VECTOR (3 downto 0);
o_out : out STD_LOGIC_VECTOR (3 downto 0));
end component;

component bit_and4
Port (
in_1 : in STD_LOGIC_VECTOR (3 downto 0);
in_2 : in STD_LOGIC_VECTOR (3 downto 0);
a_out : out STD_LOGIC_VECTOR (3 downto 0));
end component;

component cpu
Port (
clk : in STD_LOGIC;
update : in STD_LOGIC;
ax : in STD_LOGIC_VECTOR(3 downto 0);
opcode : in STD_LOGIC_VECTOR(3 downto 0);
result_add : in STD_LOGIC_VECTOR(4 downto 0);
result_mult : in STD_LOGIC_VECTOR(7 downto 0);
result_not : in STD_LOGIC_VECTOR(3 downto 0);
result_and : in STD_LOGIC_VECTOR(3 downto 0);
result_or : in STD_LOGIC_VECTOR(3 downto 0);
A : out STD_LOGIC_VECTOR(3 downto 0);
```

```
B : out STD_LOGIC_VECTOR(3 downto 0);
C : out STD_LOGIC_VECTOR(3 downto 0);
D : out STD_LOGIC_VECTOR(3 downto 0);
op_in1 : out STD_LOGIC_VECTOR(3 downto 0);
op_in2 : out STD_LOGIC_VECTOR(3 downto 0) );
end component;

signal clk_in : STD_LOGIC;
signal update_in : STD_LOGIC;
signal ax_in : STD_LOGIC_VECTOR (3 downto 0);
signal opcode_in : STD_LOGIC_VECTOR (3 downto 0);
signal add_1: STD_LOGIC_VECTOR (4 downto 0);
signal mul_1: STD_LOGIC_VECTOR (7 downto 0);
signal inv_1: STD_LOGIC_VECTOR (3 downto 0);
signal and_1: STD_LOGIC_VECTOR (3 downto 0);
signal or_1: STD_LOGIC_VECTOR (3 downto 0);
signal input1 : STD_LOGIC_VECTOR (3 downto 0);
signal input2 : STD_LOGIC_VECTOR (3 downto 0);
signal ax_temp: STD_LOGIC_VECTOR (3 downto 0):="0000";
signal BX, CX, DX : std_logic_vector(3 downto 0):="0000";
begin
cpu_tb : cpu Port Map(
clk => clk_in,
update => update_in, ax => ax_in, opcode => opcode_in, result_add => add_1, result_mult => mul_1,
result_not => inv_1, result_and => and_1, result_or => or_1, A => ax_temp, B => BX, C => CX, D =>
DX, op_in1 => input1, op_in2 => input2 );
u1 : add4 port map ( in_1 => input1, in_2 => input2, s_out => add_1 );
u2 : mul4 port map ( in_1 => input1, in_2 => input2, m_out => mul_1 );
u3 : invrt4 port map ( in_1 => input1, i_out => inv_1 );
u4 : bit_and4 port map ( in_1 => input1, in_2 => input2, a_out => and_1 );
u5 : bit_or4 port map (in_1 => input1, in_2 => input2, o_out => or_1 );
```

```
process variable a_vector : std_logic_vector(3 downto 0):=(others=>'0');
begin
update_in <= '1';
ax_in <= "0000";
for a_vector in 0 to 15 loop opcode_in <= conv_std_logic_vector(a_vector, 4);
clk_in <= '1';
wait for 10 ns;
clk_in <= '0';
wait for 10 ns;
end loop;
end process;
end;
```

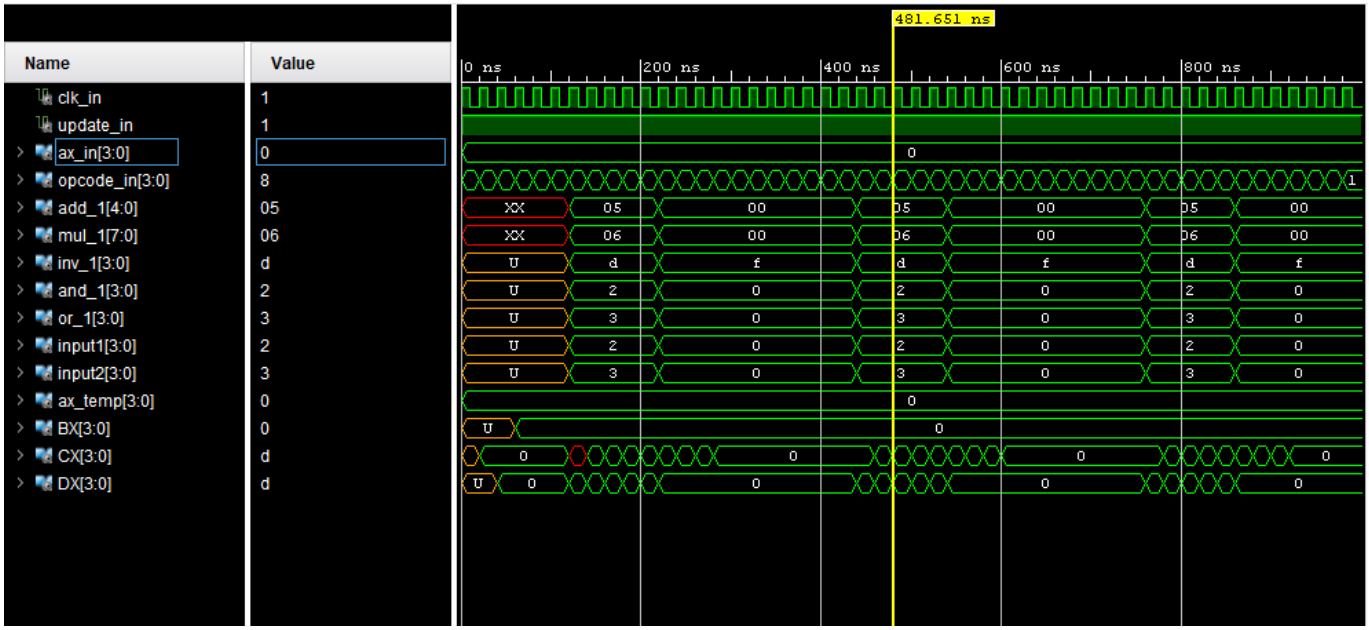

Figure 1 shows the first screenshot I provided where the registers value was not showing.

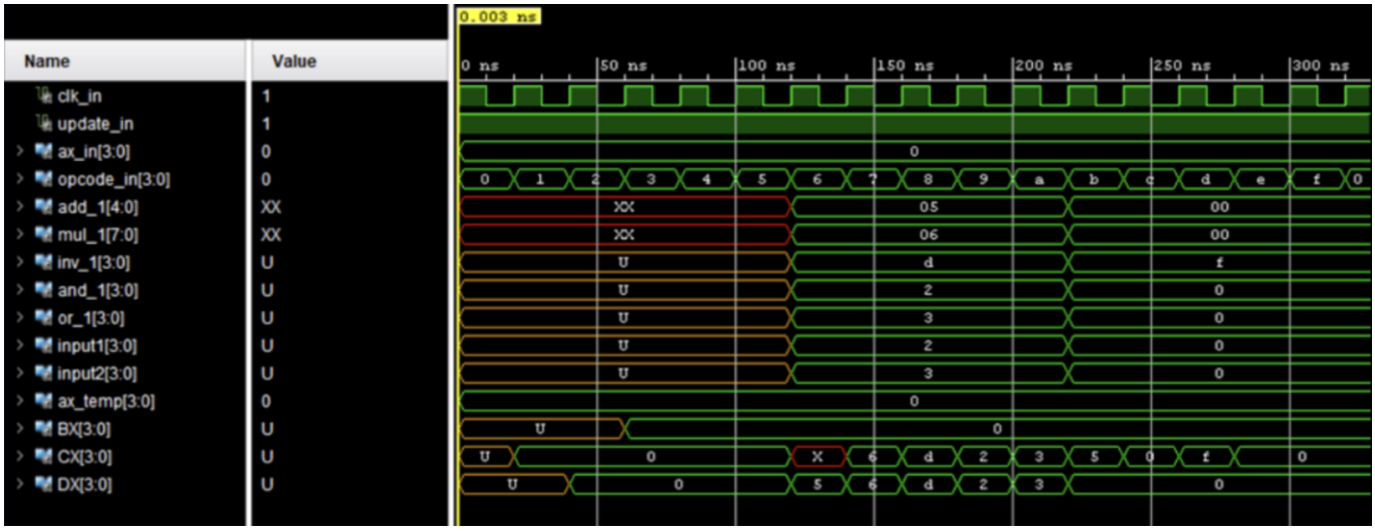

Figure 2 shows the result from the cpu.

Discussion

We can see by figure 2 that in the beginning ax_temp, BX, CX and DX are either 0 or U because they are initialized to “0000” in the testbench but after some operation their value changes. The greatest value we get is $f_{16} = 15_{10}$ because the greatest 4-bit in binary is $1111_2 = 15_{10}$.

How are the components in the cpu interconnected?

The components are interconnected through signals and ports. The specific interconnections depend on the values of input signals from clk, update, opcode and the logic defined in the code for each operation.