ELP305: DESIGN AND SYSTEM LABORATORY

EXPERIMENT 7: Design and implement a simple CPU and interface it to the gcd co-processor

GROUP NUMBER: 22
EXPERIMENT DONE ON: 30/04/2018
SUBMITTED ON: 30/04/2018

SUBMITTED BY:
AVINASH KUMAR(2015MT10319)
CHARVI NAHAR(2015MT10595)
DRASHTI KHATSURIYA(2015MT10598)

Objective: To learn and design a simple CPU. Also, implement it in VHDL.

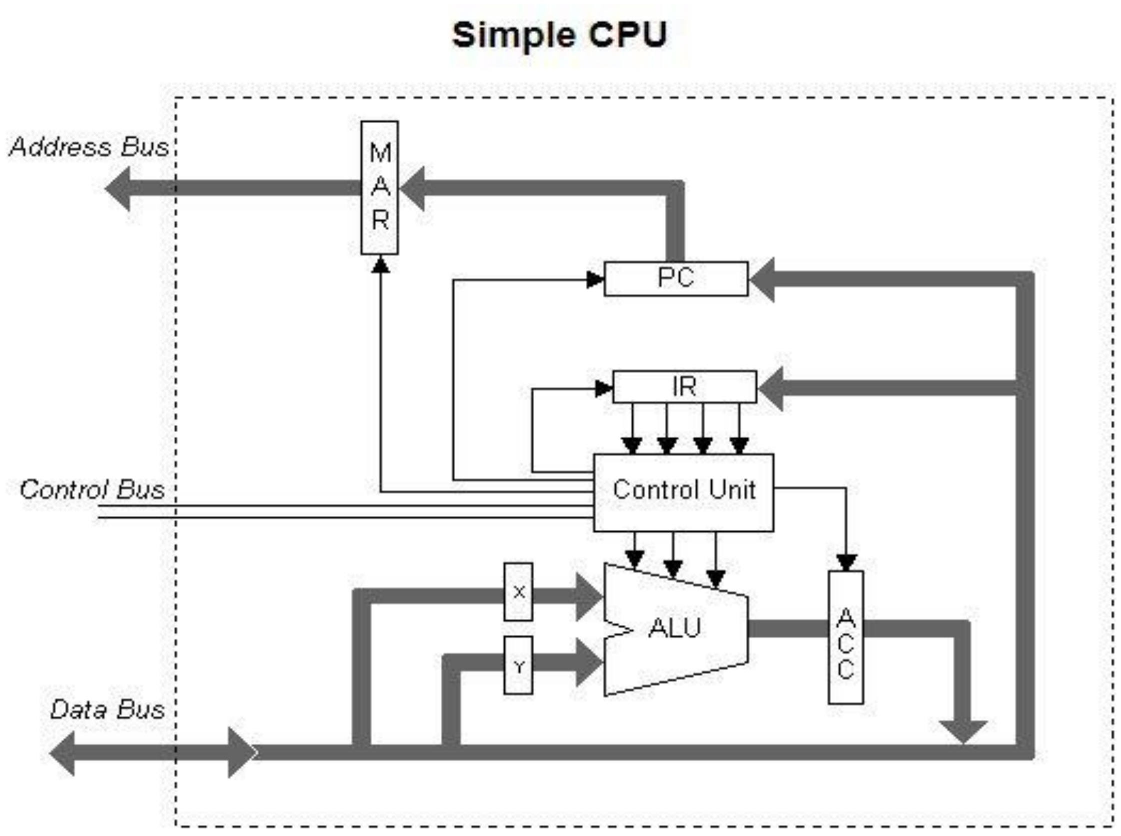
Problem Statement: Design and implement a simple CPU in VHDL. Implement some of the important operations in ALU like: add, subtract, compare, NAND, AND, OR etc. Also interface the gcd co-processor with it. CPU design contain control unit, ALU unit and registers.

Components required: Xilinx Software

Challenges Faced:

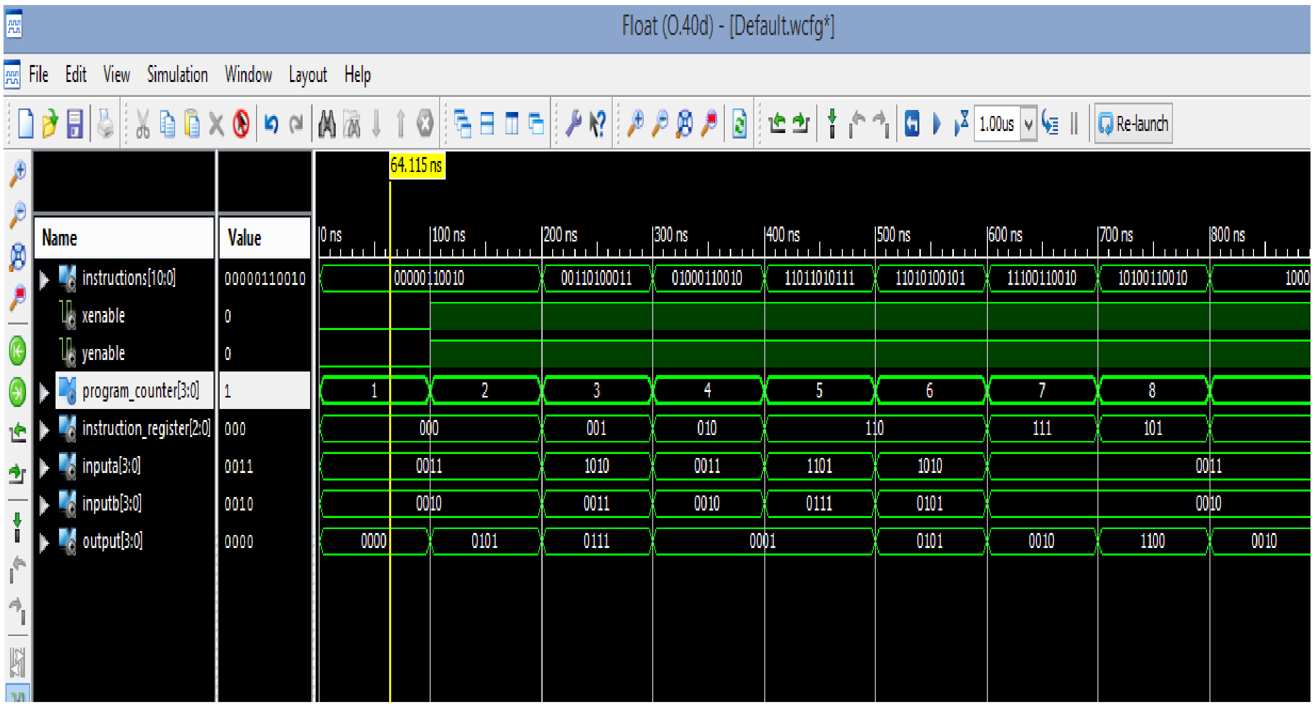
Faced difficulty to integrate more than one VHDL module and run all of them concurrently properly. But after debugging, this is working fine. Otherwise no such challenges faced since we have learnt VHDL language very well.

Block Diagram:



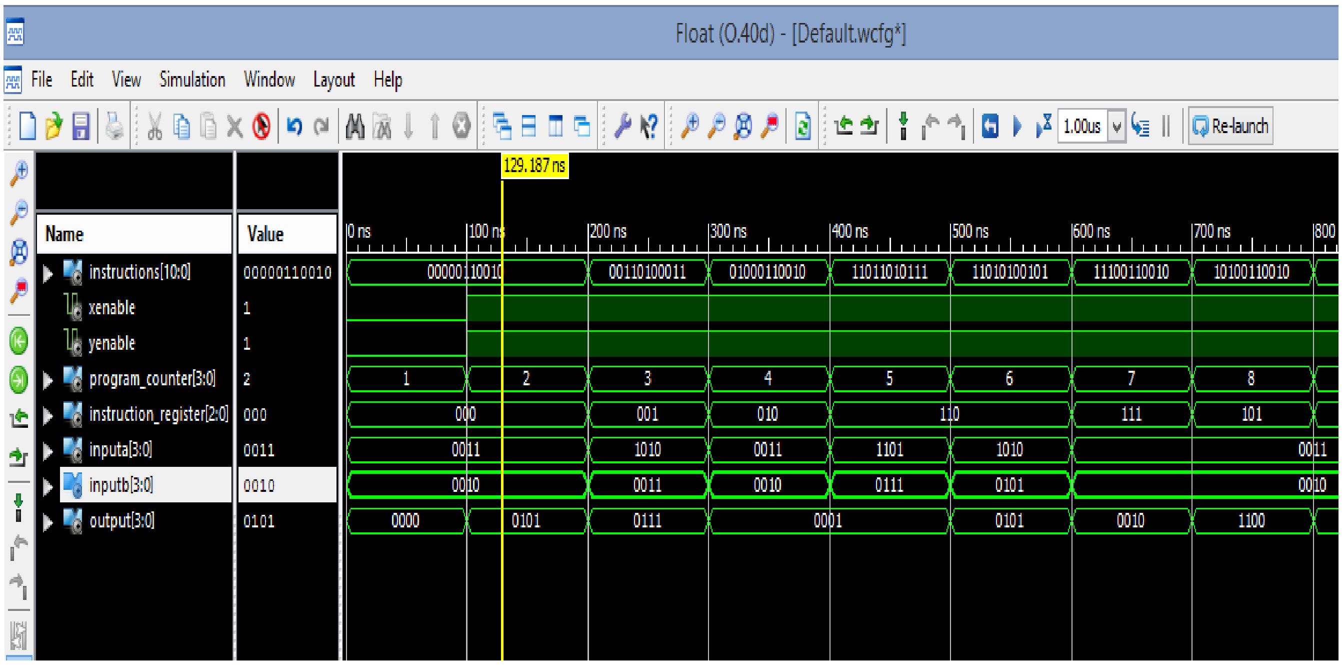
Observations:

FIG: When enable pin of x and y are 0. Means no content is load into registers and output “0000”.



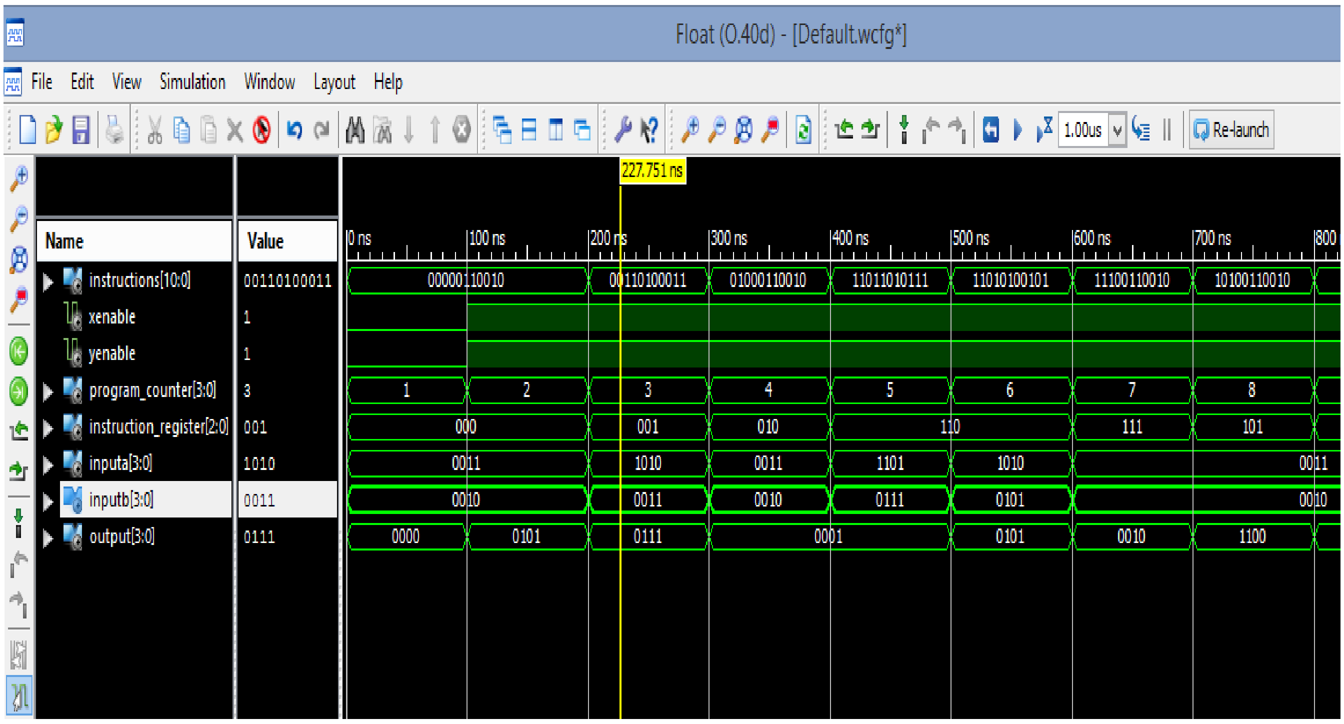
```
64.115 ns
Name                     Value         0 ns      100 ns      200 ns      300 ns      400 ns      500 ns      600 ns      700 ns      800 ns
instructions[10:0]       00000110010        00000110010        00110100011 01000110010 11011010111 11010100101 11100110010 10100110010 1000
xenable                  0
yenable                  0
program_counter[3:0]     1              1          2           3           4           5           6           7           8
instruction_register[2:0] 000                 000              001         010                110              111         101
inputa[3:0]              0011                 0011             1010        0011        1101        1010                    0011
inputb[3:0]              0010                 0010             0011        0010        0111        0101                    0010
output[3:0]              0000           0000       0101        0111                 0001           0101        0010        1100        0010
```

FIG: When enable pin of x and y are 1. Means load the content into the registers and do operation corresponding to opcode “000”. Here 3+2 = 5.


Float (O.40d) - [Default.wcfg*]
File Edit View Simulation Window Layout Help
1.00us
Re-launch
129.187 ns
0 ns
100 ns
200 ns
300 ns
400 ns
500 ns
600 ns
700 ns
800

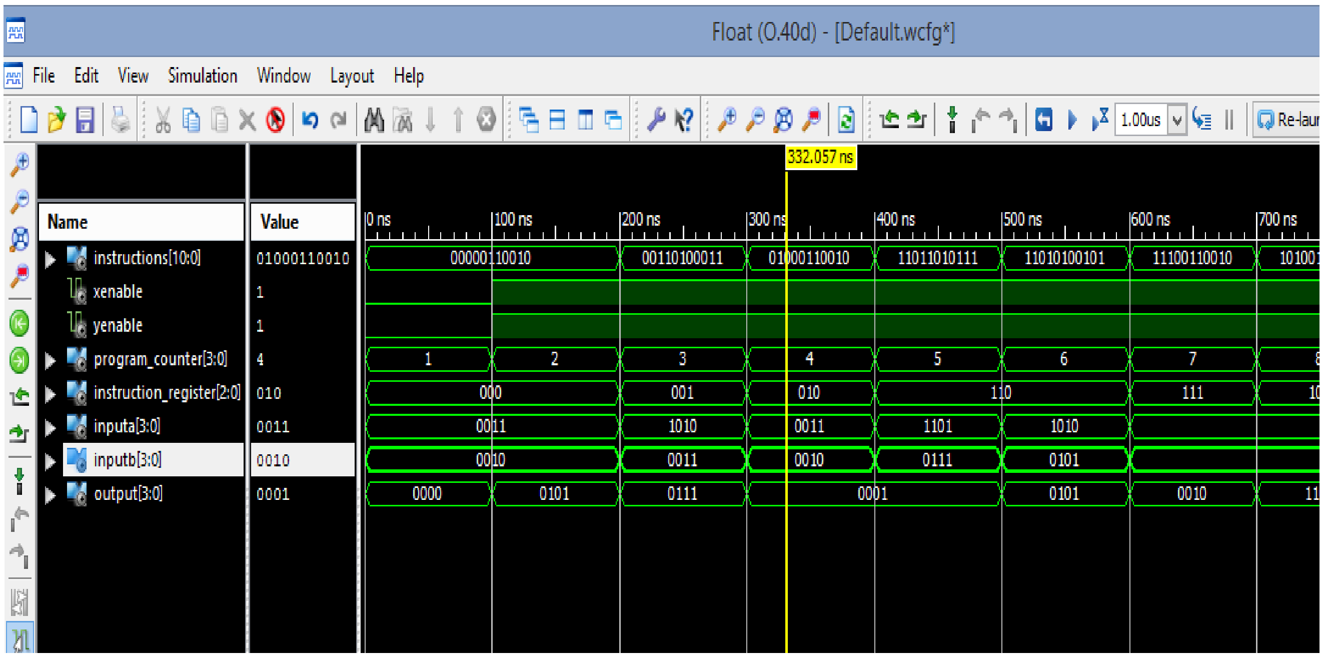
Name	Value
instructions[10:0]	00000110010
xenable	1
yenable	1
program_counter[3:0]	2
instruction_register[2:0]	000
inputa[3:0]	0011
inputb[3:0]	0010
output[3:0]	0101

FIG: When enable pin of x and y are 1. Means load the content into the registers and do operation corresponding to opcode “001”. Here 10-3 = 7.


Float (O.40d) - [Default.wcfg*]
File Edit View Simulation Window Layout Help
1.00us
Re-launch
227.751 ns
0 ns
100 ns
200 ns
300 ns
400 ns
500 ns
600 ns
700 ns
800

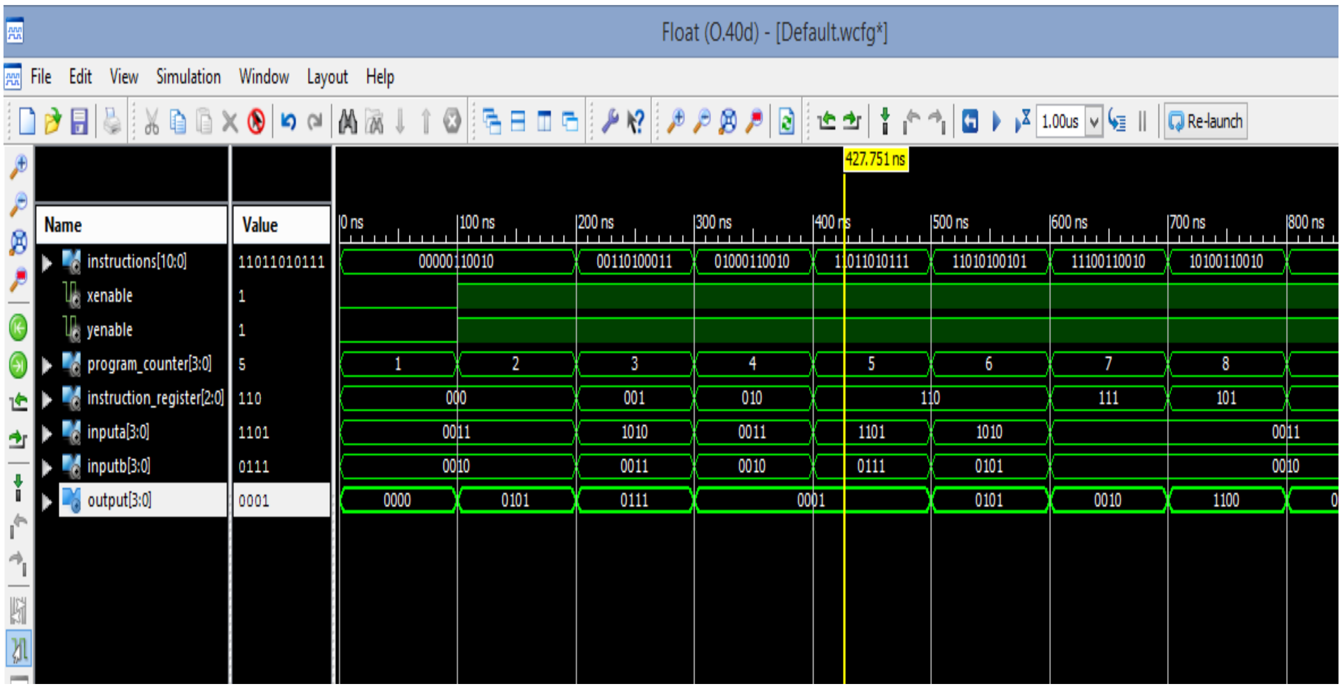
Name	Value
instructions[10:0]	00110100011
xenable	1
yenable	1
program_counter[3:0]	3
instruction_register[2:0]	001
inputa[3:0]	1010
inputb[3:0]	0011
output[3:0]	0111

FIG: When enable pin of x and y are 1. Means load the content into the registers and do operation corresponding to opcode “010”. Here 0011 XOR 0010 = 0001.

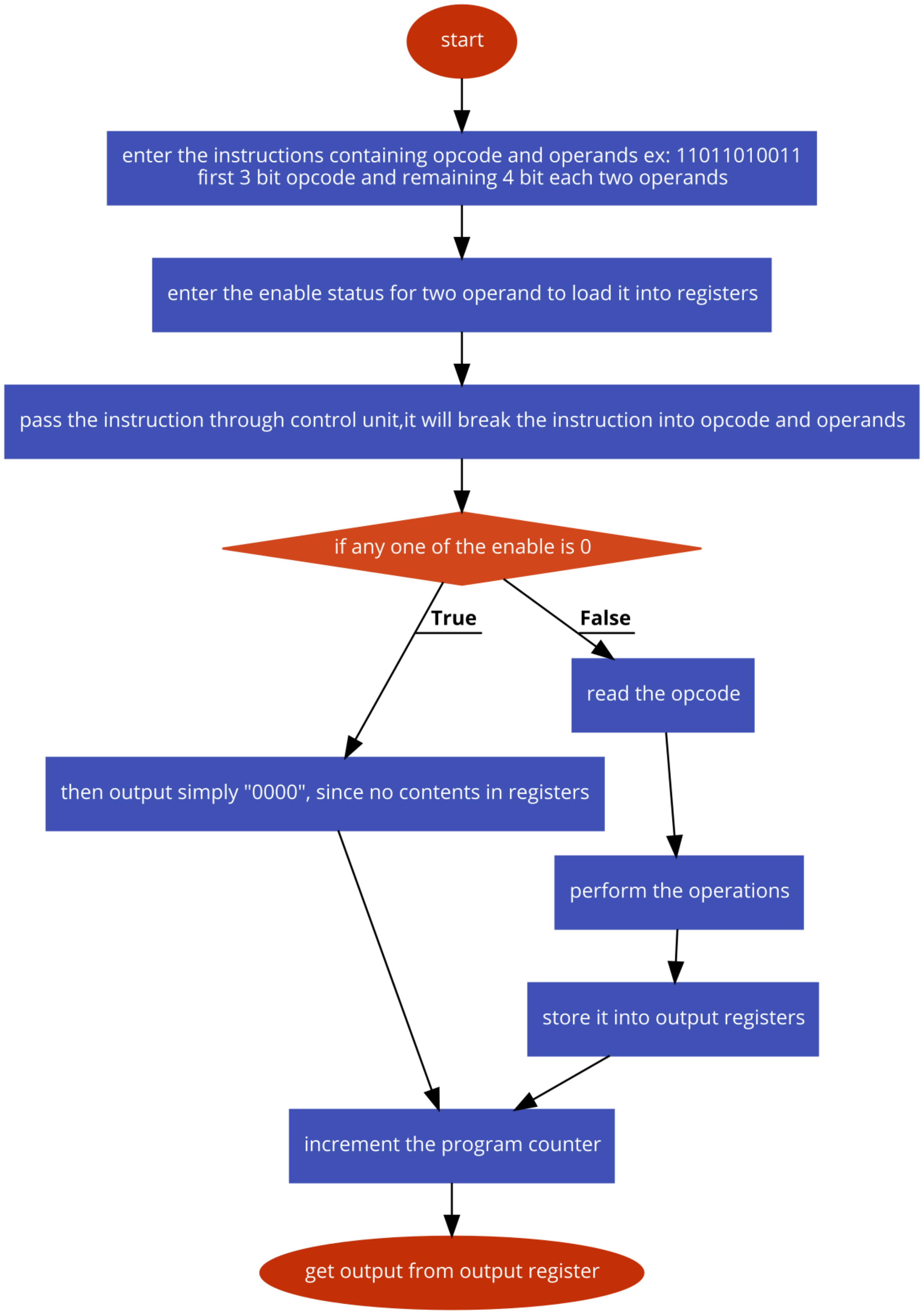


GCD INTERFACE OUTPUT

FIG: When enable pin of x and y are 1. Means load the content into the registers and do operation corresponding to opcode “110”. Here GCD of 13 & 7 = 1(shown in clock cycle 400 – 500 ns). GCD of 10 & 5 = 5. (shown in clock cycle 500 – 600 ns).



Flowchart


start
enter the instructions containing opcode and operands ex: 11011010011
first 3 bit opcode and remaining 4 bit each two operands
enter the enable status for two operand to load it into registers
pass the instruction through control unit,it will break the instruction into opcode and operands
if any one of the enable is 0
True
False
read the opcode
then output simply "0000", since no contents in registers
perform the operations
store it into output registers
increment the program counter
get output from output register

Discussions

Our CPU contains Control unit, Alu unit and registers. You can see our implementations. We have design some of the important operations in our ALU design and interface the gcd co-processor in it. We have done operations like: Add, Sub, Compare, XOR, AND, GCD Calculations, etc. We have also designed control unit which take instructions which consists of opcode and operands and break into three things opcode, Input A, Input B. and then this opcode load into instruction registers and operands load into registers according to enable pin status and ALU perform this operation and give the output into output registers.

Results

I have also shown some the results like: Add, Sub, XOR, gcd of two number above in observation. See the content written in FIG of each observations.

References

1. https://www.xilinx.com/itp/xilinx10/books/docs/qst/qst.pdf
2. http://esd.cs.ucr.edu/labs/tutorial/
3. https://www.youtube.com/watch?v=R-7u0NMLrpQ&t=10s
4. https://www.youtube.com/watch?v=-y3sFaFqSq4&t=15s

Code

Control_unit.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity control_unit is
	PORT( instruction : in STD_LOGIC_VECTOR(10 DOWNTO 0);
		opcode : out STD_LOGIC_VECTOR(2 DOWNTO 0);
		input_A : out STD_LOGIC_VECTOR(3 DOWNTO 0);
		input_B : out STD_LOGIC_VECTOR(3 DOWNTO 0));
end control_unit;
```

```
architecture Behavioral of control_unit is
begin

input_B <= instruction(3 DOWNTO 0);
input_A <= instruction(7 DOWNTO 4);
opcode <= instruction(10 DOWNTO 8);

end Behavioral;
```

alu_unit.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;

entity alu_unit is
        PORT( x_enable : in std_logic;
                y_enable : in std_logic;
                opcode_val : in std_logic_vector(2 DOWNTO 0);
                register_x : in std_logic_vector(3 DOWNTO 0);
                register_y : in std_logic_vector(3 DOWNTO 0);
                output_register : out std_logic_vector(3 DOWNTO 0));
end alu_unit;

architecture Behavioral of alu_unit is
begin

process(x_enable,y_enable,register_x,register_y,opcode_val)

variable xv,yv: STD_LOGIC_VECTOR(3 DOWNTO 0);

begin
        if(x_enable = '0' AND y_enable = '0') then
                output_register <= "0000";
        elsif(x_enable = '1' AND y_enable = '0') then
                output_register <= "0000";
        elsif(x_enable = '0' AND y_enable = '1') then
                output_register <= "0000";
        else
                case opcode_val is
                        when "000" =>
                                output_register <= register_x + register_y;
                        when "001" =>
                                output_register <= register_x - register_y;
                        when "010" =>
                                output_register <= register_x XOR register_y;
                        when "011" =>
                                output_register <= register_x OR register_y;
                        when "100" =>
                                output_register <= register_x AND register_y;
                        when "101" =>
                                output_register <= NOT register_x;
                        when "110" =>
                                xv := register_x;
                                yv:= register_y;

                                while(xv /= yv) loop
                                        if xv < yv then
                                                yv:= yv - xv;  ---- gcd co-processor
                                        else
                                                xv:= xv - yv;
                                        end if;
                                end loop;

                                output_register <= xv;
                        when "111" =>
                                if(register_x > register_y) then
                                        output_register<= "0010";
                                elsif(register_x < register_y) then
                                        output_register<= "0001";
                                else
                                        output_register<="0011";
                                end if;
                        when others =>
                                output_register <= "0000";
                end case;
        end if;
end process;

end Behavioral;
```

cpu.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity cpu is
        PORT( instructions : in std_logic_vector(10 DOWNTO 0);
                xenable : in std_logic;
                yenable : in std_logic;
                instruction_register : inout std_logic_vector(2 DOWNTO 0);
                inputA : inout std_logic_vector(3 DOWNTO 0);
                inputB : inout std_logic_vector(3 DOWNTO 0);
                output : out std_logic_vector(3 DOWNTO 0);
                program_counter : in std_logic_vector(3 DOWNTO 0));
end cpu;

architecture Behavioral of cpu is

COMPONENT control_unit is
        PORT( instruction : in std_logic_vector(10 DOWNTO 0);
                 opcode : out std_logic_vector(2 DOWNTO 0);
                 input_A: out std_logic_vector(3 DOWNTO 0);
                 input_B : out std_logic_vector(3 DOWNTO 0));
end component;

COMPONENT alu_unit is
        PORT( x_enable : in std_logic;
                 y_enable : in std_logic;
                 opcode_val : in std_logic_vector(2 DOWNTO 0);
                 register_x : in std_logic_vector(3 DOWNTO 0);
                 register_y : in std_logic_vector(3 DOWNTO 0);
                 output_register : out std_logic_vector(3 DOWNTO 0));
end component;

begin
process1 : control_unit port map(instruction => instructions,opcode =>
instruction_register, input_A => inputA, input_B => inputB);

process2 : alu_unit port map(x_enable => xenable,y_enable => yenable,
opcode_val => instruction_register,register_x => inputA, register_y =>
inputB,output_register => output);

end Behavioral;
```

cpu_test_bench.vhd

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

ENTITY cpu_test_bench IS
END cpu_test_bench;

ARCHITECTURE behavior OF cpu_test_bench IS

    COMPONENT cpu
    PORT(
         instructions : IN  std_logic_vector(10 downto 0);
         xenable : IN  std_logic;
         yenable : IN  std_logic;
         instruction_register : INOUT  std_logic_vector(2 downto 0);
         inputA : INOUT  std_logic_vector(3 downto 0);
         inputB : INOUT  std_logic_vector(3 downto 0);
         output : OUT  std_logic_vector(3 downto 0);
         program_counter : IN  std_logic_vector(3 downto 0)
        );
    END COMPONENT;

   --Inputs
   signal instructions : std_logic_vector(10 downto 0) := (others => '0');
   signal xenable : std_logic := '0';
   signal yenable : std_logic := '0';
   signal program_counter : std_logic_vector(3 downto 0) := (others => '0');

--BiDirs
   signal instruction_register : std_logic_vector(2 downto 0);
   signal inputA : std_logic_vector(3 downto 0);
   signal inputB : std_logic_vector(3 downto 0);

 --Outputs
   signal output : std_logic_vector(3 downto 0);
```

```
BEGIN

        -- Instantiate the Unit Under Test (UUT)
  uut: cpu PORT MAP (
       instructions => instructions,
       xenable => xenable,
       yenable => yenable,
       instruction_register => instruction_register,
       inputA => inputA,
       inputB => inputB,
       output => output,
       program_counter => program_counter
     );


  -- Stimulus process
  stim_proc_instruction: process
  begin
        instructions <= "00000110010";
        program_counter <= "0001";
    -- hold reset state for 100 ns.
    wait for 100 ns;
                        instructions <= "00000110010";
                        program_counter <= "0010";
    -- hold reset state for 100 ns.
    wait for 100 ns;
                        instructions <= "00110100011";
                        program_counter <= "0011";
    -- hold reset state for 100 ns.
    wait for 100 ns;
                        instructions <= "01000110010";
                        program_counter <= "0100";
    -- hold reset state for 100 ns.
                wait for 100 ns;
                        instructions <= "11011010111";
                        program_counter <= "0101";
    -- hold reset state for 100 ns.
                wait for 100 ns;
                        instructions <= "11010100101";
                        program_counter <= "0110";
    -- hold reset state for 100 ns.
                wait for 100 ns;
```

```
                     instructions <= "11100110010";
                     program_counter <= "0111";
  -- hold reset state for 100 ns.
             wait for 100 ns;
                     instructions <= "10100110010";
                     program_counter <= "1000";
  -- hold reset state for 100 ns.
             wait for 100 ns;
                     instructions <= "10000110010";
                     program_counter <= "1001";
  -- hold reset state for 100 ns.
             wait for 100 ns;
                     instructions <= "10000110010";
                     program_counter <= "1001";
  -- hold reset state for 100 ns.
      wait;
end process;

stim_proc_xenable: process
begin
      xenable <= '0';
  -- hold reset state for 100 ns.
  wait for 100 ns;
                     xenable <= '1';
  -- hold reset state for 100 ns.
  wait for 100 ns;
                     xenable <= '1';
  -- hold reset state for 100 ns.
  wait for 100 ns;
                     xenable <= '1';
                     wait for 100 ns;
                     xenable <= '1';
  -- hold reset state for 100 ns.
             wait for 100 ns;
                     xenable <= '1';
  -- hold reset state for 100 ns.
             wait for 100 ns;
                     xenable <= '1';
  -- hold reset state for 100 ns.
             wait for 100 ns;
                     xenable <= '1';
  -- hold reset state for 100 ns.
```

```
                wait for 100 ns;
                        xenable <= '1';
    -- hold reset state for 100 ns.
                wait for 100 ns;
                        xenable <= '0';
    -- hold reset state for 100 ns.
    wait;
  end process;

stim_proc_yenable: process
  begin
                        yenable <= '0';
    -- hold reset state for 100 ns.
    wait for 100 ns;
                        yenable <= '1';
    -- hold reset state for 100 ns.
    wait for 100 ns;
                        yenable <= '1';
    -- hold reset state for 100 ns.
    wait for 100 ns;
                        yenable <= '1';
                        wait for 100 ns;
                        yenable <= '1';
    -- hold reset state for 100 ns.
                wait for 100 ns;
                        yenable <= '1';
    -- hold reset state for 100 ns.
                wait for 100 ns;
                        yenable <= '1';
    -- hold reset state for 100 ns.
                wait for 100 ns;
                        yenable <= '1';
    -- hold reset state for 100 ns.
                wait for 100 ns;
                        yenable <= '1';
    -- hold reset state for 100 ns.
                wait for 100 ns;
                        yenable <= '1';
    -- hold reset state for 100 ns.
    wait;
  end process;
END;
```