Practical 1 Alu

MvM

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



use IEEE.NUMERIC_STD.ALL;


entity ALU4 is
  Port ( A : in  STD_LOGIC_VECTOR(3 downto 0);
        B : in  STD_LOGIC_VECTOR(3 downto 0);
        C_B : out  STD_LOGIC;
        Y : out  STD_LOGIC_VECTOR(3 downto 0);
        F : in  STD_LOGIC_VECTOR(2 downto 0)
 );
end ALU4;


architecture Behavioral of ALU4 is
signal result : STD_LOGIC_VECTOR (4 downto 0) := "00000";
begin
process(A, B, F)
begin
case F is
when "000" => result <= '0' & (A AND B);
when "001" => result <= '0' & (A NAND B);
when "010" => result <= '0' & (A OR B);
when "011" => result <= '0' & (A NOR B);
when "100" => result <= '0' & (A XOR B);
when "101" => result <= '0' & (A XNOR B);
when "110" => result <= ('0' & A)+('0' & B);
when others =>
if(A < B) then
result <= '0' & (not B);
result <= result + 1;
result <= (NOT(('0' & A) + ('0' & (not B)) + 1)) + 1;
else result <= ('0' & A) - ('0' & B);
end if;
end case;
end process;
Y <= result(3 downto 0);
C_B <= result(4);
end Behavioral;
```

TvM

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD.ALL;
 ENTITY alutest IS
END alutest;
ARCHITECTURE behavior OF alutest IS
 COMPONENT ALU4
    PORT(
         A : IN  std_logic_vector(3 downto 0);
         B : IN  std_logic_vector(3 downto 0);
         C_B : OUT  std_logic;
         Y : OUT  std_logic_vector(3 downto 0);
         F : IN  std_logic_vector(2 downto 0)
        );
```

```
   END COMPONENT;



  signal A : std_logic_vector(3 downto 0) := ("1001");
  signal B : std_logic_vector(3 downto 0) := ("1010");
  signal F : std_logic_vector(2 downto 0) := ("111");
 signal C_B : std_logic;
  signal Y : std_logic_vector(3 downto 0);


BEGIN
 uut: ALU4 PORT MAP (
          A => A,
          B => B,
          C_B => C_B,
          Y => Y,
          F => F
        );
  stim_proc: process
  begin
     F <= F+1;
wait for 100 ns;
  end process;

END;
```

2 .Usr

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



entity usr is
   Port ( rst : in  STD_LOGIC;
          clk : in  STD_LOGIC;
          mode : in  STD_LOGIC_VECTOR(1 downto 0);
          sin : in  STD_LOGIC;
          pin : in  STD_LOGIC_VECTOR(3 downto 0);
          sout : out  STD_LOGIC;
          pout : out  STD_LOGIC_VECTOR(3 downto 0));
end usr;

architecture Behavioral of usr is
SIGNAL temp : STD_LOGIC_VECTOR(3 downto 0) := "0000";
begin
PROCESS(rst, clk, mode, sin, pin)
BEGIN
if rst = '1' then
pout<="0000";
sout<='0';
ELSIF FALLING_EDGE(clk) THEN
CASE mode IS
WHEN "00" =>
temp(3 downto 1) <= temp(2 downto 0);
temp(0) <= sin;
sout <= temp(3);
pout <= "0000";
WHEN "01" =>
```

```
temp(3 downto 1) <= temp(2 downto 0);

temp(0) <= sin;

pout <= temp;

sout <= '0';

WHEN "10" =>

temp <= pin;

sout <= temp(3);

temp(3 downto 1) <= temp(2 downto 0);

pout <= "0000";

WHEN OTHERS =>

temp <= pin;

pout <= temp;

END CASE;

END IF;

END PROCESS;

end Behavioral;
```

TvM

```
LIBRARY ieee;

USE ieee.std_logic_1164.ALL;

 ENTITY usr_tb IS

END usr_tb;

 ARCHITECTURE behavior OF usr_tb IS

COMPONENT usr

    PORT(

         rst : IN  std_logic;

         clk : IN  std_logic;

         mode : IN  std_logic_vector(1 downto 0);

         sin : IN  std_logic;

         pin : IN  std_logic_vector(3 downto 0);
```

```
        sout : OUT  std_logic;
        pout : OUT  std_logic_vector(3 downto 0)
       );
    END COMPONENT;



   --Inputs
   signal rst : std_logic := '0';
   signal clk : std_logic := '1';
   signal mode : std_logic_vector(1 downto 0) := (others => '0');
   signal sin : std_logic := '0';
   signal pin : std_logic_vector(3 downto 0) := "0000";



   signal sout : std_logic;
   signal pout : std_logic_vector(3 downto 0);



   constant clk_period : time := 10 ns;

BEGIN

   uut: usr PORT MAP (
         rst => rst,
         clk => clk,
         mode => mode,
         sin => sin,
         pin => pin,
         sout => sout,
         pout => pout
       );
```

```
  clk_process :process
  begin
clk <= '0';
wait for clk_period/2;
clk <= '1';
wait for clk_period/2;
  end process;
rstproc : process
begin
wait for 162.5 ns;
rst <= '1';
wait for 20 ns;
rst <= '0';
wait;
end process;

  mode_proc: process
  begin
wait for 80 ns;
mode <= "00";
wait for 50 ns;
mode <= "01";
wait for 50 ns;
mode <= "10";
wait for 20 ns;
mode <= "11";
wait;
  end process;
sin_proc: process
```

```
   begin
wait for 10 ns;
sin <= '0';
wait for 10 ns;
sin <= '1';
wait for 10 ns;
sin <= '1';
wait for 10 ns;
sin <= '0';
wait for 50 ns;
sin <= '1';
wait for 10 ns;
sin <= '1';
wait for 10 ns;
sin <= '0';
wait for 10 ns;
sin <= '1';
wait;
   end process;

END;
```

3. Mod_N_counter

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD.ALL;
library UNISIM;
use UNISIM.VComponents.all;
```

```
entity MOD_N_COUNTER is
    Port ( rst : in  STD_LOGIC;
           pr : in  STD_LOGIC;
           dir : in  STD_LOGIC;
  clk : in STD_LOGIC;
           Q : out  STD_LOGIC_VECTOR (4 downto 0));
end MOD_N_COUNTER;

architecture Behavioral of MOD_N_COUNTER is
signal Qtemp : STD_LOGIC_VECTOR (4 downto 0) := "00000";
begin
process(rst, pr, clk, dir)
begin
if rst = '1' then
Qtemp <= (OTHERS => '0');
elsif pr = '1' then
Qtemp <= (OTHERS => '1');
elsif falling_edge(clk) then
if dir='1' then
if Qtemp < 24 then
Qtemp <= Qtemp + 1;
else
Qtemp <= "00000";
end if;
else
if Qtemp > 7 then
Qtemp <= Qtemp - 1;
else
Qtemp <= "11111";
end if;
end if;
end if;
end process;
Q <= Qtemp;
end Behavioral;
```

TvM

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



USE ieee.numeric_std.ALL;


ENTITY modNcounterTB IS
END modNcounterTB;


ARCHITECTURE behavior OF modNcounterTB IS




   COMPONENT MOD_N_COUNTER
   PORT(
       rst : IN  std_logic;
       pr : IN  std_logic;
       dir : IN  std_logic;
       clk : IN  std_logic;
       Q : OUT  std_logic_vector(4 downto 0)
      );
   END COMPONENT;
   signal rst : std_logic := '0';

   signal pr : std_logic := '0';

   signal dir : std_logic := '0';

   signal clk : std_logic := '0';



   signal Q : std_logic_vector(4 downto 0);



   constant clk_period : time := 10 ns;

BEGIN

   uut: MOD_N_COUNTER PORT MAP (
          rst => rst,
          pr => pr,
          dir => dir,
          clk => clk,
          Q => Q
        );



   clk_process :process
   begin
clk <= '0';
wait for clk_period/2;
clk <= '1';
wait for clk_period/2;
```

```
  end process;



  rst_process: process
  begin
wait for 212.5 ns;
rst<='1';
wait for 10 ns;
rst<='0';
end process;
dir_process :process
begin
wait for 250 ns;
dir<='1';
wait for 250 ns;
dir<='0';
end process;
pr_process :process
begin
wait for 282.5 ns;
pr<='1';
wait for 10 ns;
pr<='0';
end process;
END;
```

4. Fifo

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

```
entity fifo is
   generic (depth : integer := 16);
   port (
      clk       : in  std_logic;
      reset     : in  std_logic;
      enr       : in  std_logic;
      enw       : in  std_logic;
      data_in   : in  std_logic_vector(7 downto 0);
      fifo_empty : out std_logic;
      fifo_full  : out std_logic
   );
end fifo;

architecture fifo_arch of fifo is
   type memory_type is array (0 to depth-1) of std_logic_vector(7 downto 0);
   signal memory : memory_type := (others => (others => '0'));
   signal readptr, writeptr : integer := 0;
   signal empty, full : std_logic := '0';
begin
   fifo_empty <= empty;
   fifo_full  <= full;

   process (clk, reset)
      variable num_elem : integer := 0;
   begin
      if (reset = '1') then
         memory <= (others => (others => '0'));
         data_out <= (others => '0');
         empty <= '1';
         full <= '0';
```

```
    readptr <= 0;

    writeptr <= 0;

    num_elem := 0;

elsif (rising_edge(clk)) then


    if (enr = '1' and empty = '0') then

        data_out <= memory(readptr);

        readptr <= (readptr + 1) mod depth;

        num_elem := num_elem - 1;

    end if;




    if (enw = '1' and full = '0') then

        memory(writeptr) <= data_in;

        writeptr <= (writeptr + 1) mod depth;

        num_elem := num_elem + 1;

    end if;




    if (num_elem = 0) then

        empty <= '1';

    else

        empty <= '0';

    end if;


    if (num_elem = depth) then

        full <= '1';

    else

        full <= '0';

    end if;

end if;
```

```
    end process;
end fifo_arch;


TvM
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY fifo_tb IS
END fifo_tb;

ARCHITECTURE behavior OF fifo_tb IS
    -- Inputs and outputs
    signal Clk, reset, enr, enw : std_logic := '0';
    signal empty, full          : std_logic;
    signal data_in, data_out    : std_logic_vector(7 downto 0) := (others => '0');

    -- Temporary signals
    signal i : integer := 0;

    -- Clock period definition
    constant Clk_period : time := 10 ns;
    constant depth      : integer := 16;  -- Specify depth of FIFO here.

BEGIN

    -- Instantiate the Unit Under Test (UUT)
    uut: entity work.fifo
        generic map (depth => depth)
        port map (
            Clk       => Clk,
        reset      => reset,
        enr        => enr,
        enw        => enw,
        data_in    => data_in,
        data_out   => data_out,
        fifo_empty => empty,
        fifo_full  => full
    );

-- Clock process definition
Clk_process : process
begin
    Clk <= '0';
    wait for Clk_period / 2;
    Clk <= '1';
    wait for Clk_period / 2;
end process;

-- Stimulus process
stim_proc: process
begin
    -- Apply reset for one clock cycle
    reset <= '1';
    wait for Clk_period;
    reset <= '0';

    -- Wait for 3 clock periods
    wait for Clk_period * 3;

    -- Write 10 values to FIFO
    enw <= '1';
```

```
enr <= '0';
for i in 1 to 10 loop
   data_in <= std_logic_vector(to_unsigned(i, 8));  -- Use to_unsigned for conversion
   wait for Clk_period;
end loop;

-- Stop writing, start reading 4 values from FIFO
enw <= '0';
enr <= '1';
wait for Clk_period * 4;

-- Stop read, wait
enw <= '0';
enr <= '0';
wait for Clk_period * 10;

-- Write 10 more values to FIFO
enw <= '1';
enr <= '0';
for i in 11 to 20 loop
   data_in <= std_logic_vector(to_unsigned(i, 8));
   wait for Clk_period;
end loop;

-- Wait and read a few values
enw <= '0';
enr <= '1';
wait for Clk_period * 4;

-- Read additional values
enw <= '0';
```

```
        enr <= '1';
        wait for Clk_period * 8;

        -- Read the remaining values
        enw <= '0';
        enr <= '1';
        wait for Clk_period * 4;

        -- Stop everything
        enw <= '0';
        enr <= '0';
        wait;
    end process;

END behavior;
```

5.LCD

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

entity LCD_FSM is
    Port (
        rst      : in std_logic;             -- Reset signal
        clk_12Mhz: in std_logic;             -- High frequency clock (12 MHz
        lcd_rs   : out std_logic;            -- LCD RS contr
        lcd_en   : out std_logic;            -- LCD Enable
        lcd_data : out std_logic_vector(7 downto 0)  -- LCD Data port
    );
```

```
end LCD_FSM;


architecture Behavioral of LCD_FSM is
   signal div       : std_logic_vector(15 downto 0);  -- Delay timer
   signal clk_fsm   : std_logic;
   signal lcd_rs_s  : std_logic;

   -- LCD controller FSM states
   type state is (reset, func, mode, cur, clear, d0, d1, d2, d3, d4, hold);
   signal ps1, nx   : state;
   signal dataout_s : std_logic_vector(7 downto 0);  -- Internal data command multiplexer
begin

   -- Clock divider process
   clk_divider: process(rst, clk_12Mhz)
   begin
      if (rst = '1') then
         div <= (others => '0');
      elsif (clk_12Mhz'event and clk_12Mhz = '1') then
         div <= div + 1;
      end if;
   end process;

   clk_fsm <= div(15);

   -- Present state register process
   process(rst, clk_fsm)
   begin
      if (rst = '1') then
         ps1 <= reset;
      elsif (rising_edge(clk_fsm)) then
```

```
        ps1 <= nx;
    end if;
end process;

-- State and output decoding process
process(ps1)
begin
    case ps1 is
        when reset =>
            nx       <= func;
            lcd_rs_s  <= '0';
            dataout_s <= "00111000";  -- 38h (Function Set)

        when func =>
            nx       <= mode;
            lcd_rs_s  <= '0';
            dataout_s <= "00111000";  -- 38h (Function Set)

        when mode =>
            nx       <= cur;
            lcd_rs_s  <= '0';
            dataout_s <= "00000110";  -- 06h (Entry Mode Set)

        when cur =>
            nx       <= clear;
            lcd_rs_s  <= '0';
            dataout_s <= "00001100";  -- 0Ch (Display ON, Cursor OFF)

        when clear =>
            nx       <= d0;
            lcd_rs_s  <= '0';
```

```
   dataout_s <= "00000001";  -- 01h (Clear Display)

when d0 =>
   lcd_rs_s  <= '1';
   dataout_s <= "01010000";  -- 'P' (Decimal = 80, HEX = 50)
   nx       <= d1;

when d1 =>
   lcd_rs_s  <= '1';
   dataout_s <= "01001001";  -- 'I' (Decimal = 73, HEX = 49)
   nx       <= d2;

when d2 =>
   lcd_rs_s  <= '1';
   dataout_s <= "01000011";  -- 'C' (Decimal = 67, HEX = 43)
   nx       <= d3;

when d3 =>
   lcd_rs_s  <= '1';
   dataout_s <= "01010100";  -- 'T' (Decimal = 84, HEX = 54)
   nx       <= d4;

when d4 =>
   lcd_rs_s  <= '1';
   dataout_s <= "00100000";  -- Space (Decimal = 32, HEX = 20)
   nx       <= hold;

when hold =>
   lcd_rs_s  <= '0';
   dataout_s <= "00000000";  -- Hold (Decimal = 0, HEX = 00)
   nx       <= hold;
```

```
            when others =>
                -- Default case (if any)
                null;
        end case;
    end process;

    -- Output assignments
    lcd_en   <= clk_fsm;
    lcd_rs   <= lcd_rs_s;
    lcd_data <= dataout_s;

end Behavioral;
```

TvM

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

ENTITY LCD_Test IS
END LCD_Test;

ARCHITECTURE behavior OF LCD_Test IS
    -- Component Declaration for the Unit Under Test (UUT)
    COMPONENT LCD_FSM
        PORT(
            rst       : IN std_logic;
            clk_12Mhz : IN std_logic;
            lcd_rs    : OUT std_logic;
            lcd_en    : OUT std_logic;
            lcd_data  : OUT std_logic_vector(7 downto 0)
        );
    END COMPONENT;

    -- Inputs
    signal rst       : std_logic := '0';
    signal clk_12Mhz : std_logic := '0';

    -- Outputs
    signal lcd_rs    : std_logic;
    signal lcd_en    : std_logic;
    signal lcd_data  : std_logic_vector(7 downto 0);

    -- Clock period definition
    constant clk_12Mhz_period : time := 10 ns; -- For a 12 MHz clock

BEGIN
    -- Instantiate the Unit Under Test (UUT)
    uut: LCD_FSM
        PORT MAP (
            rst       => rst,
            clk_12Mhz => clk_12Mhz,
            lcd_rs    => lcd_rs,
            lcd_en    => lcd_en,
            lcd_data  => lcd_data
        );

    -- Clock process definitions
    clk_12Mhz_process : process
    begin
        -- Generate a 12 MHz clock signal
        clk_12Mhz <= '0';
        wait for clk_12Mhz_period / 2;
        clk_12Mhz <= '1';
        wait for clk_12Mhz_period / 2;
    end process;

    -- Stimulus process
    stim_proc: process
    begin
        -- Initialize reset
        rst <= '1';
        wait for 20 ns; -- Hold reset for 20 ns
        rst <= '0';

        -- Insert additional stimulus here if needed

        -- Simulation ends here
        wait;
    end process;

END behavior;
```