## Practical No.01: Ripple Carry Adder

```
entity OR_gate is
        Port ( p : in STD_LOGIC;
                      q : in STD_LOGIC;
                      r : out STD_LOGIC);
end OR_gate;

architecture Behavioral of OR_gate is

begin
r<= p or q;

end Behavioral;
------------------------------------------------------------------------------------------------------------------
entity Half_adder is
        Port ( a : in STD_LOGIC;
                      b : in STD_LOGIC;
                      s : out STD_LOGIC;
                      c : out STD_LOGIC);
end Half_adder;

architecture Behavioral of Half_adder is

begin
s<= a xor b;
```

```
c<= a and b;


end Behavioral;

-----------------------------------------------------------------------------------------------------------------------

entity Full_adder is

        Port ( A : in STD_LOGIC;

                      B : in STD_LOGIC;

                      Cin : in STD_LOGIC;

                      Sum : out STD_LOGIC;

                      Cout : out STD_LOGIC);

end Full_adder;


architecture struct of Full_adder is

Component Half_adder is

        Port ( a : in STD_LOGIC;

                      b : in STD_LOGIC;

                      s : out STD_LOGIC;

                      c : out STD_LOGIC);

end component;


Component OR_gate is

        Port ( p : in STD_LOGIC;

                      q : in STD_LOGIC;

                      r : out STD_LOGIC);

end component;
```

```
signal temp1,temp2,temp3 : std_logic;

begin

U1: Half_adder port map(A,B,temp1,temp2);
U2: Half_adder port map(temp1,Cin,Sum,temp3);
U3: OR_gate port map(temp2,temp3,Cout);

end struct;
```

------------------------------------------------------------------------------------------------------------------------

## With Ripple carry adder

```
entity RCA is
    Port ( X : in STD_LOGIC_VECTOR (3 downto 0);
           Y : in STD_LOGIC_VECTOR (3 downto 0);
           Zin : in STD_LOGIC;
           RCASum : out STD_LOGIC_VECTOR (3 downto 0);
           RCACout : out STD_LOGIC);
end RCA;

architecture Behavioral of RCA is
component Full_adder is
    Port ( A : in STD_LOGIC;
           B : in STD_LOGIC;
           Cin : in STD_LOGIC;
           Sum : out STD_LOGIC;
```

```
                    Cout : out STD_LOGIC);
end component;


signal Carry : std_logic_vector(2 downto 0);


begin
u1: Full_adder port map(X(0),Y(0),Zin,RCASum(0),Carry(0));
u2: Full_adder port map(X(1),Y(1),Carry(0),RCASum(1),Carry(1));
u3: Full_adder port map(X(2),Y(2),Carry(1),RCASum(2),Carry(2));
u4: Full_adder port map(X(3),Y(3),Carry(2),RCASum(3),RCACout);
end Behavioral;
```

# Practical No.02: Arithmetic Logic Unit

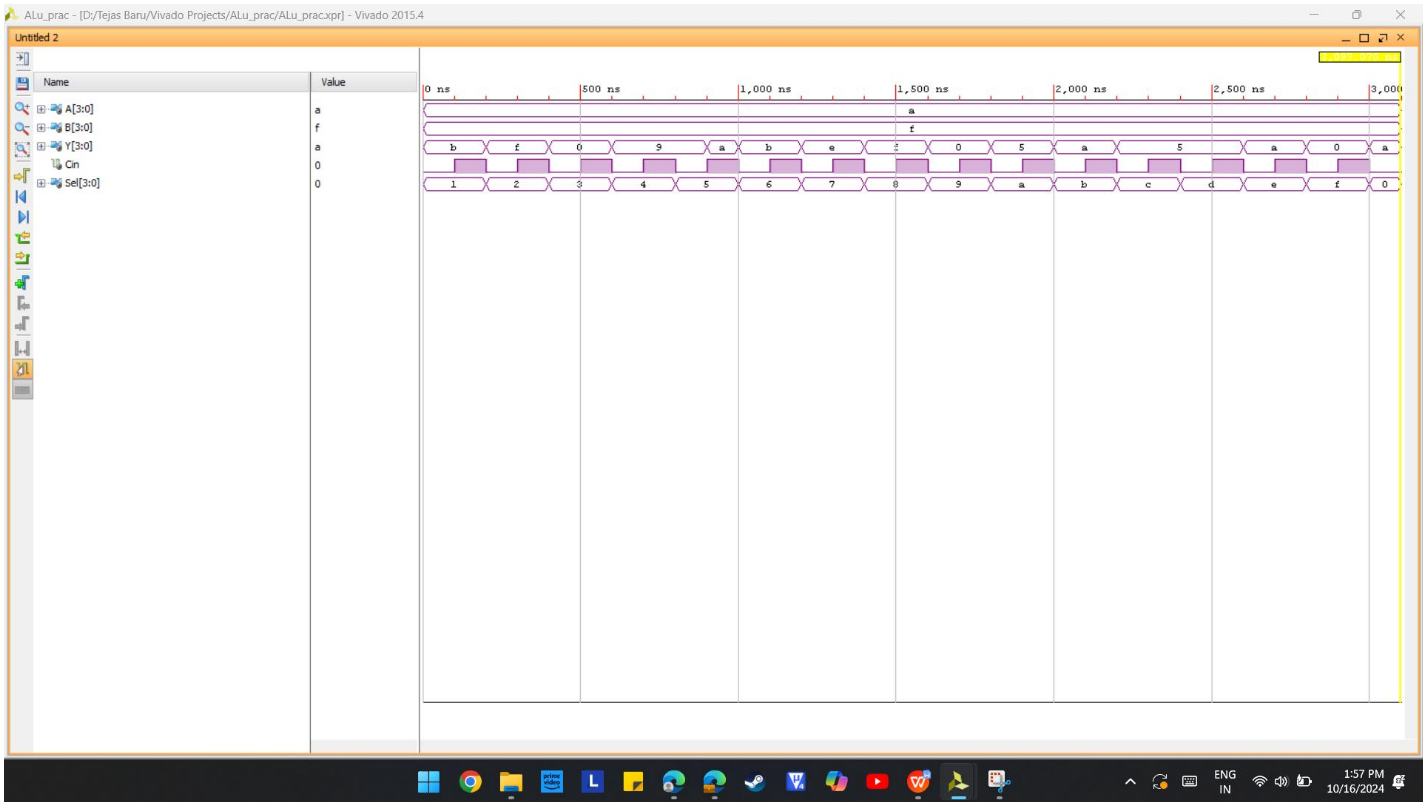
- ★ **CODE:**

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

entity ALU is
   Port (
         A : in STD_LOGIC_VECTOR (3 downto 0);
         B : in STD_LOGIC_VECTOR (3 downto 0);
         Cin : in STD_LOGIC;
         Sel : in STD_LOGIC_VECTOR (3 downto 0);
         Y : out STD_LOGIC_VECTOR (3 downto 0));
end ALU;
architecture concurrent of ALU is
begin
Y<= A when Sel = "0000" else
   A+1 when Sel = "0001" else
   B when Sel = "0010" else
   B+1 when Sel = "0011" else
   A+B when Sel = "0100" else
   A+B+Cin when Sel = "0101" else
   A-B when Sel = "0110" else
   B-1 when Sel = "0111" else
   A or B when Sel = "1000" else
   A nor B when Sel = "1001" else
   A nand B when Sel = "1010" else
   A and B when Sel = "1011" else
   not(A) when Sel = "1100" else
   A xor B when Sel = "1101" else
   A xnor B when Sel = "1110" else
   not(B) ;
end concurrent;
```

- ★ **Test Bench CODE:**

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

entity ALU_tb is
--  Port ( );
end ALU_tb;
architecture Behavioral of ALU_tb is
component ALU is
   Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
        B : in STD_LOGIC_VECTOR (3 downto 0);
        Cin : in STD_LOGIC;
        Sel : in STD_LOGIC_VECTOR (3 downto 0);
        Y : out STD_LOGIC_VECTOR (3 downto 0));
end component;
signal A, B, Y : std_logic_vector (3 downto 0);
signal Cin : std_logic;
signal Sel : std_logic_vector(3 downto 0) := "0000";
begin
u1: ALU port map (A, B, Cin, Sel, Y);
process
  begin
    A <= "1010";
    B <= "1111";
    Cin <= '0';
    Sel <= Sel + 1;
    wait for 100 ns;
    Cin <= '1';
    wait for 100 ns;
  end process;
end Behavioral;
```

→ **OUTPUT:**



ALu_prac - [D:/Tejas Baru/Vivado Projects/ALu_prac/ALu_prac.xpr] - Vivado 2015.4

Untitled 2

```
Name        Value   0 ns      500 ns      1,000 ns      1,500 ns      2,000 ns      2,500 ns      3,000
A[3:0]      a       a
B[3:0]      f       f
Y[3:0]      a       b  f  0  9  a  b  e  :  0  5  a  5  a  0  a
Cin         0
Sel[3:0]    0       1  2  3  4  5  6  7  8  9  a  b  c  d  e  f  0
```

ENG IN  1:57 PM 10/16/2024

# Practical No.03: Universal Shift Register

- **CODE:**

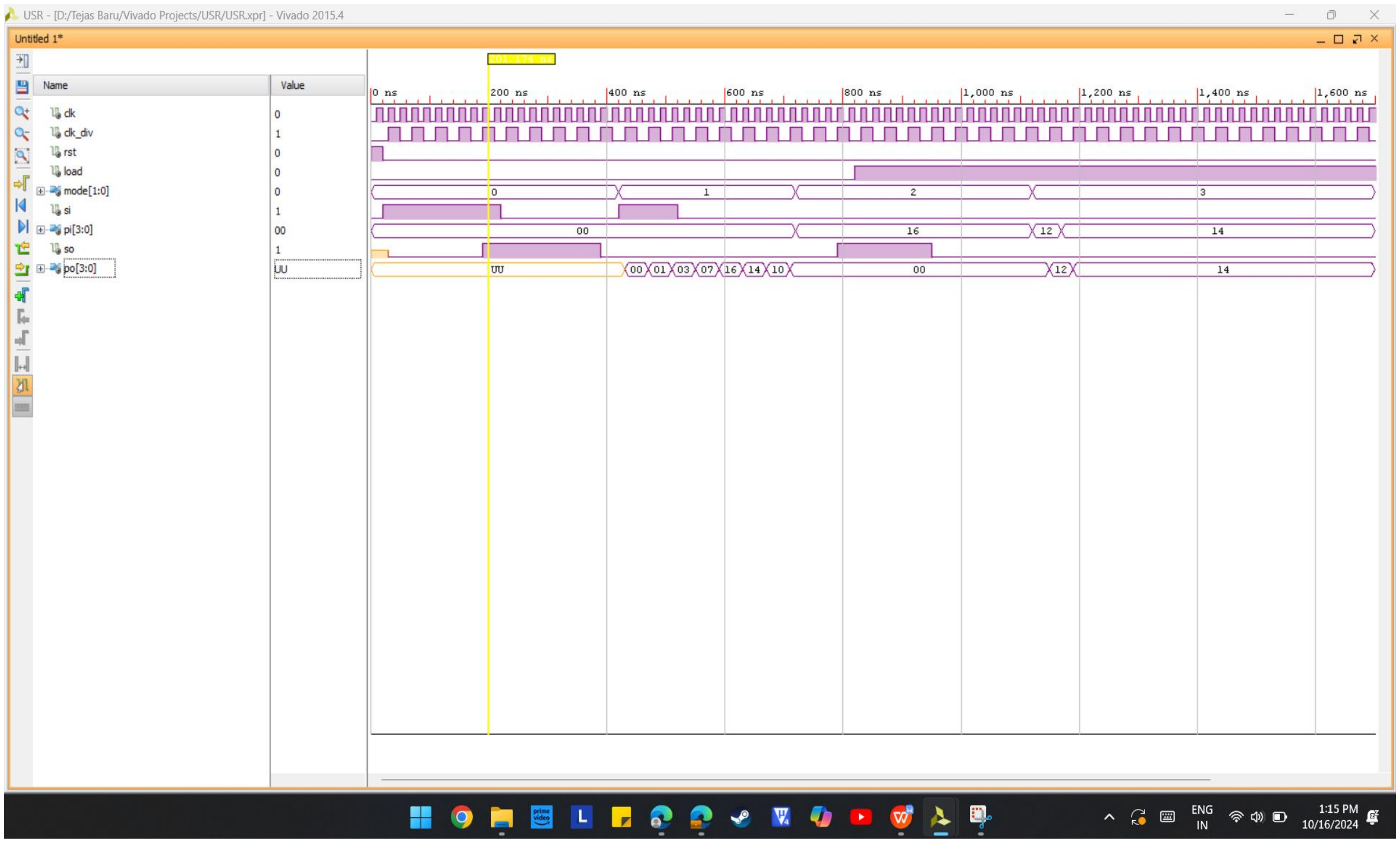
```
Use library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD.ALL;

entity usr1 is
  Port (
      clk : in  STD_LOGIC;
      clk_div : inout std_logic;
      rst : in  STD_LOGIC;
      load: in  STD_LOGIC;
      mode : in  STD_LOGIC_VECTOR(1 downto 0);  -- mode selection input
      si : in  STD_LOGIC;  -- serial input
      pi : in  STD_LOGIC_VECTOR(3 downto 0);  -- parallel input
      so : out  STD_LOGIC;  -- serial output
      po : out  STD_LOGIC_VECTOR(3 downto 0)  -- parallel output
   );
end usr1;
architecture Behavioral of usr1 is
   signal shift_reg : STD_LOGIC_VECTOR(3 downto 0);
   signal counter: std_logic_vector(26 downto 0):=( others=>'0');
begin
   process(clk_div, rst,load)
   begin
      if rst = '1' then
         shift_reg <= (others => '0');
      elsif clk_div'event and clk_div= '1' then
         case mode is
            when "00" =>  -- SISO mode
               shift_reg( 3 downto 1) <= shift_reg(2 downto 0);
               shift_reg(0)<=si;
               so <= shift_reg(3);
            when "01" =>  -- SIPO mode
               shift_reg( 3 downto 1) <= shift_reg(2 downto 0);
               shift_reg(0)<=si;
                po <= shift_reg;
           when "10" =>  -- PISO mode
                 if(load='0') then
                        shift_reg <= pi;
                  else
                     shift_reg( 3 downto 1) <= shift_reg(2 downto 0);
                  end if;
                  so <= shift_reg(3);
            when "11" =>  -- PIPO mode
```

```
                po<= pi;
            when others =>
                null;
        end case;
    end if;
 end process;
-- so <= shift_reg(3);
-- po <= shift_reg;
process(clk)
 begin
    if clk'event and clk = '1' then
        if rst = '1' then
            counter <= (others => '0');
        else
            counter <= counter + '1';
        end if;
    end if;
 end process;
clk_div<= counter(0);
end Behavioral;
```

## ★ Test Bench CODE:

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

entity usr1tb is
end usr1tb;
architecture Behavioral of usr1tb is
    signal clk,clk_div : STD_LOGIC := '0';
    signal rst : STD_LOGIC := '0';
    signal load : STD_LOGIC := '0';
    signal mode : STD_LOGIC_VECTOR(1 downto 0) := "00";
    signal si : STD_LOGIC := '0';
    signal pi : STD_LOGIC_VECTOR(3 downto 0) := (others => '0');
    signal so : STD_LOGIC;
    signal po : STD_LOGIC_VECTOR(3 downto 0);
    component usr1
        Port (
            clk : in  STD_LOGIC;
            clk_div : inout std_logic;
            rst : in  STD_LOGIC;
            load: in  STD_LOGIC;
            mode : in  STD_LOGIC_VECTOR(1 downto 0);
            si : in  STD_LOGIC;
            pi : in  STD_LOGIC_VECTOR(3 downto 0);
            so : out  STD_LOGIC;
            po : out  STD_LOGIC_VECTOR(3 downto 0)
        );
    end component;
begin
    uut : usr1
        Port Map (
            clk => clk,
            clk_div => clk_div,
            rst => rst,
            load => load,
            mode => mode,
            si => si,
            pi => pi,
            so => so,
            po => po
        );
    clk_process : process
    begin
        clk <= '0';
        wait for 10 ns;
        clk <= '1';
        wait for 10 ns;
    end process;
    stim_proc : process
    begin
        -- Reset the UUT
        rst <= '1';
        wait for 20 ns;
        rst <= '0';
        -- Test SISO mode
        mode <= "00";
        si <= '1';
        wait for 200 ns;
        si <= '0';
        wait for 200 ns;
               -- Test SIPO mode
        mode <= "01";
        si <= '1';
        wait for 100 ns;
        si <= '0';
        wait for 200 ns;
               -- Test PISO mode
        mode <= "10";
        load<='0';
        pi <= "1110";
        wait for 100 ns;
        load<='1';
        wait for 300 ns;
         -- Test PIPO mode
        mode <= "11";
        pi <= "1010";
        wait for 50 ns;
        pi <= "1100";
        wait for 50 ns;
         wait;
    end process;
end Behavioral;
```

→ **OUTPUT:**



USR - [D:/Tejas Baru/Vivado Projects/USR/USR.xpr] - Vivado 2015.4

Untitled 1*

```
Name          Value
clk           0
clk_div       1
rst           0
load          0
mode[1:0]     0
si            1
pi[3:0]       00
so            1
po[3:0]       UU
```

```
0 ns   200 ns   400 ns   600 ns   800 ns   1,000 ns   1,200 ns   1,400 ns   1,600 ns
mode[1:0]: 0 | 1 | 2 | 3
pi[3:0]:   00 | 16 | 12 | 14
po[3:0]:   UU | 00 | 01 | 03 | 07 | 16 | 14 | 10 | 00 | 12 | 14
```

ENG IN 1:15 PM 10/16/2024

# Practical No.04: MOD-N Counter

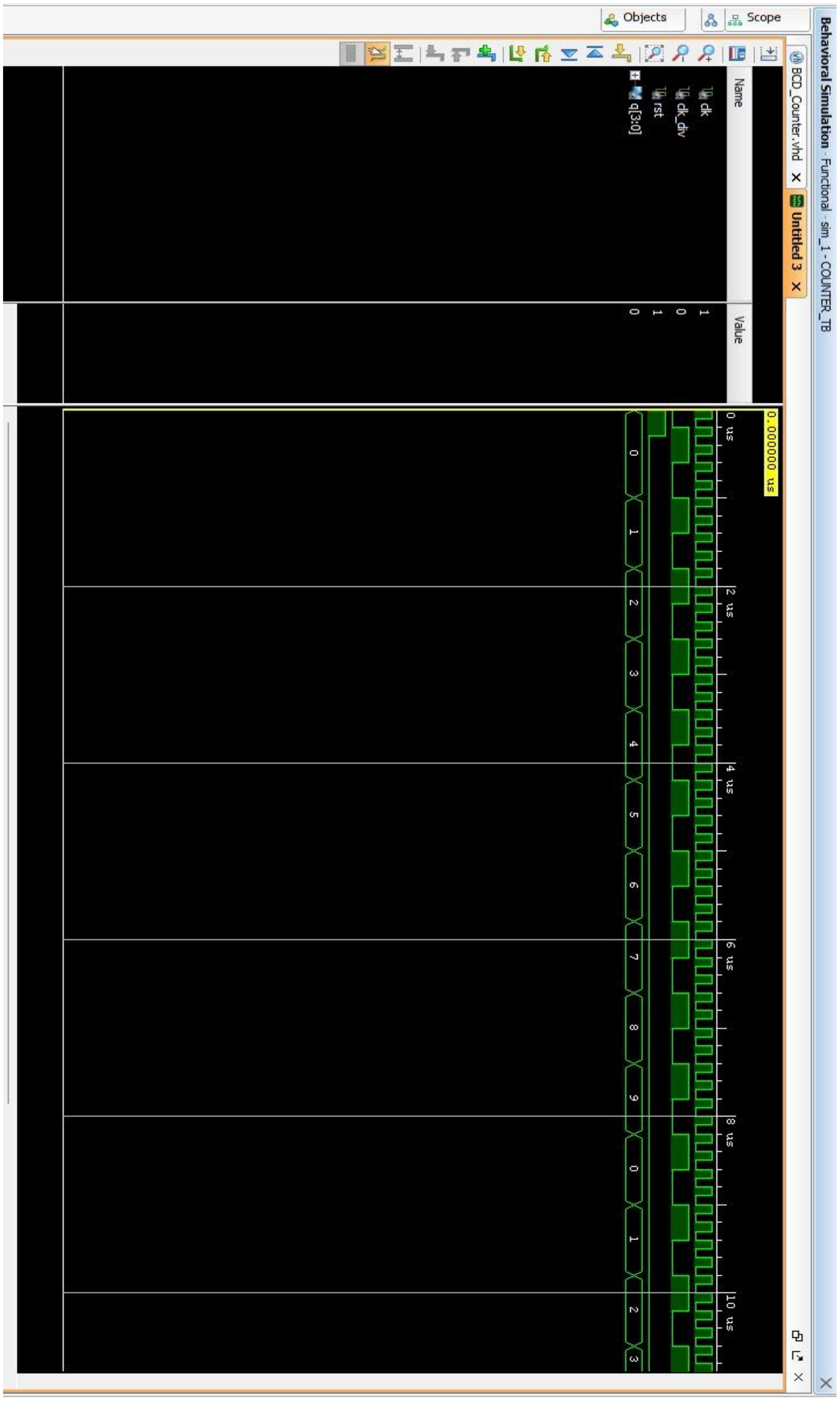
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity count is
    Port ( clk : in STD_LOGIC;
    clk_div: inout std_logic;
           rst : in STD_LOGIC;
            an      : out std_logic_vector(3 downto 0);
            seg     : out std_logic_vector(6 downto 0);
           q : inout STD_LOGIC_VECTOR (3 downto 0));
end count;

architecture Behavioral of count is
signal count : std_logic_vector( 25 downto 0):= (others => '0');

begin
an<="0001";
process(clk_div,rst)
variable temp: std_logic_vector(3 downto 0):="0000";
begin
if (rst='1') then
temp:= "0000";
elsif (clk_div'event and clk_div='1') then
temp:= temp+'1';
if( temp > "1001") then
temp := "0000";
end if;
end if;
q<=temp;
end process;
process(clk)
begin
if (clk'event and clk='1') then
count <= count + '1';
end if;
clk_div<= count(25);
end process;
--clk_div<= count(1);
process(q)
begin
 case q is
           when "0000" => seg <= "1000000"; -- 0
           when "0001" => seg <= "1001111"; -- 1
           when "0010" => seg <= "0100100"; -- 2
           when "0011" => seg <= "0110000"; -- 3
           when "0100" => seg <= "0011001"; -- 4
           when "0101" => seg <= "0010010"; -- 5
           when "0110" => seg <= "0000010"; -- 6
           when "0111" => seg <= "1111000"; -- 7
           when "1000" => seg <= "0000000"; -- 8
           when "1001" => seg <= "0010000"; -- 9
           when others => seg <= "1111111";
       end case;
   end process;
end Behavioral;
```

## Testbench for Counter :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity COUNTER_TB is
--  Port ( );
end COUNTER_TB;

architecture Behavioral of COUNTER_TB is
COMPONENT count is
   Port ( clk : in STD_LOGIC;
   clk_div: inout std_logic;
          rst : in STD_LOGIC;
          q : inout STD_LOGIC_VECTOR (3 downto 0));
end component;
signal clk,clk_div: std_logic;
signal rst:std_logic;
signal q: std_logic_vector(3 downto 0);

begin
u1: count port map(clk,clk_div,rst,q);
process
begin
clk<='1';
wait for 100 ns;
clk<='0';
wait for 100 ns;
end process;
rst<='1', '0' after 300 ns ;
```

```
end Behavioral;
```

## Counter O/P:



## Practical No.05: FIFO :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity fifo_correct is
GENERIC
(
ADDRESS_WIDTH : integer:=2;---8 bit
DATA_WIDTH : integer:=4 ---32 bit
);
port ( clk : in std_logic;
clk_div : inout std_logic;
reset : in std_logic;
enr : in std_logic; --enable read,should be '0' when not in use.
enw : in std_logic; --enable write,should be '0' when not in use.
dataout : out std_logic_vector(DATA_WIDTH-1 downto 0); --output data
datain : in std_logic_vector (DATA_WIDTH-1 downto 0); --input data
empty : out std_logic; --set as '1' when the queue is empty
err : out std_logic;
full : out std_logic --set as '1' when the queue is full
);
end fifo_correct;


architecture Behavioral of fifo_correct is
```

```
type memory_type is array (0 to ((2**ADDRESS_WIDTH)-1)) of
std_logic_vector(DATA_WIDTH-1 downto 0);

-----distributed-------

signal memory : memory_type ;-- :=(others => (others => '0')); --memory for
queue.-----

signal readptr,writeptr : std_logic_vector(ADDRESS_WIDTH-1 downto 0); --read
and write pointers.

signal full0 : std_logic;

signal empty0 : std_logic;

signal counter: std_logic_vector(28 downto 0):=( others=>'0');

begin

full <= full0;

empty <= empty0;

fifo0: process(clk_div,reset,datain,enw,enr)

begin

if reset='1' then

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

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

empty0 <='1';

full0<='0';

err<='0';

elsif  clk_div'event and clk_div = '1' then

if enw='1' and full0='0' then

memory (conv_integer(writeptr)) <= datain ;

writeptr <= writeptr + '1' ;

if (writeptr + '1' = readptr) then

full0<='1';
```

```
empty0<= '0';
else
full0<='0';
empty0<= '1';
end if ;
end if ;
if enr='1' and empty0='0' then
dataout <= memory (conv_integer(readptr));
readptr <= readptr + '1' ;
        if (readptr + '1'  = writeptr ) then
        empty0<='1';
        full0<='0';
        else
        empty0<='0';
        full0<='1';
        end if ;
end if ;
if (empty0='1' and enr='1') or (full0='1' and enw='1') then
err<='1';
else
err<= '0';
end if ;
end if;
end process;

process(clk)
```

```
begin
if clk'event and clk= '1' then
counter<= counter + '1';
end if;
end process;
clk_div<= counter(25);
end Behavioral;
```

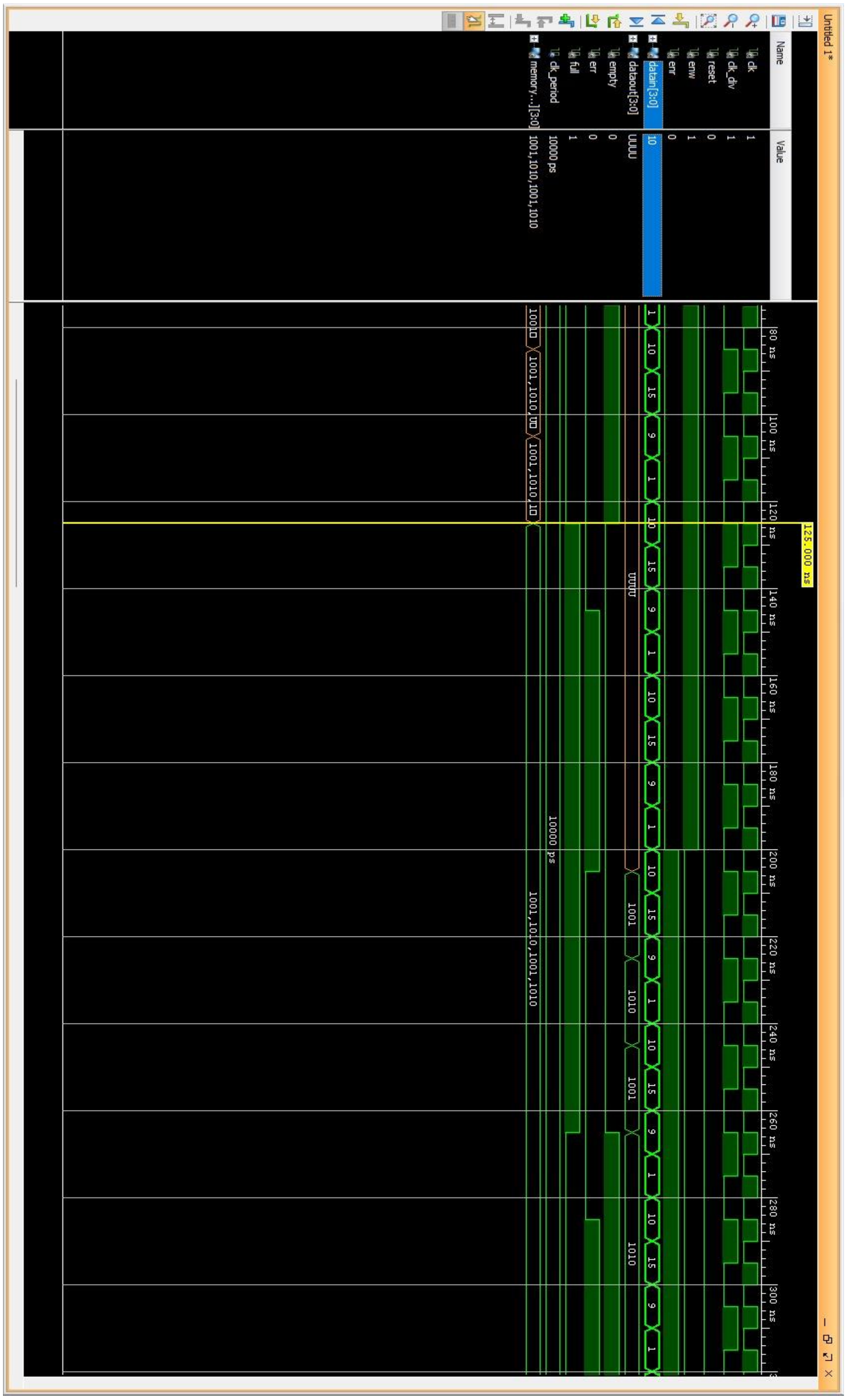
**Testbench for FIFO :**

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY fifo_test IS
END fifo_test;
ARCHITECTURE behavior OF fifo_test IS
   -- Component Declaration for the Unit Under Test (UUT)
   COMPONENT fifo_correct
   PORT(
      clk : in std_logic;
   clk_div : inout std_logic;
   reset : in std_logic;
   enr : in std_logic; --enable read,should be '0' when not in use.
   enw : in std_logic; --enable write,should be '0' when not in use.
   dataout : out std_logic_vector(3 downto 0); --output data
   datain : in std_logic_vector (3 downto 0); --input data
   empty : out std_logic; --set as '1' when the queue is empty
   err : out std_logic;
   full : out std_logic --set as '1' when the queue is full
```

```
      );
    END COMPONENT;
      --Inputs
   signal clk : std_logic := '0';
   signal reset,clk_div: std_logic;
   signal enr : std_logic := '0';
   signal enw : std_logic := '0';
   signal datain : std_logic_vector(3 downto 0) := (others => '0');
          --Outputs
   signal dataout : std_logic_vector(3 downto 0);
   signal empty : std_logic;
   signal err : std_logic;
   signal full : std_logic;
   -- Clock period definitions
   constant clk_period : time := 10 ns;
BEGIN
          -- Instantiate the Unit Under Test (UUT)
   uut: fifo_correct PORT MAP (
         clk => clk,
         clk_div=>clk_div,
         reset=>reset,
         enr => enr,
         enw => enw,
         dataout => dataout,
         datain => datain,
         empty => empty,
         err => err,
         full => full
       );
   -- Clock process definitions
   clk_process :process
   begin
                 clk <= '0';
                 wait for clk_period/2;
                 clk <= '1';
                 wait for clk_period/2;
   end process;
 reset<='1','0' after 50ns;
enw<= '1', '0' after 200 ns ;
 enr<= '0','1' after  200 ns ;
  --Stimulus process
  stim_proc: process
   begin
          datain<="1010";
          wait for 10 ns;
          datain<="1111";
                 wait for 10 ns;
          datain<="1001";
 wait for 10 ns;
          datain<="0001";
           wait for 10 ns;
 end process;
```

```
END behavior;
```

## FIFO O/P :

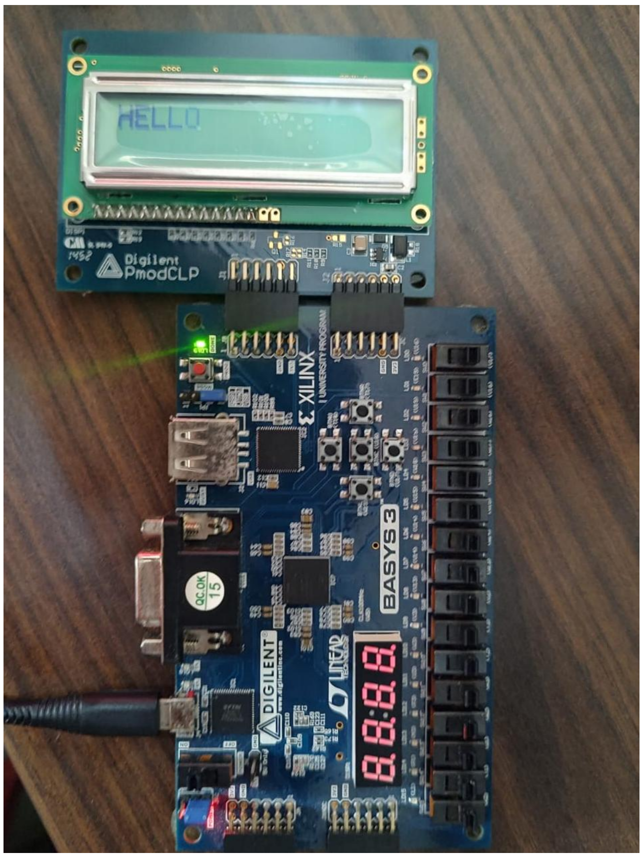


# Practical No.06: LCD

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

entity lcdm is
   Port ( clk : in  STD_LOGIC;
          rst : in  STD_LOGIC;
          lcd_data : out  STD_LOGIC_VECTOR (7 downto 0);
          lcd_rs : out  STD_LOGIC;
          lcd_rw : out  STD_LOGIC;
          lcd_en : out  STD_LOGIC);
end lcdm;

architecture Behavioral of lcdm is
   signal lcd_init : STD_LOGIC := '0';
   signal lcd_data_reg : STD_LOGIC_VECTOR (7 downto 0) := (others => '0');
   signal counter: std_logic_vector(28 downto 0):= (others => '0');
   signal en_pulse : std_logic := '0';
   signal pulse_counter : integer range 0 to 100 := 0; -- 100 clock cycles = 1 us at 100 MHz
   signal state : integer := 0;
   signal clk_enable : STD_LOGIC := '0';
begin
   process(clk)
   begin
      if clk'event and clk = '1' then

         counter <= counter + 1;
         if counter(10) = '1' then
            clk_enable <= '1';
         else
            clk_enable <= '0';
         end if;
      end if;
   end process;

   lcd_data <= lcd_data_reg;

   process(clk_enable)
   begin
      if clk_enable'event and clk_enable = '1' then

         if rst = '1' then
            lcd_init <= '0';
            lcd_data_reg <= (others => '0');
            en_pulse <= '0';
            pulse_counter <= 0;
            state <= 0;
         else
            case state is
               when 0 =>
                  -- Initialize LCD
```

```
        lcd_data_reg <= "00111000"; -- Function set command
        lcd_rs <= '0';
        lcd_rw <= '0';
        en_pulse <= '1';
        pulse_counter <= 0;
        state <= 1;

    when 1 =>
        if pulse_counter < 100 then
            lcd_en <= '1';
            pulse_counter <= pulse_counter + 1;
        else
            lcd_en <= '0';
            en_pulse <= '0';
            pulse_counter <= 0;
            state <= 2; -- Move to next state
        end if;
    when 2 =>
        -- Send Display ON command
        lcd_data_reg <= "00001100"; -- Display ON, Cursor OFF
        lcd_rs <= '0';
        lcd_rw <= '0';
        en_pulse <= '1';
        pulse_counter <= 0;
        state <= 3;

    when 3 =>
        if pulse_counter < 100 then
            lcd_en <= '1';
            pulse_counter <= pulse_counter + 1;
        else
            lcd_en <= '0';
            en_pulse <= '0';
            pulse_counter <= 0;
            state <= 4;
        end if;
    when 4 =>
        -- Send Clear Display command
        lcd_data_reg <= "00000001";
        lcd_rs <= '0';
        lcd_rw <= '0';
        en_pulse <= '1';
        pulse_counter <= 0;
        state <= 5;

    when 5 =>
        if pulse_counter < 100 then
            lcd_en <= '1';
            pulse_counter <= pulse_counter + 1;
        else
            lcd_en <= '0';
            en_pulse <= '0';
            pulse_counter <= 0;
            state <= 6;
        end if;
    when 6 =>
        -- Send H
    lcd_data_reg <= "01001000"; -- 'H'
    lcd_rs <= '1';
    lcd_rw <= '0';
    en_pulse <= '1';
    pulse_counter <= 0;
    state <= 7;

   when 7 =>
    if pulse_counter < 100 then
        lcd_en <= '1';
        pulse_counter <= pulse_counter + 1;
    else
        lcd_en <= '0';
        en_pulse <= '0';
        pulse_counter <= 0;
        state <= 8;
    end if;
  when 8 =>
    -- Send 'e'
    lcd_data_reg <= "01000101"; -- 'E'
    lcd_rs <= '1';
    lcd_rw <= '0';
    en_pulse <= '1';
    pulse_counter <= 0;
    state <= 9;
  when 9 =>
    if pulse_counter < 100 then
        lcd_en <= '1';
        pulse_counter <= pulse_counter + 1;
    else
        lcd_en <= '0';
        en_pulse <= '0';
        pulse_counter <= 0;
        state <= 10;
    end if;
  when 10 =>
    -- Send 'l'
    lcd_data_reg <= "01001100"; -- 'L'
    lcd_rs <= '1';
    lcd_rw <= '0';
    en_pulse <= '1';
    pulse_counter <= 0;
    state <= 11;
  when 11 =>
    if pulse_counter < 100 then
        lcd_en <= '1';
        pulse_counter <= pulse_counter + 1;
    else
        lcd_en <= '0';
        en_pulse <= '0';
        pulse_counter <= 0;
        state <= 12;
    end if;
  when 12 =>
    -- Send 'L'
    lcd_data_reg <= "01001100"; -- 'L'
    lcd_rs <= '1';
```

```
          lcd_rw <= '0';
          en_pulse <= '1';
          pulse_counter <= 0;
          state <= 13;
        when 13 =>
          if pulse_counter < 100 then
            lcd_en <= '1';
            pulse_counter <= pulse_counter + 1;
          else
            lcd_en <= '0';
            en_pulse <= '0';
            pulse_counter <= 0;
            state <= 14;
          end if;
        when 14 =>
          -- Send 'o'
          lcd_data_reg <= "01001111"; -- 'O'
          lcd_rs <= '1';
          lcd_rw <= '0';
          en_pulse <= '1';
          pulse_counter <= 0;
          state <= 15;
        when 15 =>
          if pulse_counter < 100 then
            lcd_en <= '1';
            pulse_counter <= pulse_counter + 1;
          else
            lcd_en <= '0';
            en_pulse <= '0';
            pulse_counter <= 0;
            state <= 0; -- Stay in this state
          end if;
        when others =>
          lcd_data_reg <= (others => '0');
          lcd_rs <= '0';
          lcd_rw <= '0';
          en_pulse <= '0';
      end case;
    end if;
  end if;
end process;
```



```
end Behavioral;
```