Reaction Game

Dimple Kochar- Roll Number 16D070010

April 20, 2018

1 Overview of the experiment/assignment

We have to implement a reaction game.
The game has two push buttons RESET and REACT, one LED and one LCD panel. The rules of the game are as follows:

- Press and release the RESET push button to start.
- After the RESET button is pushed, the following sequence is repeated 8 times.
  1. After a random delay between 1 and 2 seconds, the LED is lit up.
  2. The player has to push the REACT push button as early as possible after the LED is lit up. If the push button is pressed too early (ie before the LED is lit up), the game is over and the player gets no score. If the push button is pressed more than once after the LED is lit up, the game is over and the player gets no score. If the push-button is pressed once, the game records the delay in milliseconds between the LED being lit and the button being pushed.
- After the sequence of 8 pushes is over, the score of the player is calculated as the sum of delays (in milli seconds) observed across all 8 pushes. This number is displayed on the LCD screen. In case of a failure, the LCD screen displays all 0s.
- The player with the smallest score (other than all 0s, of course) wins.

2 Experiment setup or approach to the assignment

We divide the work in various sections.

- Generating a random number:
  For this, we use a Linear-Feedback Shift Register (LFSR). We generate a random 11 bit number using a 10 bit LFSR and keeping the most significant bit always 1. LFSR generate a pseudo random number. When we power-reset our game, we get the LFSR in its initial state. Reset doesn't reset it.

- Lighting the LED after random time:
  For this, we first make a counter, which counts to 11 bits. Once the counter number matches the random number generated, LED lights up.

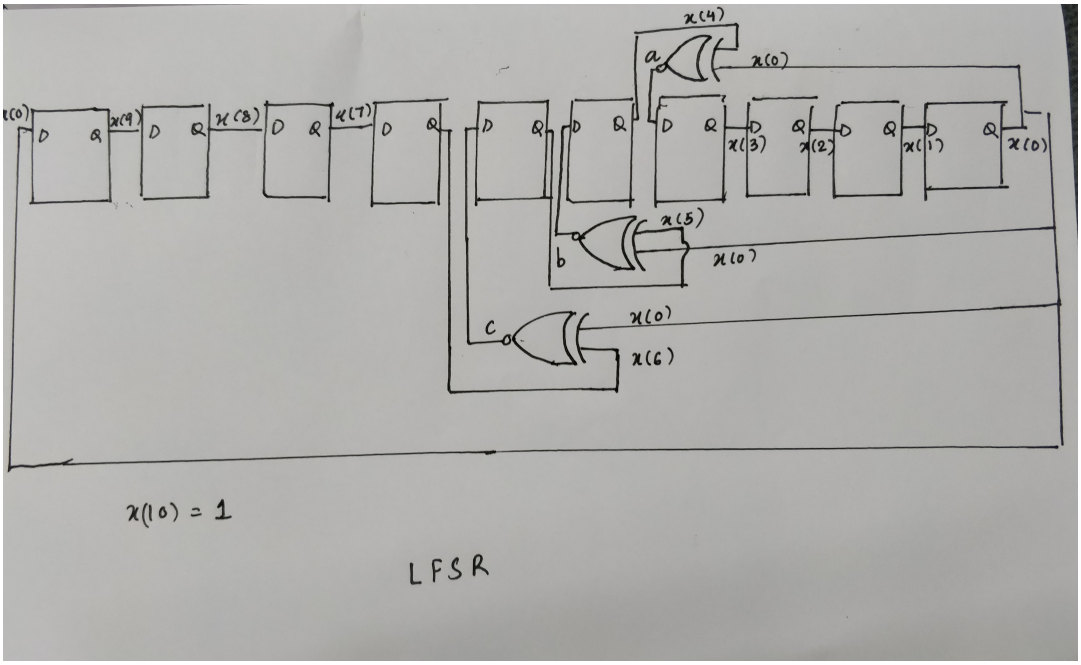
- Calculating Time:
  Once the LED lights up, the time counter starts counting the total time till the switch is pressed. Once the switch is pressed, LED goes off and counter retains its value. If the switch is pressed when the LED is not lit up, the game is over. We use a clock with 1.31 ms for counting this time. We make this clock by dividing the 50MHz by 2^{16} to get 762.939 Hz clock.

- Displaying the time:
  We use hexadecimal number system to display the time on the LCD display. First, we convert the 11 bit total time to 16 bits. Then, we convert in groups of 4 bits to integers.

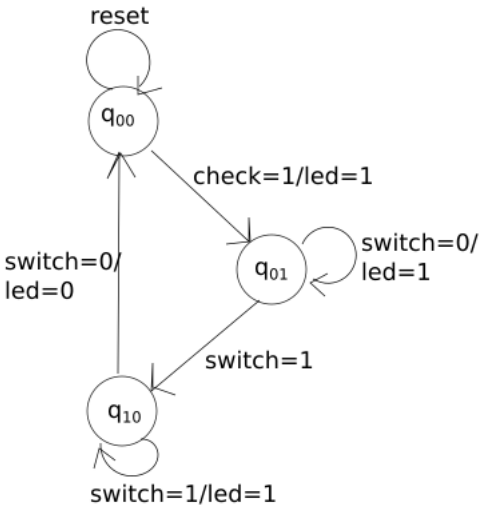
- Additional
  We have used a debouncer as well as the lcd_controller code provided to us.

3 Design/Algorithm

- LFSR



- LED FSM
  Following is the LED FSM:



State	q_1	q_0
q_{00}	0	0
q_{01}	0	1
q_{10}	1	0

Table: State Assignment

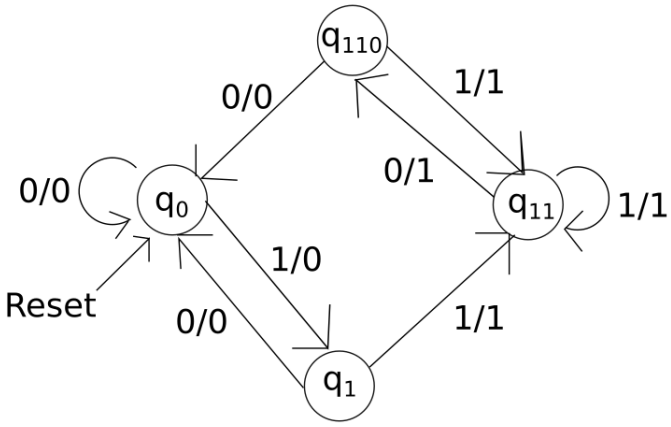
- Delay Adder
  We first make a an eleven bit adder. Then, we using this, we keep adding 1 for every clock cycle to count the delay time between led on and switch not pressed. The 11 bit adder is simply made using 11 1 bit adders.

- Counter Check
  We make an 11 bit counter which counts till it matches the random number number generated.

- One ms clock
  Using the generate statement and stacking D flip flops, we make a 1.31 ms clock. We make this clock by dividing the 50MHz by 2^{16} to get 762.939 Hz clock.

- Sampler Output
  It adds the total time across all rounds. It uses a generate statement and takes input the previous total time with the help of Delay Adder.

- Debouncer
  Following is the FSM for the debouncer, as designed in the last exper-



iment.

State	q_1	q_0
q_0	0	0
q_1	0	1
q_{11}	1	1
q_{110}	1	0

Table: State Assignment

- Game Over FSM
  We and clock with switch and (not led), because if switch is pressed without led being on, game should get over.

- Game
  We integrate all the above and some other minor components, to finally make the game. We use a process statement with the rising edge of the clock and led so that the led lights up the required 8 times. We have a generate statement to generate the next raandom number which uses clock as sample which is 1 if switch is pressed and reset is not on. The LED FSM lights up the LED and the Delay Adder adds time. So, on for the next round.

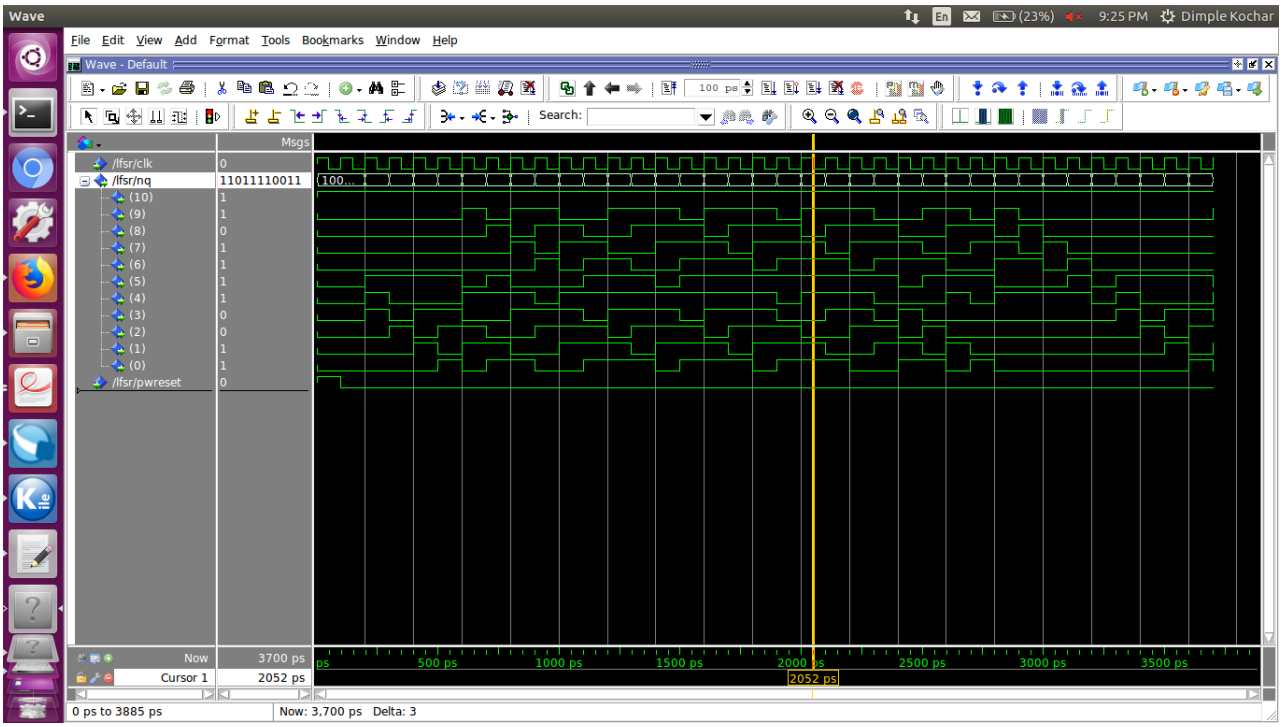
- Reaction Game
  Using the LCD controller code provided to us, and the game above, we write the output time to hexadecimal numerical system on the LCD panel provided. For this, we first convert our 11 bit delay time to 16

bits by initializing the 5 most significant bits to 0. Then we convert to integers, in batches of 4, using the to_integer function. Then, using a process statement, we write down the 4 digits of the LCD display.

4 Simulation Results

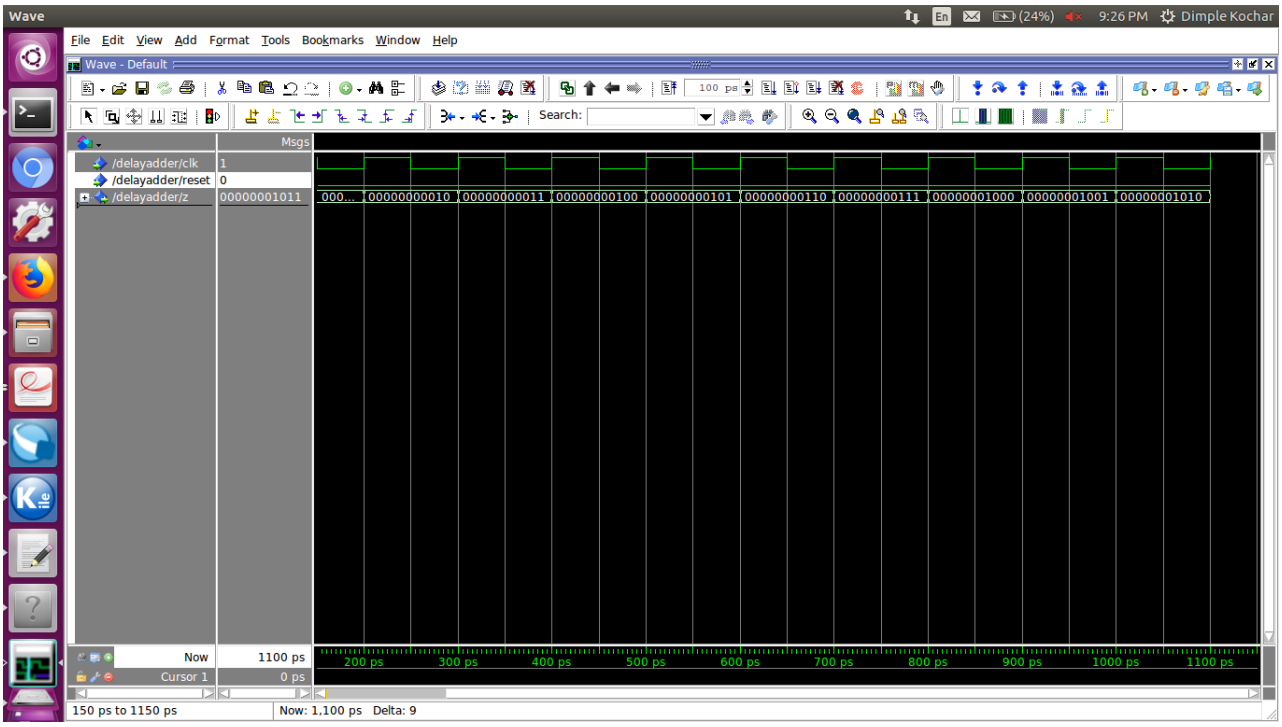
4.1 LFSR

After compiling the above code, we run the RTL simulations. This is what we get.



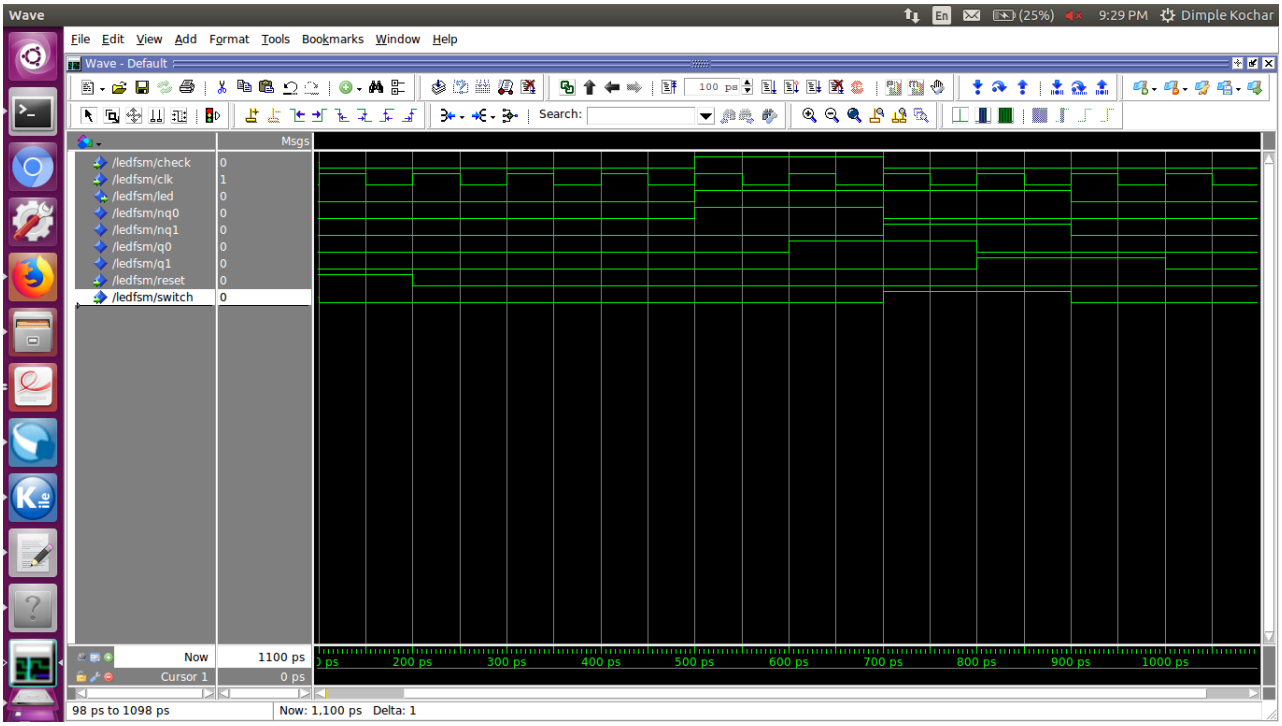
4.2 Delay Adder

After compiling the above code, we run the RTL simulations. This is what we get.



4.3 LED FSM

After compiling the above code, we run the RTL simulations. This is what we get.



5 Code

```
library std;
use std.standard.all;
library ieee;
use ieee.std_logic_1164.all;
package lab is
component Game is
port (switch1, masterclock, pwreset, reset1 : in std_logic;
tottimeout : out std_logic_vector(10 downto 0); ledoutput, go1 : out std_logic);--
end component;
component Debouncer is
port (reset, x, clkin: in std_logic; y:out std_logic);
end component;
component GameOverFSM is
```

```
port(switch, led, reset, clk : in std_logic; go : out std_logic);
end component;
component SamplerOutput is
port(tottime : in std_logic_vector (10 downto 0); reset, clkin2 : in std_logic;
vec : out std_logic_vector(10 downto 0));
end component;
component OneBitAdd is
port(x, y, cin: in std_logic; s, cout: out std_logic);
end component;
component ElevenBitAdder is
port (x, y : in std_logic_vector(10  downto 0);
z : out std_logic_vector (10  downto 0));
end component;
component DelayAdder is
port (clk, reset : in std_logic;
z : out std_logic_vector (10  downto 0));
end component;
component D_FF is
port (D, CLK, reset: in std_logic; Q: out std_logic);
end component;
component CounterCheck is
port(masterclock, reset : in std_logic; rand : in std_logic_vector(10 downto 0);
vec : out std_logic_vector(10 downto 0));
end component;
component dbtwo is
port (inclk, reset: in std_logic; outclk: out std_logic);
end component;
component oneclock is
port (inclk, reset: in std_logic; outclk: out std_logic);
end component;
component lfsr is
port (clk, pwreset: in std_logic; nq:out std_logic_vector(10 downto 0));
end component;
component ledfsm is
port (switch, check, clk, reset: in std_logic; led:  out std_logic);
end component;
component lcd_controller is
port (clk    : in std_logic;
```

```
        rst    : in std_logic;                          -- reset
        erase : in std_logic;                  --- clear position
        put_char : in std_logic;
        write_data : in std_logic_vector(7 downto 0);
        write_row : in std_logic_vector(0 downto 0);
        write_column : in std_logic_vector(3 downto 0);
        ack : out std_logic;
        lcd_rw : out std_logic;                          --read & write control
        lcd_en : out std_logic;                          --enable control
        lcd_rs : out std_logic;                          --data or command control
        lcd1  : out std_logic_vector(7 downto 0);
        b11 : out std_logic;
        b12 : out std_logic);     --data line
end component;
end lab;

library work;
use work.lab.all;
library std;
use std.standard.all;
library ieee;
use ieee.std_logic_1164.all;
entity D_FF is
  port (D, CLK, reset: in std_logic; Q: out std_logic);
end entity;
architecture WhatDoYouCare of D_FF is
begin
   process (CLK, reset)
    begin
if (reset = '1') then
Q <= '0';
elsif CLK'event and (CLK = '1') then
     Q <= D;
end if;
   end process;
end WhatDoYouCare;

library work;
```

```
use work.lab.all;
library std;
use std.standard.all;
library ieee;
use ieee.std_logic_1164.all;
entity dbtwo is
  port (inclk, reset: in std_logic; outclk: out std_logic);
end entity;
architecture struct of dbtwo is
signal nq, q: std_logic;
begin
nq<=(not(q) and not (reset));
D_FF1 : D_FF port map (D => nq, CLK => inclk, reset=>reset, Q => q);
outclk<= q and not(reset);
end struct;

library work;
use work.lab.all;
library std;
use std.standard.all;
library ieee;
use ieee.std_logic_1164.all;
entity oneclock is
port (inclk, reset: in std_logic; outclk: out std_logic);
end entity;
architecture struct of oneclock is
signal Z: std_logic_vector(16 downto 0);
begin
Z(0)<=inclk;
Q1: for I in 0 to 15 generate
dbtwox : dbtwo port map(inclk => Z(I), reset => reset, outclk => Z(I+1));
end generate;
outclk<=Z(16);
end struct;

library work;
use work.lab.all;
library std;
```

```
use std.standard.all;
library ieee;
use ieee.std_logic_1164.all;
entity Debouncer is
port (reset, x, clkin: in std_logic; y:out std_logic);
end entity;
architecture cs of Debouncer is
signal q1, q0, nq1, nq0 : std_logic;
begin
D_FF0 : D_FF port map (D=> nq0, CLK => clkin, reset => reset, Q=>q0);
D_FF1 : D_FF port map (D=> nq1, CLK => clkin, reset => reset, Q=>q1);
nq0 <= x;
nq1 <= (not(q1) and q0 and x) or (q1 and q0) or (q1 and not(q0) and x);
y <= (q1 and q0) or (not(q0) and q1);
end cs;

library work;
use work.lab.all;
library std;
use std.standard.all;
library ieee;
use ieee.std_logic_1164.all;
entity CounterCheck is
port(masterclock, reset : in std_logic; rand : in std_logic_vector(10 downto 0);
vec : out std_logic_vector(10 downto 0));
end entity;
architecture Check of CounterCheck is
signal curr : std_logic_vector(10 downto 0);
signal clock: std_logic;
begin
clock1 : oneclock port map (reset => reset, inclk => masterclock, outclk => clock)
counter1 : DelayAdder port map (clk => clock, reset => reset, z => curr);
vec <= curr;
end Check;

library work;
use work.lab.all;
library std;
```

```
use std.standard.all;
library ieee;
use ieee.std_logic_1164.all;
entity DelayAdder is
port (clk, reset : in std_logic;
z : out std_logic_vector (10  downto 0));
end entity;
architecture behave of DelayAdder is
signal y : std_logic_vector(10 downto 0);
signal y1 : std_logic_vector(10 downto 0);
signal x1 : std_logic_vector(10 downto 0);
signal x: std_logic_vector(10 downto 0);
begin
x(0) <= '1';
x(1) <= '0';
x(2) <= '0';
x(3) <= '0';
x(4) <= '0';
x(5) <= '0';
x(6) <= '0';
x(7) <= '0';
x(8) <= '0';
x(9) <= '0';
 x(10) <= '0';
random : for i in 0 to 10 generate
twox: D_FF port map(D => y1(i), CLK => clk, reset => reset,  Q => y(i));
x1(i) <= x(i) and not(reset);
end generate;
add1: ElevenBitAdder port map (x => x1, y => y, z => y1);
z <= y1;
end behave;

library work;
use work.lab.all;
library std;
use std.standard.all;
library ieee;
use ieee.std_logic_1164.all;
```

```
entity ElevenBitAdder is
port (x, y : in std_logic_vector(10  downto 0);
z : out std_logic_vector (10  downto 0));
end entity;
architecture behave of ElevenBitAdder is
signal c : std_logic_vector(10  downto 0);
begin
a1 : OneBitAdd port map (x=>x(0), y=>y(0), cin=>'0', s=>z(0), cout=>c(1));
a2 : OneBitAdd port map (x=>x(1), y=>y(1), cin=>c(1), s=>z(1), cout=>c(2));
a3 : OneBitAdd port map (x=>x(2), y=>y(2), cin=>c(2), s=>z(2), cout=>c(3));
a4 : OneBitAdd port map (x=>x(3), y=>y(3), cin=>c(3), s=>z(3), cout=>c(4));
a5 : OneBitAdd port map (x=>x(4), y=>y(4), cin=>c(4), s=>z(4), cout=>c(5));
a6 : OneBitAdd port map (x=>x(5), y=>y(5), cin=>c(5), s=>z(5), cout=>c(6));
a7 : OneBitAdd port map (x=>x(6), y=>y(6), cin=>c(6), s=>z(6), cout=>c(7));
a8 : OneBitAdd port map (x=>x(7), y=>y(7), cin=>c(7), s=>z(7), cout=>c(8));
a9 : OneBitAdd port map (x=>x(8), y=>y(8), cin=>c(8), s=>z(8), cout=>c(9));
a10 : OneBitAdd port map (x=>x(9), y=>y(9), cin=>c(9), s=>z(9), cout=>c(10));
a11 : OneBitAdd port map (x=>x(10), y=>y(10), cin=>c(10), s=>z(10), cout=>c(0));
end behave;

library work;
use work.lab.all;
library std;
use std.standard.all;
library ieee;
use ieee.std_logic_1164.all;
entity OneBitAdd is
port(x, y, cin: in std_logic; s, cout: out std_logic);
end entity;
architecture oneadd of OneBitAdd  is
signal c1, c2, c3, s1, or2 : std_logic;
begin
s1<= x xor y;
s <= s1 xor cin;
c1 <= x and y;
c2 <= x and cin;
c3 <= cin and y;
or2 <= c1 or c2;
```

```
cout <= or2 or c3;
end oneadd;

library work;
use work.lab.all;
library std;
use std.standard.all;
library ieee;
use ieee.std_logic_1164.all;
entity lfsr is
  port (clk, pwreset: in std_logic; nq:out std_logic_vector(10 downto 0));
end entity lfsr;
architecture struct of lfsr is
signal x: std_logic_vector(9 downto 0);
signal a, b, c : std_logic;
begin
nq(0)<=x(0);
nq(1)<=x(1);
nq(2)<=x(2);
nq(3)<=x(3);
nq(4)<=x(4);
nq(5)<=x(5);
nq(6)<=x(6);
nq(7)<=x(7);
nq(8)<=x(8);
nq(9)<=x(9);
a<=(x(4) xnor x(0));
b<=(x(5) xnor x(0));
c<=(x(6) xnor x(0));
dff0 : D_FF port map (D => x(1), CLK => clk, reset=>pwreset, Q => x(0));
dff1 : D_FF port map (D => x(2), CLK => clk, reset=>pwreset, Q => x(1));
dff2 : D_FF port map (D => x(3), CLK => clk, reset=>pwreset, Q => x(2));
dff3 : D_FF port map (D => a, CLK => clk, reset=>pwreset, Q => x(3));
dff4 : D_FF port map (D => b, CLK => clk, reset=>pwreset, Q => x(4));
dff5 : D_FF port map (D => c, CLK => clk, reset=>pwreset, Q => x(5));
dff6 : D_FF port map (D => x(7), CLK => clk, reset=>pwreset, Q => x(6));
dff7 : D_FF port map (D => x(8), CLK => clk, reset=>pwreset, Q => x(7));
dff8 : D_FF port map (D => x(9), CLK => clk, reset=>pwreset, Q => x(8));
```

```
dff9 : D_FF port map (D => x(0), CLK => clk, reset=>pwreset, Q => x(9));
nq(10) <= '1';
end struct;

library work;
use work.lab.all;
library std;
use std.standard.all;
library ieee;
use ieee.std_logic_1164.all;
entity ledfsm is
    port (switch, check, clk, reset: in std_logic; led:  out std_logic);
end entity;
architecture struct of ledfsm is
signal q1, q0, nq1, nq0: std_logic;
begin
DFF0: D_FF port map (D => nq0, reset => reset, CLK => clk, Q => q0);
DFF1: D_FF port map (D => nq1, reset => reset, CLK => clk, Q => q1);
nq0 <= (not reset) and ((not switch) and (q0 or check));
nq1 <= (not reset) and (switch and (q0 or q1));
led <= (not reset) and ((check and (not switch)) or ((not q1) and q0)
or (q1 and switch));
end struct;

library work;
use work.lab.all;
library std;
use std.standard.all;
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity Game is
port (switch1, masterclock, pwreset, reset1 : in std_logic;
tottimeout : out std_logic_vector(10 downto 0); ledoutput, go1 : out std_logic);--
end entity;
architecture behave of Game is
signal curr, rand, rand3, gameover, tottime1, tottime : std_logic_vector(10 downto
signal dd : std_logic_vector(2 downto 0);
```

```
signal temp1 : unsigned(3 downto 0) := to_unsigned(0, 4);
signal tempv : std_logic_vector (3 downto 0);
signal switchb, go, clkin2, ledoutput1, switch, check, addclock, checkerreset, one
reset, led, led1, sample, endd, ledclk : std_logic;
begin
go1 <= go;
process(led1, reset)
begin
if reset='1' then
   temp1 <= "0000";
elsif rising_edge(led1) and temp1<17   then
temp1 <= temp1+1;
elsif rising_edge(led1) and temp1>16 then
    temp1(3) <= '1';
 temp1(2) <= '0';
 temp1(1) <= '0';
 temp1(0) <= '1';
end if;
end process;
tempv <= std_logic_vector(temp1);
ledoutput1 <= led1 and ((not tempv(3)) or (tempv(2)) or (tempv(1)) or (not tempv(0
switchb <= not switch1;
debounce : Debouncer port map(reset => reset, clkin => oneeclock, x => switchb, y
led1 <= led;
reset <= reset1 or pwreset;
sample <= switch or not reset;
ledclk <= masterclock;
checkerreset <= reset or switch;
onclk : oneclock port map (inclk => masterclock, reset => reset, outclk => oneeclo
random : for i in 0 to 10 generate
tottimeout(i) <= tottime(i) and not(go);
twox: D_FF port map(D => rand(i), CLK => sample, reset => reset, Q => rand3(i));
end generate;
rand1 : lfsr port map (clk => masterclock, pwreset => pwreset, nq => rand);
checker1 : CounterCheck port map (masterclock => masterclock, reset => checkerrese
vec=>curr);
check <= not ((rand(0) xor curr(0)) or (rand(1) xor curr(1)) or (rand(2) xor curr(
xor curr(3)) or (rand(4) xor curr(4)) or (rand(5) xor curr(5)) or (rand(6) xor cur
```

```
xor curr(7)) or (rand(8) xor curr(8)) or (rand(9) xor curr(9)) or (rand(10) xor cu
ledfsm1 : ledfsm port map (switch=>switch, clk=>ledclk, reset=>reset , check=>chec
addclock <= oneeclock and ledoutput1 and not(switch);
da : DelayAdder port map(clk => addclock, reset=>reset, Z => tottime1);
ledoutput <= ledoutput1 and not(go);
clkin2 <= switch and led and not(go);
outsample : SamplerOutput port map (tottime => tottime1, reset => reset, clkin2 =>
vec => tottime);
gameisover1 : GameOverFSM port map (switch => switch, led => led1, clk => masterCl
reset => reset, go => go);
end behave;

library work;
use work.lab.all;
library std;
use std.standard.all;
library ieee;
use ieee.std_logic_1164.all;
entity GameOverFSM is
port(switch, led, reset, clk : in std_logic; go : out std_logic);
end entity;
architecture gameisover of GameOverFSM is
signal clk1, nq, q : std_logic;
begin
dff : D_FF port map (D => nq, CLK => clk1, Q => q, reset => reset);
nq <= '1';
clk1 <= clk and (switch and (not led));
go <= q;
end gameisover;

library work;
use work.lab.all;
library std;
use std.standard.all;
library ieee;
use ieee.std_logic_1164.all;
entity SamplerOutput is
port(tottime : in std_logic_vector (10 downto 0); reset, clkin2 : in std_logic;
```

```
vec : out std_logic_vector(10 downto 0));
end entity;
architecture sampler of SamplerOutput is
begin
random : for i in 0 to 10 generate
twox: D_FF port map(D => tottime(i), CLK => clkin2, reset => reset,  Q => vec(i));
end generate;
end sampler;

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

entity lcd_controller is
port (clk    : in std_logic;
      rst    : in std_logic;                          -- reset
      erase : in std_logic;                  --- clear position
      put_char : in std_logic;
      write_data : in std_logic_vector(7 downto 0);
      write_row : in std_logic_vector(0 downto 0);
      write_column : in std_logic_vector(3 downto 0);
      ack : out std_logic;
      lcd_rw : out std_logic;                          --read & write control
      lcd_en : out std_logic;                          --enable control
      lcd_rs : out std_logic;                          --data or command control
      lcd1  : out std_logic_vector(7 downto 0);
      b11 : out std_logic;
      b12 : out std_logic);     --data line
end lcd_controller;


architecture Behavioral of lcd_controller is

type arr is array (0 to 4) of std_logic_vector(7 downto 0);
 constant lcd_cmd : arr := (x"38", x"01", x"0C", x"80", x"04");  -- cmd for LCD
```

```
signal lcd : std_logic_vector (7 downto 0);
signal count_next_cmd, count_cmd :integer range 0 to 5;
signal count_next_data, count_data :integer range 0 to 6;
signal count_next_data1, count_data1 :integer range 0 to 4;
signal cmd_line, cmd_line_next :integer range 0 to 10;
signal cmd_position : std_logic_vector(7 downto 0);
signal data_dis : std_logic_vector(7 downto 0);


type state_type is (S0, S1, S2, S3, S4, S5, S6, S7, S8, S9, S10, S11);
signal state : state_type := S0;

begin

count_next_cmd <= count_cmd + 1;
count_next_data <= count_data + 1;
count_next_data1 <= count_data1 + 1;
cmd_line_next <= cmd_line + 1;
b11 <= '1';
b12 <= '0';
lcd1 <= lcd;

process(clk)
begin
if rising_edge(clk) then
if (write_row(0) = '0') then          -- first row
   cmd_position <= x"80" + write_column;
   elsif (write_row(0) = '1') then        -- second row
    cmd_position <= x"C0" + write_column;
end if;
end if;
end process;


PROCESS(clk)

BEGIN
```

```
if rising_edge(clk) then

if (rst='0') then
state <= S0;
count_cmd <= 0;
count_data <= 0;
count_data1 <= 0;
cmd_line <= 0;
ack <= '0';
else

case state is

when S0=>-- S0 to S2: Send LCD commands
if (count_cmd < 5) then
lcd <= lcd_cmd(count_cmd);
lcd_rs <= '0';
lcd_rw <= '0';
lcd_en <= '0';
ack <= '0';
state <= S1;
else
state <= S3;
end if;

when S1=>
lcd <= lcd_cmd (count_cmd);
lcd_rs <= '0';
lcd_rw <= '0';
lcd_en <= '1';
ack <= '0';
state <= S2;


when S2 =>
lcd <= lcd_cmd (count_cmd);
lcd_rs <= '0';
lcd_rw <= '0';
```

```
lcd_en <= '0';
ack <= '0';
state <= S0;
count_cmd <= count_next_cmd;


when S3 =>            -- S3 to S5: Send LCD command position
lcd <= cmd_position;
lcd_rs <= '0';
lcd_rw <= '0';
lcd_en <= '0';
ack <= '0';
state <= S4;

when S4 =>
lcd <= cmd_position;
lcd_rs <= '0';
lcd_rw <= '0';
lcd_en <= '1';
ack <= '0';
state <= S5;


when S5 =>
lcd <= cmd_position;
lcd_rs <= '0';
lcd_rw <= '0';
lcd_en <= '0';
ack <= '0';
state <= S6;

when S6=>                     -- S6 to S8: Send LCD data or clear
   if (erase = '1') then
      data_dis <= x"20";  -- to clear the data
   lcd <=  data_dis;
   lcd_rs <= '1';
   lcd_rw <= '0';
   lcd_en <= '0';
```

```
   ack <= '0';
   state <= S7;
elsif (put_char = '1') then
   data_dis <= write_data;       -- lcd data
lcd <=  data_dis;
   lcd_rs <= '1';
   lcd_rw <= '0';
   lcd_en <= '0';
   ack <= '0';
   state <= S7;
elsif (erase = '0' and put_char = '0') then
   state <= S3;
end if;

when S7=>
lcd <= data_dis;
lcd_rs <= '1';
lcd_rw <= '0';
lcd_en <= '1';
ack <= '0';
state <= S8;


when S8=>
lcd <= data_dis ;
lcd_rs <= '1';
lcd_rw <= '0';
lcd_en <= '0';
ack <= '1';
state <= S9;

when S9 =>
   ack <= '0';
   state <= s3;

when others =>
state <= S0;
count_cmd <= 0;
```

```
end case;
end if;
   end if;
end process;

end Behavioral;

library work;
use work.lab.all;
library std;
use std.standard.all;
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity ReactionGame is
port (switch1, masterclock, pwreset, reset1 : in std_logic;
lcd_rw : out std_logic;                          --read & write control
      lcd_en : out std_logic;                          --enable control
      lcd_rs : out std_logic;                          --data or command control
      lcd1  : out std_logic_vector(7 downto 0);
      b11 : out std_logic;
      b12 : out std_logic;
tottimeout1 : out std_logic_vector(10 downto 0);
ledoutput, go : out std_logic);--
end entity;
architecture final of ReactionGame is
type dataarr1 is array (0 to 17) of std_logic_vector(7 downto 0);
type intarr is array (3 downto 0) of integer range 0 to 9;
signal finaltime : std_logic_vector(15 downto 0);
signal tottimeout : std_logic_vector(10 downto 0);
signal oneeclock : std_logic;
signal j, i, temp : integer := 0;
signal digit : intarr;
signal lcdreset, smallclk1, led1, erase, ack1, gameOver, g1, go1 : std_logic;
signal put_char, temp3 :  std_logic:='1';
signal write_data :  std_logic_vector(7 downto 0);
signal write_row :  std_logic_vector(0 downto 0);
```

```
signal write_column :  std_logic_vector(3 downto 0) := "0000";
constant lcd_data1 : dataarr1 := (x"30", x"31", x"32", x"33", x"34", x"35", x"36",
x"38", x"39", x"41", x"42", x"43", x"44", x"45", x"46", x"4B", x"55");
begin
lcdreset <= not reset1;
tottimeout1 <= tottimeout;
go <= go1;
oonclk : oneclock port map (inclk => masterclock, reset => reset1,
outclk => oneeclock);
game1 : Game port map (switch1 => switch1, masterclock => masterclock,
pwreset => pwreset,
reset1 => reset1, tottimeout => tottimeout, ledoutput => ledoutput,
go1 => go1);
finaltime(0) <= tottimeout(0);
finaltime(1) <= tottimeout(1);
finaltime(2) <= tottimeout(2);
finaltime(3) <= tottimeout(3);
finaltime(4) <= tottimeout(4);
finaltime(5) <= tottimeout(5);
finaltime(6) <= tottimeout(6);
finaltime(7) <= tottimeout(7);
finaltime(8) <= tottimeout(8);
finaltime(9) <= tottimeout(9);
finaltime(10) <= tottimeout(10);
finaltime(11) <= '0';
finaltime(12) <= '0';
finaltime(13) <= '0';
finaltime(14) <= '0';
finaltime(15) <= '0';
display : lcd_controller port map (clk => oneeclock, rst => lcdreset,
erase => erase,
put_char => put_char, write_data => lcd_data1(i), write_row => write_row,
write_column => write_column, lcd_rw => lcd_rw, lcd_en => lcd_en,
lcd_rs => lcd_rs,
lcd1 => lcd1, b11 => b11, b12 => b12, ack => ack1);
temp <= to_integer(unsigned(finaltime));
digit(0) <= to_integer(unsigned(finaltime(3 downto 0)));
digit(1) <= to_integer(unsigned(finaltime(7 downto 4)));
```

```
digit(2) <= to_integer(unsigned(finaltime(11 downto 8)));
digit(3) <= to_integer(unsigned(finaltime(15 downto 12)));
process (oneeclock, ack1, reset1)
begin
if (rising_edge(oneeclock)) then
erase <= '0';
if(ack1 = '1')then
if (j = 0) then
write_column <= "0011";
i <= 16;
i <= digit(0);
put_char <= '1';
write_row <= "0";
elsif (j = 1)then
write_column <= "0010";
i <= digit(1);
put_char <= '1';
write_row <= "0";
elsif (j = 2)then
write_column <= "0001";
i <= digit(2);
put_char <= '1';
write_row <= "0";
elsif (j = 3)then
write_column <= "0000";
i <= 15;
i <= digit(3);
put_char <= '1';
write_row <= "0";
end if;
if(j = 3)then
j <= 0;
else
j <= j + 1;
end if;
end if;
end if;
end process;
```

```
end final;
```

References

[1] *http://wel.ee.iitb.ac.in/teaching_labs/WEL%20Site/ee214/Labsheets2016/LabManual.pdf*