EEE102 Project Report

Semih Akkoç

23 December 2022

Abstract

EEE 102 term project will be to implement Li-Fi protocol using Basys3 and via using this protocol demonstration of a text file transfer and sound file transfer.

Introduction and Description:

Li-Fi is a wireless communication technology that utilizes light to transmit data. This system is capable of transmitting data at high speeds over visible light, ultraviolet, and infrared spectrum. Also, as a technology, Li-Fi is able to function in areas without being affected by electromagnetic interference. Due to the extensive benefits of Li-Fi technology and the theories behind encoding and decoding certain file types. I wish to do the term project on this topic. The project will be implemented using the Basys3 board and LDR sensor to detect the signal, an LED to send the signals, and a buzzer to play the received sound file through it.

Design of the Project:

The project is working on Basys3 and the decoder part has been implemented in a way that it decodes the transmitted data and display the resulting outcome in a specific way, such as if the received data is a text file, then it displays it on the seven-segment display or if the data is sound file it plays it through the buzzer. When it comes to the encoder part, the desired data is going to be stored in the ROMs which have been implemented on Basys3. For instance, the text files have been parsed into their characters, and the characters' ASCII code has been encoded into binary to transmit via LED. When LED is on, the LDR sensor yields higher output, and it translates into 1, and contrarily, it yields a low output of 0, and the received data is then rendered by the decoder to determine the outputs of anode and cathode voltages on seven segment display.

Components:

- Basys3 board
- Photoresistor (LDR sensor)
- Light emitting diode (LED)
- Buzzer

Modules:

Modules used in the project can be summarized in the following list:

- LiFi
  - fClock
  - sClock
  - Transmitter
    - AddressCounter
    - textROM
    - soundROM
    - 2x2 crossbar switch
    - PISO 8-bit Shift Register (Parallel IN Serial OUT)
  - Reciever
    - POSI 8-bit Shift Register (Parallel OUT Serial IN)
    - 8-bit Register (for 1st char)
    - 8-bit Register (for 2nd char)
    - 8-bit Register (for 3rd char)
    - 8-bit Register (for 4th char)
    - refreshCounter (for 7-segment display)
    - Driver
      - Decoder

To clarify what each module's role is, modules will be briefly explained below:

LiFi:

The LiFi module is the top module that implements the receiver and transmitter as intended.

fClock:

Due to the fact that Basys3 has a 10ns clock and the circuit operates at 0.1s, this module generates a clock with a period of 0.1 seconds.

sClock:

Since the circuit operates at 0.1s and the transfer happens with 8-bit chunks, this module generates a clock with a period of 0.8 seconds.

Transmitter:

This module is the transmitter module where the data stored in the ROMs are transmitted via laser.

AddressCounter:

The AddressCounter output is used to change the address inputs of the ROMs so that after changing the counter, ROMs can be searched.

textROM:

The ROM stores the encoded text file information and has been stored in 8-bit chars.

soundROM:

The ROM, which stores the encoded sound file information and has been stored in 8-bit chunks.

2x2 crossbar switch:

This switch is used to swap back and forth with the buzzer and seven-segment display.

PISO 8-bit Shift Register (Parallel IN Serial OUT):

The parallel in serial out 8-bit shift register is utilized to send the data read from the ROMs.

Receiver:

This module is the receiver module where the data received by the light dependant resistor is stored in tentative 8-bit registers and displayed on the seven-segment display or buzzer depending on the select input.

POSI 8-bit Shift Register (Parallel OUT Serial IN)

The parallel-out serial in an 8-bit shift register receives data and stores it temporarily until the 8-bit has been received.

8-bit Register

These registers store the incoming characters for a short amount of time to display them on the seven-segment display. The outputs of these registers then get concatenated and sent to the seven-segment driver as 32-bit vector.

refreshCounter

This counter is used to refresh the anodes and the cathodes therefore the output of this counter is connected to the driver input.

Driver

The driver module has been implemented to show the outputs of the encoded vector in the seven-segment display.

decoder

This module takes the 8-bit encoded data and decodes it according to the ASCII table, and turns it into anodes and cathodes signals.

The Schematics of the Project:

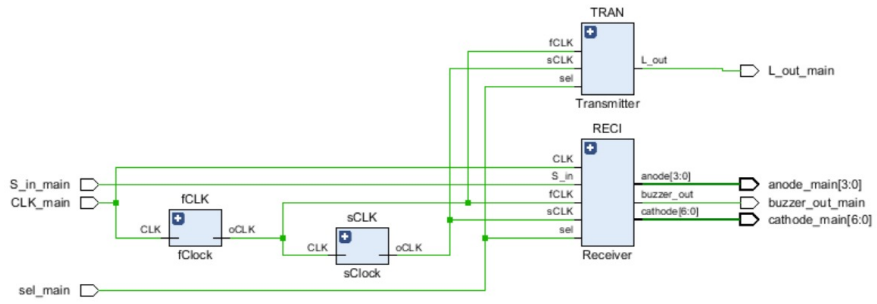

Figure 1: Schematic of the Li-Fi Module

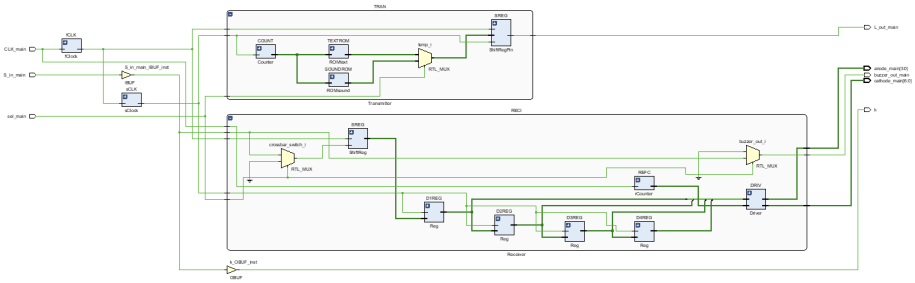

Figure 2: Inside the transistor and receiver module.

YouTube video link:

Implementation of Li-Fi on Basys3 — EEE102 Term Project:
https://youtu.be/tLhvn5wmTDc

Code:

Code 1: (LiFi.vhd)

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

entity LiFi is
    Port (
        CLK_main : in std_logic;
        S_in_main : in std_logic;
        sel_main : in std_logic;
        buzzer_out_main : out std_logic;
        anode_main : out std_logic_vector(3 downto 0);
        cathode_main: out std_logic_vector(6 downto 0);
        k: out std_logic;
        L_out_main : out std_logic
    );
end LiFi;

architecture Behavioral of LiFi is

signal fCLK_main : std_logic := '0';
signal sCLK_main : std_logic := '0';

component Receiver
    Port (
        CLK  : in std_logic;
        fCLK : in std_logic;
        sCLK : in std_logic;
        S_in : in std_logic;
        sel  : in std_logic;
        buzzer_out: out std_logic;
        anode : out std_logic_vector(3 downto 0);
        cathode: out std_logic_vector(6 downto 0)
    );
end component;

component Transmitter
    Port (
        fCLK : in std_logic;
        sCLK : in std_logic;
        sel : in std_logic;
        L_out : out std_logic
     );
end component;

component fClock
    Port (
        CLK : in std_logic;
        oCLK : out std_logic
    );
end component;

component sClock
    Port (
        CLK : in std_logic;
```

```
        oCLK : out std_logic
    );
end component;


begin

TRAN: Transmitter PORT MAP (
    fCLK  => fCLK_main,
    sCLK  => sCLK_main,
    sel   => sel_main,
    L_out => L_out_main
);

RECI: Receiver PORT MAP (
    CLK  => CLK_main,
    fCLK => fCLK_main,
    sCLK => sCLK_main,
    S_in => S_in_main,
    sel  => sel_main,
    buzzer_out => buzzer_out_main,
    anode => anode_main,
    cathode => cathode_main
);

-- generate fCLK sCLK
fCLK: fClock PORT MAP (
    CLK => CLK_main,
    oCLK => fCLK_main
);

sCLK: sClock PORT MAP (
    CLK => fCLK_main,
    oCLK => sCLK_main
);

k<=S_in_main;

end Behavioral;
```

Code 2: (Transmitter.vhd)

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

entity Transmitter is
    Port (
        fCLK : in std_logic;
        sCLK : in std_logic;
        sel : in std_logic;
        L_out : out std_logic
     );
end Transmitter;

architecture Behavioral of Transmitter is

component ShiftRegPin
```

```
    Port(
        CLK : in std_logic;
        LOAD : in std_logic;
        Din : in std_logic_vector(7 downto 0);
        Dout : out std_logic
        );
end component;

component ROMtext
    Port (
        address : in std_logic_vector(5 downto 0);
        data : out std_logic_vector(7 downto 0)
     );
end component;

component ROMsound
    Port (
        address : in std_logic_vector(5 downto 0);
        data : out std_logic_vector(7 downto 0)
     );
end component;

component Counter
    Port (
        CLK: in std_logic; -- clock input
        -- RST: in std_logic;
        q: out std_logic_vector(5 downto 0) -- output 4-bit counter
     );
end component;

signal count_add : std_logic_vector(5 downto 0);

signal temp_text : std_logic_vector(7 downto 0) := (others => '0');
signal temp_sound : std_logic_vector(7 downto 0) := (others => '0');
signal temp : std_logic_vector(7 downto 0) := (others => '1');
signal reset : std_logic := '0';

begin

-- tentative counter for text
COUNT: Counter PORT MAP (
    CLK=>sCLK,
    -- RST=>reset,
    q=>count_add
);

-- process (count_add) begin
--     if (count_add = "111111") then
--         reset <= '1';
--     else
--         reset <= '0';
--     end if;
-- end process;

TEXTROM: ROMtext PORT MAP (
    address=>count_add,
    data=>temp_text
);
```

```
SOUNDROM: ROMsound PORT MAP (
    address=>count_add,
    data=>temp_sound
);

SREG: ShiftRegPin PORT MAP (
    CLK => fCLK,
    LOAD => sCLK, -- not sure about this think
    Din => temp,
    Dout => L_out
);
-- may need to add some process for sCLK in shiftreg

process (sel, temp_text, temp_sound) begin
    if (sel = '0') then
        temp <= temp_text;
    elsif (sel = '1') then
        temp <= temp_sound;
    end if;
end process;

end Behavioral;
```

Code 3: (Reciever.vhd)

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

entity Receiver is
    Port (
    CLK  : in std_logic;
    fCLK : in std_logic;
    sCLK : in std_logic;
    S_in : in std_logic;
    sel  : in std_logic;
    buzzer_out: out std_logic;
    anode : out std_logic_vector(3 downto 0);
    cathode: out std_logic_vector(6 downto 0)
    );
end Receiver;

architecture Behavioral of Receiver is
-- how to sel??

component ShiftReg
    Port (
        CLK  : in std_logic;
        S_in : in std_logic;
        Pout : out std_logic_vector(7 downto 0)
    );
end component;

component Reg
    Port(
        CLK : in std_logic;
        D: in std_logic_vector(7 downto 0);
```

```
        Q: out std_logic_vector(7 downto 0)
    );
end component;

component Driver
    Port (
        sel : in std_logic_vector(1 downto 0); -- anode selector
        in_letter: in std_logic_vector(31 downto 0);
        anode : out std_logic_vector(3 downto 0);
        cathode: out std_logic_vector(6 downto 0)
     );
end component;

component rCounter
    Port(
        CLK: in std_logic; -- clock input
        -- RST: in std_logic;
        refresh: out std_logic_vector(1 downto 0) -- output 4-bit counter
    );
end component;

signal reset: std_logic := '0';
signal temp_read: std_logic_vector (7 downto 0);

-- signal d1in : std_logic_vector (7 downto 0) := "00000000";
-- signal d2in : std_logic_vector (7 downto 0) := "00000000";
-- signal d3in : std_logic_vector (7 downto 0) := "00000000";
-- signal d4in : std_logic_vector (7 downto 0) := "00000000";

signal d1out : std_logic_vector (7 downto 0); -- do zero zero zero 00000
signal d2out : std_logic_vector (7 downto 0);
signal d3out : std_logic_vector (7 downto 0);
signal d4out : std_logic_vector (7 downto 0);

signal in_letter: std_logic_vector(31 downto 0);

signal crossbar_switch: std_logic;
signal refresh_counter: std_logic_vector(1 downto 0);


begin

SREG : ShiftReg PORT MAP(
    CLK=>fCLK,
    S_in=>crossbar_switch,
    Pout=>temp_read
);

D1REG : Reg PORT MAP(
    CLK=>sCLK,
    D=>temp_read,
    Q=>d1out
);

D2REG : Reg PORT MAP(
    CLK=>sCLK,
    D=>d1out,
    Q=>d2out
);
```

```
D3REG : Reg PORT MAP(
    CLK=>sCLK,
    D=>d2out,
    Q=>d3out
);

D4REG : Reg PORT MAP(
    CLK=>sCLK,
    D=>d3out,
    Q=>d4out
);

DRIV: Driver PORT MAP(
    sel=>refresh_counter,
    in_letter=>in_letter,
    anode=>anode,
    cathode=>cathode
); -- something might go wrong with reset

REFC: rCounter PORT MAP(
    CLK=>CLK,
    -- RST=>reset,
    refresh=>refresh_counter
);

-- process (refresh_counter) begin
--     if (refresh_counter = "11") then
--         reset <= '1';
--     else
--         reset <= '0';
--     end if;
-- end process;

process (sel, S_in) begin
    if (sel = '0') then
        buzzer_out <= '0';
    elsif (sel = '1') then
        buzzer_out <= S_in;
    end if;
end process;

process (sel, S_in) begin
    if (sel = '0') then
        crossbar_switch <= S_in;
    elsif (sel = '1') then
        crossbar_switch <= '0';
    end if;
end process;


-- refresh_counter <= (others => '0') when refresh_counter =
    "11111111111111111111" else (others => '1'); -- to reset

--in_letter(31 downto 24) <= d4out;
--in_letter(23 downto 16) <= d3out;
--in_letter(15 downto 8) <= d2out;
--in_letter(7 downto 0) <= d1out;
```

```
 in_letter <= d4out & d3out & d2out & d1out;

end Behavioral;
```

Code 5: (ClockDiv.vhd)

```
 library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
entity ClockDiv is
port (
   clk_in: in std_logic; -- clock input on FPGA
   clk_out: out std_logic -- clock output
  );
end ClockDiv;

architecture Behavioral of ClockDiv is

signal divisor: std_logic_vector(27 downto 0):=(others =>'0');

begin
 process(clk_in)
 begin
 if(rising_edge(clk_in)) then
 divisor <= divisor + x"0000001";

 if(divisor>=x"04C4B40") then -- for running simulation -- comment when
     running on FPGA
 -- divisor = 1 => divide clock by half for simulation purposes
 divisor <= x"0000000";
 end if;

 if(divisor<x"04C4B40") then -- replace x"0000001" by x"17D7840" when
     running on FPGA

 clk_out <= '1';

 else

 clk_out <= '0';

 end if;

 end if;
 end process;

end Behavioral;
```

Code 6: (Counter.vhd)

```
 library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use ieee.numeric_std.all;
```

```
entity Counter is
    Port (
        CLK: in std_logic; -- clock input
        -- RST: in std_logic;
        q: out std_logic_vector(5 downto 0) -- output 4-bit counter
     );
end Counter;

architecture Behavioral of Counter is

--signal counter_up: std_logic_vector(2 downto 0) := (others => '0');
--signal counter_up2: std_logic_vector(2 downto 0) := (others => '0');

signal counter_up : integer range 0 to 8 := 0;

begin

process (CLK) begin
    if (rising_edge(CLK)) then
        if (counter_up = 8) then
            counter_up <= 0;
        else
            counter_up <= counter_up + 1;
        end if;
    end if;
end process;


-- counter_up <= (others => '0') when counter_up = "111111" else (others =>
    '1'); -- to reset

q <= std_logic_vector(to_unsigned(counter_up, q'length));

end Behavioral;
```

Code 7: (decoder.vhd)

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

entity Decoder is
    Port (
        letter : in std_logic_vector(7 downto 0);
        cathode : out std_logic_vector(6 downto 0)
     );
end Decoder;

architecture Behavioral of Decoder is

begin

process(letter) begin
    -- report "letter: " & std_logic'image(letter(0)) &
        std_logic'image(letter(1)) & std_logic'image(letter(2)) &
```

```
    std_logic'image(letter(3)) & std_logic'image(letter(4)) &
    std_logic'image(letter(5)) & std_logic'image(letter(6)) &
    std_logic'image(letter(7)) ;
case letter is
    when "00100000" => cathode <= "1111111"; -- /* (space) */
    when "00100001" => cathode <= "1001111"; -- /* ! */
    when "00100010" => cathode <= "1011101"; -- /* " */
    when "00100011" => cathode <= "1000000"; -- /* # */
    when "00100100" => cathode <= "0100100"; -- /* $ */
    when "00100101" => cathode <= "1011010"; -- /* % */
    when "00100110" => cathode <= "1001110"; -- /* & */
    when "00100111" => cathode <= "1111101"; -- /* ' */
    when "00101000" => cathode <= "0110101"; -- /* ( */
    when "00101001" => cathode <= "0010111"; -- /* ) */
    when "00101010" => cathode <= "0111101"; -- /* * */
    when "00101011" => cathode <= "1111000"; -- /* + */
    when "00101100" => cathode <= "1111011"; -- /* , */
    when "00101101" => cathode <= "1111110"; -- /* - */
    when "00101110" => cathode <= "1111111"; -- /* . */
    when "00101111" => cathode <= "1011010"; -- /* / */
    when "00110000" => cathode <= "0000001"; -- /* 0 */
    when "00110001" => cathode <= "1001111"; -- /* 1 */
    when "00110010" => cathode <= "0010010"; -- /* 2 */
    when "00110011" => cathode <= "0000110"; -- /* 3 */
    when "00110100" => cathode <= "1001100"; -- /* 4 */
    when "00110101" => cathode <= "0100100"; -- /* 5 */
    when "00110110" => cathode <= "0100000"; -- /* 6 */
    when "00110111" => cathode <= "0001111"; -- /* 7 */
    when "00111000" => cathode <= "0000000"; -- /* 8 */
    when "00111001" => cathode <= "0000100"; -- /* 9 */
    when "00111010" => cathode <= "0110111"; -- /* : */
    when "00111011" => cathode <= "0100111"; -- /* ; */
    when "00111100" => cathode <= "0111100"; -- /* < */
    when "00111101" => cathode <= "1110110"; -- /* = */
    when "00111110" => cathode <= "0011110"; -- /* > */
    when "00111111" => cathode <= "0011010"; -- /* ? */
    when "01000000" => cathode <= "0000010"; -- /* @ */
    when "01000001" => cathode <= "0001000"; -- /* A */
    when "01000010" => cathode <= "1100000"; -- /* B */
    when "01000011" => cathode <= "0110001"; -- /* C */
    when "01000100" => cathode <= "1000010"; -- /* D */
    when "01000101" => cathode <= "0110000"; -- /* E */
    when "01000110" => cathode <= "0111000"; -- /* F */
    when "01000111" => cathode <= "0100001"; -- /* G */
    when "01001000" => cathode <= "1001000"; -- /* H */
    when "01001001" => cathode <= "1111001"; -- /* I */
    when "01001010" => cathode <= "1000011"; -- /* J */
    when "01001011" => cathode <= "0101000"; -- /* K */
    when "01001100" => cathode <= "1110001"; -- /* L */
    when "01001101" => cathode <= "0101011"; -- /* M */
    when "01001110" => cathode <= "0001001"; -- /* N */
    when "01001111" => cathode <= "0000001"; -- /* O */
    when "01010000" => cathode <= "0011000"; -- /* P */
    when "01010001" => cathode <= "0010100"; -- /* Q */
    when "01010010" => cathode <= "0011001"; -- /* R */
    when "01010011" => cathode <= "0100100"; -- /* S */
    when "01010100" => cathode <= "1110000"; -- /* T */
    when "01010101" => cathode <= "1000001"; -- /* U */
    when "01010110" => cathode <= "1000001"; -- /* V */
```

```
        when "01010111" => cathode <= "1010101"; -- /* W */
        when "01011000" => cathode <= "1001000"; -- /* X */
        when "01011001" => cathode <= "1000100"; -- /* Y */
        when "01011010" => cathode <= "0010010"; -- /* Z */
        when "01011011" => cathode <= "0110001"; -- /* [ */
        when "01011100" => cathode <= "1101100"; -- /* \ */
        when "01011101" => cathode <= "0000111"; -- /* ] */
        when "01011110" => cathode <= "0011101"; -- /* ^ */
        when "01011111" => cathode <= "1110111"; -- /* _ */
        when "01100000" => cathode <= "1011111"; -- /* ` */
        when "01100001" => cathode <= "0000010"; -- /* a */
        when "01100010" => cathode <= "1100000"; -- /* b */
        when "01100011" => cathode <= "1110010"; -- /* c */
        when "01100100" => cathode <= "1000010"; -- /* d */
        when "01100101" => cathode <= "0010000"; -- /* e */
        when "01100110" => cathode <= "0111000"; -- /* f */
        when "01100111" => cathode <= "0000100"; -- /* g */
        when "01101000" => cathode <= "1101000"; -- /* h */
        when "01101001" => cathode <= "1111011"; -- /* i */
        when "01101010" => cathode <= "1100111"; -- /* j */
        when "01101011" => cathode <= "0101000"; -- /* k */
        when "01101100" => cathode <= "1111001"; -- /* l */
        when "01101101" => cathode <= "1101011"; -- /* m */
        when "01101110" => cathode <= "1101010"; -- /* n */
        when "01101111" => cathode <= "1100010"; -- /* o */
        when "01110000" => cathode <= "0011000"; -- /* p */
        when "01110001" => cathode <= "0001100"; -- /* q */
        when "01110010" => cathode <= "1111010"; -- /* r */
        when "01110011" => cathode <= "0100100"; -- /* s */
        when "01110100" => cathode <= "1110000"; -- /* t */
        when "01110101" => cathode <= "1100011"; -- /* u */
        when "01110110" => cathode <= "1100011"; -- /* v */
        when "01110111" => cathode <= "1101011"; -- /* w */
        when "01111000" => cathode <= "1001000"; -- /* x */
        when "01111001" => cathode <= "1000100"; -- /* y */
        when "01111010" => cathode <= "0010010"; -- /* z */
        when "01111011" => cathode <= "1001110"; -- /* { */
        when "01111100" => cathode <= "1111001"; -- /* | */
        when "01111101" => cathode <= "1111000"; -- /* } */
        when "01111110" => cathode <= "0111111"; -- /* ~ */
        when "01111111" => cathode <= "1111111"; -- /* (del) */
        when others => cathode <= "1111111"; --off
    end case;
end process;

end Behavioral;
```

Code 8: (Driver.vhd)

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


entity Driver is
  Port (
```

```
    sel : in std_logic_vector(1 downto 0); -- anode selector
    in_letter: in std_logic_vector(31 downto 0);
    anode : out std_logic_vector(3 downto 0);
    cathode: out std_logic_vector(6 downto 0)
   );
end Driver;

architecture Behavioral of Driver is


component Decoder
    Port (
        letter : in std_logic_vector(7 downto 0);
        cathode : out std_logic_vector(6 downto 0)
     );
end component;


signal letter: std_logic_vector(7 downto 0);

begin

D : decoder PORT MAP(letter=>letter, cathode=>cathode);

process (sel, in_letter) begin
    case sel is
        when "00" =>
            anode <= "0111"; -- 1st digit
            letter <= in_letter(31 downto 24);

        when "01" =>
            anode <= "1011"; -- 2nd digit
            letter <= in_letter(23 downto 16);

        when "10" =>
            anode <= "1101"; -- 3rd digit
            letter <= in_letter(15 downto 8);

        when "11" =>
            anode <= "1110"; -- 4th digit
            letter <= in_letter(7 downto 0);

        when others => anode <= "1111";
    end case;
end process;

end Behavioral;
```

Code 9: (fClock.vhd)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use ieee.numeric_std.all;

entity fClock is
```

```
    Port (
        CLK : in std_logic;
        oCLK : out std_logic
    );
end fClock;

architecture Behavioral of fClock is

signal fCounter : integer range 0 to 16777216 := 0; -- in order to get 10
    million -> 0.1s
signal fCLK_main : std_logic := '0';

begin

-- clock with 0.1s
process (CLK) begin
    if (rising_edge(CLK)) then
        if (fCounter = 10000000) then
            fCounter <= 0;
        else
            fCounter <= fCounter + 1;
        end if;
    end if;
end process;

fCLK_main <= '1' when fCounter >= 5000000 else '0'; -- until 5 million

oCLK <= fCLK_main;


end Behavioral;
```

Code 10: (rClock.vhd)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use ieee.numeric_std.all;

entity rCounter is
    Port(
        CLK: in std_logic; -- clock input
        -- RST: in std_logic;
        refresh: out std_logic_vector(1 downto 0) -- output 4-bit counter
    );
end rCounter;

architecture Behavioral of rCounter is

signal rc : integer range 0 to 1048576 := 0;
signal t: std_logic_vector(19 downto 0);

begin

-- display refresher signal
process (CLK) begin
```

```
    if (rising_edge(CLK)) then
        if (rc = 1048576) then
            rc <= 0;
        else
            rc <= rc + 1;
        end if;
    end if;
end process;

t <= std_logic_vector(to_unsigned(rc, t'length));
refresh <= t(19 downto 18);

end Behavioral;
```

Code 11: (Reg.vhd)

```
library ieee;
use ieee.std_logic_1164.all;

entity Reg is
    Port(
    CLK : in std_logic;
    D: in std_logic_vector(7 downto 0);
    Q: out std_logic_vector(7 downto 0)
    );
end Reg;

architecture Behavioral of Reg is

begin

process (CLK) begin
    if (rising_edge(CLK)) then
        Q <= D;
    end if;
end process;

end Behavioral;
```

Code 12: (ROMsound.vhd)

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

entity ROMsound is
    Port (
        address : in std_logic_vector(5 downto 0);
        data : out std_logic_vector(7 downto 0)
     );
end ROMsound;

architecture Behavioral of ROMsound is

type mem is array ( 0 to 61 ) of std_logic_vector(7 downto 0);
```

```
constant my_rom : mem := (
    0 => "0100111",
    1 => "0011011",
    2 => "1101110",
    3 => "1000010",
    4 => "0000011",
    5 => "1010001",
    6 => "1010000",
    7 => "1100001",
    8 => "0111010",
    9 => "0001000",
    10 => "0001101",
    11 => "0000110",
    12 => "0001011",
    13 => "1001001",
    14 => "1100110",
    15 => "1101000",
    16 => "0010000",
    17 => "0011000",
    18 => "1001110",
    19 => "1010111",
    20 => "0100001",
    21 => "0000001",
    22 => "1100110",
    23 => "1101111",
    24 => "0110110",
    25 => "1011001",
    26 => "0101110",
    27 => "1000110",
    28 => "1000011",
    29 => "0100101",
    30 => "1011100",
    31 => "1100111",
    32 => "0010000",
    33 => "0011010",
    34 => "0101101",
    35 => "1010111",
    36 => "0000011",
    37 => "0111101",
    38 => "1100100",
    39 => "1110100",
    40 => "0110000",
    41 => "1011011",
    42 => "1001110",
    43 => "1000010",
    44 => "0000010",
    45 => "0100100",
    46 => "1000000",
    47 => "1100111",
    48 => "0111010",
    49 => "1011001",
    50 => "0101110",
    51 => "0110111",
    52 => "0011001",
    53 => "0000001",
    54 => "1110010",
    55 => "1110101",
    56 => "0111001",
    57 => "1011101",
```

```
    58 => "0101100",
    59 => "1100010",
    60 => "0000001",
    61 => "1110000");

begin

process (address) begin
case address is
    when "000000" => data <= my_rom(0);
    when "000001" => data <= my_rom(1);
    when "000010" => data <= my_rom(2);
    when "000011" => data <= my_rom(3);
    when "000100" => data <= my_rom(4);
    when "000101" => data <= my_rom(5);
    when "000110" => data <= my_rom(6);
    when "000111" => data <= my_rom(7);
    when "001000" => data <= my_rom(8);
    when "001001" => data <= my_rom(9);
    when "001010" => data <= my_rom(10);
    when "001011" => data <= my_rom(11);
    when "001100" => data <= my_rom(12);
    when "001101" => data <= my_rom(13);
    when "001110" => data <= my_rom(14);
    when "001111" => data <= my_rom(15);
    when "010000" => data <= my_rom(16);
    when "010001" => data <= my_rom(17);
    when "010010" => data <= my_rom(18);
    when "010011" => data <= my_rom(19);
    when "010100" => data <= my_rom(20);
    when "010101" => data <= my_rom(21);
    when "010110" => data <= my_rom(22);
    when "010111" => data <= my_rom(23);
    when "011000" => data <= my_rom(24);
    when "011001" => data <= my_rom(25);
    when "011010" => data <= my_rom(26);
    when "011011" => data <= my_rom(27);
    when "011100" => data <= my_rom(28);
    when "011101" => data <= my_rom(29);
    when "011110" => data <= my_rom(30);
    when "011111" => data <= my_rom(31);
    when "100000" => data <= my_rom(32);
    when "100001" => data <= my_rom(33);
    when "100010" => data <= my_rom(34);
    when "100011" => data <= my_rom(35);
    when "100100" => data <= my_rom(36);
    when "100101" => data <= my_rom(37);
    when "100110" => data <= my_rom(38);
    when "100111" => data <= my_rom(39);
    when "101000" => data <= my_rom(40);
    when "101001" => data <= my_rom(41);
    when "101010" => data <= my_rom(42);
    when "101011" => data <= my_rom(43);
    when "101100" => data <= my_rom(44);
    when "101101" => data <= my_rom(45);
    when "101110" => data <= my_rom(46);
    when "101111" => data <= my_rom(47);
    when "110000" => data <= my_rom(48);
    when "110001" => data <= my_rom(49);
```

```
    when "110010" => data <= my_rom(50);
    when "110011" => data <= my_rom(51);
    when "110100" => data <= my_rom(52);
    when "110101" => data <= my_rom(53);
    when "110110" => data <= my_rom(54);
    when "110111" => data <= my_rom(55);
    when "111000" => data <= my_rom(56);
    when "111001" => data <= my_rom(57);
    when "111010" => data <= my_rom(58);
    when "111011" => data <= my_rom(59);
    when "111100" => data <= my_rom(60);
    when "111101" => data <= my_rom(61);
    when others => data <= "01000001";
end case;

end process;

end Behavioral;
```

Code 13: (ROMtext.vhd)

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

entity ROMtext is
    Port (
        address : in std_logic_vector(5 downto 0);
        data : out std_logic_vector(7 downto 0)
     );
end ROMtext;

architecture Behavioral of ROMtext is
                        -- can change this size
type mem is array ( 0 to 33 ) of std_logic_vector(7 downto 0);

constant my_rom : mem := (
    0 => "0100001",
    1 => "0011101",
    2 => "0101101",
    3 => "1100111",
    4 => "0101001",
    5 => "0000001",
    6 => "1110010",
    7 => "1100001",
    8 => "0111101",
    9 => "0011000",
    10 => "0101101",
    11 => "1100010",
    12 => "0000011",
    13 => "1010001",
    14 => "1011110",
    15 => "1110011",
    16 => "0111010",
    17 => "1011011",
    18 => "1000100",
    19 => "0000110",
```

```
    20 => "1111011",
    21 => "0101101",
    22 => "1101010",
    23 => "1111001",
    24 => "0110000",
    25 => "1011011",
    26 => "1001100",
    27 => "0010010",
    28 => "0000011",
    29 => "0101101",
    30 => "1011110",
    31 => "1110011",
    32 => "0111010",
    33 => "1011011");

begin

process (address) begin
-- case address is
--     when "000000" => data <= my_rom(0);
--     when "000001" => data <= my_rom(1);
--     when "000010" => data <= my_rom(2);
--     when "000011" => data <= my_rom(3);
--     when "000100" => data <= my_rom(4);
--     when "000101" => data <= my_rom(5);
--     when "000110" => data <= my_rom(6);
--     when "000111" => data <= my_rom(7);
--     when "001000" => data <= my_rom(8);
--     when "001001" => data <= my_rom(9);
--     when "001010" => data <= my_rom(10);
--     when "001011" => data <= my_rom(11);
--     when "001100" => data <= my_rom(12);
--     when "001101" => data <= my_rom(13);
--     when "001110" => data <= my_rom(14);
--     when "001111" => data <= my_rom(15);
--     when "010000" => data <= my_rom(16);
--     when "010001" => data <= my_rom(17);
--     when "010010" => data <= my_rom(18);
--     when "010011" => data <= my_rom(19);
--     when "010100" => data <= my_rom(20);
--     when "010101" => data <= my_rom(21);
--     when "010110" => data <= my_rom(22);
--     when "010111" => data <= my_rom(23);
--     when "011000" => data <= my_rom(24);
--     when "011001" => data <= my_rom(25);
--     when "011010" => data <= my_rom(26);
--     when "011011" => data <= my_rom(27);
--     when "011100" => data <= my_rom(28);
--     when "011101" => data <= my_rom(29);
--     when "011110" => data <= my_rom(30);
--     when "011111" => data <= my_rom(31);
--     when "100000" => data <= my_rom(32);
--     when "100001" => data <= my_rom(33);
--     when others => data <= "01000001";
--end case;
    data <= my_rom(to_integer(unsigned(address)));

end process;
```

```
end Behavioral;
```

Code 14: (ShiftRegPin.vhd)

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

entity ShiftRegPin is
    Port(
        CLK : in std_logic;
        LOAD : in std_logic;
        Din : in std_logic_vector(7 downto 0);
        Dout : out std_logic
        );
end ShiftRegPin;

architecture Behavioral of ShiftRegPin is

signal temp: std_logic_vector(7 downto 0);

begin

process (CLK, LOAD, Din) begin
    if (rising_edge(CLK)) then
        if (LOAD = '1') then
            temp <= Din;
        else
            temp <= temp(6 downto 0) & '0';
        end if;
    end if;
end process;


Dout <= temp(7);

end Behavioral;
```

Code 15: (sClock.vhd)

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

entity sClock is
    Port (
        CLK : in std_logic;
        oCLK : out std_logic
     );
end sClock;

architecture Behavioral of sClock is

signal sCounter : std_logic_vector(2 downto 0) := (others => '0'); --
    second fake clk -> 0.8s
signal sCLK_main : std_logic := '0';
```

```
begin

-- clock with 0.8s
process (CLK) begin
    if (rising_edge(CLK)) then
        if (sCounter = "111") then
            sCounter <= (others => '0');
        else
            sCounter <= sCounter + '1';
        end if;
    end if;
end process;

sCLK_main <= '1' when sCounter >= "100" else '0'; -- until 4

oCLK <= sCLK_main;

end Behavioral;
```

Code 16: (ShiftReg.vhd)

```
library ieee;
use ieee.std_logic_1164.all;

entity ShiftReg is
    Port (
        CLK, S_in : in std_logic;
        Pout : out std_logic_vector(7 downto 0)
    );
end ShiftReg;

architecture Behavorial of ShiftReg is

signal temp: std_logic_vector(7 downto 0);

begin

process (CLK) begin
    if (rising_edge(CLK)) then
        temp <= temp(6 downto 0) & S_in; -- 6 downto 0
    end if ;
end process;

Pout <= temp;
end Behavorial;
```