Project 4: Reaction Timer

Tenzin Choden Thinley

CS232

Project Overview

This report documents the design, simulation and testing of a simple programmable light display using a state machine. The programmable light system follows a predefined instruction set and operates using a read-only memory(ROM) component that stores a sequence of commands. This system consists of two primary VHDL files:

- Lights.vhd: implements the state machine that controls the execution of instructions
- Lightrom.vhd: defines the read-only memory that controls the execution of instructions.

1. **Designing LightsRom and Lights For the Lights Display**

The goal of this section of the project is to use VHDL to design a simple read-only memory(ROM) that holds a program and a state machine that executes stored instructions.

a. VHDL Design:

The lights display consists of a state machine that can execute eight different instructions over a 16-line instruction program. The main components include:

1. Program Counter: keeps track of the current instruction.
2. Instruction Register: holds the current instruction
3. Light Register: An 8-bit register that modifies the display state.
4. ROM: Stores a predefined instruction set to be executed.

The following is an image of the Lights.vhd that was implemented:

```
    port(
       clk       : in std_logic;
       reset     : in std_logic;
       lightsig  : out   std_logic_vector(7 downto 0);  -- output signal that will drive the LED display on th
       IRView    : out   std_logic_vector(2 downto 0);  -- IR and PC will let you monitor the values of the IR
       PCView    : out unsigned (3 downto 0)
    );

end entity;

architecture rtl of lights is
    -- uncomment the following section if lightrom
    --component lightrom
       --port(
          --addr      : in std_logic_vector (3 downto 0);
          --data   : out std_logic_vector(2 downto 0)
       --);
    --end component;

    -- uncomment the following secntion if lightromMyprog
    --component lightromMyprog
       --port(
          --addr    : in std_logic_vector (3 downto 0);
          --data   : out std_logic_vector(2 downto 0)
       --);
    --end component;

    -- uncomment the following secntion if lightrom2Myprog
    component lightrom2Myprog
       port(
          addr   : in std_logic_vector (3 downto 0);
```

Figure 1. Image of the first part of lights.vhd.

```
    -- Build an enumerated type for the state machine
    type state_type is (sFetch, sExecute);
    signal IR  : std_logic_vector(2 downto 0);
    signal PC  : unsigned(3 downto 0);
    signal LR  : unsigned (7 downto 0);
    signal ROMvalue  : std_logic_vector(2 downto 0);
    signal state   : state_type;

begin

    -- Logic to advance to the next state
    process (clk, reset)
    begin
       if reset = '0' then
          state <= sFetch;
          IR <= "000";
          PC <= "0000";
          LR <= "00000000";

       elsif (rising_edge(clk)) then
          case state is
             when sFetch=>
                IR <= ROMValue;
                PC <= PC + 1;
                state <= sExecute;
             when sExecute=>
                case IR is
                   when "000" =>
                      LR <= "00000000";
                   when "101" =>
                      LR <= NOT LR;
                   when "001" =>
```

Figure 2. Image of the second part of lights.vhd that was implemented.

```
                        LR <= LR(6 downto 0) & '0';
                     when "011" =>
                        LR <= LR + 1;
                     when "100" =>
                        LR <= LR - 1;
                     when others =>
                        LR <= (others => '0');
                  end case;
                  state <= sFetch;
            end case;
         end if;
      end process;

      -- Output depends solely on the current state

      lightsig <= std_logic_vector(LR);
      IRView <= IR;
      PCView <= PC;

      -- Uncomment below section if lightrom
      --lightrom1 : lightrom
         --port map (
            --addr => std_logic_vector(PC),
            --data => ROMValue
         --);

      -- Uncomment below section if lightromMyprog
      --lightromMyprog1 : lightromMyprog
         --port map (
            --addr => std_logic_vector(PC),
            --data => ROMValue
         --);
```

Figure 3. Image of the third part of lights.vhd that was implemented.

```
   -- Uncomment below section if lightromMyprog
   --lightromMyprog1 : lightromMyprog
      --port map (
         --addr => std_logic_vector(PC),
         --data => ROMValue
      --);

   -- Uncomment below section if lightrom2Myprog
   lightrom2Myprog1 : lightrom2Myprog
      port map (
         addr => std_logic_vector(PC),
         data => ROMValue
      );



end rtl;
```

Figure 4. Image of the fourth part of lights.vhd that was implemented.

The LightRom has two states:

- sFetch: Fetches the instruction from memory and increments the program counter
- sExecute: Executes the instruction based on the machine's instruction set from ROM.

The following is a picture of lightrom.vhd that was implemented:

```
entity lightrom is

   port
   (
      addr     : in std_logic_vector (3 downto 0);
      data   : out std_logic_vector(2 downto 0)
   );

end entity;

architecture rtl of lightrom is

begin
     data <=
          "000" when addr = "0000" else -- move 0s to LR  00000000
          "101" when addr = "0001" else -- bit invert LR  11111111
          "101" when addr = "0010" else -- bit invert LR  00000000
          "101" when addr = "0011" else -- bit invert LR  11111111
          "001" when addr = "0100" else -- shift LR right 01111111
          "001" when addr = "0101" else -- shift LR right 00111111
          "111" when addr = "0110" else -- rotate LR left 01111110
          "111" when addr = "0111" else -- rotate LR left 11111100
          "111" when addr = "1000" else -- rotate LR left 11111001
          "111" when addr = "1001" else -- rotate LR left 11110011
          "010" when addr = "1010" else -- shift LR left  11100110
          "010" when addr = "1011" else -- shift LR left  11001100
          "011" when addr = "1100" else -- add 1 to LR    11001101
          "100" when addr = "1101" else -- sub 1 from LR  11001100
          "101" when addr = "1110" else -- bit invert LR  00110011
          "011";                        -- add 1 to LR    00110100
end rtl;
```

Figure 2. Image of lightrom.vhd.

b. Testing lights.vhd and lightsrom.vhd using lightsbench.vhd:

The system was tested using the GHDL Wave Simulation and an animation using the vcd_movie.py file. This is the link to the video that validates my design. The following picture of the GHDL wave simulation also validates the design of the lights system.

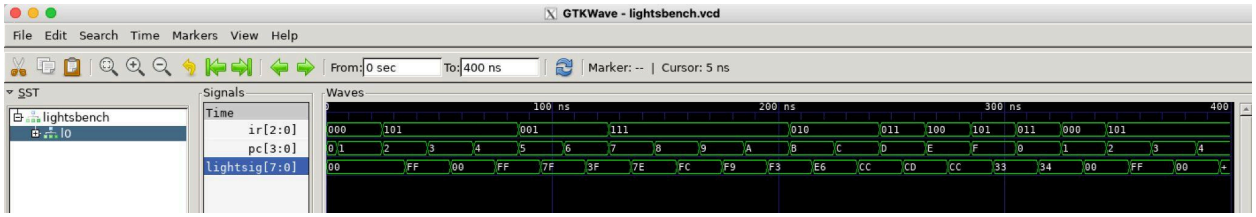

Figure 3. Image of the GHDL Wave Simulation of lightsbench.vhd.

Both the GHDL wave simulation and the video validate that my files accurately translate the instructions to the light display. For the first four signals, the LEDs are supposed to blink. This is because the instruction first starts with all the LEDs turned off, so the signals are 00000000. It then inverts the LED signals and turns them all on, 11111111. After that, it is again inverted and turns all of the LEDs off, 00000000, and again they are inverted and turned back on. This behavior is clearly reflected in the video as the LEDs flash for the first few seconds. Similarly, in GTKWave, the lightsig[7:0] transitions from 00 to FF, back to 00, and then to FF, confirming the correctness of execution. Similarly, after this pattern, the instruction shifts the LR to the right two times and this is also depicted in the video where the first two bits turn off after a few seconds. Thereby, we can say that my design has valid.

2. Designing Other Instruction Sequences

Another key task of this project was to design and implement two different instruction sequences to be stored in Lightrom. These sequences execute different behaviours, demonstrating the programmable nature of this system.

a. VHDL Files:

To do this I created the following files:

1. lightromMyprog.vhd: The following is a picture of the file and this is the link to the video that confirms that my files are functioning correctly.

```
entity lightromMyprog is

    port
    (
        addr    : in std_logic_vector (3 downto 0);
        data    : out std_logic_vector(2 downto 0)
    );

end entity;

architecture rtl of lightromMyprog is

begin
     data <=
          "000" when addr = "0000" else
          "101" when addr = "0001" else
          "000" when addr = "0010" else
          "101" when addr = "0011" else
          "001" when addr = "0100" else
          "001" when addr = "0101" else
          "001" when addr = "0110" else
          "001" when addr = "0111" else
          "101" when addr = "1000" else
          "010" when addr = "1001" else
          "010" when addr = "1010" else
          "010" when addr = "1011" else
          "010" when addr = "1100" else
          "000" when addr = "1101" else
          "101" when addr = "1110" else
          "011";
end rtl;
```

Figure 4. Image of the lightromMyprog.vhd file.

2. lightrom2Myprog.vhd: The following is a picture of the file and this is the link to the video that confirms that my files are functioning correctly.

```
entity lightrom2Myprog is

    port
    (
        addr    : in std_logic_vector (3 downto 0);
        data    : out std_logic_vector(2 downto 0)
    );

end entity;

architecture rtl of lightrom2Myprog is

begin
     data <=
          "000" when addr = "0000" else
          "101" when addr = "0001" else
          "101" when addr = "0010" else
          "101" when addr = "0011" else
          "010" when addr = "0100" else
          "010" when addr = "0101" else
          "010" when addr = "0110" else
          "010" when addr = "0111" else
          "101" when addr = "1000" else
          "111" when addr = "1001" else
          "010" when addr = "1010" else
          "010" when addr = "1011" else
          "010" when addr = "1100" else
          "000" when addr = "1101" else
          "101" when addr = "1110" else
          "011";
end rtl;
```

Figure 5. Image of the lightrom2Myprog.vhd file.

3. Extensions: More Complicated Programmable Circuits

a. Creating Longer Instructions:

As an extension, I modified the system so that it can support longer instruction sequences, increasing the flexibility in light display patterns. In order to do this, I added two bits to the signal that acts as a counter for the ROM and the instructions (addr). I then added more instructions and extended the duration to 160ns. With this modification, the system can execute more complex sequences, allowing for greater customization in light display behavior.

The following is an image of the extensionROM.vhd file:

```
entity extensionrom is
    port (
        addr : in  std_logic_vector(5 downto 0);
        data : out std_logic_vector(2 downto 0)
    );
end entity;

architecture rtl of extensionrom is
begin
    data <=
        "000" when addr = "000000" else -- move 0s to LR  00000000
        "101" when addr = "000001" else -- bit invert LR  11111111
        "101" when addr = "000010" else -- bit invert LR  00000000
        "101" when addr = "000011" else -- bit invert LR  11111111
        "001" when addr = "000100" else -- shift LR right 01111111
        "001" when addr = "000101" else -- shift LR right 00111111
        "001" when addr = "000110" else -- rotate LR left 01111110
        "111" when addr = "000111" else -- rotate LR left 11111100
        "111" when addr = "001001" else -- rotate LR left 11111001
        "111" when addr = "001010" else -- rotate LR left 11110011
        "010" when addr = "001011" else -- shift LR left  11100110
        "111" when addr = "001100" else -- shift LR left  11001100
        "111" when addr = "001101" else -- add 1 to LR    11001101
        "101" when addr = "001110" else -- sub 1 from LR  11001100
        "000" when addr = "001111" else -- bit invert LR  00110011
        "101" when addr = "010000" else -- BRZ (Branch if Zero)
        "001" when addr = "010001" else -- move 0s to LR  00000000
        "001" when addr = "010010" else -- bit invert LR  11111111
        "011" when addr = "010011" else -- bit invert LR  00000000
        "011" when addr = "010100" else -- bit invert LR  11111111
        "011" when addr = "010101" else -- shift LR right 01111111
```

Figure 6. This is the first part of the extensionrom.vhd file.

```
        "110" when addr = "100000" else -- BRZ (Branch if Zero)
        "111" when addr = "100001" else -- rotate LR left 11110011
        "010" when addr = "100010" else -- shift LR left  11100110
        "010" when addr = "100011" else -- shift LR left  11001100
        "011" when addr = "100100" else -- add 1 to LR    11001101
        "100" when addr = "100101" else -- sub 1 from LR  11001100
        "101" when addr = "100110" else -- bit invert LR  00110011
        "110" when addr = "100111" else -- BRZ (Branch if Zero)
        "010" when addr = "101000" else -- shift LR left  11001100
        "000" when addr = "101001" else -- add 1 to LR    11001101
        "101" when addr = "101010" else -- sub 1 from LR  11001100
        "010" when addr = "101011" else -- bit invert LR  00110011
        "110" when addr = "101100" else -- BRZ (Branch if Zero)
        "011" when addr = "101101" else -- sub 1 from LR  11001100
        "010" when addr = "101110" else -- bit invert LR  00110011
        "110" when addr = "101111" else -- BRZ (Branch if Zero)
        "101" when addr = "110000" else -- shift LR left  11001100
        "011" when addr = "110001" else -- add 1 to LR    11001101
        "010" when addr = "110010" else -- sub 1 from LR  11001100
        "110" when addr = "110011" else -- bit invert LR  00110011
        "110" when addr = "110100" else -- BRZ (Branch if Zero)
        "100" when addr = "110101" else -- sub 1 from LR  11001100
        "101" when addr = "110110" else -- bit invert LR  00110011
        "110" when addr = "110111" else -- BRZ (Branch if Zero)
        "000" when addr = "111000" else -- shift LR left  11001100
        "101" when addr = "111001" else -- add 1 to LR    11001101
        "000" when addr = "111010" else -- sub 1 from LR  11001100
        "101" when addr = "111011" else -- bit invert LR  00110011
        "110" when addr = "111100" else -- BRZ (Branch if Zero)
        "011";                          -- add 1 to LR    00110100
end rtl;
```

Figure 7. This is the second part of the extensionrom.vhd file.

b. Creating Speed Buttons:

In order to get more control over the simulation, I also added a speed control feature with two buttons. The first button makes the simulation 2 times faster and the second button makes the simulation 4 times faster. In order to implement this system, I modified the light.vhd file to make it an extension.vhd file and added two more external inputs, a speed_button and another speed_button_4x. I then two added more internal signals, a clk_enable and a clk_counter.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;  -- defines the unsigned type

entity extension is

   port(
      clk       : in std_logic;
      reset     : in std_logic;
      speed_button: in  std_logic;  -- New speed control button
      speed_button_4x: in std_logic;
      lightsig  : out   std_logic_vector(7 downto 0);  -- output signal that will drive the LED display on the board
      IRview    : out   std_logic_vector(2 downto 0);  -- IR and PC will let you monitor the values of the IR and PC registers.
      PCView    : out unsigned (5 downto 0)
   );

end entity;

architecture rtl of extension is

   component extensionrom
      port(
         addr   : in std_logic_vector (5 downto 0);
         data   : out std_logic_vector(2 downto 0)
      );
   end component;


   type state_type is (sFetch, sExecute);
   signal IR  : std_logic_vector(2 downto 0);
   signal PC  : unsigned(5 downto 0);
```

Figure 8. Image of the first part of the extension.vhd file.

```
   signal IR  : std_logic_vector(2 downto 0);
   signal PC  : unsigned(5 downto 0);
   signal LR  : unsigned (7 downto 0);
   signal ROMvalue  : std_logic_vector(2 downto 0);
   signal state   : state_type;
   signal clk_enable : std_logic;
   signal clk_counter : integer := 0;

begin


   process(clk, reset)
   begin
       if reset = '0' then
           clk_enable <= '0';
           clk_counter <= 0;
       elsif rising_edge(clk) then
           if speed_button_4x = '1' then
               clk_enable <= '1';
               clk_counter <= 0;
           elsif speed_button = '1' then
               if clk_counter = 1 then
                   clk_enable <= '1';
                   clk_counter <= 0;
               else
                   clk_enable <= '0';
                   clk_counter <= clk_counter + 1;
               end if;
           else
               if clk_counter = 3 then
                   clk_enable <= '1';
                   clk_counter <= 0;
```

Figure 9. Image of the second part of the extension.vhd file.

```
                else
                    clk_enable <= '0';
                    clk_counter <= clk_counter + 1;
                end if;
            else
                if clk_counter = 3 then
                    clk_enable <= '1';
                    clk_counter <= 0;
                else
                    clk_enable <= '0';
                    clk_counter <= clk_counter + 1;
                end if;
            end if;
        end if;
    end process;

    process(clk, reset)
    begin
        if reset = '0' then
            state <= sFetch;
            IR <= "000";
            PC <= "000000";
            LR <= "00000000";
        elsif rising_edge(clk) and clk_enable = '1' then
            case state is
                when sFetch =>
                    IR <= ROMvalue;
                    PC <= PC + 1;
                    state <= sExecute;
                when sExecute =>
                    case IR is
                        when "000" =>
```

Figure 10. Image of the third part of the extension.vhd file.

```
                            LR <= "00000000";
                        when "001" =>
                            LR <= '0' & LR(7 downto 1);
                        when "010" =>
                            LR <= LR(6 downto 0) & '0';
                        when "011" =>
                            LR <= LR + 1;
                        when "100" =>
                            LR <= LR - 1;
                        when "101" =>
                            LR <= not LR;
                        when "110" =>
                            if LR = "00000000" then
                                PC <= unsigned("000" & ROMvalue);
                            end if;
                        when "111" =>
                            PC <= unsigned("000" & ROMvalue);
                        when others =>
                            LR <= (others => '0');
                    end case;
                    state <= sFetch;
            end case;
        end if;
    end process;
```

Figure 11. Image of the fourth part of the extension.vhd file.

```
    -- Output depends solely on the current state

    lightsig <= std_logic_vector(LR);
    IRView <= IR;
    PCView <= PC;

    extensionrom1 : extensionrom
        port map (
            addr => std_logic_vector(PC),
            data => ROMValue
        );


end rtl;
```

Figure 12. Image of the fifth part of the extension.vhd file.

c. Creating Branching

A branching instruction was also added to enable conditional execution of specific instructions. To do this, I added a BRZ instruction. When the instruction is “110” the system checks if LR is all zeros. If LR is all zeros, PC is updated to a new address that is specified by the ROMvalue. Since ROMvale is only 3 bits wide, it is extended to 6 bits by concatenating it with “000”. This makes sure that the PC can address any of the 64 possible locations in the ROM. If LR is not zero, the PC will simply increment to the next institution, allowing the program to continue execution without branching.

In addition to the conditional branching instruction (BRZ), I also implemented a BRA (Branch Always) instruction to enable unconditional branching in the system. The BRA instruction is triggered when the instruction code "111" is fetched from the ROM. Unlike BRZ, which only branches if the LR register is zero, BRA unconditionally updates the program counter (PC) to a new address specified by the ROMvalue. Since ROMvalue is only 3 bits wide, it is extended to 6 bits by concatenating it with "000" (e.g., "000" & ROMvalue). This ensures that the PC can correctly address any of the 64 possible locations in the ROM. The BRA instruction is useful for implementing jumps to specific sections of the program, such as loops or subroutines, without requiring any condition to be met.

The following is the image of the extensionrom.vhd that also includes the branching instructions.

```
entity extensionrom is
    port (
        addr : in  std_logic_vector(5 downto 0);
        data : out std_logic_vector(2 downto 0)
    );
end entity;

architecture rtl of extensionrom is
begin
    data <=
        "000" when addr = "000000" else -- move 0s to LR  00000000
        "101" when addr = "000001" else -- bit invert LR  11111111
        "101" when addr = "000010" else -- bit invert LR  00000000
        "101" when addr = "000011" else -- bit invert LR  11111111
        "001" when addr = "000100" else -- shift LR right 01111111
        "001" when addr = "000101" else -- shift LR right 00111111
        "001" when addr = "000110" else -- rotate LR left 01111110
        "111" when addr = "000111" else -- rotate LR left 11111100
        "111" when addr = "001001" else -- rotate LR left 11111001
        "111" when addr = "001010" else -- rotate LR left 11110011
        "010" when addr = "001011" else -- shift LR left  11100110
        "111" when addr = "001100" else -- shift LR left  11001100
        "111" when addr = "001101" else -- add 1 to LR    11001101
        "101" when addr = "001110" else -- sub 1 from LR  11001100
        "000" when addr = "001111" else -- bit invert LR  00110011
        "101" when addr = "010000" else -- BRZ (Branch if Zero)
        "001" when addr = "010001" else -- move 0s to LR  00000000
        "001" when addr = "010010" else -- bit invert LR  11111111
        "011" when addr = "010011" else -- bit invert LR  00000000
        "011" when addr = "010100" else -- bit invert LR  11111111
        "011" when addr = "010101" else -- shift LR right 01111111
```

Figure 13. Image of extensionrom.vhd file. Note that the comments do not accurately represent the instructions as I had to edit after writing the comments.

```
        "110" when addr = "100000" else -- BRZ (Branch if Zero)
        "111" when addr = "100001" else -- rotate LR left 11110011
        "010" when addr = "100010" else -- shift LR left  11100110
        "010" when addr = "100011" else -- shift LR left  11001100
        "011" when addr = "100100" else -- add 1 to LR    11001101
        "100" when addr = "100101" else -- sub 1 from LR  11001100
        "101" when addr = "100110" else -- bit invert LR  00110011
        "110" when addr = "100111" else -- BRZ (Branch if Zero)
        "010" when addr = "101000" else -- shift LR left  11001100
        "000" when addr = "101001" else -- add 1 to LR    11001101
        "101" when addr = "101010" else -- sub 1 from LR  11001100
        "010" when addr = "101011" else -- bit invert LR  00110011
        "110" when addr = "101100" else -- BRZ (Branch if Zero)
        "011" when addr = "101101" else -- sub 1 from LR  11001100
        "010" when addr = "101110" else -- bit invert LR  00110011
        "110" when addr = "101111" else -- BRZ (Branch if Zero)
        "101" when addr = "110000" else -- shift LR left  11001100
        "011" when addr = "110001" else -- add 1 to LR    11001101
        "010" when addr = "110010" else -- sub 1 from LR  11001100
        "110" when addr = "110011" else -- bit invert LR  00110011
        "110" when addr = "110100" else -- BRZ (Branch if Zero)
        "100" when addr = "110101" else -- sub 1 from LR  11001100
        "101" when addr = "110110" else -- bit invert LR  00110011
        "110" when addr = "110111" else -- BRZ (Branch if Zero)
        "000" when addr = "111000" else -- shift LR left  11001100
        "101" when addr = "111001" else -- add 1 to LR    11001101
        "000" when addr = "111010" else -- sub 1 from LR  11001100
        "101" when addr = "111011" else -- bit invert LR  00110011
        "110" when addr = "111100" else -- BRZ (Branch if Zero)
        "011";                          -- add 1 to LR    00110100
end rtl;
```

Figure 14. Image of the second part of extensionrom.vhd file. Note that the comments do not accurately represent the instructions as I had to edit after writing the comments.

d. Simulation and Testing

To validate the functionality of my extension, I modified the lightbench.vhd testbench to account for the new features, including the BRZ (Branch if Zero) and BRA (Branch Always) instructions. The testbench was designed to exercise all the instructions in the ROM, including conditional and unconditional branching, to ensure correct behavior under various scenarios. To visualize the results, I used the vcd_movie.py script to create an animation.

This is the link to the video that validates my extension.

In this video, it starts without pressing any of the buttons. It then presses the second button which makes the counter go by 4 times faster and then it switches between pressing the other button and not pressing any button.

This is the image of the extensionbench.vhd that was used to create the simulation:

```
  constant num_cycles : integer := 160;
  signal clk : std_logic := '1';
  signal reset : std_logic;
  signal speed_button : std_logic := '0';
  signal speed_button_4x : std_logic := '0';
  component extension
    port( clk, reset : in std_logic;
          speed_button: in std_logic;
          speed_button_4x : in std_logic;
          lightsig   : out std_logic_vector(7 downto 0) );
  end component;
  signal lightsig : std_logic_vector(7 downto 0);
begin
  reset <= '0', '1' after 5 ns;
  process begin
    for i in 1 to num_cycles loop
      clk <= not clk;
      wait for 5 ns;
      clk <= not clk;
      wait for 5 ns;
    end loop;
    wait;
  end process;
  process begin
        wait for 25 ns;
        speed_button_4x <= '1';
        wait for 200 ns;
        speed_button_4x <= '0';
        wait for 100 ns;
        speed_button <= '1';
        wait for 200 ns;
        speed_button <= '0';
```

Figure 15. Image of the extensionbench.vhd file that was used to create the simulation.

Acknowledgements:

https://www.fpga4student.com/2017/06/vhdl-code-for-counters-with-testbench.html