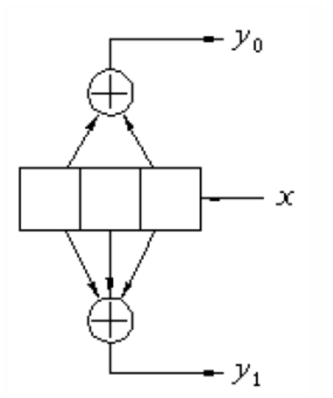
FPGA laboratory report

Diego Figueroa – 2485776

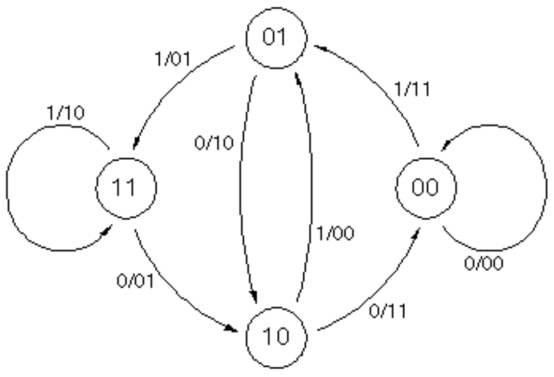
July 10, 2025

1 Convolutional encoder

The goal of this practice is to implement a convolutional encoder, of a specific code of rate $R = 1/2$, with memory $m = 2$, and polynomials generators $g_0(x) = x^2 + 1$ and $g_1(x) = x^2 + x + 1$. This specific convolutional encoder can be represented by a shift register and some xor operations, as seen in figure 1a, or as a finite state machine as seen in figure 1b.



(a) Representation as shift register



(b) Representation as a finite state machine

Figure 1: Caption place holder

1.1 Symbol schematic implementation

To implement this encoder in the FPGA, the first Method used was to use the schematic interface of Quartus II. For this propose the representation of a shift register of the convolutional encoder is well suited, so it is implemented as seen in figure **??**. There the shift register is done with 3 D-flipflops conected in series, with a common clock and reset, and no preset is used. Finally the two output bits are generated with xor gates using the signals stored in the registers.

Finally the circuit is simulated, and the results can be seen in section 1.4.

1.2 VHDL implementation

For VHDL the representation of the code as a FSM is used. The code describing this machine can be seen listing 1 in the Annex, this code is based in an example of a state machine in [1].

In the architecture declaration, the first thing done is defining a new enum type to store the states of the state machine, then some signals are defined. Two processes are used, one for the state logic i.e. describing the outputs and what transitions are done in each state; and the other to update the state and read the incoming data on every rising edge of the clock.

Finally the circuit is simulated, and the results can be seen in section 1.4.

1.3 AHDL implementation

For the AHDL implementation a FSM is also used. The code in this case can be seen in listing 2, and is inspired by the example found in [2].

The first three lines correspond to the declaration of the component, then two variables are declared, one of type `MACHINE`, where the states are declared, and one of type `node`. In the description of the behavior, the clock and the reset of the state machine ara assigned, `d_in` is implemented as a D-flipflop to store the incoming data over a clock period, and the behavior of the FSM is described as a table.

It is important to notice that the state machine is implemented with asynchronous outputs, that means that while being in one state, the outputs can change if the inputs change, even before the clock edge. That is why it is important the flip flop for `d_in`, so it has a stable value over a clock period.

Finally the circuit is simulated, and the results can be seen in section 1.4.

1.4 Simulation results

In figure 2 can be seen the simulations result of the encoders, the first two signals (from top to bottom) are the clock and a reference clock with a phase difference to change the data. The third signal is the input binary data, the forth signal is the reset signal.

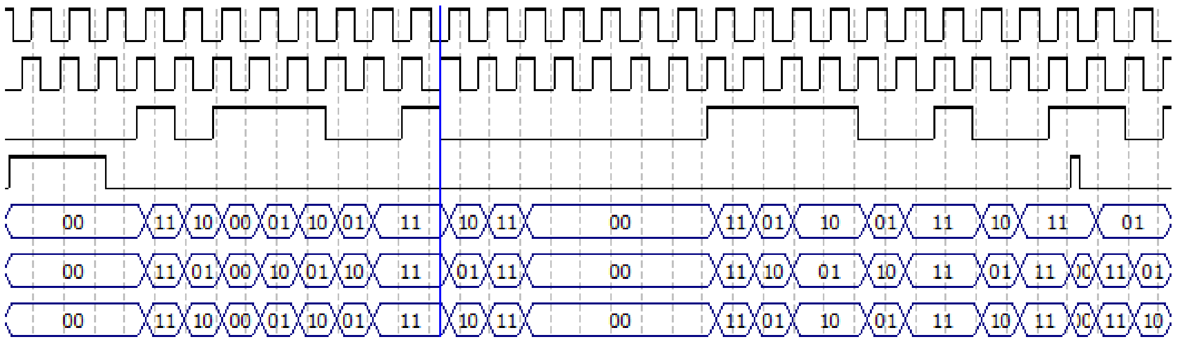


Figure 2: Simulation results of the three convolutional codes encoders.

Finally we see the signals of the tree encoders: AHDL, VHDL and Schematic implementation. The first part of the data is de binary representation of 0xB9, and then there is some random data. It is evident that the three encoders have the same output which is also the expected one given the desired code. The only difference is at the end of the simulation when the reset signal goes to one for a small time. Here the two last encoders behave the same, but the first is different, this is because the AHDL implementation has a synchronous reset while the other two have an asynchronous reset.

2 Running light

The goal of this practice is to familiarize with the FPGA board, its built elements, such as leds push buttons, switches and seven segments displays, and program the FPGA to test the results of a circuit in real life

2.1 Functional description

The circuit has a series of functional requirements:

- Using the leds of the board, make a running light that goes from left to right and back, by turning off the current led and turning on the next one in a line, so it seems like a moving light (it is important to take into account the speed of the moving light, so it is possible to perceive the effect with the eyes).
- Use the seven segments to count how many times the light has gone from one side to the other.
- Add the possibility to change the speed of the running light with the use of some switches.
- Add de ability of giving a desired pattern that moves through the leds, by using the switches.
- Change the operation mode of the circuit between **Start**, **Stop**, **Reset, Load pattern**, by using the push buttons.
- Change the direction of the lights between **left**, **right** or **for- and backward**

2.2 Digital Design and VHDL implementation

To accomplish the functional specifications of the circuit, the design is broken into smaller pieces, which are then interconnected in a top level entity. Each of the sub-systems and the top level entity are describen next.

2.2.1 lr_ring_reg

To accomplish the moving pattern in both directions the entity `lr_ring_reg` is used. It is a shift register that shift all the values to the left (in a circular manner) on every rising edge of the clock. It also has a `pattern_in` signal that is hard set in the registers if the signal `load` is set to high. Finally the output is a pattern moving left, and other moving right, which is implemented simply by reversing the order of the ring register.

The implementation in VHDL can be seen in listing 3. In line 7 a generic is defined to specify the number of cells in the ring register. In the architecture some intermediate signals are defined. Then in line 16 - 19 a concurrent generate statement is used to define `pattern_out_l` as simply `pattern_out`, and `pattern_out_r` as the reversed version of pattern_out. A process is used to do the shift operation, and an other process is used to load a pattern and update pattern_out.

2.2.2 clk_div_n

A clock divider is needed for different proposes: first to make the effect of the moving light visible, but also to change the speed of the light. Also a clock that is n_{led} (number of leds) slower than the clock used for the ring register is useful to signal when a light has done a full run from one side to the other (which is needed to count the runs of the light).

For this reason an entity for a clock divider is defined, which can change the division number during operation. The code for this module is in listing 4. In the entity declaration a generic is used to define the counter width. The only process of the architecture resets the counter cnt and the pulse to 0 if rst is 1. Otherwise on a clk_in rising edge if n changes, set cnt to 1 and pulse_reg to 0, if cnt = n - 1, cnt is set again to 0 and pulse_reg to 1, otherwise cnt is increased by one and pulse_reg set to 0. The result is a small positive pulse every n pulses of clk_in.

2.2.3 decimal_cnt

This module is used to count the number of runs of the light. It is based on the module digint_cnt, and is just a series connection of n decimal_cnt's, as can be seen in listing 5. And digint_cnt just counts on every inc rising edge, and if gets to 9 it goes back to 0 and generate a pulse on inc_out (see code on listing 6).

2.2.4 seven_segments

This module is really simple, it just outputs the needed signals for a seven segments display depending on the given number it receives, it is just implemented as a look up table in a concurrent statement as seen in listing 7.

2.2.5 speed_rom

To change the speed of the light a ROM is implemented, where different values of n are stored and used as inputs of clk_div_n, this file is automatically generated using a python code (see listing 8) depending in some parameters, an example the VHDL rom can be seen in listing 9.

2.2.6 const_types_pkg

A package to define some types and constants is also used to improve the readability of the code in running_light (see code in listing 10).

2.2.7 running_light

Finally the top level entity is described in listing 11. A structural schematic of the model can be seen in figure **??**. And the different modules are controlled using the FSM seen in figure **??**. In general the fast clock of the FPGA is divided by speed, this clock goes to the lr_ring_reg, and is also divided to count the number of runs whit the module decimal_cnt.

The signal going to the leds is taken from a multiplexer depending on the current value `dir`, and with the FSM the circuits switches between the modes **Start**, **Stop**, **Reset, Load pattern**.

3 FIR filters

3.1 Low pass

3.2 High pass

3.3 Band pass

3.4 Band stop

3.5 Project with the 4 filters

References

[1] D. L. Perry, *VHDL /*, 3. ed. New York, NY [u.a.] : McGraw-Hill, 1998, Previous ed.: 1994. [Online]. Available: `http://www.loc.gov/catdir/enhancements/fy0602/98016663-t.html`.

[2] *Designing state machines (ahdl)*. [Online]. Available: `http://www.xilinx.info/_altera/html/_sw/q2help/source/ahdl/ahdl_pro_design_state_machine.htm`.

Annex

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

entity convolutional_code_VHDL is
  port(data_in, clk, rst: in std_logic;
      data_out0, data_out1: out std_logic);
end convolutional_code_VHDL;

architecture state_machine of convolutional_code_VHDL is
  type state is (s00, s01, s10, s11);
  signal curr_st, next_st: state;
  signal d_tmp, d_out0, d_out1: std_logic;
  begin
    process(curr_st, d_tmp) -- state logic
    begin
      case curr_st is
      when s00 =>
        if d_tmp = '0' then
          d_out0 <= '0';
          d_out1 <= '0';
          next_st <= s00;
        else
```

```
          d_out0 <= '1';
          d_out1 <= '1';
          next_st <= s01;
        end if;
      when s01 =>
        if d_tmp = '0' then
          d_out0 <= '0';
          d_out1 <= '1';
          next_st <= s10;
        else
          d_out0 <= '1';
          d_out1 <= '0';
          next_st <= s11;
        end if;
      when s10 =>
        if d_tmp = '0' then
          d_out0 <= '1';
          d_out1 <= '1';
          next_st <= s00;
        else
          d_out0 <= '0';
          d_out1 <= '0';
          next_st <= s01;
        end if;
      when s11 =>
        if d_tmp = '0' then
          d_out0 <= '1';
          d_out1 <= '0';
          next_st <= s10;
        else
          d_out0 <= '0';
          d_out1 <= '1';
          next_st <= s11;
        end if;
      end case;
    end process;

    process(clk, rst) -- go to next state
    begin
      if rst = '1' then
        curr_st <= s00;
        d_tmp <= '0';
      elsif (clk = '1' and clk'event) then
        curr_st <= next_st;
        d_tmp <= data_in;
      end if;
    end process;

    -- concurrent assigment of the output
    data_out0 <= d_out0;
    data_out1 <= d_out1;
```

```
end state_machine;
```

Listing 1: Code in VHDL for the FSM of the convolutional code encoder.

```
subdesign convolutional_code_AHDL(
  data_in, clk, rst: input;
  data_out0, data_out1: output
)
variable
  conv_code_state: MACHINE
  WITH STATES (
    s00 = b"00",
    s01 = b"01",
    s10 = b"10",
    s11 = b"11");
  d_in: node;

begin
  conv_code_state.clk = clk;
  conv_code_state.reset = rst;

  d_in = DFF(data_in, clk, VCC, VCC);

  TABLE
  conv_code_state, d_in => data_out1, data_out0,
     conv_code_state;
  s00, 0 => 0, 0, s00;
  s00, 1 => 1, 1, s01;
  s01, 0 => 1, 0, s10;
  s01, 1 => 0, 1, s11;
  s10, 0 => 1, 1, s00;
  s10, 1 => 0, 0, s01;
  s11, 0 => 0, 1, s10;
  s11, 1 => 1, 0, s11;
  END TABLE;
end;
```

Listing 2: Code in VHDL for the FSM of the convolutional code encoder.

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

use work.const_types_pkg.all;

entity lr_ring_reg is
    generic(num_cells: integer := 8);
    port(clk, load: in std_logic;
         pattern_in: in std_logic_vector(num_cells - 1 downto
            0);
         pattern_out_l, pattern_out_r: inout std_logic_vector(
            num_cells - 1 downto 0));
end lr_ring_reg;

```

```vhdl
architecture behave of lr_ring_reg is
    signal pattern_out, pattern_shift: std_logic_vector(
       num_cells - 1 downto 0);
begin
    gen_x: for i in pattern_out'range generate
        pattern_out_l(i) <= pattern_out(i);
        pattern_out_r(i) <= pattern_out(pattern_out'left +
           pattern_out'right - i);
    end generate;
    process (pattern_out)
    begin
        pattern_shift(num_cells - 2 downto 0) <= pattern_out(
           num_cells - 1 downto 1);
        pattern_shift(num_cells - 1) <= pattern_out(0);
    end process;

    process (clk, load, pattern_in)
    begin
        if load = '1' then
            pattern_out <= pattern_in;
        elsif clk'event and clk = '1' then
            pattern_out <= pattern_shift;
        end if;
    end process;
end behave;
```

Listing 3: Code in VHDL for `lr_ring_reg`.

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

entity clk_div_n is
    generic (
        CNT_WIDTH : integer := 32
    );
    port (
        clk_in, rst: in  std_logic;
        n: in  unsigned(CNT_WIDTH-1 downto 0);
        clk_out: out std_logic
    );
end clk_div_n;

architecture rtl of clk_div_n is
    signal cnt: unsigned(CNT_WIDTH-1 downto 0);
    signal pulse_reg: std_logic;
    signal n_prev: unsigned(CNT_WIDTH-1 downto 0);
begin

    process(clk_in, rst, n)
    begin
        if rst = '1' then
```

```
                cnt <= (others => '0');
                pulse_reg <= '0';
                --n_prev <= n;
            elsif clk_in'event and clk_in = '1' then
                if n /= n_prev then
                    cnt <= to_unsigned(1, CNT_WIDTH);
                    pulse_reg <= '0';
                    n_prev <= n;
                elsif cnt = (n - 1) then
                    cnt <= (others => '0');
                    pulse_reg <= '1';
                else
                    cnt <= cnt + 1;
                    pulse_reg <= '0';
                end if;
            end if;
        end process;

        clk_out <= pulse_reg;

end rtl;
```

Listing 4: Code in VHDL for `clk_div_n`.

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

use work.const_types_pkg.all;

entity decimal_cnt is
    port(inc, rst: in std_logic;
        cnt_out: inout digit_array);
end decimal_cnt;

architecture behave of decimal_cnt is
    component digit_cnt
        port (inc, rst: in std_logic;
            inc_out: out std_logic;
            cnt_out: inout digit);
    end component;
    signal inc_vec: std_logic_vector(cnt_out'high + 1 downto 0)
        ;
begin
    inc_vec(0) <= inc;
    gen_uutx: for i in 0 to cnt_out'high generate
    begin
        uutx: digit_cnt port map (inc_vec(i), rst, inc_vec(i+1)
            , cnt_out(i));
    end generate gen_uutx;
end behave;
```

Listing 5: Code in VHDL for `decimal_cnt`.

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

use work.const_types_pkg.all;

entity digit_cnt is
    port (inc, rst: in std_logic;
          inc_out: out std_logic;
          cnt_out: inout digit);
end digit_cnt;

architecture behave of digit_cnt is
    signal cnt_tmp: digit;
    signal inc_out_tmp: std_logic;
begin
    process (cnt_out)
    begin
        if cnt_out = 9 then
            cnt_tmp <= 0;
            inc_out_tmp <= '1';
        else
            cnt_tmp <= cnt_out + 1;
            inc_out_tmp <= '0';
        end if;
    end process;

    process (inc, rst)
    begin
        if rst = '1' then
            cnt_out <= 0;
            inc_out <= '0';
        elsif inc'event and inc = '1' then
            cnt_out <= cnt_tmp;
            inc_out <= inc_out_tmp;
        end if;
    end process;
end behave;
```

Listing 6: Code in VHDL for `digit_cnt`.

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

use work.const_types_pkg.all;

entity seven_segments is
  port (digit_in: in digit;
          seven_seg_leds: out sev_seg_disp);
end seven_segments;

architecture behave of seven_segments is
begin
```

```
    seven_seg_leds <= ss_disp_0 when digit_in = 0 else
                      ss_disp_1 when digit_in = 1 else
                      ss_disp_2 when digit_in = 2 else
                      ss_disp_3 when digit_in = 3 else
                      ss_disp_4 when digit_in = 4 else
                      ss_disp_5 when digit_in = 5 else
                      ss_disp_6 when digit_in = 6 else
                      ss_disp_7 when digit_in = 7 else
                      ss_disp_8 when digit_in = 8 else
                      ss_disp_9 when digit_in = 9;
end behave;
```

Listing 7: Code in VHDL for seven_segments.

```
import numpy as np

run_min_freq = 0.1
run_max_freq = 5
num_adr_bits = 5

num_steps = 2 ** num_adr_bits
fpga_clk_freq = 50e6
num_leds = 18

min_led_clk_freq = run_min_freq * num_leds
max_led_clk_freq = run_max_freq * num_leds
# led_clk_freq_range = np.linspace(min_led_clk_freq,
    max_led_clk_freq, num_steps)
led_clk_freq_range = np.geomspace(min_led_clk_freq,
    max_led_clk_freq, num_steps)

cnt_div_clk = (fpga_clk_freq // led_clk_freq_range).astype(np.
    int64)
num_cnt_bits = len(bin(cnt_div_clk.max()))
bin_format = f"#0{num_cnt_bits}b"

rom_def = f"\tconstant clk_leds_cnt_len: integer := {
    num_cnt_bits-2};\n"
rom_def += f"\tconstant adr_len: integer := {num_adr_bits};\n"
rom_def += f"\tconstant num_speeds: integer := {num_steps};\n"
rom_def += f"\ttype rom_speed is array (0 to num_speeds - 1) of
     unsigned(clk_leds_cnt_len - 1 downto 0);\n"
rom_def += "\tconstant speeds: rom_speed := (\n"
for i, cnt in enumerate(cnt_div_clk):
    rom_def += f'\t\t{i} => "{format(cnt, bin_format)[2:]}",\n'
rom_def = rom_def[:-2] + ");\n"


with open('running_light/speed_rom.vhd', 'w') as f:
    f.write("library ieee;\n")
    f.write("use ieee.std_logic_1164.all;\n")
    f.write("use ieee.numeric_std.all;\n\n")
    f.write("package speed_rom is\n")
    f.write(rom_def)
    f.write("end package;\n")
```

Listing 8: Code in python for creating the speed room.

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

package speed_rom is
  constant clk_leds_cnt_len: integer := 25;
  constant adr_len: integer := 5;
  constant num_speeds: integer := 32;
  type rom_speed is array (0 to num_speeds - 1) of unsigned(
     clk_leds_cnt_len - 1 downto 0);
  constant speeds: rom_speed := (
    0 => "1101001111101101011110001",
    1 => "1011101011001101010111111",
    2 => "1010010010100111110101111",
    3 => "1001000100100010011110001",
    4 => "0111111111101101100100011",
    5 => "0111000011000010111001011",
    6 => "0110001101100100100010101",
    7 => "0101011110011011111011011",
    8 => "0100110100111000111100111",
    9 => "0100010000010001001101110",
    10 => "0011101111111111010110000",
    11 => "0011010011100010011001010",
    12 => "0010111010011101010101010",
    13 => "0010100100010110100011110",
    14 => "0010010000110111100001000",
    15 => "0001111111101100010100011",
    16 => "0001110000100011011100000",
    17 => "0001100011001101011011000",
    18 => "0001010111011100101010001",
    19 => "0001001101000101001000111",
    20 => "0001000011111100010010001",
    21 => "0000111011111000110001000",
    22 => "0000110100110010010111100",
    23 => "0000101110100001110101110",
    24 => "0000101001000000110011000",
    25 => "0000100100001001100111001",
    26 => "0000011111110111010100010",
    27 => "0000011100000101100010110",
    28 => "0000011000110000011011101",
    29 => "0000010101110100100101101",
    30 => "0000010011001111000000111",
    31 => "0000010000111101000100011");
end package;
```

Listing 9: Code in VHDL for `speed_rom`.

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

package const_types_pkg is

    constant num_sev_seg: integer := 4;
    constant num_dip_sws: integer := 18;
    constant num_lights: integer := 18;
    constant clk_cnt_len: integer := 5;    -- at least ceil(log2(num_lights))

    subtype digit is integer range 0 to 9;
    type digit_array is array (natural range <>) of digit;
    subtype sev_seg_disp is std_logic_vector (0 to 6);
    type sev_seg_disp_array is array (num_sev_seg - 1 downto 0) of sev_seg_disp;
    constant ss_disp_0: sev_seg_disp := "0000001";
    constant ss_disp_1: sev_seg_disp := "1001111";
    constant ss_disp_2: sev_seg_disp := "0010010";
    constant ss_disp_3: sev_seg_disp := "0000110";
    constant ss_disp_4: sev_seg_disp := "1001100";
    constant ss_disp_5: sev_seg_disp := "0100100";
    constant ss_disp_6: sev_seg_disp := "0100000";
    constant ss_disp_7: sev_seg_disp := "0001111";
    constant ss_disp_8: sev_seg_disp := "0000000";
    constant ss_disp_9: sev_seg_disp := "0000100";

    constant num_lights_bit_vec: unsigned(clk_cnt_len - 1 downto 0) := to_unsigned(num_lights, clk_cnt_len);
    type direction is (left, right);
end package;
```

Listing 10: Code in VHDL for `const_types_pkg`.

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

use work.const_types_pkg.all;
use work.speed_rom.all;

entity running_light is
    port(clk, start_fpga, stop_sys_fpga, rst_fpga, load_pattern_fpga: in std_logic;
        dip_sws: in std_logic_vector(num_dip_sws - 1 downto 0);
        seven_segs: out sev_seg_disp_array;
        leds: out std_logic_vector(num_lights - 1 downto 0);
        leds_status: out std_logic_vector(adr_len + 2 downto 0));
end running_light;

```

```vhdl
architecture behave of running_light is
    component lr_ring_reg
        generic (num_cells: integer);
        port (clk, load: in std_logic;
            pattern_in: in std_logic_vector(num_lights - 1
               downto 0);
            pattern_out_l, pattern_out_r: inout
               std_logic_vector(num_lights - 1 downto 0));
    end component;
    component clk_div_n
        generic (CNT_WIDTH: integer);
        port (clk_in, rst: in std_logic;
            n: in unsigned;
            clk_out: out std_logic);
    end component;
    component seven_segments
        port (digit_in: in digit;
            seven_seg_leds: out sev_seg_disp);
    end component;
    component decimal_cnt
        port(inc, rst: in std_logic;
            cnt_out: inout digit_array);
    end component;

    signal rst, start, stop_sys, load_pattern: std_logic;
    signal rst_dec_cnt, rst_clk_leds, rst_clk_cnt: std_logic;
    signal inc_cnt, leds_clk, load: std_logic;
    signal cnt_runs: digit_array (num_sev_seg - 1 downto 0);
    signal speed: unsigned(clk_leds_cnt_len - 1 downto 0);
    signal pattern_in, pattern_out_l, pattern_out_r:
       std_logic_vector(num_lights - 1 downto 0);
    type state is (ss_reset, ss_stop_sys, ss_run_light,
       ss_load_pattern);
    signal curr_state, next_state: state;
    signal dir: direction := left;
    signal dir_mode: std_logic_vector(2 downto 0);

begin
    dec_cnt: decimal_cnt
        port map(inc_cnt, rst_dec_cnt, cnt_runs);
    gen_ss : for i in 0 to num_sev_seg - 1 generate
        ssx: seven_segments port map(cnt_runs(i), seven_segs(i)
           );
    end generate;
    clk_leds: clk_div_n
        generic map(CNT_WIDTH => clk_leds_cnt_len)
        port map(clk, rst_clk_leds, speed, leds_clk);
    clk_cnt: clk_div_n
        generic map(CNT_WIDTH => clk_cnt_len)
        port map(leds_clk, rst_clk_cnt, num_lights_bit_vec,
           inc_cnt);
    lr_rr: lr_ring_reg
        generic map(num_cells => num_lights)
        port map(leds_clk, load, pattern_in, pattern_out_l, pattern_out_r);

    speed <= speeds(to_integer(unsigned(dip_sws(adr_len - 1 downto 0))));
    dir_mode <= dip_sws(adr_len + 2 downto adr_len);
    leds_status(adr_len - 1 downto 0) <= dip_sws(adr_len - 1 downto 0);
    leds <= pattern_out_l when dir = left else
            pattern_out_r when dir = right;
    rst <= not rst_fpga;
    start <= not start_fpga;
    stop_sys <= not stop_sys_fpga;
    load_pattern <= not load_pattern_fpga;

    process(inc_cnt, dir_mode, dir)
    begin
        if dir_mode = "000" then
            leds_status(adr_len + 2 downto adr_len) <= "001";
            dir <= left;
        elsif dir_mode(0) = '1' then
            dir <= left;
            leds_status(adr_len + 2 downto adr_len) <= "001";
        elsif dir_mode(1) = '1' then
            dir <= right;
            leds_status(adr_len + 2 downto adr_len) <= "010";
        else
            leds_status(adr_len + 2 downto adr_len) <= "100";
            if inc_cnt'event and inc_cnt = '1' then
                if dir = left then
                    dir <= right;
                else
                    dir <= left;
                end if;
            end if;
        end if;
    end process;

    process(curr_state, start, load_pattern, stop_sys, dip_sws)
    begin
        case curr_state is
            when ss_reset =>
                pattern_in <= (num_lights - 1 => '1', others => '0');
                load <= '1';
                rst_dec_cnt <= '1';
                rst_clk_leds <= '1';
                rst_clk_cnt <= '1';
                if start = '1' then
```

```vhdl
                    next_state <= ss_run_light;
                elsif load_pattern = '1' then
                    next_state <= ss_load_pattern;
                else
                    next_state <= ss_reset;
                end if;
            when ss_stop_sys =>
                pattern_in <= (others => '0');
                load <= '0';
                rst_dec_cnt <= '0';
                rst_clk_leds <= '1';
                rst_clk_cnt <= '1';
                if start = '1' then
                    next_state <= ss_run_light;
                elsif load_pattern = '1' then
                    next_state <= ss_load_pattern;
                else
                    next_state <= ss_stop_sys;
                end if;
            when ss_run_light =>
                pattern_in <= (others => '0');
                load <= '0';
                rst_dec_cnt <= '0';
                rst_clk_leds <= '0';
                rst_clk_cnt <= '0';
                if stop_sys = '1' then
                    next_state <= ss_stop_sys;
                elsif load_pattern = '1' then
                    next_state <= ss_load_pattern;
                else
                    next_state <= ss_run_light;
                end if;
            when ss_load_pattern =>
                pattern_in <= dip_sws;
                rst_dec_cnt <= '1';
                rst_clk_leds <= '1';
                rst_clk_cnt <= '1';
                if start = '1' then
                    load <= '0';
                    next_state <= ss_run_light;
                else
                    load <= '1';
                    next_state <= ss_load_pattern;
                end if;
        end case;
    end process;

    process (clk, rst)
    begin
        if rst = '1' then
            curr_state <= ss_reset;
        elsif clk'event and clk = '1' then
            curr_state <= next_state;
        end if;
    end process;
end behave;
```

Listing 11: Code in VHDL for `running_light`.