

FPGA laboratory report

Diego Figueroa – 2485776

July 15, 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.

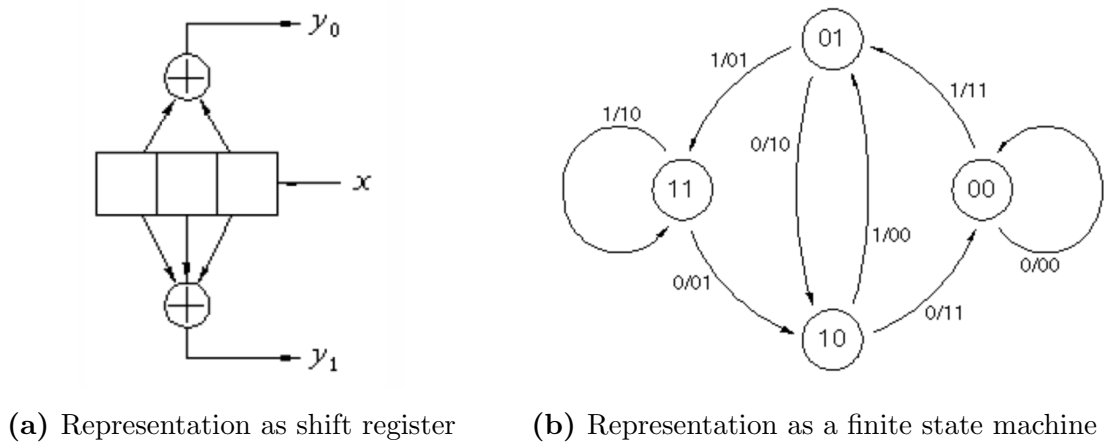


Figure 1: Caption place holder

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 are 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.

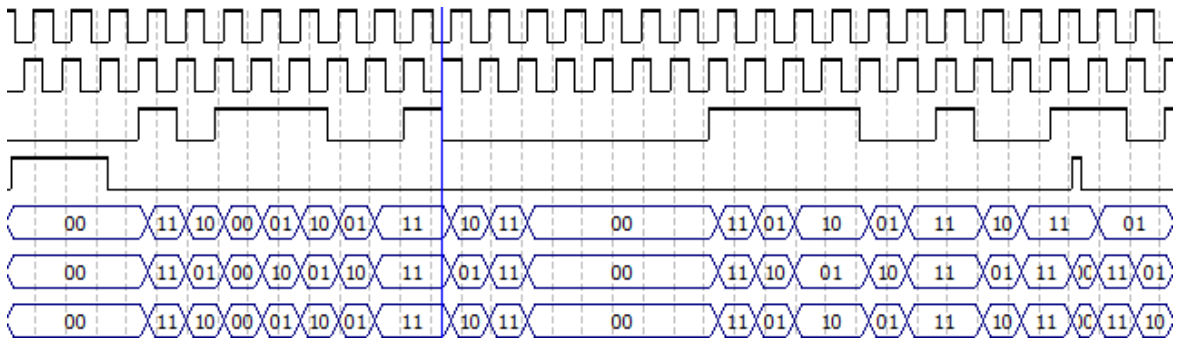


Figure 2: Simulation results of the three convolutional codes encoders.

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 the 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 described 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 shifts 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`.

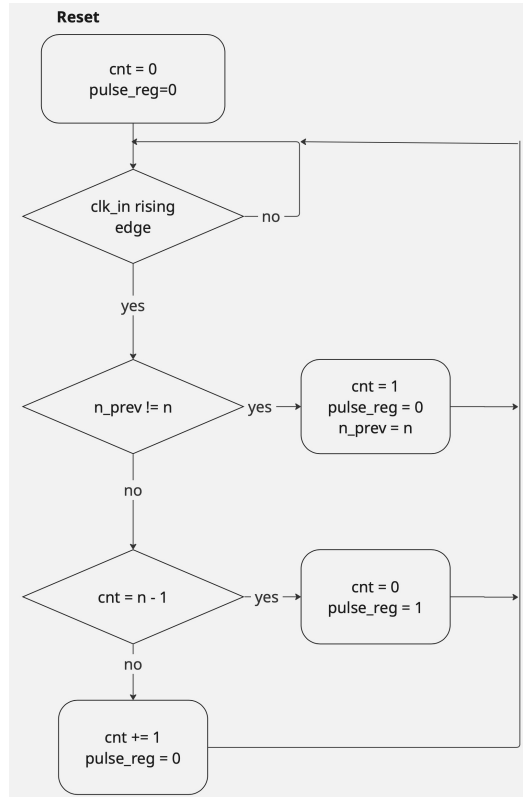
2.2.2 `clk_div_n`

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`.



(a) `clk_dic_n` symbol block

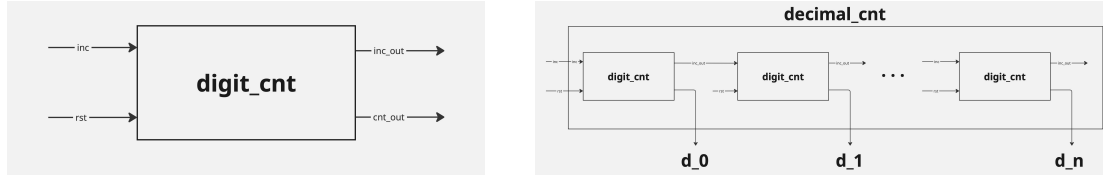


(b) `clk_dic_n` flow diagram

Figure 3: `clk_dic_n` desing

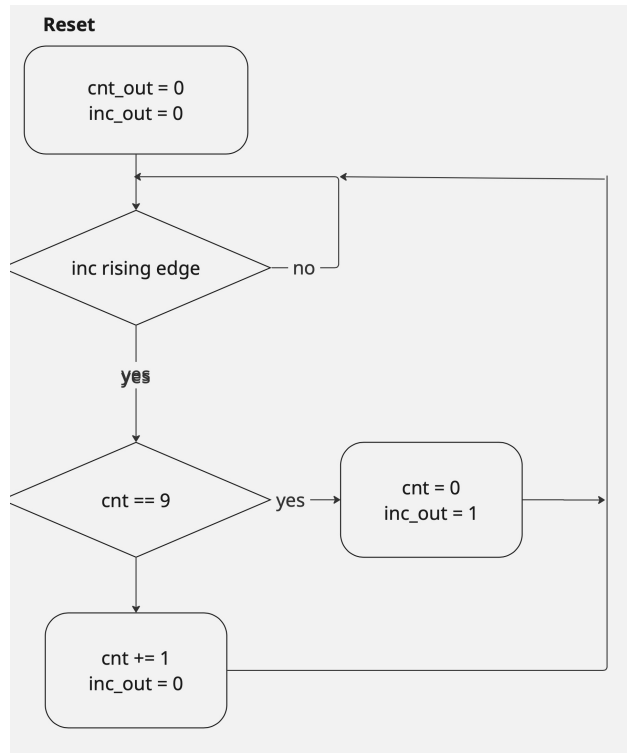
2.2.3 decimal_cnt

This module is used to count the number of runs of the light. It is based on the module `digit_cnt`, and is just a series connection of n `decimal_cnt`'s, as can be seen in listing 5. And `digit_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).



(a) `digit_cnt` symbol block

(b) `decimal_cnt` schematic



(c) `digit_cnt` flow diagram

Figure 4: `decimal_cnt` design.

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 5. And the different modules are controlled using the FSM seen in figure 6. 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**.

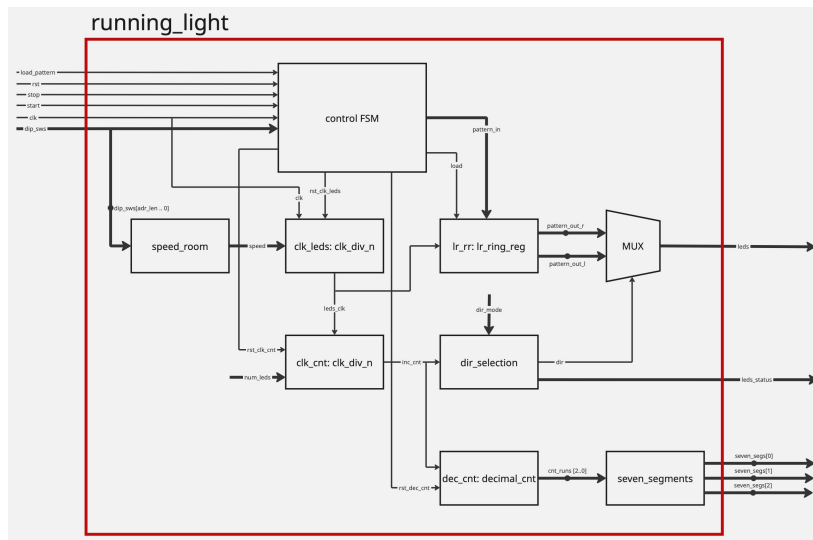


Figure 5: running_light schematic

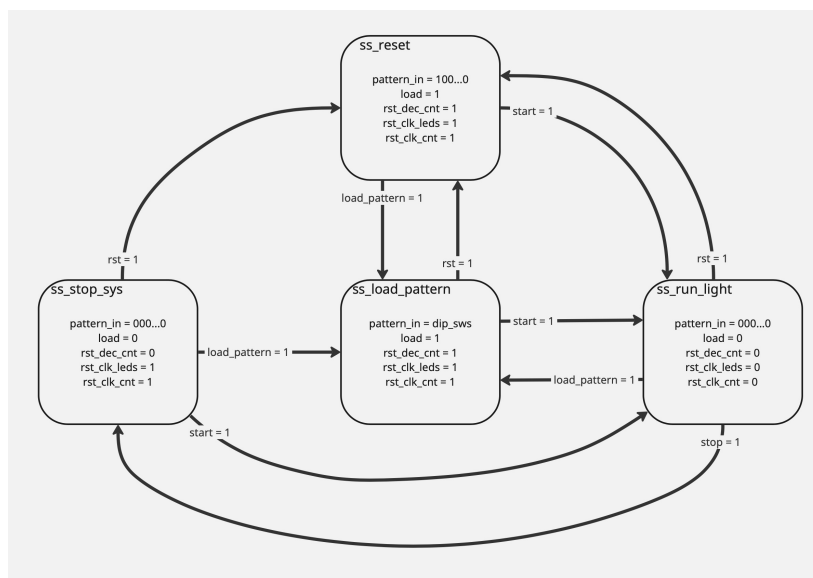


Figure 6: running_light control finite state machine.

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

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3
4 entity convolutional_code_VHDL is
5     port(data_in, clk, rst: in std_logic;
6          data_out0, data_out1: out std_logic);
7 end convolutional_code_VHDL;
8
9 architecture state_machine of convolutional_code_VHDL is
10     type state is (s00, s01, s10, s11);
11     signal curr_st, next_st: state;
12     signal d_tmp, d_out0, d_out1: std_logic;
13 begin
14     process(curr_st, d_tmp) -- state logic
15     begin
16         case curr_st is
17         when s00 =>
18             if d_tmp = '0' then
19                 d_out0 <= '0';
20                 d_out1 <= '0';
21                 next_st <= s00;
22             else
23                 d_out0 <= '1';
24                 d_out1 <= '1';
25                 next_st <= s01;
26             end if;
27         when s01 =>
```

```

28         if d_tmp = '0' then
29             d_out0 <= '0';
30             d_out1 <= '1';
31             next_st <= s10;
32         else
33             d_out0 <= '1';
34             d_out1 <= '0';
35             next_st <= s11;
36         end if;
37     when s10 =>
38         if d_tmp = '0' then
39             d_out0 <= '1';
40             d_out1 <= '1';
41             next_st <= s00;
42         else
43             d_out0 <= '0';
44             d_out1 <= '0';
45             next_st <= s01;
46         end if;
47     when s11 =>
48         if d_tmp = '0' then
49             d_out0 <= '1';
50             d_out1 <= '0';
51             next_st <= s10;
52         else
53             d_out0 <= '0';
54             d_out1 <= '1';
55             next_st <= s11;
56         end if;
57     end case;
58 end process;
59
60 process(clk, rst) -- go to next state
61 begin
62     if rst = '1' then
63         curr_st <= s00;
64         d_tmp <= '0';
65     elsif (clk = '1' and clk'event) then
66         curr_st <= next_st;
67         d_tmp <= data_in;
68     end if;
69 end process;
70
71 -- concurrent assignment of the output
72 data_out0 <= d_out0;
73 data_out1 <= d_out1;
74 end state_machine;

```

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

```

1 subdesign convolutional_code_AHDL(
2     data_in, clk, rst: input;

```



```

3   data_out0, data_out1: output
4 )
5 variable
6   conv_code_state: MACHINE
7   WITH STATES (
8     s00 = b"00",
9     s01 = b"01",
10    s10 = b"10",
11    s11 = b"11");
12   d_in: node;
13
14 begin
15   conv_code_state.clk = clk;
16   conv_code_state.reset = rst;
17
18   d_in = DFF(data_in, clk, VCC, VCC);
19
20   TABLE
21   conv_code_state, d_in => data_out1, data_out0,
22     conv_code_state;
23   s00, 0 => 0, 0, s00;
24   s00, 1 => 1, 1, s01;
25   s01, 0 => 1, 0, s10;
26   s01, 1 => 0, 1, s11;
27   s10, 0 => 1, 1, s00;
28   s10, 1 => 0, 0, s01;
29   s11, 0 => 0, 1, s10;
30   s11, 1 => 1, 0, s11;
31 END TABLE;
32 end;

```

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

```

1 library ieee;
2 use ieee.std_logic_1164.all;
3
4 use work.const_types_pkg.all;
5
6 entity lr_ring_reg is
7   generic(num_cells: integer := 8);
8   port(clk, load: in std_logic;
9         pattern_in: in std_logic_vector(num_cells - 1 downto
10           0);
11         pattern_out_l, pattern_out_r: inout std_logic_vector(
12           num_cells - 1 downto 0));
13 end lr_ring_reg;
14
15 architecture behave of lr_ring_reg is
16   signal pattern_out, pattern_shift: std_logic_vector(
17     num_cells - 1 downto 0);
18 begin
19   gen_x: for i in pattern_out'range generate

```

```

17         pattern_out_l(i) <= pattern_out(i);
18         pattern_out_r(i) <= pattern_out(pattern_out'left +
        pattern_out'right - i);
19     end generate;
20     process (pattern_out)
21     begin
22         pattern_shift(num_cells - 2 downto 0) <= pattern_out(
        num_cells - 1 downto 1);
23         pattern_shift(num_cells - 1) <= pattern_out(0);
24     end process;
25
26     process (clk, load, pattern_in)
27     begin
28         if load = '1' then
29             pattern_out <= pattern_in;
30         elsif clk'event and clk = '1' then
31             pattern_out <= pattern_shift;
32         end if;
33     end process;
34 end behave;

```

Listing 3: Code in VHDL for lr_ring_reg.

```

1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.NUMERIC_STD.ALL;
4
5  entity clk_div_n is
6      generic (
7          CNT_WIDTH : integer := 32
8      );
9      port (
10         clk_in, rst: in  std_logic;
11         n: in  unsigned(CNT_WIDTH-1 downto 0);
12         clk_out: out std_logic
13     );
14 end clk_div_n;
15
16 architecture rtl of clk_div_n is
17     signal cnt: unsigned(CNT_WIDTH-1 downto 0);
18     signal pulse_reg: std_logic;
19     signal n_prev: unsigned(CNT_WIDTH-1 downto 0);
20 begin
21
22     process(clk_in, rst, n)
23     begin
24         if rst = '1' then
25             cnt <= (others => '0');
26             pulse_reg <= '0';
27             --n_prev <= n;
28         elsif clk_in'event and clk_in = '1' then
29             if n /= n_prev then

```

```

30         cnt <= to_unsigned(1, CNT_WIDTH);
31         pulse_reg <= '0';
32         n_prev <= n;
33     elsif cnt = (n - 1) then
34         cnt <= (others => '0');
35         pulse_reg <= '1';
36     else
37         cnt <= cnt + 1;
38         pulse_reg <= '0';
39     end if;
40 end if;
41 end process;
42
43 clk_out <= pulse_reg;
44
45 end rtl;

```

Listing 4: Code in VHDL for clk_div_n.

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  use work.const_types_pkg.all;
5
6  entity decimal_cnt is
7      port(inc, rst: in std_logic;
8           cnt_out: inout digit_array);
9  end decimal_cnt;
10
11 architecture behave of decimal_cnt is
12     component digit_cnt
13         port (inc, rst: in std_logic;
14              inc_out: out std_logic;
15              cnt_out: inout digit);
16     end component;
17     signal inc_vec: std_logic_vector(cnt_out'high + 1 downto 0)
18         ;
19 begin
20     inc_vec(0) <= inc;
21     gen_uutx: for i in 0 to cnt_out'high generate
22     begin
23         uutx: digit_cnt port map (inc_vec(i), rst, inc_vec(i+1)
24             , cnt_out(i));
25     end generate gen_uutx;
26 end behave;

```

Listing 5: Code in VHDL for decimal_cnt.

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  use work.const_types_pkg.all;
5

```

```

6 entity digit_cnt is
7     port (inc, rst: in std_logic;
8           inc_out: out std_logic;
9           cnt_out: inout digit);
10 end digit_cnt;
11
12 architecture behave of digit_cnt is
13     signal cnt_tmp: digit;
14     signal inc_out_tmp: std_logic;
15 begin
16     process (cnt_out)
17     begin
18         if cnt_out = 9 then
19             cnt_tmp <= 0;
20             inc_out_tmp <= '1';
21         else
22             cnt_tmp <= cnt_out + 1;
23             inc_out_tmp <= '0';
24         end if;
25     end process;
26
27     process (inc, rst)
28     begin
29         if rst = '1' then
30             cnt_out <= 0;
31             inc_out <= '0';
32         elsif inc'event and inc = '1' then
33             cnt_out <= cnt_tmp;
34             inc_out <= inc_out_tmp;
35         end if;
36     end process;
37 end behave;

```

Listing 6: Code in VHDL for digit_cnt.

```

1 library ieee;
2 use ieee.std_logic_1164.all;
3
4 use work.const_types_pkg.all;
5
6 entity seven_segments is
7     port (digit_in: in digit;
8           seven_seg_leds: out sev_seg_disp);
9 end seven_segments;
10
11 architecture behave of seven_segments is
12 begin
13     seven_seg_leds <= ss_disp_0 when digit_in = 0 else
14                     ss_disp_1 when digit_in = 1 else
15                     ss_disp_2 when digit_in = 2 else
16                     ss_disp_3 when digit_in = 3 else
17                     ss_disp_4 when digit_in = 4 else

```

```

18             ss_disp_5 when digit_in = 5 else
19             ss_disp_6 when digit_in = 6 else
20             ss_disp_7 when digit_in = 7 else
21             ss_disp_8 when digit_in = 8 else
22             ss_disp_9 when digit_in = 9;
23 end behave;

```

Listing 7: Code in VHDL for seven_segments.

```

1 import numpy as np
2
3 run_min_freq = 0.1
4 run_max_freq = 5
5 num_adr_bits = 5
6
7 num_steps = 2 ** num_adr_bits
8 fpga_clk_freq = 50e6
9 num_leds = 18
10
11 min_led_clk_freq = run_min_freq * num_leds
12 max_led_clk_freq = run_max_freq * num_leds
13 # led_clk_freq_range = np.linspace(min_led_clk_freq,
14     max_led_clk_freq, num_steps)
15 led_clk_freq_range = np.geomspace(min_led_clk_freq,
16     max_led_clk_freq, num_steps)
17
18 cnt_div_clk = (fpga_clk_freq // led_clk_freq_range).astype(np.
19     int64)
20 num_cnt_bits = len(bin(cnt_div_clk.max()))
21 bin_format = f"#0{num_cnt_bits}b"
22
23 rom_def = f"\tconstant clk_leds_cnt_len: integer := {
24     num_cnt_bits-2};\n"
25 rom_def += f"\tconstant adr_len: integer := {num_adr_bits};\n"
26 rom_def += f"\tconstant num_speeds: integer := {num_steps};\n"
27 rom_def += f"\ttype rom_speed is array (0 to num_speeds - 1) of
28     unsigned(clk_leds_cnt_len - 1 downto 0);\n"
29 rom_def += "\tconstant speeds: rom_speed := (\n"
30 for i, cnt in enumerate(cnt_div_clk):
31     rom_def += f'\t\t\t{i} => "{format(cnt, bin_format)[2:]}"',\n'
32 rom_def = rom_def[:-2] + ");\n"
33
34 with open('running_light/speed_rom.vhd', 'w') as f:
35     f.write("library ieee;\n")
36     f.write("use ieee.std_logic_1164.all;\n")
37     f.write("use ieee.numeric_std.all;\n\n")
38     f.write("package speed_rom is\n")
39     f.write(rom_def)
40     f.write("end package;\n")

```

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

```

1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.numeric_std.all;
4
5 package speed_rom is
6     constant clk_leds_cnt_len: integer := 25;
7     constant adr_len: integer := 5;
8     constant num_speeds: integer := 32;
9     type rom_speed is array (0 to num_speeds - 1) of unsigned(
10         clk_leds_cnt_len - 1 downto 0);
11     constant speeds: rom_speed := (
12         0 => "1101001111101101011110001",
13         1 => "101110101100110101011111",
14         2 => "101001001010011111010111",
15         3 => "1001000100100010011110001",
16         4 => "0111111111101101100100011",
17         5 => "0111000011000010111001011",
18         6 => "0110001101100100100010101",
19         7 => "0101011110011011111011011",
20         8 => "0100110100111000111100111",
21         9 => "0100010000010001001101110",
22         10 => "00111011111111111010110000",
23         11 => "0011010011100010011001010",
24         12 => "0010111010011101010101010",
25         13 => "0010100100010110100011110",
26         14 => "0010010000110111100001000",
27         15 => "00011111111101100010100011",
28         16 => "0001110000100011011100000",
29         17 => "0001100011001101011011000",
30         18 => "0001010111011100101010001",
31         19 => "0001001101000101001000111",
32         20 => "0001000011111100010010001",
33         21 => "0000111011111000110001000",
34         22 => "0000110100110010010111100",
35         23 => "0000101110100001110101110",
36         24 => "0000101001000000110011000",
37         25 => "0000100100001001100111001",
38         26 => "0000011111110111010100010",
39         27 => "0000011100000101100010110",
40         28 => "0000011000110000011011101",
41         29 => "0000010101110100100101101",
42         30 => "0000010011001111000000111",
43         31 => "0000010000111101000100011");
44 end package;

```

Listing 9: Code in VHDL for speed_rom.

```

1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.numeric_std.all;
4
5 package const_types_pkg is

```

```

6
7     constant num_sev_seg: integer := 4;
8     constant num_dip_sws: integer := 18;
9     constant num_lights: integer := 18;
10    constant clk_cnt_len: integer := 5;      -- at least ceil(
        log2(num_lights))
11
12    subtype digit is integer range 0 to 9;
13    type digit_array is array (natural range <>) of digit;
14    subtype sev_seg_disp is std_logic_vector (0 to 6);
15    type sev_seg_disp_array is array (num_sev_seg - 1 downto 0)
        of sev_seg_disp;
16    constant ss_disp_0: sev_seg_disp := "0000001";
17    constant ss_disp_1: sev_seg_disp := "1001111";
18    constant ss_disp_2: sev_seg_disp := "0010010";
19    constant ss_disp_3: sev_seg_disp := "0000110";
20    constant ss_disp_4: sev_seg_disp := "1001100";
21    constant ss_disp_5: sev_seg_disp := "0100100";
22    constant ss_disp_6: sev_seg_disp := "0100000";
23    constant ss_disp_7: sev_seg_disp := "0001111";
24    constant ss_disp_8: sev_seg_disp := "0000000";
25    constant ss_disp_9: sev_seg_disp := "0000100";
26
27    constant num_lights_bit_vec: unsigned(clk_cnt_len - 1
        downto 0) := to_unsigned(num_lights, clk_cnt_len);
28    type direction is (left, right);
29 end package;

```

Listing 10: Code in VHDL for const_types_pkg.

```

1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.numeric_std.all;
4
5 use work.const_types_pkg.all;
6 use work.speed_rom.all;
7
8 entity running_light is
9     port(clk, start_fpga, stop_sys_fpga, rst_fpga,
        load_pattern_fpga: in std_logic;
10         dip_sws: in std_logic_vector(num_dip_sws - 1 downto 0);
11         seven_segs: out sev_seg_disp_array;
12         leds: out std_logic_vector(num_lights - 1 downto 0);
13         leds_status: out std_logic_vector(adr_len + 2 downto 0)
            );
14 end running_light;
15
16 architecture behave of running_light is
17     component lr_ring_reg
18         generic (num_cells: integer);
19         port (clk, load: in std_logic;
20             pattern_in: in std_logic_vector(num_lights - 1

```

```

        downto 0);
21         pattern_out_l, pattern_out_r: inout
            std_logic_vector(num_lights - 1 downto 0));
22     end component;
23     component clk_div_n
24         generic (CNT_WIDTH: integer);
25         port (clk_in, rst: in std_logic;
26             n: in unsigned;
27             clk_out: out std_logic);
28     end component;
29     component seven_segments
30         port (digit_in: in digit;
31             seven_seg_leds: out sev_seg_disp);
32     end component;
33     component decimal_cnt
34         port(inc, rst: in std_logic;
35             cnt_out: inout digit_array);
36     end component;
37
38     signal rst, start, stop_sys, load_pattern: std_logic;
39     signal rst_dec_cnt, rst_clk_leds, rst_clk_cnt: std_logic;
40     signal inc_cnt, leds_clk, load: std_logic;
41     signal cnt_runs: digit_array (num_sev_seg - 1 downto 0);
42     signal speed: unsigned(clk_leds_cnt_len - 1 downto 0);
43     signal pattern_in, pattern_out_l, pattern_out_r:
        std_logic_vector(num_lights - 1 downto 0);
44     type state is (ss_reset, ss_stop_sys, ss_run_light,
        ss_load_pattern);
45     signal curr_state, next_state: state;
46     signal dir: direction := left;
47     signal dir_mode: std_logic_vector(2 downto 0);
48
49     begin
50         dec_cnt: decimal_cnt
51             port map(inc_cnt, rst_dec_cnt, cnt_runs);
52         gen_ss : for i in 0 to num_sev_seg - 1 generate
53             ssx: seven_segments port map(cnt_runs(i), seven_segs(i)
                );
54         end generate;
55         clk_leds: clk_div_n
56             generic map(CNT_WIDTH => clk_leds_cnt_len)
57             port map(clk, rst_clk_leds, speed, leds_clk);
58         clk_cnt: clk_div_n
59             generic map(CNT_WIDTH => clk_cnt_len)
60             port map(leds_clk, rst_clk_cnt, num_lights_bit_vec,
                inc_cnt);
61         lr_rr: lr_ring_reg
62             generic map(num_cells => num_lights)
63             port map(leds_clk, load, pattern_in, pattern_out_l,
                pattern_out_r);
64

```



```

65     speed <= speeds(to_integer(unsigned(dip_sws(adr_len - 1
        downto 0))));
66     dir_mode <= dip_sws(adr_len + 2 downto adr_len);
67     leds_status(adr_len - 1 downto 0) <= dip_sws(adr_len - 1
        downto 0);
68     leds <= pattern_out_l when dir = left else
69         pattern_out_r when dir = right;
70     rst <= not rst_fpga;
71     start <= not start_fpga;
72     stop_sys <= not stop_sys_fpga;
73     load_pattern <= not load_pattern_fpga;
74
75     process(inc_cnt, dir_mode, dir)
76     begin
77         if dir_mode = "000" then
78             leds_status(adr_len + 2 downto adr_len) <= "001";
79             dir <= left;
80         elsif dir_mode(0) = '1' then
81             dir <= left;
82             leds_status(adr_len + 2 downto adr_len) <= "001";
83         elsif dir_mode(1) = '1' then
84             dir <= right;
85             leds_status(adr_len + 2 downto adr_len) <= "010";
86         else
87             leds_status(adr_len + 2 downto adr_len) <= "100";
88             if inc_cnt'event and inc_cnt = '1' then
89                 if dir = left then
90                     dir <= right;
91                 else
92                     dir <= left;
93                 end if;
94             end if;
95         end if;
96     end process;
97
98     process(curr_state, start, load_pattern, stop_sys, dip_sws)
99     begin
100         case curr_state is
101             when ss_reset =>
102                 pattern_in <= (num_lights - 1 => '1', others =>
                    '0');
103                 load <= '1';
104                 rst_dec_cnt <= '1';
105                 rst_clk_leds <= '1';
106                 rst_clk_cnt <= '1';
107                 if start = '1' then
108                     next_state <= ss_run_light;
109                 elsif load_pattern = '1' then
110                     next_state <= ss_load_pattern;
111                 else
112                     next_state <= ss_reset;

```

```

113         end if;
114     when ss_stop_sys =>
115         pattern_in <= (others => '0');
116         load <= '0';
117         rst_dec_cnt <= '0';
118         rst_clk_leds <= '1';
119         rst_clk_cnt <= '1';
120         if start = '1' then
121             next_state <= ss_run_light;
122         elsif load_pattern = '1' then
123             next_state <= ss_load_pattern;
124         else
125             next_state <= ss_stop_sys;
126         end if;
127     when ss_run_light =>
128         pattern_in <= (others => '0');
129         load <= '0';
130         rst_dec_cnt <= '0';
131         rst_clk_leds <= '0';
132         rst_clk_cnt <= '0';
133         if stop_sys = '1' then
134             next_state <= ss_stop_sys;
135         elsif load_pattern = '1' then
136             next_state <= ss_load_pattern;
137         else
138             next_state <= ss_run_light;
139         end if;
140     when ss_load_pattern =>
141         pattern_in <= dip_sws;
142         load <= '1';
143         rst_dec_cnt <= '1';
144         rst_clk_leds <= '1';
145         rst_clk_cnt <= '1';
146         if start = '1' then
147             next_state <= ss_run_light;
148         else
149             next_state <= ss_load_pattern;
150         end if;
151     end case;
152 end process;
153
154 process (clk, rst)
155 begin
156     if rst = '1' then
157         curr_state <= ss_reset;
158     elsif clk'event and clk = '1' then
159         curr_state <= next_state;
160     end if;
161 end process;
162 end behave;

```

Listing 11: Code in VHDL for running_light.