ENGINE

Teaching online electronics, microcontrollers and programming in Higher Education

# Hardware Implementation of Algorithms

## 7. Incremental encoder.

**Lead Partner: Warsaw University of Technology**

**Autor: Lukasz Mik**

University of Applied Sciences in Tarnow

Erasmus+

GRANT NUMBER: 2020-1-PL01-KA226-HE-095653

## Declaration

This laboratory instruction has been prepared in the context of the ENGINE project. Where other published and unpublished source materials have been used, these have been acknowledged.

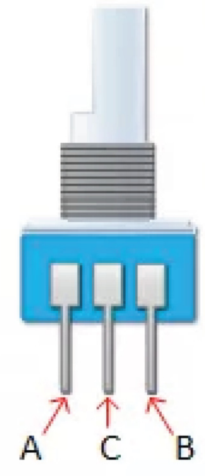
## Copyright

## Funding Disclaimer

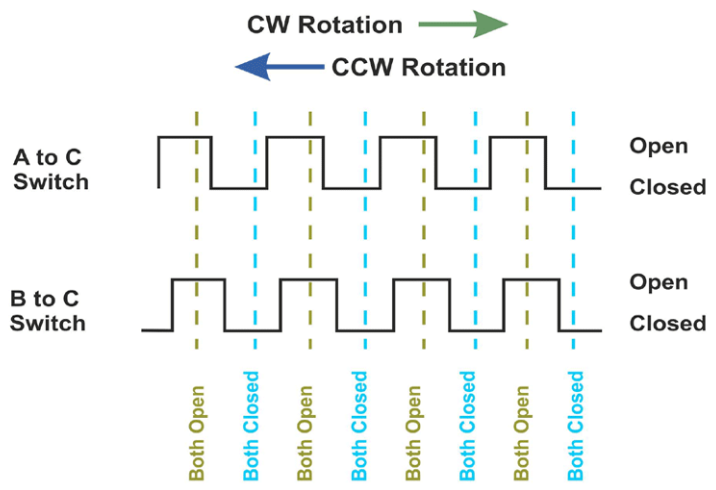
This project has been funded with support from the European Commission. This report reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

# I. Incremental encoder - principle of operation.

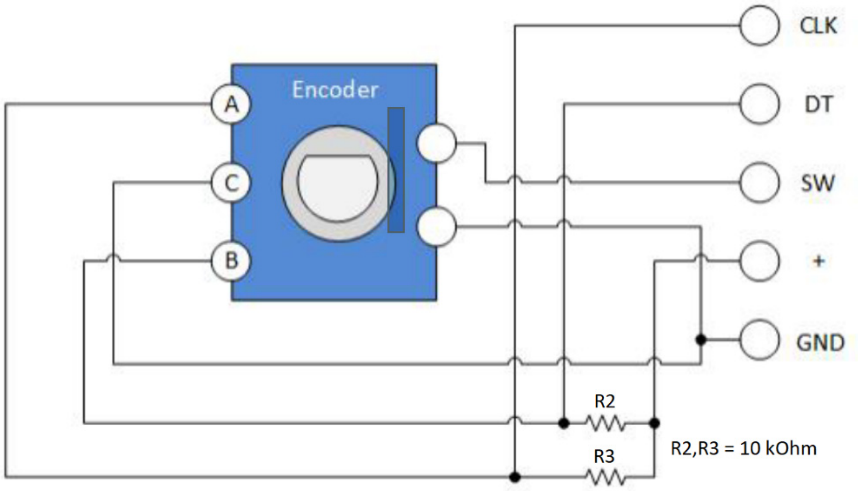
The incremental rotary encoder module by Keyes with the symbol KY-040 or alternatively HW-040 will be used for this exercise. Pull-up resistors for the A and B lines of the encoder are already provided in this module.
The rotary encoder is an element that allows you to read the position of the shaft rotation (in this case, the knob) and the direction of its rotation. It has a strictly defined number of positions per full turn of the knob. The encoder itself has 3 leads: A, B and C.



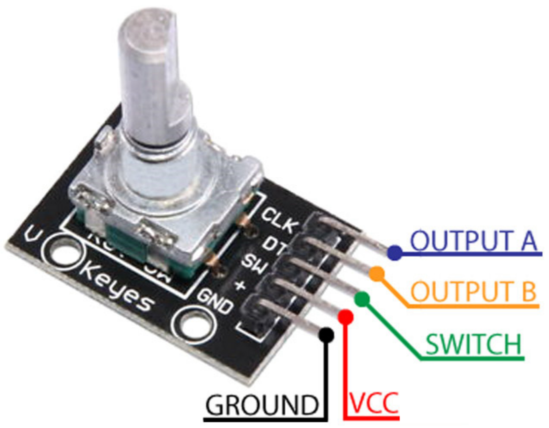
On pins A and B, pulses are generated when the knob is rotated. These pulses are out of phase with each other, which makes it possible to determine the direction of rotation. Pin C is a common ground for both A and B signals, which are generated by switches.



When the knob is turned clockwise, it causes the output of switch A (pins A and C) to be earlier than the output of switch B (pins B and C). The rotation speed also affects the frequency of the pulses. The diagram of the encoder module is shown in the figure below.



The module has been designed in such a way that when the A or B contacts are shorted, 0V voltage appears on their outputs, and in the case of opening - the supply voltage thanks to pull-up resistors with a resistance of 10 kOhm. The description of the pins on the module looks like this:

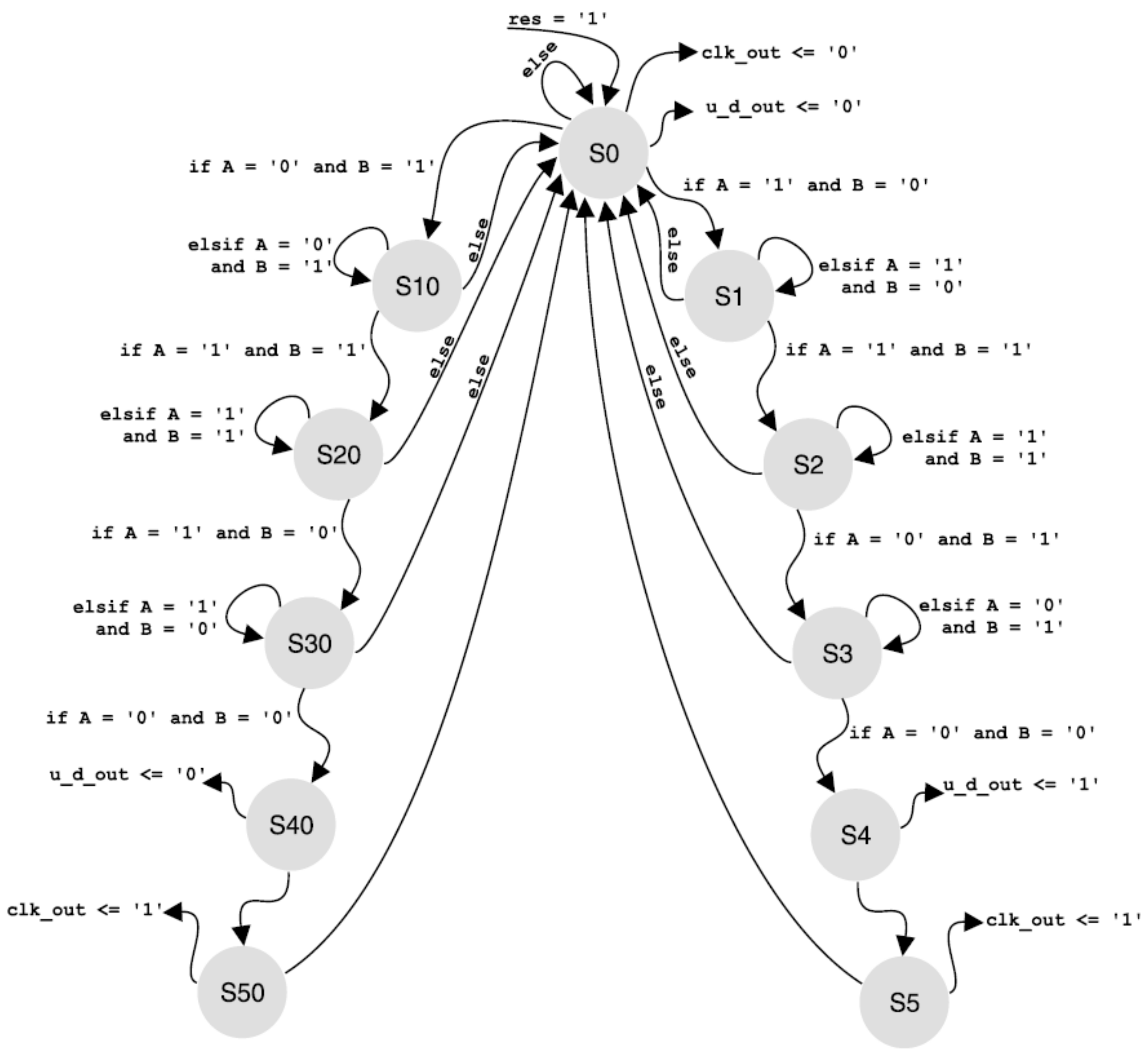


In order to adapt the application of the encoder module to the FPGA, connect the supply voltage to the + and GND pins. CLK (A) and DT (B) pins should be connected to selected FPGA pins, available on goldpin connectors. We omit the SW pin, which is the output of the switch, shorted when the knob is pressed.

Taking into account the inevitable impulse interference generated during closing and opening of contacts, such a pair of signals is not suitable for use directly in other digital circuits. Timing circuits are needed to eliminate contact jitter in FPGAs. The simplest way to implement a timing system in an FPGA is to build a state machine clocked by an external clock signal. It is also necessary to set such a working cycle of this automaton that will minimize the impact of contact vibrations on the quality of decoding.

# II. Signal decoder implementation for incremental encoder.

In modern HDL languages, including VHDL, it is possible to describe the work of an automaton in many ways. To ensure the clarity of the description of the automaton's operation, it has been presented in the form of a graph, the individual states of which are explicitly predefined by the constant Sx declaration, where x identifies the state. Although explicitly assigning values to individual states requires some work, it is easier to check the correctness of the logical description (verification of the description), because on the basis of the value of the **state** vector it is easy to determine in which state (in what place in the graph) the automaton is at a given moment..



Due to the principle of operation, the designed automatic unit is equipped with a reset input (asynchronous) **res** - each time after switching on the power supply, a short impulse of high level should be given to it. Hardware zeroing forces the automaton to go to the S0 state, which is the initial state - the automaton remains in this state until the level changes on one of the inputs: A or B. Depending on which of them the high level occurs first (which determines the rotation direction encoder axis), the automatic unit goes to state S1 (counting direction up) or S10 (counting direction down).

## Implementation of a decoder for encoder signals in VHDL

**STEP 1:** In ISE Design Suite 14.7 (Webpack) create a new project called encoder, set the appropriate parameters of the target FPGA.

**STEP 2:** In the project, create a new *VHDL Module* file called **encoder**. After creating it, an editing window with the contents of the *encoder.vhd* file will open. First, complete the header part with libraries.

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

**STEP 3:** Then declare the unit name as the project encoder along with its I/O ports.

```
entity encoder is
port ( A : in std_logic;          -- input A
       B : in std_logic;          -- input B
       res: in std_logic;         -- reset input
       clk: in std_logic;         -- clock input
       EN : out std_logic_vector(2 downto 0); -- enable for 7-segment display
       SSEG: out std_logic_vector(7 downto 0) -- 7-segment display
);
end encoder;
```

**STEP 4:** After declaring the unit, move on to the description of the architecture. In the declaration part of the architecture, define the constants from **S0** to **S5** and from **S10** to **S50** representing individual states from the state machine diagram presented earlier. In addition, declare the **encoder_rol** signal (encoder right or left), which indicates the direction of rotation of the encoder knob, and the **encoder_clk**, whose output generates rectangular pulses in the S5 or S50 states. In addition, declare the **bcd_counter** signal, responsible for counting pulses from the encoder. This signal will directly drive the input of the BCD code decoder to the seven-segment display code.

```
architecture arch of encoder is
  signal state: std_logic_vector(3 downto 0);
  constant S0 : std_logic_vector(3 downto 0):= "0000";
  constant S1 : std_logic_vector(3 downto 0):= "0001";
  constant S2 : std_logic_vector(3 downto 0):= "0010";
  constant S3 : std_logic_vector(3 downto 0):= "0011";
  constant S4 : std_logic_vector(3 downto 0):= "0100";
  constant S5 : std_logic_vector(3 downto 0):= "0101";
  constant S10 : std_logic_vector(3 downto 0):= "1001";
  constant S20 : std_logic_vector(3 downto 0):= "1010";
  constant S30 : std_logic_vector(3 downto 0):= "1011";
  constant S40 : std_logic_vector(3 downto 0):= "1100";
  constant S50 : std_logic_vector(3 downto 0):= "1101";
  signal encoder_clk : std_logic;
  signal encoder_rol : std_logic;
  signal clk_1kHz : std_logic:='0';
  signal bcd_counter : std_logic_vector(3 downto 0):="0000";
```

**STEP 5:** In the architecture description, the first process is responsible for handling signals from the encoder. It starts with the handling of the reset signal - return to the zero state, as well as the handling of the transition from the S0 to S1 state (turning the encoder 1 tact to the right) or from the state S0 to S10 (turning the encoder 1 tact to the left).

```
BEGIN
process (clk_1kHz, res)
begin
  if res = '0' then
       state <= "0000";
  elsif clk_1kHz'event and clk_1kHz = '1' then
    case state is
      when S0 =>
         encoder_clk <= '0';
         encoder_rol <= '0';
         if A = '1' and B = '0' then
           state <= S1;
         elsif A = '0' and B = '1' then
           state <= S10;
         else state <= S0;
         end if;
```

**STEP 6:** If there was a change from S0 to S1, then go to the path of the state machine responsible for detecting rotational movement to the right ("increase"). This path of the state machine is also started on every cycle (combination) of signals A and B for the direction of rotation to the right (states S1 to S5).

```
-- path of the state machine for motion detection in the "increase" direction
        when S1 =>
          if A = '1' and B = '1' then
            state <= S2;
          elsif A = '1' and B = '0' then
            state <= S1;
          else state <= S0;
          end if;
        when S2 =>
          if A = '0' and B = '1' then
            state <= S3;
          elsif A = '1' and B = '1' then
            state <= S2;
          else state <= S0;
          end if;
        when S3 =>
          if A = '0' and B = '0' then
```

```
            state <= S4;
          elsif A = '0' and B = '1' then
            state <= S3;
          else state <= S0;
          end if;
        when S4 =>
          encoder_rol <= '1';
          state <= S5;
        when S5 =>
          encoder_clk <= '1';
          state <= S0;
```

**STEP 7:** If there was a change from S0 to S10, we go to the path of the state machine, responsible for detecting rotational movement to the left ("decrease"). This path of the state machine is also started on every cycle (combination) of signals A and B for the direction of rotation to the left (states from S10 to S50).

```
-- path of the state machine for motion detection in the "decrease" direction
        when S10 =>
          if A = '1' and B = '1' then
            state <= S20;
          elsif A = '0' and B = '1' then
            state <= S10;
          else state <= S0;
          end if;
        when S20 =>
          if A = '1' and B = '0' then
            state <= S30;
          elsif A = '1' and B = '1' then
            state <= S20;
          else state <= S0;
          end if;
        when S30 =>
          if A = '0' and B = '0' then
            state <= S40;
          elsif A = '1' and B = '0' then
            state <= S30;
          else state <= S0;
          end if;
        when S40 =>
          encoder_rol <= '0';
          state <= S50;
        when S50 =>
          encoder_clk <= '1';
          state <= S0;
        when others =>
          state <= S0;
        end case;
  end if;
end process;
```

In both paths of the state machine, if there is a different combination of signals A and B than assumed (when changing the direction of rotation), the state returns to S0.

**STEP 8:** After the process that handles the encoder, add a process responsible for generating a clock signal called **clk_1kHz** and frequency 1kHz. This frequency (period equal to 10 ms) is suitable to eliminate the influence of vibration of the encoder contacts.

```
--clock 1kHz
process(clk)
  variable clock_cnt : integer:=0;
begin
  if rising_edge(clk) then
    if clock_cnt < 6000 then
         clock_cnt := clock_cnt+1;
       else
         clock_cnt := 0;
         clk_1kHz <= not(clk_1kHz);
       end if;
  end if;
end process;
```

**STEP 9:** The next process in the architecture will be responsible for handling the counter mod 10, clocked by the **encoder_clk** signal. Counting direction of the counter will be controlled by the **encoder_rol** signal. If the encoder axis rotates to the left, the counter will be incremented by 1 with each cycle of the encoder_clk signal, otherwise it will be decremented by 1.

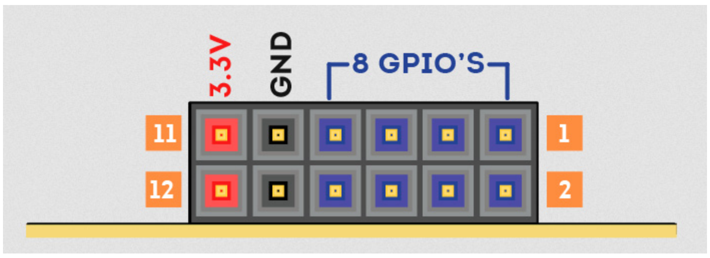
```
process(encoder_clk)
begin
  if rising_edge(encoder_clk) then
       if bcd_counter < 10 then
         if(encoder_rol = '1') then
              bcd_counter <= bcd_counter + 1;  --encoder knob rotates right
         else
              bcd_counter <= bcd_counter - 1;  --encoder knob rotates left
         end if;
       else
         bcd_counter <= "0000";
       end if;
  end if;
end process;
```

**STEP 10:** At the end of the description of the architecture, add the process responsible for decoding the BCD code into the code of the seven-segment display. It is also necessary to set the bits of the EN vector responsible for selecting the active display.

```
process(bcd_counter)
begin
  case bcd_counter is
       --------------------abcdefgp
       when "0000"=>SSEG<="00000011";
       when "0001"=>SSEG<="10011111";
       when "0010"=>SSEG<="00100101";
       when "0011"=>SSEG<="00001101";
       when "0100"=>SSEG<="10011001";
       when "0101"=>SSEG<="01001001";
```

```
        when "0110"=>SSEG<="01000001";
        when "0111"=>SSEG<="00011111";
        when "1000"=>SSEG<="00000001";
        when "1001"=>SSEG<="00001001";
        when others=>SSEG<="11111111";
    end case;
  end process;
  EN <= "110";
  END arch;
```

After completing all the above steps, it remains to connect the encoder to the pins of the FPGA. Connect the encoder to the pins of the P1 socket called GPIO HEADER P1.



The connection scheme is described in the table below.

| Encoder pin | GPIO HEADER „P1” |
|---|---|
| + | 3.3V |
| GND | GND |
| CLK | 1 |
| DT | 2 |

For the project to work properly, the file with the assignment of the FPGA pins to the I/O ports of the project unit is still missing. Therefore, create a file called **encoder.ucf** (when creating a new file, select the *Implementation Constraints File* option) and paste the following content.

```
NET "A"   LOC = P31   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
NET "B"   LOC = P32   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
NET "clk"  LOC = P129  | IOSTANDARD = LVCMOS33 | PERIOD = 12MHz;
NET "res" LOC = P80   | PULLUP   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;

NET "SSEG[7]"  LOC = P117  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
NET "SSEG[6]"  LOC = P116  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
NET "SSEG[5]"  LOC = P115  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
NET "SSEG[4]"  LOC = P113  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
NET "SSEG[3]"  LOC = P112  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
NET "SSEG[2]"  LOC = P111  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
NET "SSEG[1]"  LOC = P110  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
NET "SSEG[0]"  LOC = P114  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
NET "EN[2]"  LOC = P124 | IOSTANDARD = LVCMOS33 | DRIVE = 8 | SLEW = FAST ;
NET "EN[1]"  LOC = P121 | IOSTANDARD = LVCMOS33 | DRIVE = 8 | SLEW = FAST ;
NET "EN[0]"  LOC = P120  | IOSTANDARD = LVCMOS33 | DRIVE = 8 | SLEW = FAST ;
```

After re-saving and synthesizing the project, generate a configuration file for the FPGA.

**NOTE:** Observe how many encoder ticks correspond to a single change of the counter displayed on the display? Can you explain what is happening?

**TASK:** Modify the design so that the rotation of the encoder increments or decrements the counter in the range from 0 to 99 (two seven-segment displays must be used).

## References

- J. Majewski, P. Zbysiński – *Układy FPGA w przykładach.* Wydawnictwo BTC, Legionowo 2007.