ENGINE

Teaching online electronics, microcontrollers and programming in Higher Education

Hardware Implementation of Algorithms

4. BCD to seven-segment display decoder. Counters.

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 Where other published and unpublished source materials have been used, these have been acknowledged.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.

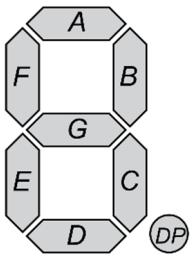
I. BCD code and seven segment display

The BCD (Binary-Coded Decimal) code is a method of writing a number consisting in encoding successive decimal digits of this numer in a binary system. Only four bits (half-byte) are used for this purpose. Due to the way numbers are represented in digital systems, it is widely used in electronics and computer science. This notation allows you to easily convert a decimal number to binary and vice versa.

Digit	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

The BCD code is most often used in electronic devices with a digital display (e.g. calculators, digital meters).

A seven-segment display is a type of display consisting of 7 luminous elements (segments), arranged in a shape that allows the display of all decimal digits from 0 to 9. Lighting elements are most often LED diodes (see the figure below).

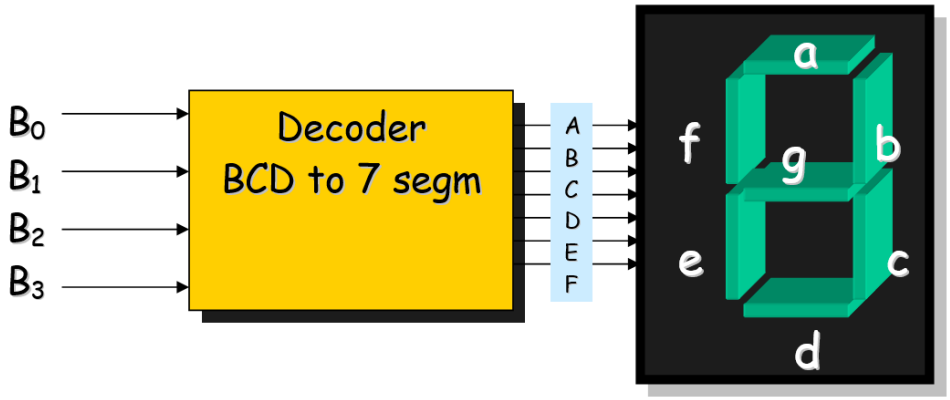


Segment designations of a single seven-segment display

The segments are labeled **A** through **G** clockwise, starting with the highest one. An additional 8th segment called DP is used to mark a decimal point in a number with a fractional part.

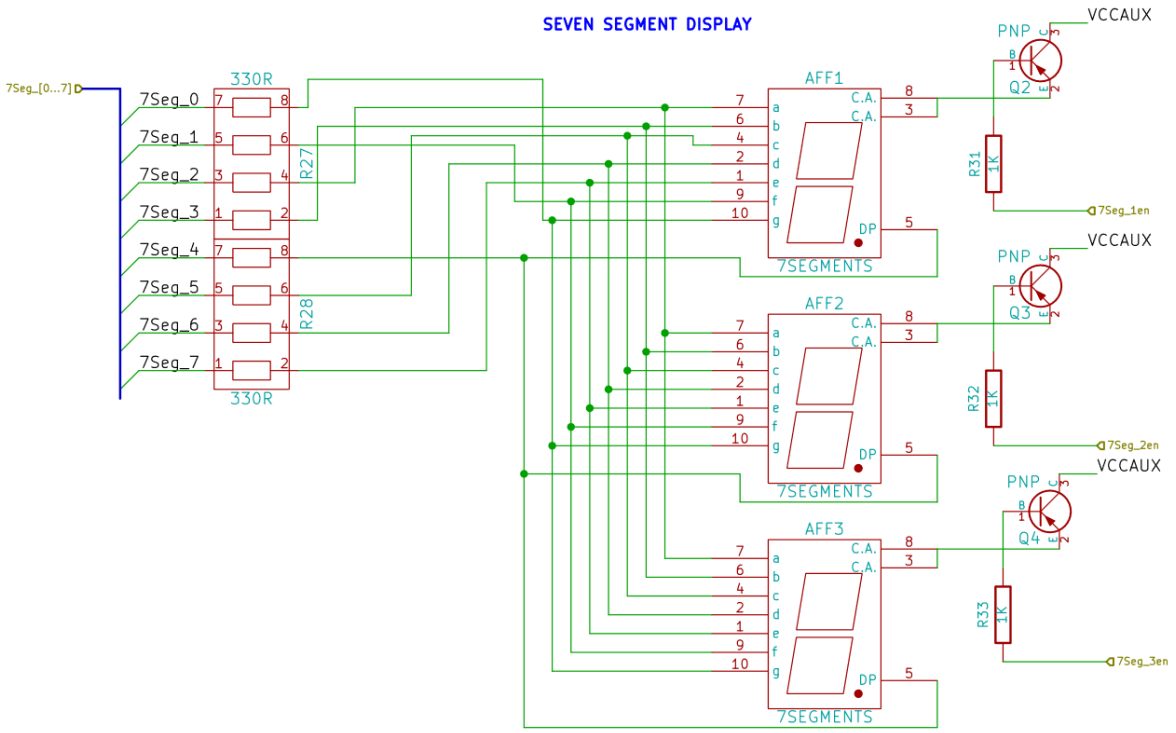
II. BCD to seven-segment display decoder.

Direct connection of 4 lines with BCD code to a 7-segment display is not possible. It is necessary to use a special decoder that will convert the 4-bit string given at its input to the 7-bit string generated at its output. A simplified connection diagram of the decoder with the display is shown in the figure below.



Simplified decoder and display connection diagram

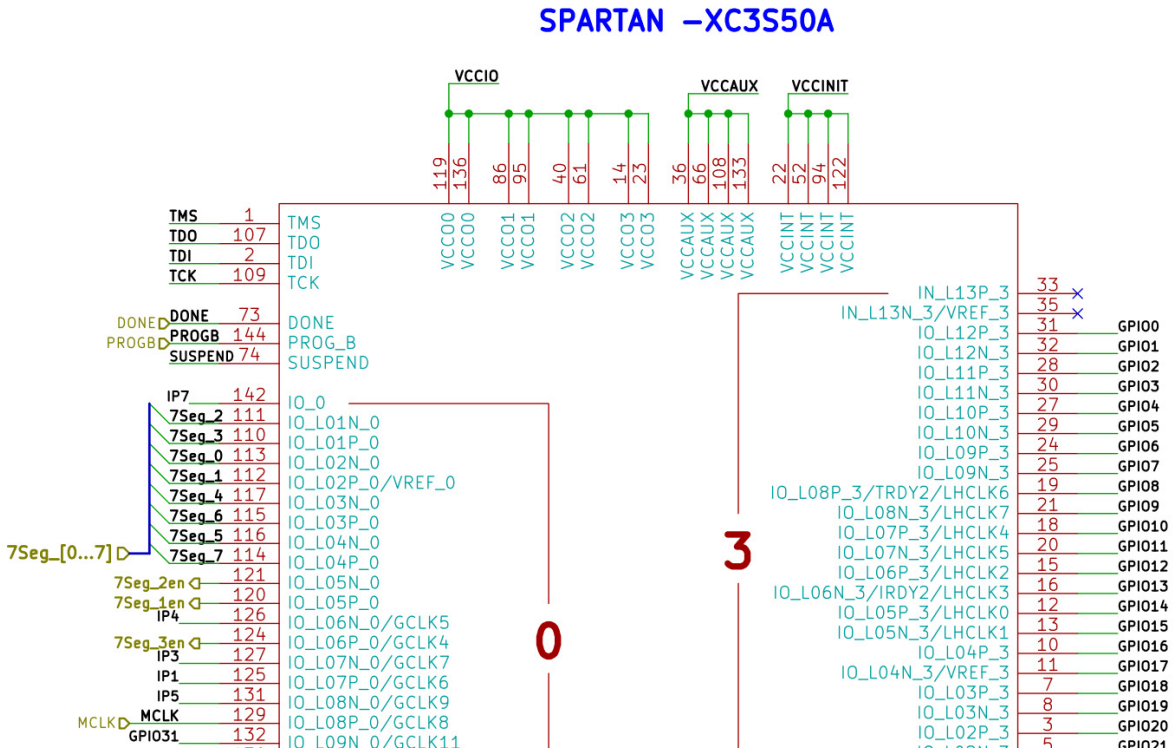
The Numato Elbert v2 evaluation board has three 7-segment displays that share the same FPGA port lines.



Fragment of a diagram showing section 3 of 7-segment displays

As can be seen in the figure above, all displays are of the common anode type. Therefore, to light each of the segments from **a** to **g**, it is necessary to short these pins to ground (set 0 logic state on FPGA pin). The bases of Q2, Q3 and Q4 are connected to the FPGA pins and marked as **en** (enable) lines. The active display is selected by applying a logic 0 to the base of one of the transistors. The resistor ladder R27, R28 is used to limit the current flowing through the LEDs in the seven-segment display as well as the output port of the FPGA.

From the FPGA side, the display lines have been connected to pins 110 to 117. The enable lines have been connected to pins 120, 121 and 124. The manufacturer on the product page provides an appropriate file with definitions of port names along with assigned pin numbers.



Fragment of the diagram showing the connection of the 7-segment display line to the FPGA

III. Implementation of the BCDto7SEG decoder in VHDL.

To support displays connected to the Spartan3A chip on the Elbert board, the following ports need to be defined:

- 4 – bit input port called BCD, which we will physically connect to DIP switches.
- 8 – bit output port called SSEG, which we will connect to the appropriate segments of the display (we include the DP dot as 8 bit)
- 3 – bit output port called EN, responsible for selecting the active display.

The header part (declaration of used libraries) together with the declaration of the design unit will look like this:

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

entity bcdto7seg is
port
(
	BCD: in std_logic_vector(3 downto 0);
	SSEG: out std_logic_vector(7 downto 0);
	EN : out std_logic_vector(1 downto 0);
);
end bcdto7seg;
```

The description of the architecture should take into account all states on the BCD input and assign the appropriate states on the SSEG output. If we want to display digits on one of the displays, then it is enough to clear one bit of the EN vector. The architecture description template is presented below. Please note that the display is of the common anode type, therefore the active state at the input of each segment is logical 0. The DIP switch that is connected to the BCD port of our decoder also works in reverse logic, so in the architecture we will use the auxiliary signal BCD_temp, by means of which we will negate the bits from the BCD input.

```
architecture Behavioral of bcdto7seg is
signal BCD_temp : std_logic_vector(3 downto 0);
begin
EN <= "110";
BCD_temp <= not BCD;
process(BCD_temp)
begin
	case BCD_temp is
		--------------------edcpbafg
		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;
end Behavioral;
```

To create a decoder project, it is still necessary to generate a UCF file, which will contain connections of the ports of the bcdto7seg design unit to the physical pins of the system. Note that each bit of an input or output vector must be assigned separately. The contents of the UCF file are shown below.

```
NET "BCD[0]"  LOC = P70   | PULLUP  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
NET "BCD[1]"  LOC = P69   | PULLUP  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
NET "BCD[2]"  LOC = P68   | PULLUP  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
NET "BCD[3]"  LOC = P64   | 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 ;
```

In ISE WebPack, follow these steps:

- create a new project,
- select a destination chip,
- create a new source file (VHDL module)
- paste the design unit declaration along with a description of its architecture

- create a new file with the assignment of signals to pins (*Implementation Constraints File*)
- Compile the project and generate the configuration file (*.bit or *.bin)
- Program the target system using the ElbertV2Config software.

After completing these steps, you should check if the project works properly. In case of problems with compilation, please use the source files included with the exercise.

ZADANIE 1: Based on what you've learned so far in this and previous classes, try adding support for a second display and another section of 4 DIP switches to your project. Remember that displaying on 2 displays at the same time will require multiplexing, i.e. a quick change of bits in the EN vector, responsible for selecting the active display. The switching frequency must be high enough so that the human eye is not able to notice the flickering of the diodes.

IV. Implementation of a binary counter in VHDL.

To implement counters, it is necessary to use additional libraries that allow performing arithmetic operations on vectors interpreted as unsigned numbers.

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

In the declaration of the design unit of the counter, instead of the BCD input, we give the input from the CLK clock

```
entity cntmod10 is
port
(
	CLK : in std_logic_vector(3 downto 0);
	SSEG : out std_logic_vector(7 downto 0);
	EN : out std_logic_vector(2 downto 0)
);
end cntmod10;
```

In the architecture description code, we define 2 signals: ***clk_1Hz***, which will be responsible for generating a 1 Hz clock signal, and a ***counter***, which will count from 0 to 9 (modulo 10 counter).

```
architecture Behavioral of cntmod10 is
	signal clk_1Hz : std_logic:='0';
	signal counter : std_logic_vector(3 downto 0):="0000";
begin

...

end Behavioral;
```

We place 3 processes in the body of the architecture. The task of the first one will be to generate a clock signal with a frequency of 1 Hz.

```
process(clk)
 variable clock_cnt : integer:=0;
 begin
    if rising_edge(clk) then
        if clock_cnt < 6000001 then
            clock_cnt := clock_cnt+1;
        else
            clock_cnt := 0;
            clk_1Hz <= not(clk_1Hz);
        end if;
    end if;
end process;
```

The second process will be responsible for counting 1 Hz clock cycles.

```
process(clk_1Hz)
begin
    if rising_edge(clk_1Hz) then
        if counter < 10 then
            counter <= counter + 1;
        else
            counter <= "0000";
        end if;
    end if;
end process;
```

The ***counter*** value is incremented with each rising edge of the clk_1Hz signal. After reaching the value of 10, it is immediately reset, which is why we call it a modulo 10 counter - it counts in the range from 0 to 9.

The task of the third process will be to display the counter status from the second process on a seven-segment display.

```
process(counter)
begin
      case 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;
```

The modified UCF file will look like this:

```
NET "CLK"  LOC = P129  | IOSTANDARD = LVCMOS33 | PERIOD = 12MHz;

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 ;
```

As in point III, create a project by adding the appropriate files. After completing the source codes, compile the project and generate the configuration file. In the event that errors occur during compilation, use the source files provided for classes. Check if the system is working properly, i.e. the counter is incremented every second and its value is displayed on the 7-segment display.

ZADANIE 2: Modify the project from point IV so that inside the architecture they are 2 counters modulo 10. One counting up and one counting down. Both counters are to be incremented and decremented at a frequency of 1 Hz. Note that counter values must be displayed on separate displays.

References

- P. Zbysiński, J. Pasierbiński – *Układy programowalne – pierwsze kroki.* Wydawnictwo BTC, Warszawa 2004

- Elbert V2 S3A FPGA Development Board*, https://numato.com/product/elbert-v2-spartan-3a-fpga-development-board/*