
UNIVERISTY: FACULTY OF AUTOMATION AND COMPUTER SCIENCE

DEPARTMENT: COMPUTER SCIENCE

DISCIPLINE: DIGITAL SYSTEM DESIGN

PROJECT: POCKET COMPUTER WITH BASIC NUMERICAL OPERATIONS

Laboratory teacher

Diana Irena Pop

Student

Andrei-Călin Cadar

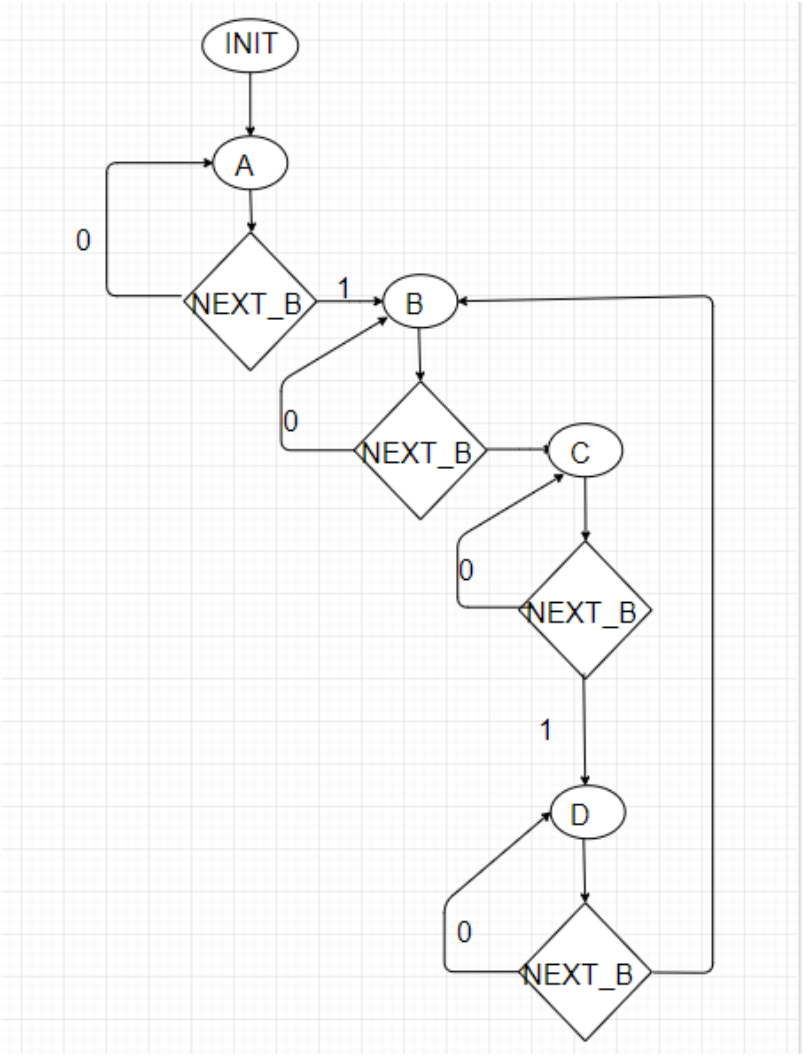
**Constrains**

1. Project specification
2. State Diagram
3. Block Diagram
4. Components
5. VHDL code
6. FPGA board Nexys 4
7. Instructions for use
8. Justification for chosen solution
9. Possibilities for further development
10. Bibliography

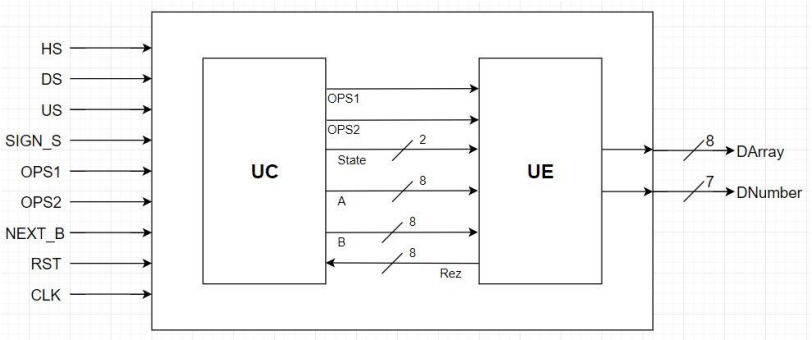
## 1) Project specification

Requirement: Design a pocket computer with basic arithmetic operations (addition, subtraction, multiplication, division). Multiplication and division operations will be implemented using specific algorithms, not language operators. The operands are represented by 8 bits with a sign. Operants and operators will be entered sequentially in decimal form. The 7-segment displays on the FPGA boards will be used.

## 2) State Diagram



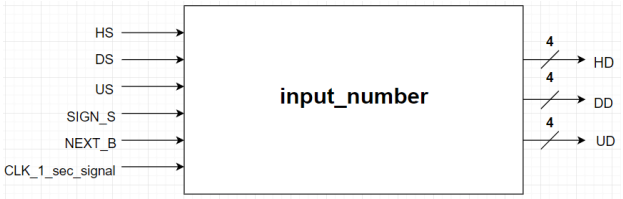

## 3) Block Diagram



**4) Components**

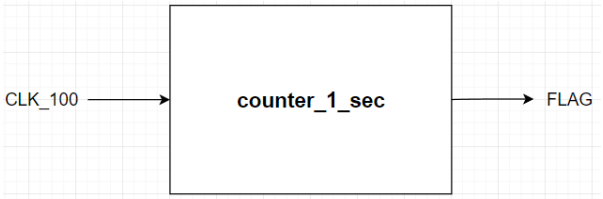
In this project, the following entities were used **I**

a) input_number



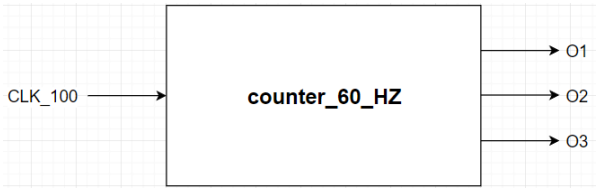

This entity forms the number, it uses the 1 second clock to increment each stored digit( on 4 bits) in the 0 -> 9 loop if the corresponding switch is

b) counter_1_sec



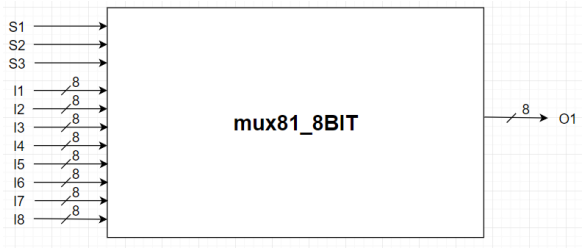

This is a frequency divider with the period of the clock equal to one second. We use it for the input of the number.

c) counter_60_HZ




This is a 60HZ frequency divider and we use it to iterate trough the digit output array and the active display array.

d) mux81_8BIT




Mux 8:1 with data bus width equal to 8 bits.
Used to iterate trough active display array.

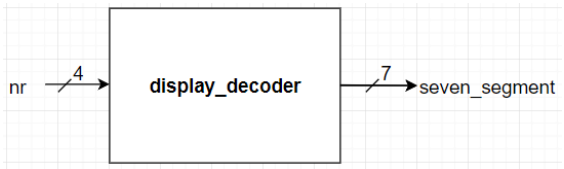
e) mux41_8BIT


Mux 4:1 with data bus width equal to 8 bits.
Used to determine the numbers we are currently working with.

f) mux81_4BIT

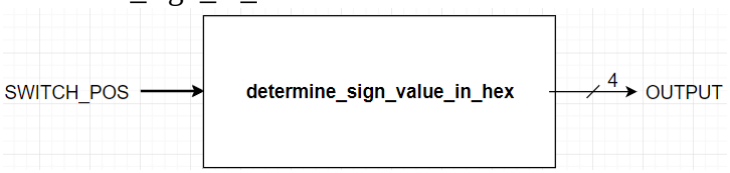

Mux 8:1 with data bus width equal to 4 bits.
Used to iterate trough digit display array

g) display_decoder



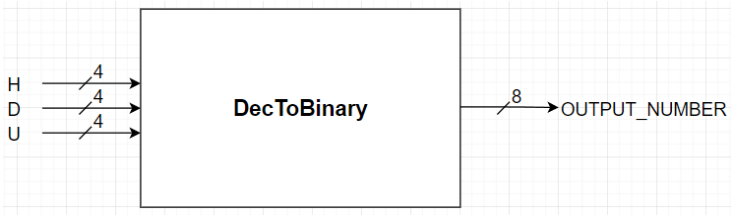

Used to convert digit or sign to 7 segments code.

h) determine_sign_in_hex



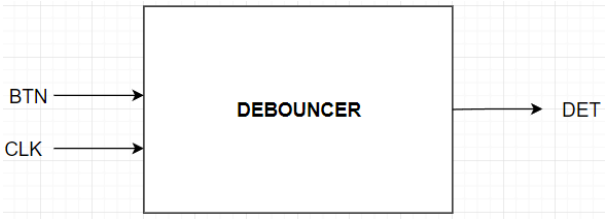

Used to convert sign to 7 segments code.

i) DecToBinary



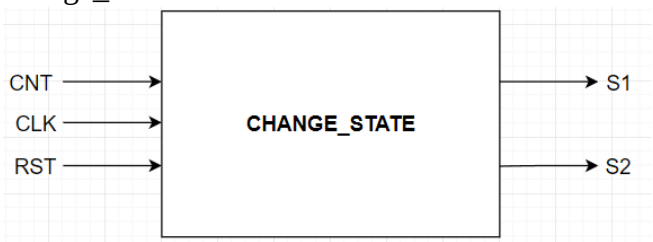

Converts a 3 digit number, each digit on 4 bits to an 8 bit number in binary using the reverse double dabble algorithm.

j) Debouncer




A debouncer for the Next_B button.

k) change_state

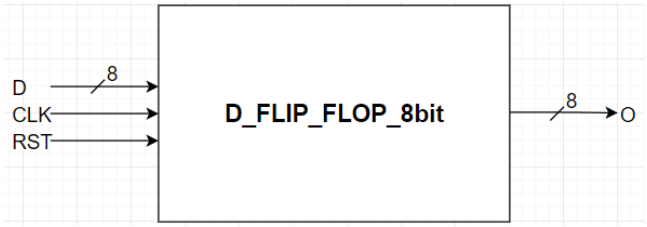



A counter which counts in the loop 0->1->2->3->1, which has as count signal the output from the debouncer.

l) binaryToDec

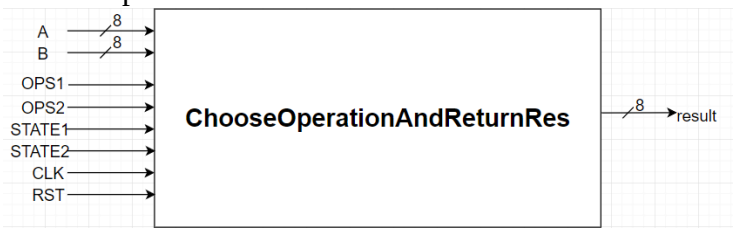

Converts an 8 bit number in binary to base ten on 3 digits using the double dabble algorithm.

m) D_FLIP_FLOP_8bit



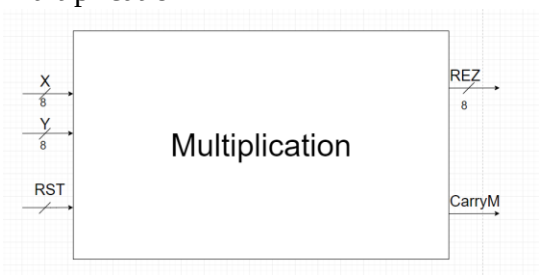

Used as a register to store numbers between states.

n) ChooseOperationAndReturnRes




The actual calculator, computes all 4 operations, and using a MUX 4:1 selects which operation to display based on the OPS1 and OPS2 switches.

o) Multiplication




Used to multiply X and Y. Carry is set if REZ > xFF.

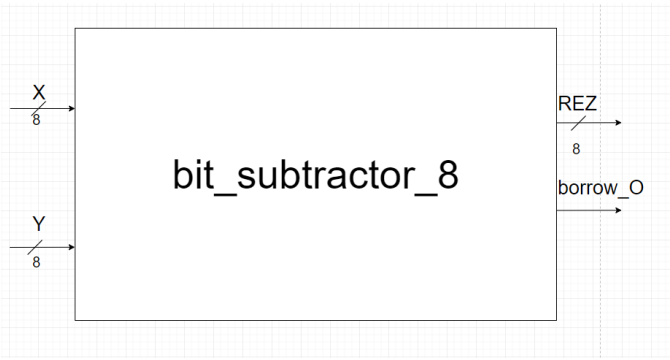
p) Division


Used to divide X to Y, returns xFF if Y is 0.

p) bit_adder_8


Used to add X and Y in REZ and we have carry_O for overflow.

q) bit_subtractor_8




Used to subtract X from Y and set borrow_O if abs(X) < abs (Y);

### 5) VHDL code

Active-HDL Designer Edition provides FPGA designers with a mixed RTL simulator that includes: industry proven IEEE mixed-language simulation support for VHDL, Verilog® and SystemVerilog (Design), with 2X-plus performance gains over FPGA supplied RTL simulators, encrypted IP support and no limitations on FPGA device size.

I wrote the code in this project using VHDL language which is one of the commonly used Hardware Description Languages (HDL) in digital circuit design. VHDL stands for VHSIC Hardware Description Language. In turn, VHSIC stands for Very-High-Speed Integrated Circuit.

VHDL was initiated by the US Department of Defense around 1981. The cooperation of companies such as IBM and Texas Instruments led to the release of VHDL's first version in 1985. Xilinx, which invented the first FPGA in 1984, soon supported VHDL in its products. Since then, VHDL has evolved into a mature language in digital circuit design, simulation, and synthesis.

The VHDL code on the main page of the project can be divided into three important parts the libraries used, the entity (black box) and the architecture (inside the black box).

```
1  library IEEE;
2  use IEEE.STD_logic_1164.all;
3  use IEEE.STD_logic_unsigned.all;
4  use IEEE.numeric_std.all;
```

```
23 entity calculator is
24     port(CLK_100, HS, DS, US, SIGN_S, OPS1, OPS2, NEXT_B, RST: in std_logic;
25     DArray: out std_logic_vector(7 downto 0);
26     DNumber: out std_logic_vector(0 to 6);
27     rez_o: out std_logic_vector(7 downto 0));
28 end entity;
```

Figure 4.1 Entity of project

```
type DisplaySelectorType is array (0 to 7) of std_logic_vector(7 downto 0);
signal DSelector:DisplaySelectorType := ("11111110", "11111101", "11110111", "11101111", "11011111", "10111111", "01111111");
signal OutDigit, HD, DD, UD: std_logic_vector(3 downto 0);
signal Out_of_60HZ_1, Out_of_60HZ_2, Out_of_60HZ_3, FLAG_1_SEC, STATE1, STATE2, Last_NEXT_B_STATE: std_logic;
signal A, A_AUX, A_AUX_2, A_DELAYED, B, B_AUX, B_AUX_2, REZ, THE_INPUT: std_logic_vector(7 downto 0);
signal A_DOUBLE_DABBLE, B_DOUBLE_DABBLE: std_logic_vector(11 downto 0);
signal signValue: std_logic_vector(3 downto 0) :="1010";
signal change_state_signal, not_clock: std_logic;

begin
    not_clock <= not CLK_100;
    --freq divider
    Counter_60_HZ_C: counter_60_HZ port map(CLK_100, Out_of_60HZ_1, Out_of_60HZ_2, Out_of_60HZ_3);
    Counter_1_sec_C: counter_1_sec port map(CLK_100, FLAG_1_SEC);


    --display
    MUX81_8BIT_C: mux81_8BIT port map(Out_of_60HZ_1, Out_of_60HZ_2, Out_of_60HZ_3, DSelector(0), DSelector(1), DSelector(2), DSelector(3), DSelector(4), DSelector(5), DSelector(6), DSelector(7), DArray);
    Mux81_4BIT_C: Mux81_4BIT port map(Out_of_60HZ_1, Out_of_60HZ_2, Out_of_60HZ_3, A_DOUBLE_DABBLE(11 downto 8), A_DOUBLE_DABBLE(7 downto 4), A_DOUBLE_DABBLE(3 downto 0), signValue, "0000", "0000", "0000", "0000", OutDigit);
    Display_decoder_C: display_decoder port map(OutDigit, DNumber);
    Determine_sign_value_in_hex_C: determine_sign_value_in_hex port map(SIGN_S, signValue);

    --Input number
    --Input_number_C: input_number port map(FLAG_1_SEC, HS, DS, US, SIGN_S, HD, DD, UD);
    Input_number_C: input_number port map(CLK_100, HS, DS, US, SIGN_S, NEXT_B, HD, DD, UD);

    --decode a from 8bits to 3 sets fo 4 bits
    binaryToDec_C: binaryToDec port map(A, A_DOUBLE_DABBLE(11 downto 8), A_DOUBLE_DABBLE(7 downto 4), A_DOUBLE_DABBLE(3 downto 0));

    --Here we do A
    D_FLIP_FLOP_8bit_C1: D_FLIP_FLOP_8bit port map(A_AUX, not_clock, RST, A_AUX_2);
    D_FLIP_FLOP_8bit_C2: D_FLIP_FLOP_8bit port map(A, NEXT_B, RST, A_DELAYED);
    mux41_8BIT_C: mux41_8BIT port map(STATE2, STATE1, THE_INPUT, THE_INPUT, A_AUX_2, REZ,  A_AUX);
    A <= A_AUX;

    --Here we do B
    D_FLIP_FLOP_8bit_C3: D_FLIP_FLOP_8bit port map(B_AUX, not_clock, RST, B_AUX_2);
    mux41_8BIT_C2: mux41_8BIT port map(STATE2, STATE1, "00000000", A_DELAYED, B_AUX_2, B_AUX_2, B_AUX);
    B <= B_AUX;

    --decode the input to 8 bits
    DecToBinary_C : DecToBinary port map(HD, DD, UD, THE_INPUT);

    --States
    -- pentru simulator DEBOUNCER_C: DEBOUNCER port map(NEXT_B, CLK_100, change_state_signal);
    --change_state_C: change_state port map(change_state_signal, CLK_100, RST, STATE2, STATE1);
    change_state_C: change_state port map(NEXT_B, CLK_100, RST, STATE2, STATE1);

    --Calculator
    ChooseOperationAndReturnRes_C: ChooseOperationAndReturnRes port map(B, A_DELAYED, OPS1, OPS2, STATE2, STATE1, CLK_100, RST, REZ);
    rez_o <= rez;

end architecture;
```

Figure 4.2 Architecture of the project

```
component input_number is
    port(CLK_1_sec_signal, HS, DS, US, SIGN_S, NEXT_B: in std_logic;
    HD, DD, UD: out std_logic_vector(3 downto 0));
end component;

component counter_1_sec is
    port(CLK_100: in std_logic;
    FLAG: out std_logic);
end component;

component counter_60_HZ is
    port(CLK_100: in std_logic;
    O1, O2, O3: out std_logic);
end component;

component mux81_8BIT is
    port(S1, S2, S3: in std_logic;
    I1, I2, I3, I4, I5, I6, I7, I8: in std_logic_vector(7 downto 0);
    O1: out std_logic_vector(7 downto 0));
end component;

component mux41_8BIT is
    port(S1, S2: in std_logic;
    I1, I2, I3, I4: in std_logic_vector(7 downto 0);
    O1: out std_logic_vector(7 downto 0));
end component;

component mux81_4BIT is
    port(S1, S2, S3: in std_logic;
    I1, I2, I3, I4, I5, I6, I7, I8: in std_logic_vector(3 downto 0);
    O1: out std_logic_vector(3 downto 0));
end component;

component display_decoder is
    port(nr: in std_logic_vector(3 downto 0);
    seven_segment: out std_logic_vector(0 to 6));
end component;
```

```
component determine_sign_value_in_hex is
    port(SWITCH_POS: in std_logic;
    OUTPUT: out std_logic_vector(3 downto 0));
end component;

component DecToBinary is
    port(H, D, U: in std_logic_vector(3 downto 0);
    OUTPUT_NUMBER: out std_logic_vector(7 downto 0));
end component;

component DEBOUNCER is
    Port ( BTN : in STD_LOGIC;
           CLK : in STD_LOGIC;
           DET : out STD_LOGIC);
end component;

component change_state is
    port(CNT, CLK, RST: in std_logic;
        S2, S1: out std_logic);
end component;

component binaryToDec is
    port(inputNumber: in std_logic_vector(7 downto 0);
    H, D, U: out std_logic_vector(3 downto 0));
end component;

component D_FLIP_FLOP_8bit is
    port(D: in std_logic_vector(7 downto 0);
    CLK, RST: in std_logic;
    O: out std_logic_vector(7 downto 0));
end component;

component ChooseOperationAndReturnRes is
    port(A, B: in std_logic_vector(7 downto 0);
    OPS1, OPS2, STATE1, STATE2: in std_logic;
    CLK, RST: in std_logic;
    result: out std_logic_vector(7 downto 0));
end component;
```

Figure 4.3 Components

Each component is built separately, using processes or structurally by connecting with other smaller components, for example in Figure 4.4.

```
architecture multiplication_a of multiplication is
begin

    process(RST)
    variable A, B: std_logic_vector(15 downto 0):=(others => '0');
    variable Q: std_logic_vector(7 downto 0);
    variable N: std_logic_vector(3 downto 0);
    begin
        if RST = '1' then
            Q:=Y;
            B(7 downto 0) := X;
            carryM <= '0';
            A:= x"0000";
        else
            for i in 0 to 7 loop
                if Q(0) = '1' then
                    A := A + B;
                end if;
                B(15 downto 1) := B(14 downto 0);
                B(0) := '0';
                Q(6 downto 0) := Q(7 downto 1);
                Q(7) := '0';

                if A > x"FF" then
                    carryM <= '1';
                end if;
            end loop;
        end if;
            rez <= A(7 downto 0);
    end process;
end architecture;
```

## 6) FPGA board Nexys 4

The Nexys4 board is a complete, ready-to-use digital circuit development platform based on the latest Artix-7™ Field Programmable Gate Array (FPGA) from Xilinx®. With its large, high-capacity FPGA (Xilinx part number XC7A100T-1CSG324C), generous external memories, and collection of USB, Ethernet, and other ports, the Nexys4 can host designs ranging from introductory combinational circuits to powerful embedded processors.

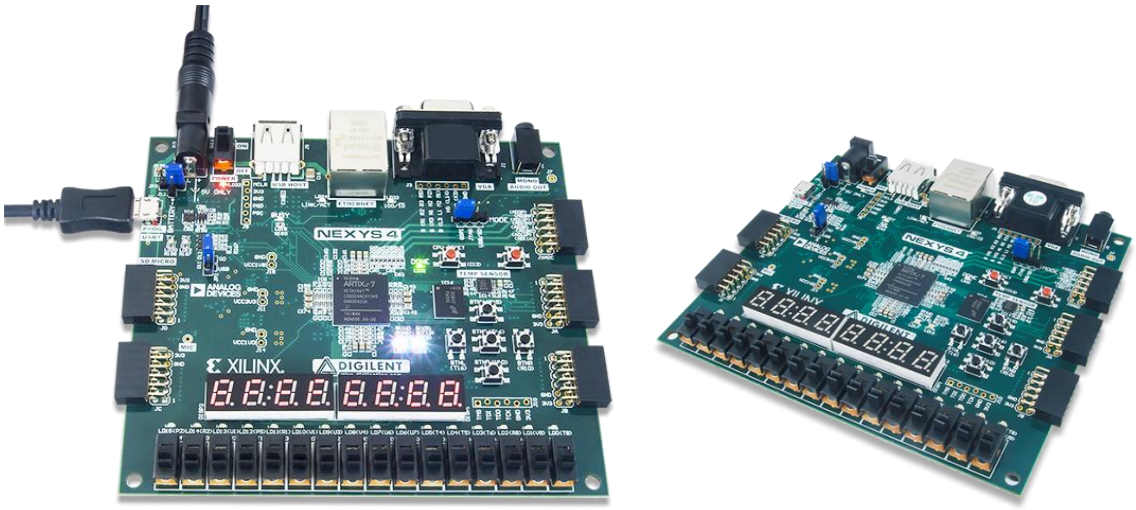



Figure 5.1 FPGA Nexys 4

The Nexys4 offers an improved collection of ports and peripherals, including:

- 4,860 Kbits of fast block RAM
- Six clock management tiles, each with phase-locked loop (PLL)

- Internal clock speeds exceeding 450 MHz
- 16 user switches
- 16 user LEDs
- Two tri-color LEDs
- 12-bit VGA output
- Two 4-digit 7-segment displays

Each digit is comprised of seven LEDs connected such that their anodes are all tied together, and each LEDs cathode is independently controllable. So, at a basic level, knowing nothing else about their control circuit, we can expect the light up segments turning the common anode high, and pulling low cathode pins that correspond to the LED segments we want to light up (and pulling high those that we do not by default).

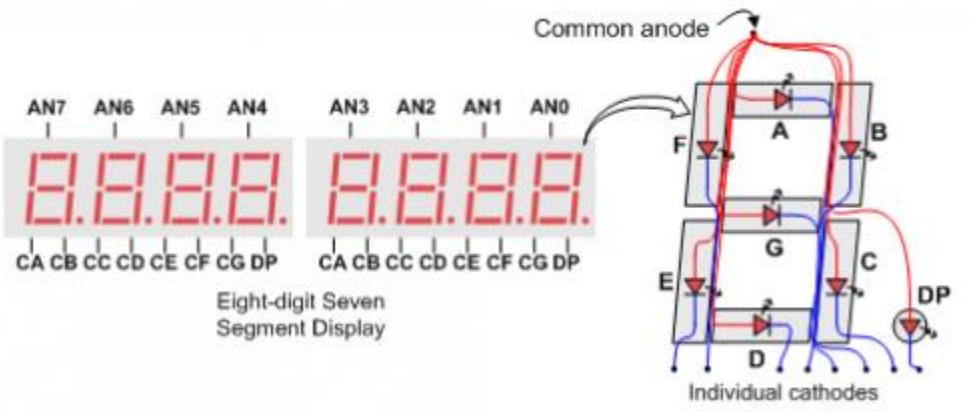



Figure 5.2 The 7-Segment Displays

## 7) Instructions for use

1. Program the FPGA with the written code.
2. Write the constraints file
3. Reset using the reset switch
4. Introduce the first number
5. Press next
6. Introduce the second number
7. Press next
8. Choose operation
9. Press next
10. Now the result is displayed
11. Press next to go to the 6$^{th}$ step using the previous stored result as the first number.

## 8) Justification for chosen solution

This was not an easy project and I have known it from the start and I have taken it as a challenge. Honestly the project was mostly built on the fly because a lot of problems appeared when building it according to the scheme. There were a lot of signals to take into account and how to synchronise more multiplexers depending on each other. I took advantage of the fact we can only store 3 numbers (like building the Fibonacci sequence). I used conditioned delayed registers to move the first and the second

number around but not in every state. The reading and printing is always done from the first number register. I have chosen the used division algorithm, because I already needed to know it from Assembly.

**9) Possibilities for further development**

We could add the A^B operation by building a counter with B steps and multiply A in a register at each step. Division with fractional part can also be added by changing the division algorithm to support rational numbers.

**10) Bibliography**


https://www.aldec.com/en/products/fpga_simulation/designeredition

https://www.allaboutcircuits.com/technical-articles/hardware-description-langauge-getting-started-vhdl-digital-circuit-design/

https://www.xilinx.com/support/documents/university/XUP%20Boards/XUPNexys4DDR/documentation/Nexys4-DDR_rm.pdf

https://web.mit.edu/6.111/volume2/www/f2019/handouts/labs/lab2_19/index.html

https://en.wikipedia.org/wiki/Double_dabble

Octavian Augustin Cret's lectures at the university.