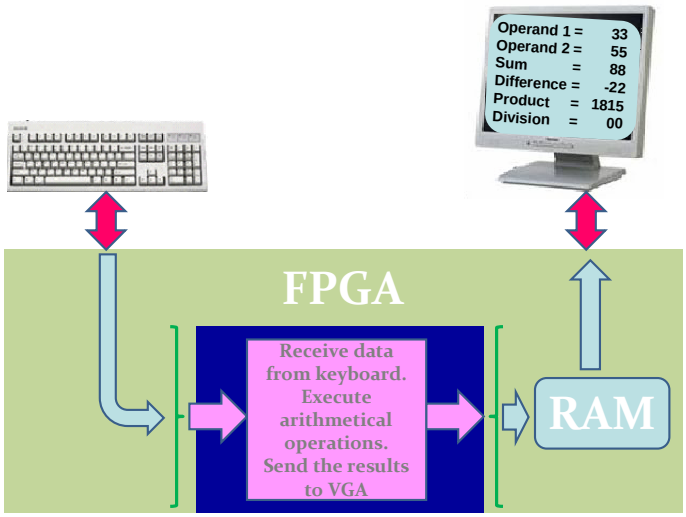
LAB. 2

Objectives

- Allow the FPGA circuits to interact with peripheral devices;
- Provide interface with a VGA monitor;
- Provide interface with a keyboard;
- Design and implement a simple arithmetical circuit, which receives input data from a keyboard and displays the results of arithmetical operations on a VGA monitor screen.

Task

Implement a calculator which executes operations of addition, subtraction, multiplication, and integer division over two operands that range from 0 to 99. The operands are specified with the aid of keyboard and the results of all the operations have to be displayed on a VGA monitor screen as shown on figure below.



Details

1. Blocks that manage interface with keyboard and VGA monitor will be provided.

2. The arithmetical block must have the following external interface:

```
entity ArithmeticBlock is
    generic (AddressBits : natural;
             StateMaxValue : positive;
             NumberOfColumns : natural);
    port ( ASCII_in        : in  std_logic_vector (7 downto 0);
           ASCII_out       : out  std_logic_vector (7 downto 0);
           Address_in      : in  std_logic_vector (AddressBits - 1 downto 0);
           Address_out     : out  std_logic_vector (AddressBits - 1 downto 0);
           clk             : in std_logic;
           rst             : in std_logic;
```

```
            WE_in           : in std_logic;
            WE_out          : out std_logic);
end ArithmeticBlock;
```

where

`AddressBits` is the width of address bus for VGA RAM (fixed);

`StateMaxValue` is the number of states required to display all your data (one state per one graphical symbol to be displayed on monitor screen – you have to calculate this parameter);

`NumberOfColumns` is the number of text columns which can be displayed with available monitor resolution (fixed);

`ASCII_in` - ASCII code received from the keyboard;

`Address_in` – serial number of ASCII code received from the keyboard (all codes received from the keyboard are numbered from 0 to $2^{AddressBits}$-1) - you would use this address in order to distinguish the first operand from the second;

`Address_out` – address of VGA RAM to which you would like to write a byte of information (ASCII code of a symbol that you would like to display at position `Address_out`);

`clk` – main clock to be used by VGA controller, keyboard, and your arithmetical circuit;

`rst` – reset signal generated by DCM (it is active only while DCM has not achieved lock);

`WE_in` – is active when new ASCII code has arrived from the keyboard;

`WE_out` – controls writing operation to VGA RAM.

3. Start with one-digit operands (that range from 0 to 9). Receive the respective ASCII codes from the keyboard – assume that the first dialed digit (and all subsequent odd dialed digits) corresponds to the first operand and the second digit (and all subsequent even dialed digits) – to the second operand (discard all non-digits). Convert the received ASCII codes to 4-bit binary values.

4. Realize the operations of addition, subtraction (note that the result might be negative), and multiplication.

5. Convert the results of arithmetical operations to BCD and then to ASCII. Note that the result of multiplication operation might range from 0 to 81. Consequently, you would need two digits in order to display the product. You can use a simple ROM to perform conversion of product result to BCD.

6. Send everything (the operands, the results, and auxiliary text) to VGA RAM and test your circuit on the prototyping board.

7. Provide support for integer division. You may use a simple algorithm which sequentially subtracts the second operand (different from 0!) from the first until the first operand turns out to be smaller than the second operand, incrementing each time an auxiliary counter.

8. Now switch to two-digit operands (that range from 0 to 99) and repeat all previous steps. Note that the result of multiplication operation now might range from 0 to 9801. Use "**Shift and Add-3**" algorithm to convert the operands and the results of operations to BCD.

Appendix

Shift and Add-3 Algorithm

1. Shift the binary number left one bit.

2. If 16 shifts have taken place, the BCD number is in the *Thousands*, *Hundreds*, *Tens*, and *Units* columns.

3. If the binary value in any of the BCD columns is 5 or greater, add 3 to that value in that BCD column.

4. Go to 1.

Example:

Convert number 9801 to BCD. $9801_{10} = 2649_{16} = 0010\ 0110\ 0100\ 1001_2 = ?_{BCD}$

Operation	*Thousands*	*Hundreds*	*Tens*	*Units*	Binary number
start					**0010 0110 0100 1001**
shift left (1)				0	010 0110 0100 1001
shift left (2)				00	10 0110 0100 1001
shift left (3)				001	0 0110 0100 1001
shift left (4)				0010	0110 0100 1001
shift left (5)			0	0100	110 0100 1001
shift left (6)			00	1001	10 0100 1001
add 3 and shift left (7)			001	1001	0 0100 1001
add 3 and shift left (8)			0011	1000	0100 1001
add 3 and shift left (9)		0	0111	0110	100 1001
add 3 and shift left (10)		01	0101	0011	00 1001
add 3 and shift left (11)		011	0000	0110	0 1001
add 3 and shift left (12)		0110	0001	0010	1001
add 3 and shift left (13)	1	0010	0010	0101	001
add 3 and shift left (14)	10	0100	0101	0000	01
add 3 and shift left (15)	100	1001	0000	0000	1
add 3 and shift left (16)	1001	1000	0000	0001	
BCD	**9**	**8**	**0**	**1**	