

Pocket CALCULATOR

DSD first year project



May 31, 2023

Borzan călina-annemary

PROJECT SUPERVISOR: Ing. Diana Pop

Table of Contents

[1 Specifications 2](#_Toc130314624)

[2 Design 3](#_Toc130314625)

[2.1 Black Box 3](#_Toc130314626)

[2.2 Control and Execution Unit 3](#_Toc130314627)

[2.2.1 Resources (breakdown of the Execution Unit) 4](#_Toc130314629)

[2.2.4 State diagram of the Control Unit 7](#_Toc130314631)

[2.2.5 Detailed diagram of the project 9](#_Toc130314632)

[3 User manual 10](#_Toc130314633)

[4 Technical justifications for the design 13](#_Toc130314634)

[5 Future developments 14](#_Toc130314635)

**Pocket Calculator**

# 1.Specifications

We have to design a pocket calculator with basic arithmetic operations(addition, subtraction, multiplication, division).The calculator must operate as follows:

* The multiplication and division operations will be implemented using specific algorithms, not language operators.
* The operators will be represented on 8 bits with sign.
* The operands and operators will be entered sequentially in decimal form.
* The 7-segment display on the FPGA board will be us

# 2.Design

## Black Box



## INPUTS AND OUTPUTS

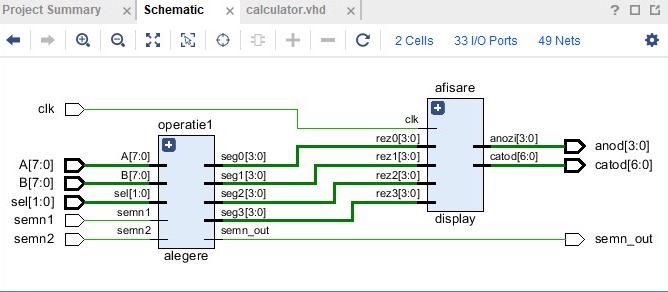
The black box will have as main inputs the data entered from the switches> numbers: A(7:0),B(7:0),which will be represented in hexadecimal, the sign of the first and the second number:semn1,semn2 , a selection that will help us choose the operation which we want to be implemented, a clk and a reset. The main outputs are the sign of the result and the result, which will be displayed on the 7-segment BCD( anod, catod).

## Control and Execution Unit

The system's black box must be further broken down in order to find implementable components. We will do a **top-down** breakdown of the problem until we get to known circuits, and then we will implement **bottom-up**.

The first breakdown of any system is one in which we will differentiate between the **control logic** in the system and the **system resources**. The control logic is represented by the Control Unit (CU) which is the one that controls all the resources and the resources are represented by the Execution Unit (EU).

The Control Unit in this project will be represented by the Calculator component, which generates the control signals for the entire system, direct the data flow and makes sure that the data gets on the board. We have the two input numbers, the two selection buttons, in order to choose the operation the EU performs, the sign, in order for the EU to compute the sign of the final result, and the clock signal.



### 

### Resources (breakdown of the Execution Unit)

In order to further establish the links between the CU and the EU, we must first identify **the resources on the basis of which we make decisions**. The EU has the role of executing some instructions in order to execute mathematical calculations. Any decision-making information must come from a resource that generates that information and passes it on to UC. The EU contains the resources that realize the implementation of the operations such as: a 1-bit full adder, an 8-bit adder/subtractor, a divider, a displayer, a comparator, a multiplier.

**RESOURCES**

1. 1 Bit Full-Adder

This will be a simple 1-bit full adder that will add two 1-bit numbers. As input it will have X1,X2,and SEL0 and as outputs COUT and S.



1. 8 Bit Adder-Subtractor

Because we need to make the sum of two numbers with 8 bits we need to cascade eight 1-bit full adders. This will be a component that will tell us, according to the least bit of the selection input, if we generate the addition or subtraction of the 2 numbers. If sel(0) is 0 we simply add the numbers, but if it is 1 we must invert all the bits of the second number, in order to realize the subtraction.



3.Multiplication algorithm

This operation represents the multiplication of 2 numbers on 8 bits. It is a process that depends on the selection and the operands N1 and N2.It is realized with the help of the eight-bit adder-subtractor created previously. This algorithm is done almost the same as when we want to multiply two numbers with more then one digit. We obtain every row by multiplying every bit of the first element with the rest of the elements from the second number. Then we shift the first row to the right and add it with the help of the adder to the second one. We repeat this process till we reach the last row.



4.Division algorithm

This operation represents the division of two numbers on 8 bits. This process depends on the operands A and B. It is realized with the following algorithm: shift the divisor to the left and compare it with the divident.If divisor is bigger than dividend, we shift to the next bit of the quotient. If the divisor is smaller than the dividend and greater than 0 we subtract to get new dividend and shit 1 as next bit of quotient. The shifting is done by multiplying with power of 2.

sh(8 to 15)<=B;

sh(0 to 7)<=”0000000”;

sh6<=sh7+sh7;--shifted once,2^1

sh5<=sh6+sh6;--shifted twice,2^2

sh4<=sh5+sh5;--shifted third,2^3

sh3<=sh4+sh4;--shifted for time,2^4

s2<=sh3+sh3;--shifted 5 times,2^5

sh1<=sh2+sh2;--shifted 6 times,2^6

sh0<=sh1+sh1;--shifted 7 times,2^7



5.Comparator

I decided to use a comparator for the two 8 bits number as to be able to decide with more certainty how an operation should be done and, in some case, to decide event the sign of the result. If N1 is greater or equal then N2 it will return 1,and if not, it will return 0,which is why the output Y will have only one bit.



6.Display

The input variables, namely clk, rez0, rez1, rez2, and rez3, are used to transmit signals. These signals represent different numbers that will be shown on four screens. The numbers displayed on the screens are divided into four groups: rez3 (first four bits of the result), rez2 (next four bits), rez1 (next four bits), and rez0 (last four bits).

The device operates on a clock signal called clk, which has a frequency of 100 mHz. To enable simultaneous display on all four screens, the clock signal is divided into 20 units. This division ensures that the information is properly synchronized.

Additionally, there is a signal called selector , which determines which anode will be activated based on the digit to be displayed. Initially, selector is set to 0, and it increments when a different digit needs to be displayed. The values of selector determine which anode is activated: 00 for the first anode, 01 for the second anode, 10 for the third anode, and 11 for the last anode.



### 

### 2.2.2 State diagram of the Control Unit

### First, I made a block diagram that helps us understand the implementation of the

### project. This is the diagram for the first steps needed for the implementation.



# 3.User manual

To begin using Vivado, start by opening the software. Once Vivado is launched, navigate to the "File" menu and select "Project" and then "New." This action will open a new window where you can specify the project name. After entering the name, proceed by clicking "Next."

If you are working with the Basys3 board, in the subsequent window, select the following options: "Family" - Artix\_7, "Package" - cpg236, and "Part" - xc7a35tcpg236-1. Once these options are chosen, click "Next" and then "Finish."

To incorporate the necessary files into the project, go to the "Add Sources" function. Locate the relevant VHDL files and select them all. Finally, click "Open" to add them to the project.

Next, you need to create a constraints file. Follow these steps:

Click on "Add Sources" and then "Add or create constraints."

Click the plus sign to add a new file and select "Create file."

Give the file a name and click "Finish."

Now, select the constraints file from the "Sources" window and assign the pins as follows:

For the pin assignments, use the following format:

For each pin, use the set\_property command:

Set the PACKAGE\_PIN to the pin number.

Use [get\_ports nume\_input/output] to specify the input/output name.

Set the IOSTANDARD to LVCMOS33.

Here are the pin assignments I used:

set\_property PACKAGE\_PIN R2 [get\_ports {A[7]}]

set\_property PACKAGE\_PIN T1 [get\_ports {A[6]}]

set\_property PACKAGE\_PIN U1 [get\_ports {A[5]}]

set\_property PACKAGE\_PIN W2 [get\_ports {A[4]}]

set\_property PACKAGE\_PIN R3 [get\_ports {A[3]}]

set\_property PACKAGE\_PIN T2 [get\_ports {A[2]}]

set\_property PACKAGE\_PIN T3 [get\_ports {A[1]}]

set\_property PACKAGE\_PIN V2 [get\_ports {A[0]}]

set\_property PACKAGE\_PIN W13 [get\_ports {B[7]}]

set\_property PACKAGE\_PIN W14 [get\_ports {B[6]}]

set\_property PACKAGE\_PIN V15 [get\_ports {B[5]}]

set\_property PACKAGE\_PIN W15 [get\_ports {B[4]}]

set\_property PACKAGE\_PIN W17 [get\_ports {B[3]}]

set\_property PACKAGE\_PIN W16 [get\_ports {B[2]}]

set\_property PACKAGE\_PIN V16 [get\_ports {B[1]}]

set\_property PACKAGE\_PIN V17 [get\_ports {B[0]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {A[7]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {A[6]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {A[5]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {A[4]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {A[3]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {A[2]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {A[1]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {A[0]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {B[7]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {B[6]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {B[5]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {B[4]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {B[3]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {B[2]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {B[1]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {B[0]}]

set\_property PACKAGE\_PIN T18 [get\_ports {sel[1]}]

set\_property PACKAGE\_PIN U17 [get\_ports {sel[0]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {sel[1]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {sel[0]}]

set\_property PACKAGE\_PIN W19 [get\_ports semn1]

set\_property PACKAGE\_PIN T17 [get\_ports semn2]

set\_property IOSTANDARD LVCMOS33 [get\_ports semn1]

set\_property IOSTANDARD LVCMOS33 [get\_ports semn2]

set\_property PACKAGE\_PIN L1 [get\_ports semn\_out]

set\_property IOSTANDARD LVCMOS33 [get\_ports semn\_out]

set\_property PACKAGE\_PIN W5 [get\_ports clk]

set\_property IOSTANDARD LVCMOS33 [get\_ports clk]

set\_property PACKAGE\_PIN W4 [get\_ports {anod[3]}]

set\_property PACKAGE\_PIN V4 [get\_ports {anod[2]}]

set\_property PACKAGE\_PIN U4 [get\_ports {anod[1]}]

set\_property PACKAGE\_PIN U2 [get\_ports {anod[0]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {anod[3]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {anod[2]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {anod[1]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {anod[0]}]

set\_property PACKAGE\_PIN U7 [get\_ports {catod[0]}]

set\_property PACKAGE\_PIN V5 [get\_ports {catod[1]}]

set\_property PACKAGE\_PIN U5 [get\_ports {catod[2]}]

set\_property PACKAGE\_PIN V8 [get\_ports {catod[3]}]

set\_property PACKAGE\_PIN U8 [get\_ports {catod[4]}]

set\_property PACKAGE\_PIN W6 [get\_ports {catod[5]}]

set\_property PACKAGE\_PIN W7 [get\_ports {catod[6]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {catod[6]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {catod[5]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {catod[4]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {catod[3]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {catod[2]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {catod[1]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {catod[0]}]



This program works in this way: we choose the numbers with the help of the switches. Then with the help of the switches we establish the operation. If we don’t press any of the selections it will automatically add the numbers and display them on the fpga. If we press sel0 it makes the subtraction of the two numbers. If we press sel1 it makes the multiplication and if we press both of the buttons it makes the division. We must remember that the numbers are represented in hexadecimal so if we see on the display 20 it means 16 in decimal. If we want negative numbers we must press the buttons for semn1 or semn2 and if the led for the semn\_out start it means we have a negative result.

# 4.Technical justifications for the design

I selected this approach because, after analyzing all possible options, it proved to be the most efficient and straightforward to implement. From the beginning, our implementation plan aimed to find an optimal solution and break the project into smaller parts. This approach made simulations easier and facilitated the detection of potential errors.

Initially, I considered representing numbers in Two's complement. However, I realized that using signed magnitude would be much more user-friendly. To accommodate this, I added two additional inputs for the signs of each number and another output for the sign of the result.

To optimize the implementation, I leveraged the fact that addition and subtraction can be computed using the same component by recognizing that A-B is equivalent to A+(-B).

In order to illustrate the functionality of the pocket computer system more effectively, I utilized six components (as described in main.vdh). This selection of components facilitated the binding process and ensured smooth integration.

I believe that the chosen solution is easily understandable and traceable, even for users with moderate experience. Furthermore, there is potential for further improvement to enable more complex operations to be performed.

# 5. Future developments

# One potential enhancement that could be implemented is to allow the input numbers to be in decimal format. Instead of requiring the user to input binary numbers using switches on the device, they can simply press buttons representing decimal digits. This modification would simplify the computing process for users, as it would eliminate the need for them to manually convert between decimal and binary representations. For instance, if the user wishes to enter the number 123, they would only need to press the buttons for 1, 2, and 3, rather than configuring the switches on the board to represent "01111011."

# Another optimization could involve incorporating additional outputs to display the results of each operation. To achieve this, a larger FPGA board could be employed. Furthermore, the size of the result obtained from addition could be larger than 8 bits. Consequently, there is room for further optimization in this regard. To enhance the user experience, a memory feature could be added to store previous results. This would enable users to review their previous calculations. Looking ahead, future improvements could involve creating an erase button that will make possible deleting the last digit introduced.