POCKET CALCULATOR

DSD first year project

JUNE 5, 2024

FLOREA DEBORAH RUXANDRA MARIA
PROJECT SUPERVISOR: Ing. Diana Pop

Table of Contents

Pocket Calculator

1 Specifications

This project follows the design of a pocket computer with basic arithmetic operations (addition, subtraction, multiplication and division). Multiplication and division operations are implemented using specific algorithms, not language operators.
The operands are represented by 8 bits with an additional 9th bit for the sign. Operands and operators will be entered sequentially in decimal form. The 7-segment displays on the FPGA boards will be used for displaying the numbers.

Algorithm for Multiplication

The algorithm used for multiplication is called Booth's multiplication algorithm, being invented by Andrew Donald Booth in 1950 while doing research on crystallography. This algorithm multiplies two signed binary numbers in two's complement notation (given that our numbers are positive, their representation in signed magnitude and two's complement coincide).

This algorithm can be described as follow:

If *x* is the number of bits of the multiplicand (in two's complement notation) and *y* is the number of bits of the multiplier (also in two's complement notation):

- Create a grid with 3 rows and *x* + *y* + 1 columns. Call the rows *A* (addition), *S* (subtraction), and *P* (product).
- Fill in the first *x* bits of each row with:
  - A: the multiplicand
  - S: the negative of the multiplicand
  - P: zeros
- Fill in the next *y* bits of each row with:
  - A: zeros
  - S: zeros
  - P: the multiplier
- Put zero at the last bit of each row.
- Repeat the procedure below *y* times:
  1. If the last two bits of the product are:
     - 00 or 11: do nothing.
     - 01: P = P + A. Ignore any overflow.
     - 10: P = P + S. Ignore any overflow.
  2. Shift *P* one bit to the right. In this step, the *P* signal must be preserved, that is, if the most significant bit is 1, then after shifting the new most significant bit must also be

1. If the most significant bit is 0, after shifting the new most significant bit should also be 0.

- Drop the least significant (rightmost) bit from *P* for the final result.

The technique presented above is inadequate when the multiplicand is a negative number longer than what can be represented by the positive equivalent value (i.e. if the multiplicand is 8 bits, then that value is -128). A possible fix for this problem is to add one more bit to the left of *A*, *S*, and *P*. this problem does not affect us, since for the multiplication we only use positive numbers (or the sign can be determined by the usual rules, if the signs of the input numbers differ the sign bit of the result is 1, otherwise 0).

Algorithm for Division

The algorithm used for division is called the restoring division algorithm. Initially, the dividend is loaded into the right half of the 2n-bit A register, and the divisor is loaded into the left half of the 2n-bit B register. The n-bit quotient register (Q) is set to 0, and the counter N is set to n+1. In order to determine whether the divisor is smaller than the partial remainder, the divisor register (B) is subtracted from the remainder register (A). If the result is negative, the next step is to restore the previous value by adding the divisor back to the remainder, generating a 0 in the Q0 position of the quotient register. This is the reason why this method is called restoring division. If the result is positive, a 1 is generated in the Q0 position of the quotient register. In the next step, the divisor is shifted to the right, aligning the divisor with the dividend for the next iteration. The first improvement that can be made is by shifting the partial remainder to the left instead of shifting the divisor to the right, thus producing the same alignment and simplifying the hardware necessary for the ALU and the divisor register. Both the divisor register and the ALU could have the size reduced to half (n bits instead of 2n bits). The second improvement comes from the fact that the first step of the algorithm cannot generate a digit of 1 in the quotient, because, in this case, the quotient would be too large for its register. By switching the order of the operations to shift and then to subtract, one iteration of the algorithm can be removed. Another observation is that the size of the A register could be reduced to half, and the A and Q registers could be combined, shifting the bits of the quotient into the A register instead of shifting in zeros as in the preceding algorithm. The A and Q registers are shifted left

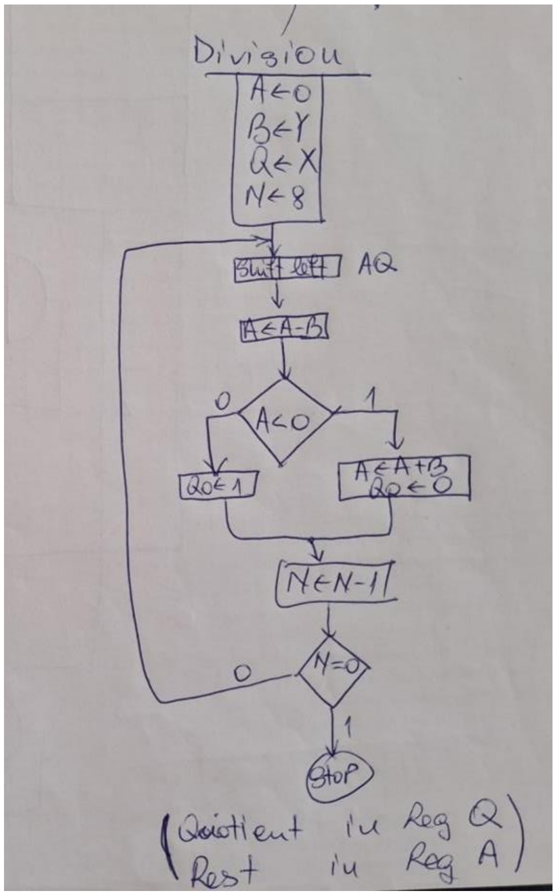


Figure 1: Flowcharts for Multiplication and Division algorithms

together. Consequently, no component will surpass the size of 8 bits, the quotient ending up in the register Q, while the rest will be found in the register A.

2 Design

2.1 Black Box

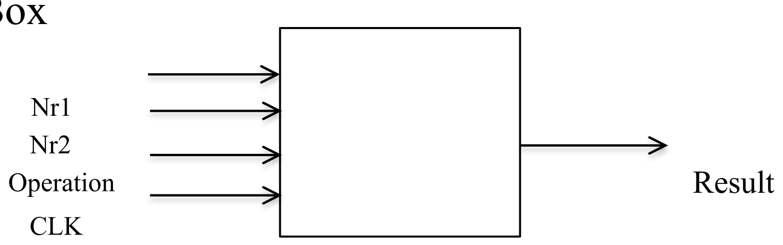


Figure 2: Black Box of the system

The numbers used in the calculation are on 8 bits, with an additional 9th bit for the sign. Therefore, their value will not surpass 255, -255 respectively, each of their decimal digits being displayed on one of the 7-segment displays, using the fourth one for the sign display. The result will then also be displayed on several 7-segment displays as a decimal number. A more detailed black box of the project, taking into account the available resources of the FPGA board will be having the following data input and outputs.

Inputs:
CLK: the clock of the FPGA board, which will be divided later using a frequency divider
Sw[0:2], Sw[3:5], Sw[6:8]: each trilogy of switches controls a counter for a digit, with the properties clear (first switch), enable (second switch) and hold (third switch)
Sw9: sign bit (off for positive numbers, on for negative ones)
Sw11: execute addition operation
Sw12: execute subtraction operation
Sw13: execute multiplication operation
Sw14: execute division operation
Sw15: execute equal operation

Outputs:
Anod: what digit to display
Catod: where to display it (on the 7-segment display)

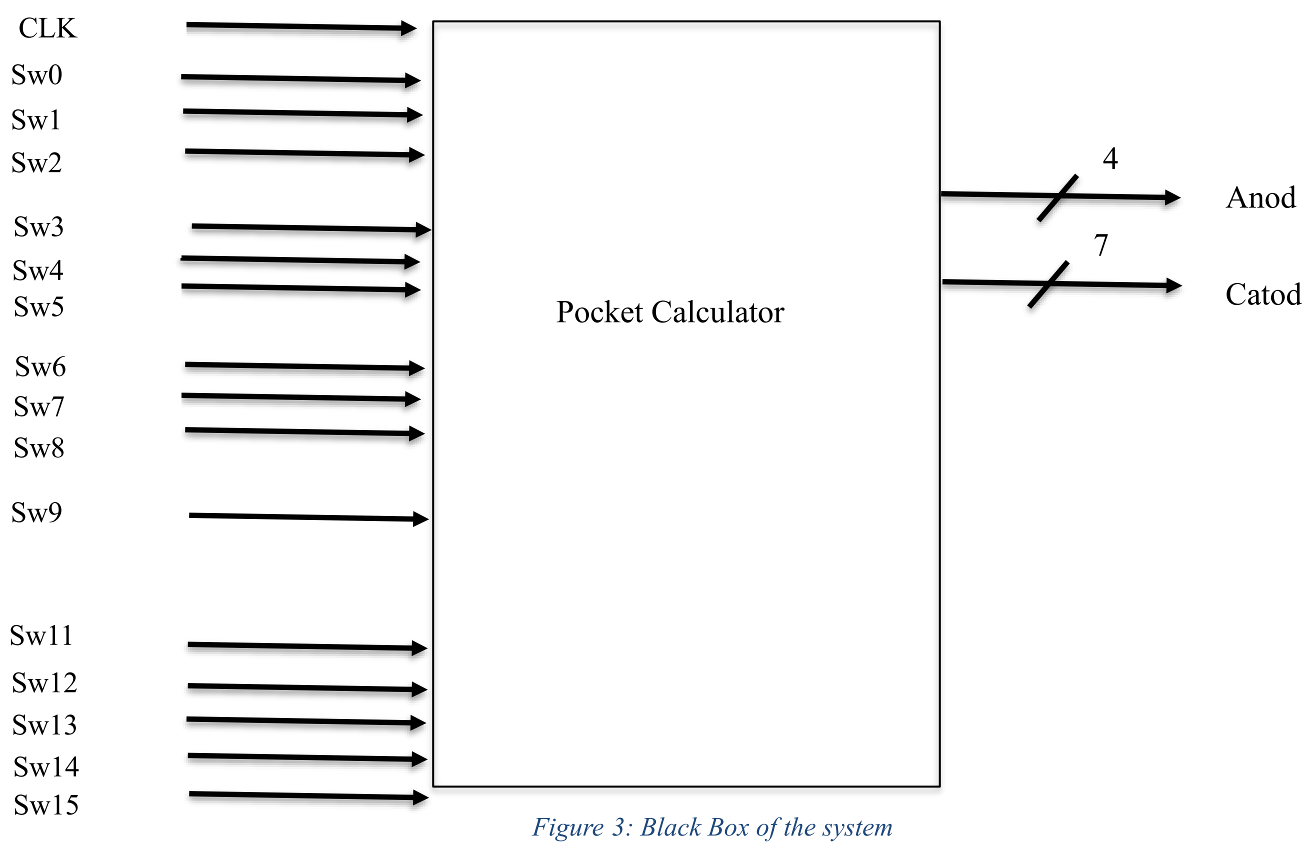


Figure 3: Black Box of the system according to FPGA resources

2.2 Control and Execution Unit

The project will be further broken down in the control logic of the system, represented by the Control Unit (CU) and the system resources, represented by the Execution Unit (EU).

2.2.1 Mapping the inputs and outputs of the black box on the two components

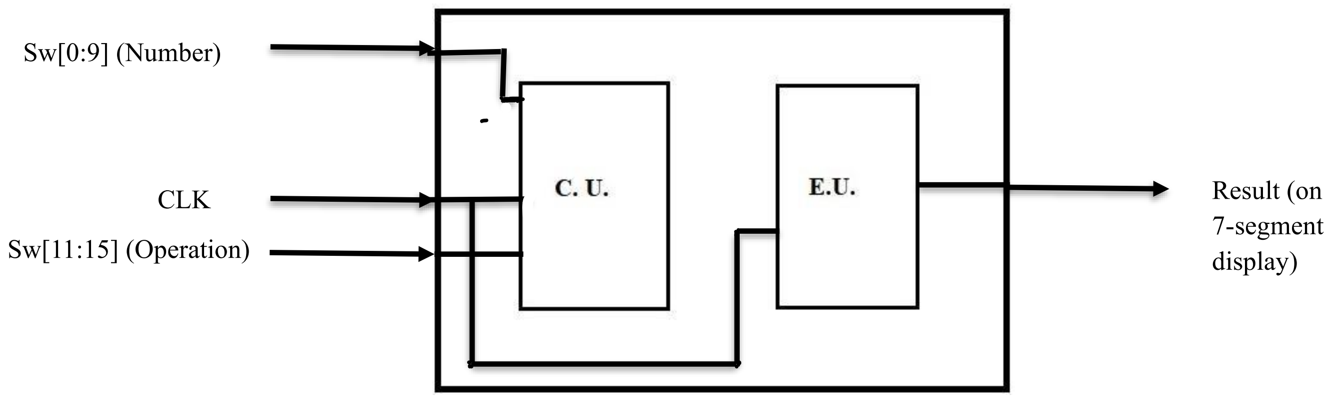


Figure 4: Mapping the inputs and outputs of the black box on the inputs and outputs of the units

2.2.2 Resources (breakdown of the Execution Unit)

RESOURCES

1. Counters
   - There are 3 types of counters implemented:
     - modulo 2 counter for the hundreds digit, which cannot surpass 2 (max number on 8 bits is 255)
     - decimal counter for the tens digit with an additional signal that enables when the hundred digit is 2, such that it will not surpass the number 5
     - decimal counter for the units digit with two additional signals, one similar to the previous tens counter, that enables when the hundred digit is 2, and one that enables when the tens digit is 5, these being the conditions needing to be fulfilled, such that the counter will not surpass the digit 5
2. Registers
   - Loads and holds the values of the numbers and result; gets cleared in case of cancellation. It was mostly created to make it easier when creating shift registers for the algorithms.
3. MUX 2:1
   - Decides if the digits displayed belong to the counter or the register A
4. MUX 8:1
   - There are 2 such components, one on 8 bits that decides on which anod to display the given digit and one on 4 bits that decides what to display
5. Segment decoder
   - Presents the way catods are being lighten up according to the given digit
6. Display
   - Displays the digits using catods on an anod at a time, by port mapping the previous 3 components mentioned above
7. Frequency divider
   - Allows the steps to be executed at a pace at which the user can give input
8. Transformation from number to digits and vice versa

- Allows the digits received from the counters to be transformed into a useable number and the number received from the result to be transformed in displayable digits

9. Multiplication component
   - Multiplies two 8-bit binary numbers using Booth's multiplication algorithm

10. Division component
    - Counter modulo 8 (only 8 (=size of numbers in bits) steps must be executed)
    - Subtractor on 8 bits (the borrow will be used as selection for the MUX)
    - MUX 2:1 (decides whether the value to be loaded will be the outputs of A or of the subtractor)
    - Shift left register

All the resources and their inputs/outputs can be seen on the EU drawing in the chapter 2.2.5.

2.2.3 Block Diagram for first breakdown

The order in which the operations of the project must be carried out is determined by the control unit (CU), which will enable or disable signals accordingly and send them to the execution unit to ensure the flow of the program. The signals that the units send amongst them are detailed in the figure below:

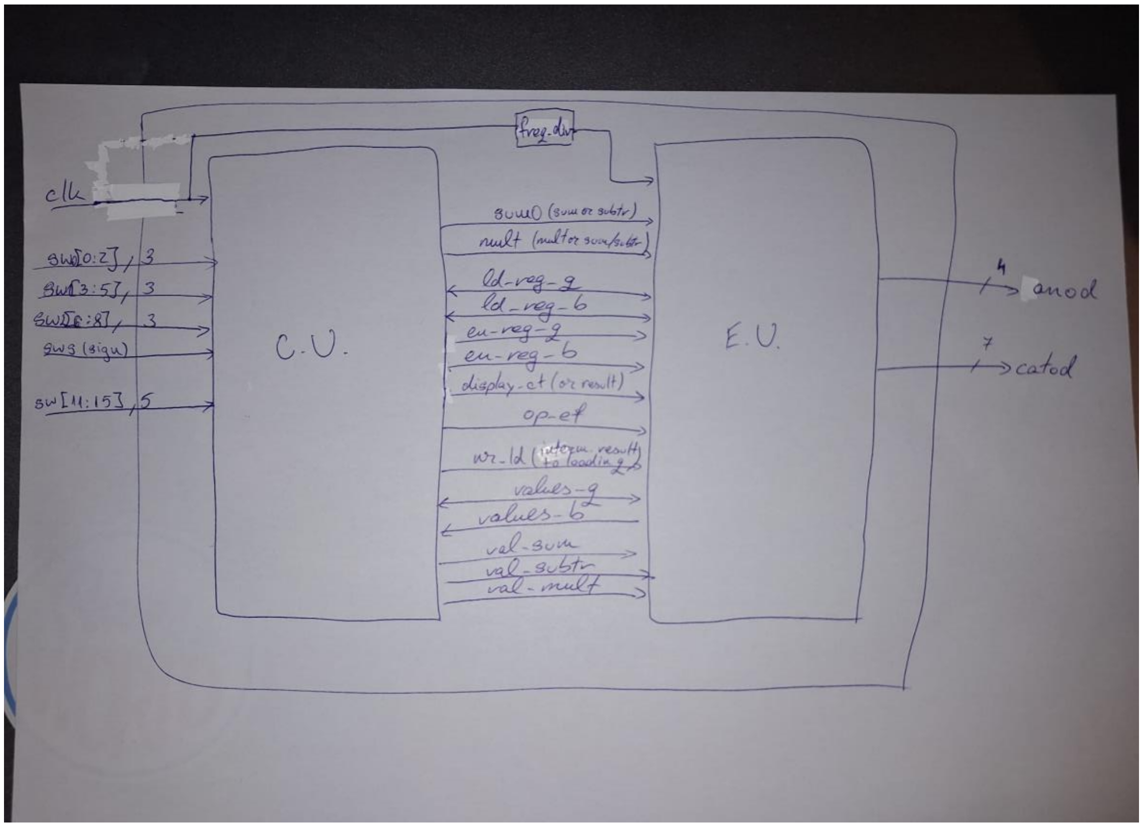


Figure 5: Mapping the connections found between the control and execution units

2.2.4 State diagram of the Control Unit

States:

A – set counters (the ones controlled by switches) and all operations to 0
B – enable counters and their properties (hold and clear)
C – load first number in register Q
D – choose operation
E – load second number in register B
F – calculate result
G – set everything to 0
H – display result and stop
I – display intermediate result and continue calculation J – set operations to 0

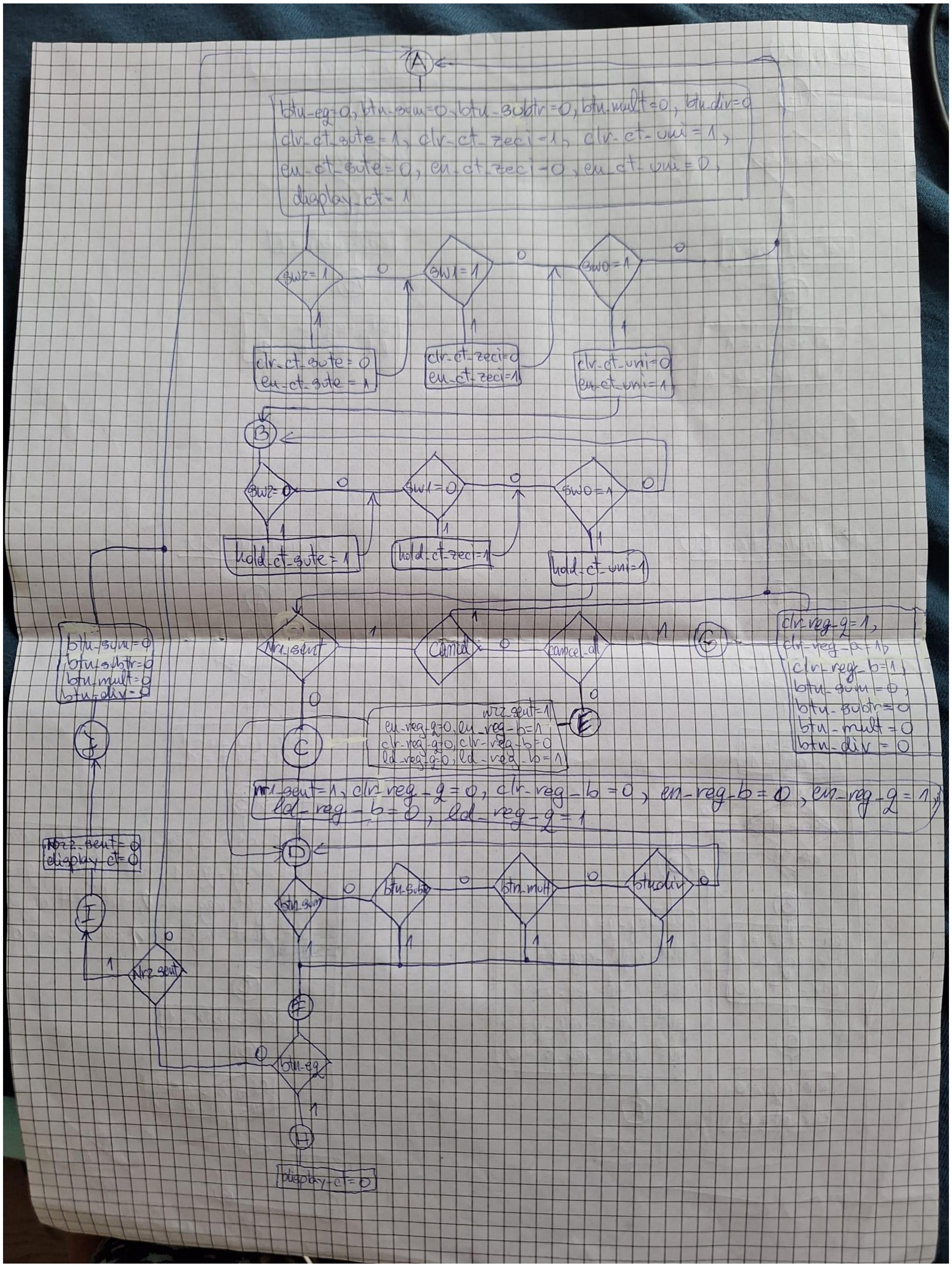


Figure 6: State diagram

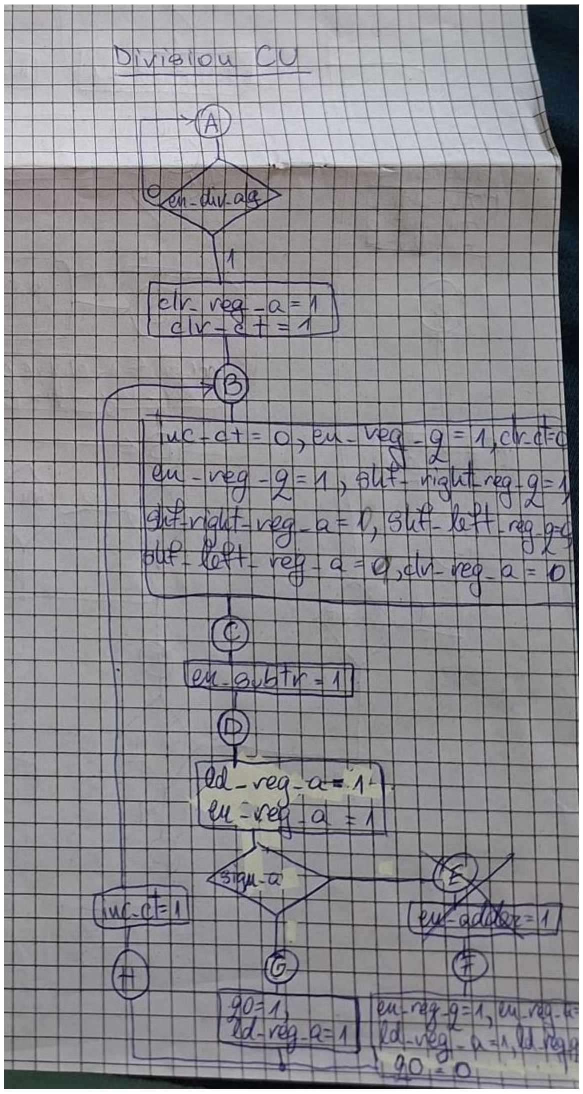


Figure 7: State diagrams of the individual operations

The state diagram of the division algorithm will be broken down into 3 processes:

- first process ensures that the current state is moved into the next state
- second process assigns to the next state the value according to the inputs
- third process assigns values to the outputs corresponding to the current state

Meanwhile the state diagram of the control unit is broken down in 2 processes: one that identifies the chosen operation and sets signals accordingly and another that calculates the needed result according to some signals.

2.2.5 Detailed diagram of the project

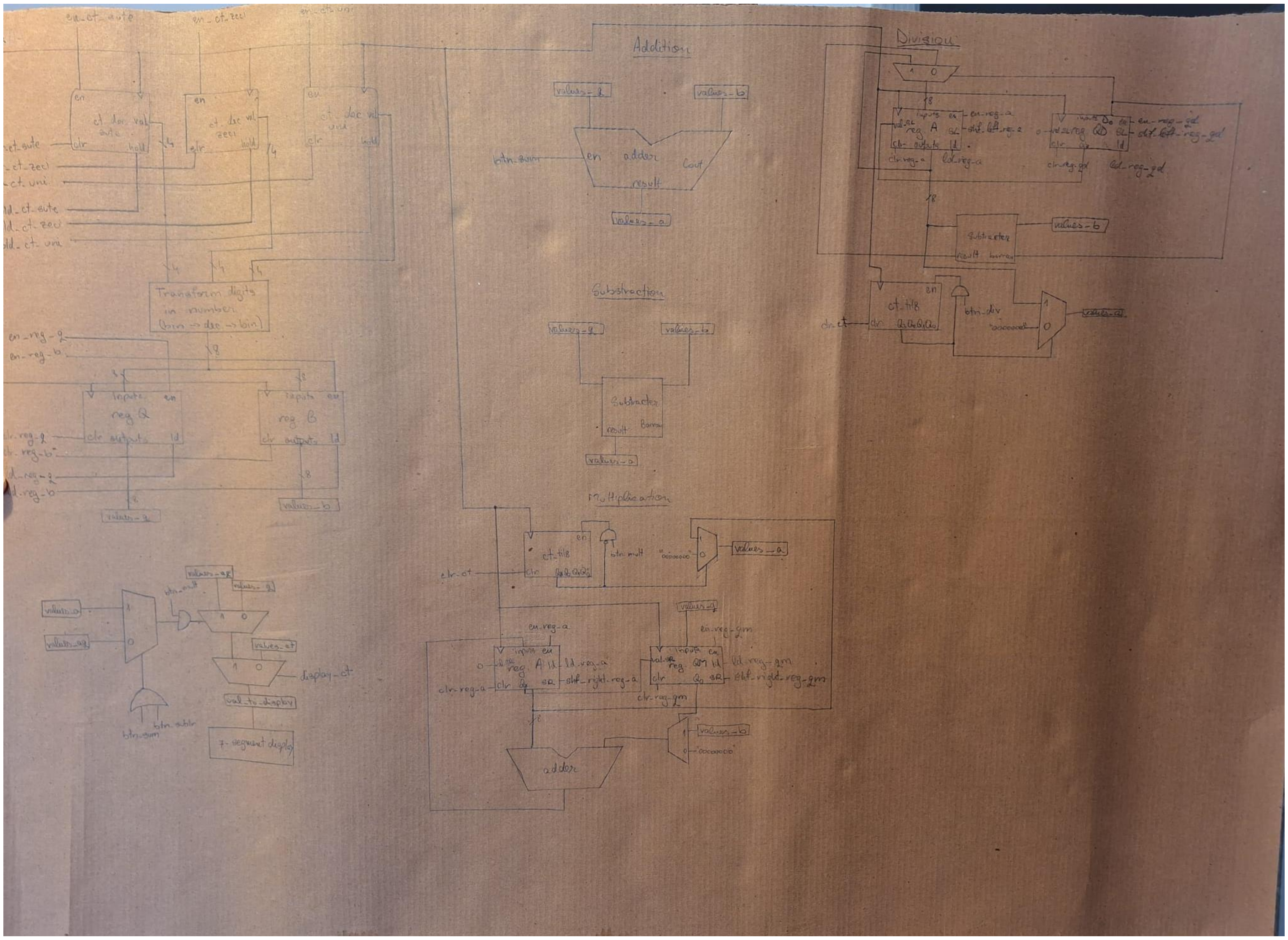


Figure 8: Execution Unit

As can be seen in the diagram, the more complex operations (multiplication and division) have been selected as individual parts of the project. This means that they have their own CU and EU and are being included in the EU of the bigger project using the "port map" instruction and a signal that enables each of them in the corresponding case.

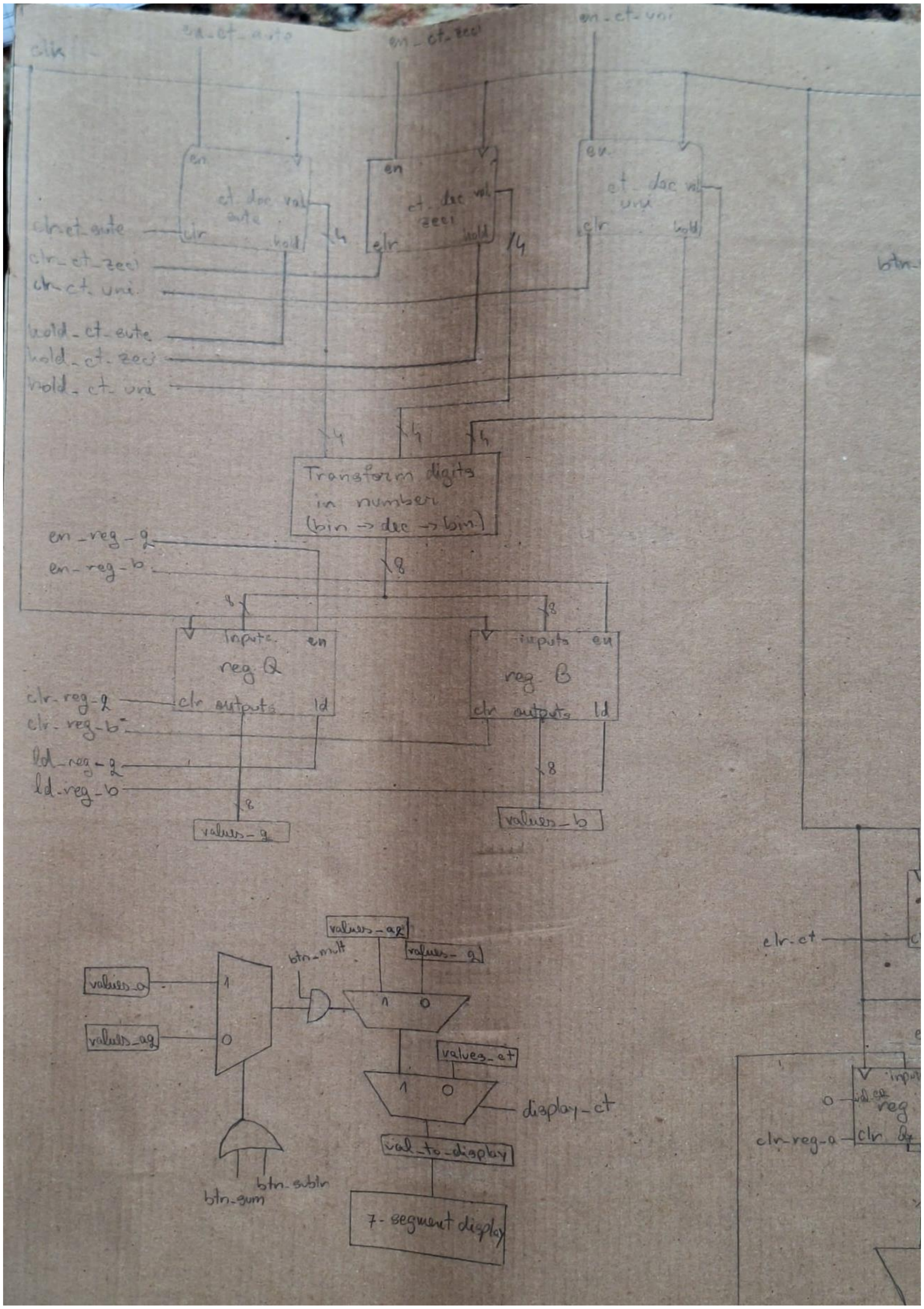


Figure 9: Execution Unit (Left)

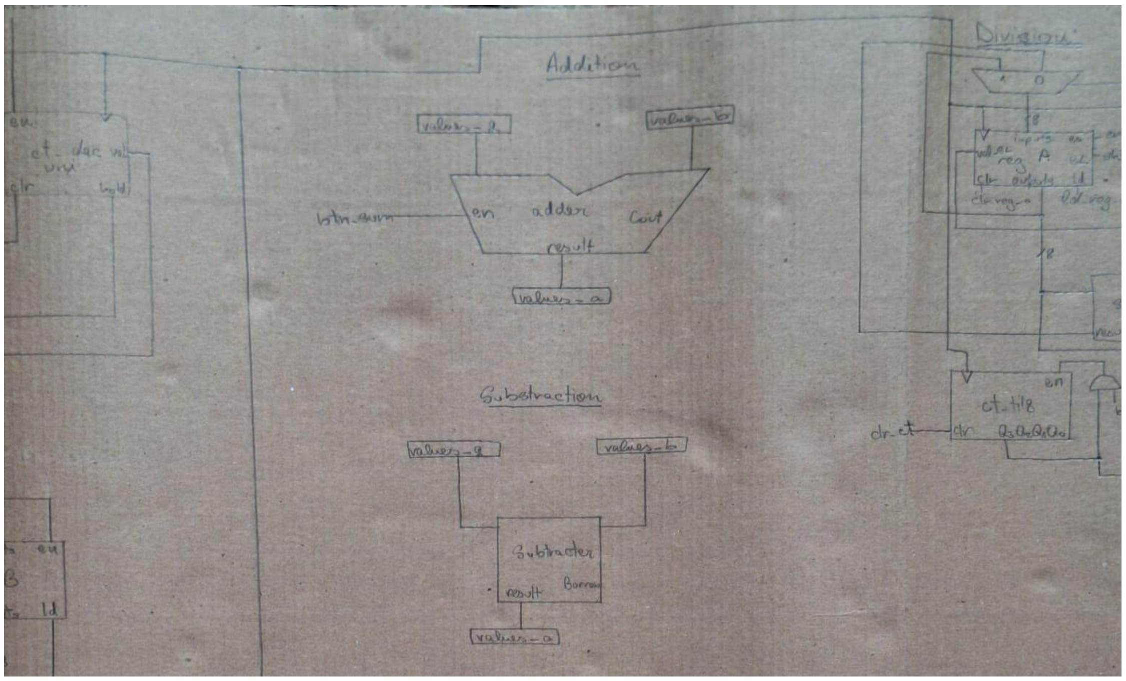
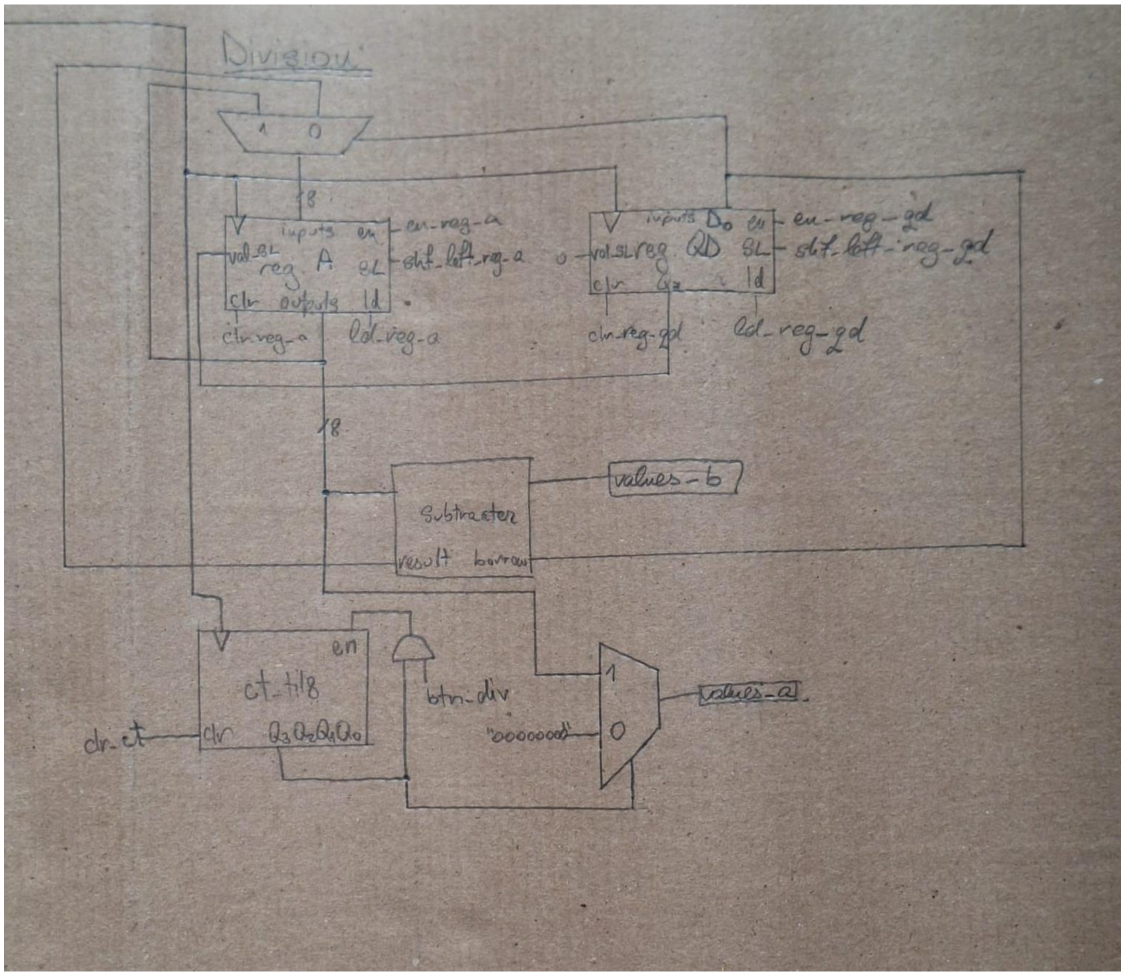
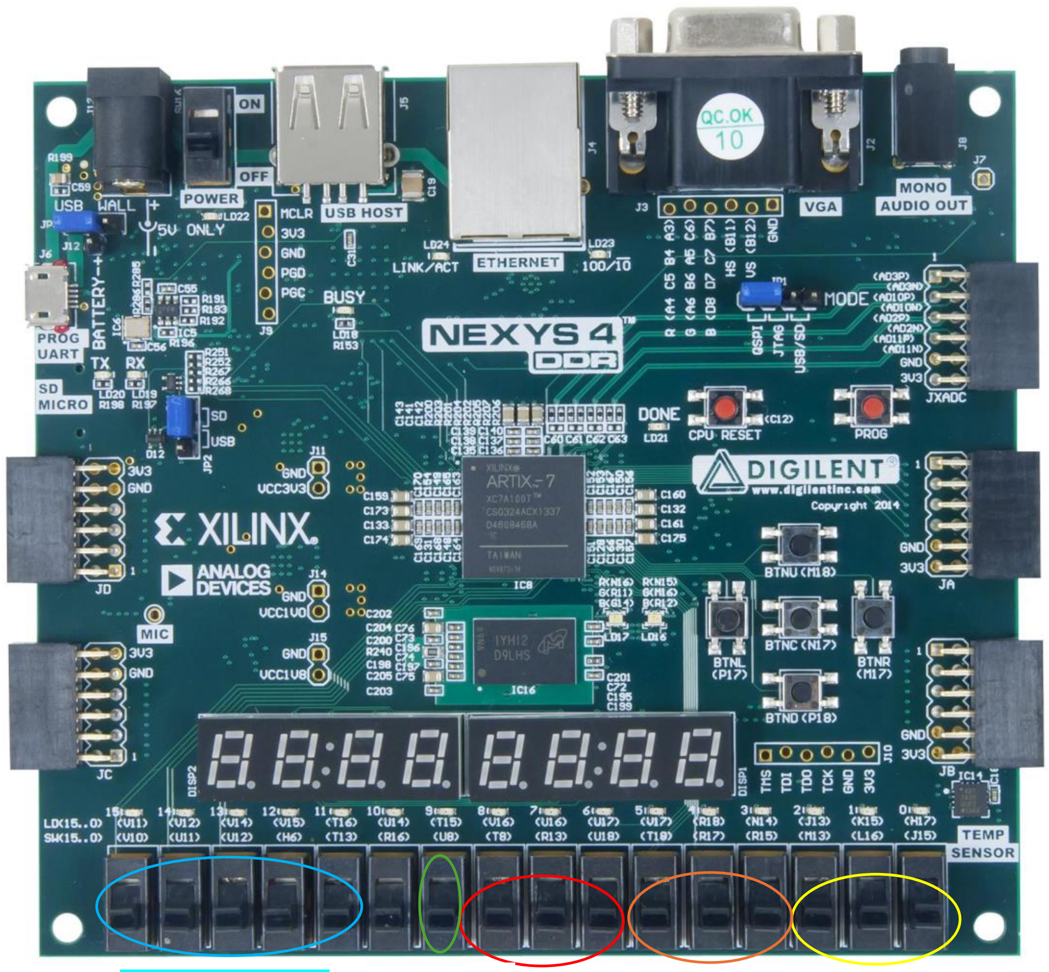


Figure 10&11: Execution Unit (Center and Right)



3 User manual



The user begins by turning on sw1, which enables the counter for the hundred digit and turning on sw2, which stops counting and holds the value when the desired digit has been reached. The hundreds digit must be chosen first such that the tens digit can be conditioned to not surpass the number 5 (it must be taken into consideration that the largest number on 8 bits is 255). Although the digit displayed by the counter does surpass the limit (number 3 also gets displayed), the user will not be able to choose it. After the hundreds digit was chosen, the tens digit counter must be enabled using sw5 and held to the desired position with sw6. This conditions the units digit, which will be chosen lastly using sw8 and sw9 and will not surpass 5 if the hundreds digit is 2 and the tens digit is 5. Does the user want to change one of the digits to 0 or restart the counting while at a certain step, it can use the sw0, sw3 and sw6 to clear the counters. When sw2, sw5 and sw8 are all simultaneously turned on, the digits represented by each counter will be fetched and transformed into the first operand of the following operation.

The next step will be for the user to turn on one of the switches signifying the possible operations to confirm they have entered the first number and wish to start the calculation.

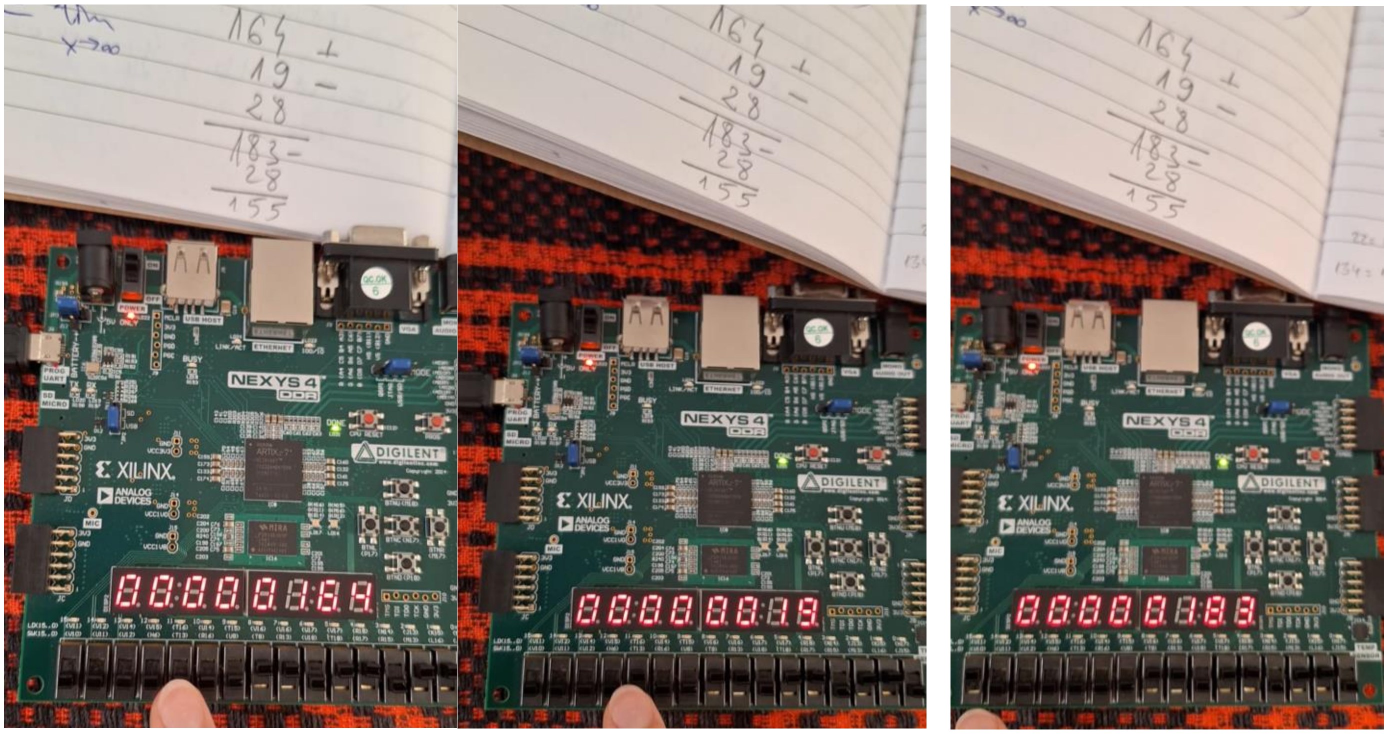


Figure 12&13&14: 1st number input (plus confirmation) and 2nd number input (plus operation) and result

After that the user is expected to give another number as input following the same steps as for the first number input. After having both numbers given as input, the user is expected to choose the preferred operation. To view the expected result of the chosen operation, the user must turn on the 15th switch, which stands for the equals operation. After turning the switch off again, the user can continue to input numbers and operations and view their results using the last switch.

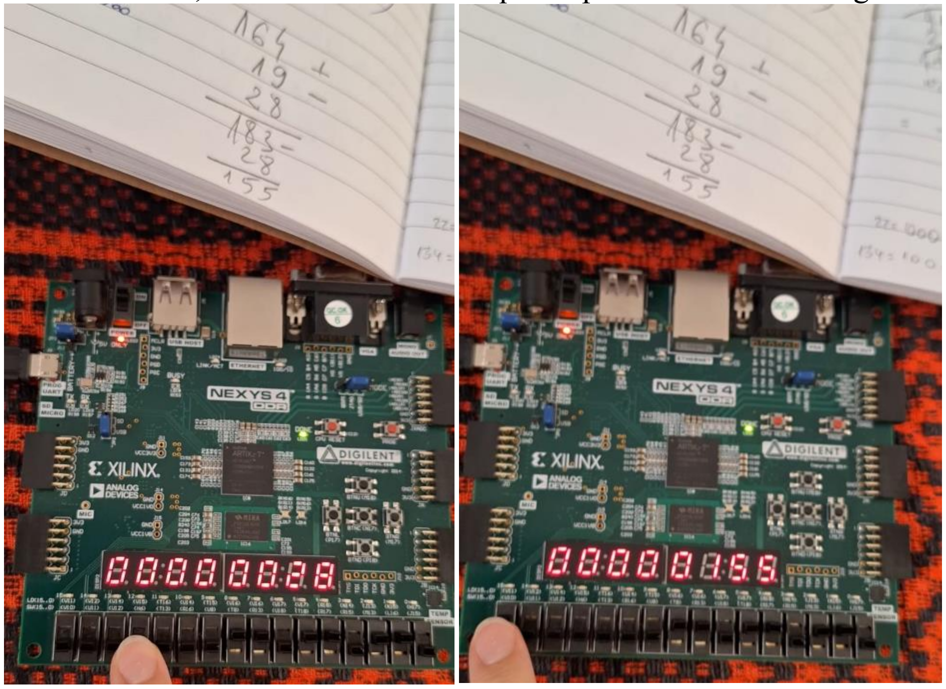


Figure 15&16: 3rd number input (plus operation) and result

4 Technical justifications for the design

I have chosen to separate the operations from the entire system because I found that to be the easiest way to handle the complex structure of the project. Having individual control and execution units, the algorithms are easier to handle and can be executed only when needed. Choosing the switches in reverse order of the number's digits (hundreds digit is controlled by the first switches not the last) is also an indirect constraint for the user to choose a number that will not surpass the 8-bit maximum which is 255. Will the user not conform to the chosen norms, then the results will also be incorrect.

5 Future developments

The most obvious improvement that could be implemented is the use of numbers bigger than 8 bits. As long as the result can be displayed on the 8 7-segment displays, there should be no problem in just increasing the size of the components. The *generic* instruction can be used to create components of arbitrary size to generalize the calculator for any numbers given as input. Another possible future development could be to move the functionality of the equal operation from the button to a switch such that the button could be used as inserting a bracket. This would mean storing the previous values in several registers. Maybe this could be done by using the *generate* instruction and creating a register for the numbers that are progressively being given as input and then executing the calculation in reverse order.

6 References

Octavian Creţ, Lucia Văcariu – Probleme de proiectare logică a sistemelor numerice. Logic Design Problems for Digital Systems. Editura UTPres, Cluj-Napoca, ROMÂNIA, 2008, ISBN 978-973-662-412-4, 258 pagini. [Editura UTPres este recunoscută CNCSIS, având codul 161]

Octavian Creţ, Lucia Văcariu, Aurel Neţin – Limbajul VHDL. Îndrumător de laborator. Editura UTPres, Cluj-Napoca, ROMÂNIA, 2002, ISBN 973-8335-38-8, 227 pagini. [Editura UTPres este recunoscută CNCSIS, având codul 161]

Baruch, Z. F., *Structure of Computer Systems*, U.T.PRES, Cluj-Napoca, 2002, ISBN 973-833544-2.

https://en.wikipedia.org/wiki/Booth%27s_multiplication_algorithm

https://github.com/gustavohb/booth-multiplier/tree/master