TECHNICAL UNIVERSITY OF CLUJ-NAPOCA

Faculty of Automation and Computer Science, Computer Science Department

Documentation for Digital System Design project

- Pocket Calculator -

Student: Stânea Georgiana Grațiela, 1st year, group 30414

Coordinating teacher: Maier Noema - Laura

Table of contents

1. Specifications and general information

The aim of this project is to compute simple operations like addition, subtraction, multiplication and division. The numbers are in the range [-255, 255], 8 bits denoting the magnitude and one more being for the sign (for both the inputs and the result).

The project is implemented using the hardware description language VHDL.

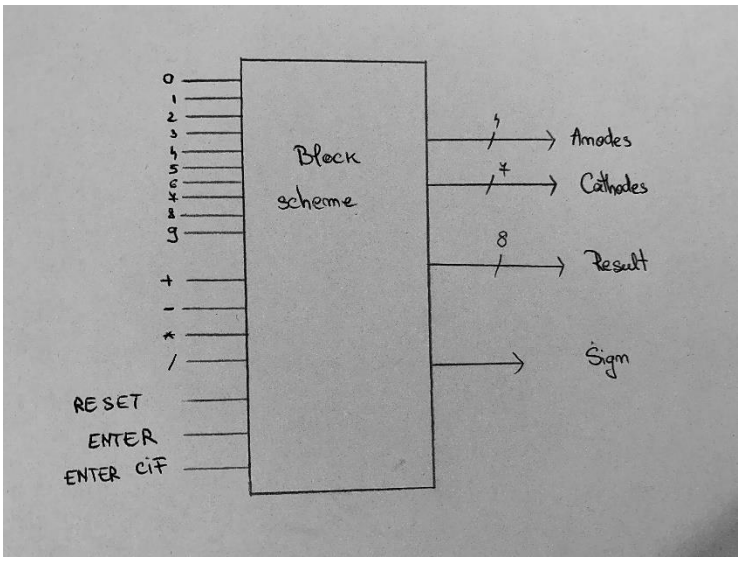
The request:

A10) Sa se proiecteze un calculator de buzunar cu operatii aritmetice fundamentale (adunare, scadere, inmultire, impartire). Operatiile de inmultire si impartire se vor implementa folosind algoritmi specifici, nu operatorii limbajului. Operanzii sunt reprezentati pe 8 biti cu semn. Operanzii si operatorii vor fi introdusi secvential in forma zecimala. Se vor folosi afisajele cu 7 segmente de pe placutele cu FPGA. Proiectul va fi realizat de 1 student.

2. The project description

In order to implement the project, first I needed to draw some schematics of the most important components. Thus, we have:

I) The block scheme



Here you can see the inputs and outputs.

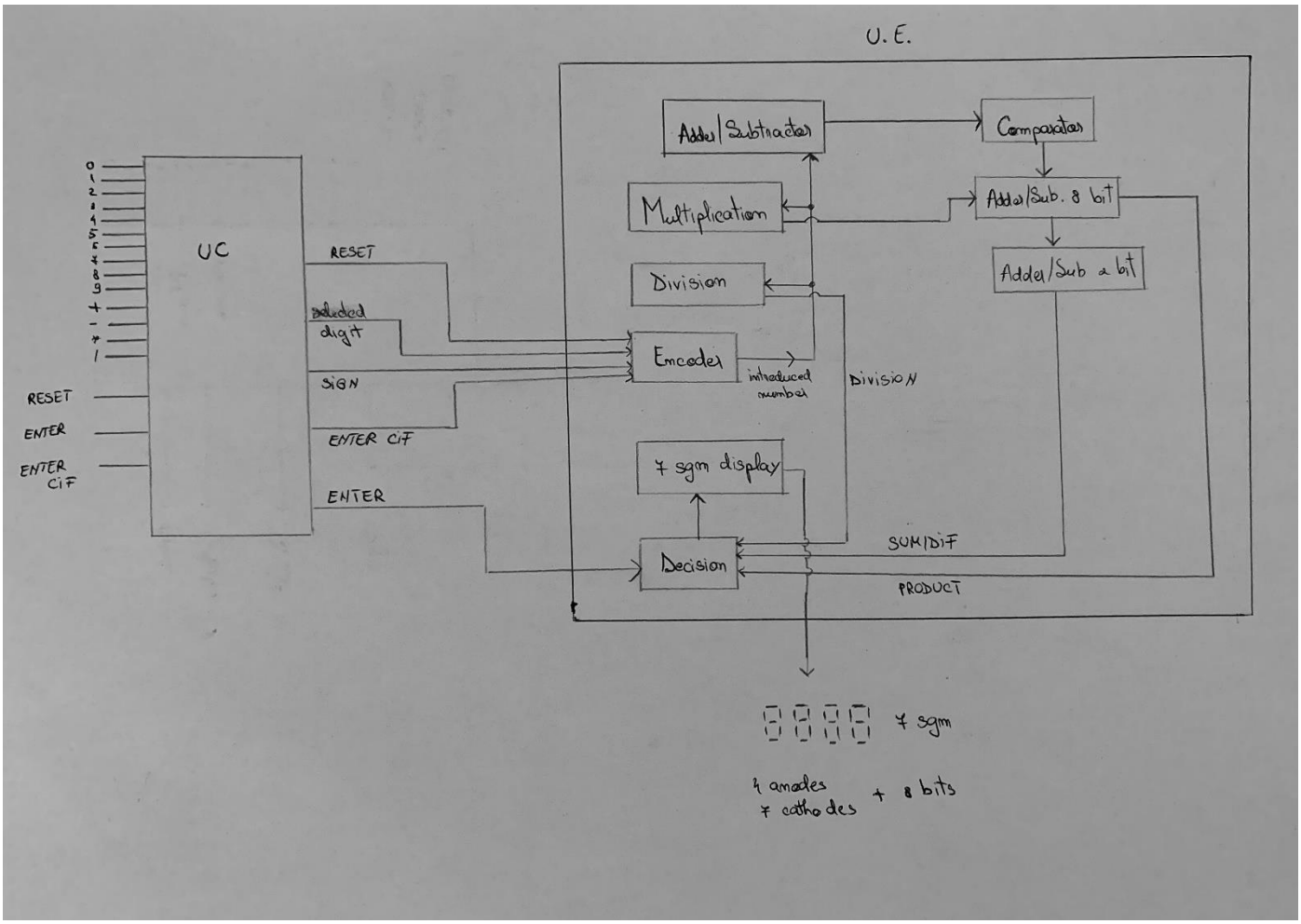
The inputs are:

- 0,1,2,3,4,5,6,7,8 : representing the digits which the user will introduce;
- **RESET** : will initialize the numbers with 0;
- +,-,*,/ : the operands
- **ENTER** : displays the result
- **ENTER_CIF/NEXT** : confirms an introduced digit and the follow-up of another operand.
- For the implementation on the FPGA board another input will be needed: the clock signal (CLK)

The outputs are represented by 4 anodes and 7 cathodes that will blink one by one at high frequency so that the human eye can observe the difference. Moreover, we will introduce 8 outputs for the 8-bit number that is calculated, 4 outputs for the remainder and one for the sign.

II) The Command and Execution units

In the following picture you can see the Command Unit, the Execution Unit and the main components:



The Command Unit is given by the inputs that were specified at the previous point and take part at the computation of the final result. The Execution Unit is made of the main components (you can also see some secondary components in the picture) which are connected between them (they depend on each other) and the decisions are made with respect to the inputs.

3. The main components

a) Adder/Subtractor (8 bits + Sign)

It represents the main component and is designed totally structural. This adds or subtracts two 8-bit numbers created using the digits. The result will be on 8 bits + the sign bit.

b) Multiplication (8 bits+ sign)

This is the main component that multiplies two 8-bit numbers. The result will be on 8 bits (due to the request) so, the maximum operation is 15*15=255 + the sign.

c) Division

This is the main component that divides two 8-bit numbers. The result will be on 8 bits for de quotient, 4 bits for the remainder and one bit for the sign. There are some forbidden operations like : 0/0, X/0, where X- integer from [-255,255].

d) Encoder

This is the main component that converts the introduced number to binary. The maximum number of digits introduced is 3 for each number, then the next number is required (successive operations), because the selection interval is [-255,255]. The result is calculated and can be later modified if another digit is introduced.

e) Decision

This is the component that gives the final result by checking the **ENTER** switch.

f) 7sgm (7 segment)

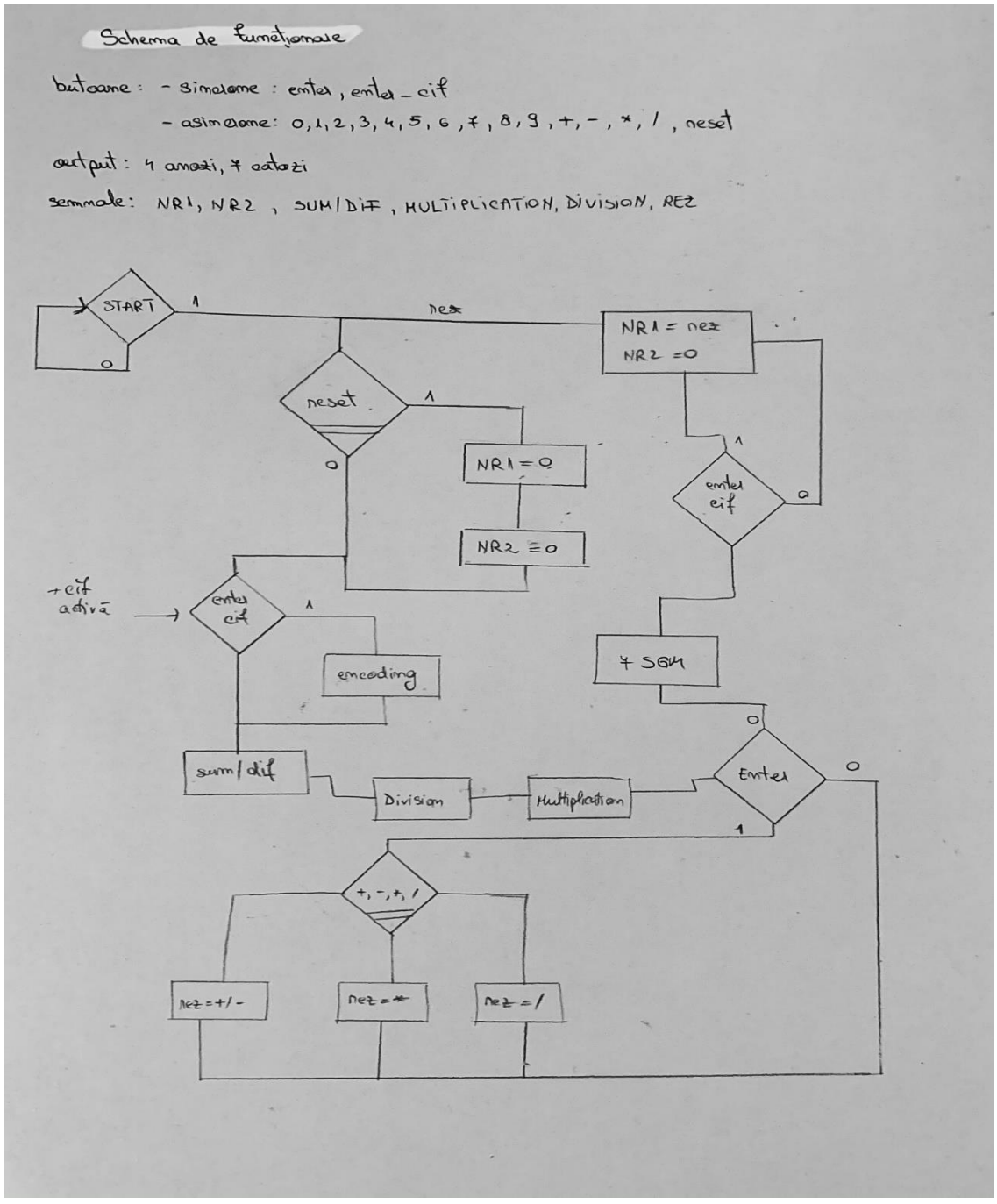
This is the component that decodes the final result and displays it on the 7-segment display of the board. The number is divided by 10 so that we obtain every decimal. According to their order and value, they are shown on the display.

g) The main component (main)

This is the final component where all the other components are linked together. It is actually the Execution Unit.

4. Steps for implementation

In the following picture is presented the general schematic and ideas behind this project (flowchart):



The schematic presents a cyclic fuctionality of the circuit. The algorithm repeats until some input is actived (for example: **ENTER** and **ENTER_CIF** => the content is modified/displayed).

The decision **START** represents the power-up of the board. When this switch is turned, the algorithm starts. The first thing that we need to check is the **RESET** switch. If it is active, the numbers recieve the value 0; if not, the last values are kept. The decision **ENTER_CIF** represents the confirmation of a digit followed by the construction of the number with that digit. If there is no impulse on the **ENTER_CIF** input the computation is done with the existing values of **NR1** and **NR2**. The value of each computation is stored into a specific signal. Next, we check if the **ENTER** input is active. If it is active, we check which of the operations is selected:

- „+" is active: the final result will recieve the value of the signal Addition/Subtraction;
- „*" is active : the final result will recieve the value of the signal Multiplication
- „/" is active : the final result will recieve the value of the signal Division

This process is done in the „Decision"component.

If the input **ENTER** has the value ‚0' the system skips this part and goes directly to displaying the result on 7 segments and the extra outputs (on bits). If there is no change on the inputs, the display will show the same value as before (at the beginning it is 0).

After this procedure, we check again whether the inputs **ENTER_CIF** and **ENTER** are active (represented by the decision enter,cif on the right side of the schematic). After the result is displayed due to the activation of the **ENTER** switch, the **ENTER_CIF** switch is turned too. If both the conditions are true, the computation continues: NR1 takes the value of the result and NR2 takes the value ‚0'. By doing this we solve the problem of succesive operations. Lastly, the process is resumed at the activation of the **RESET** switch.

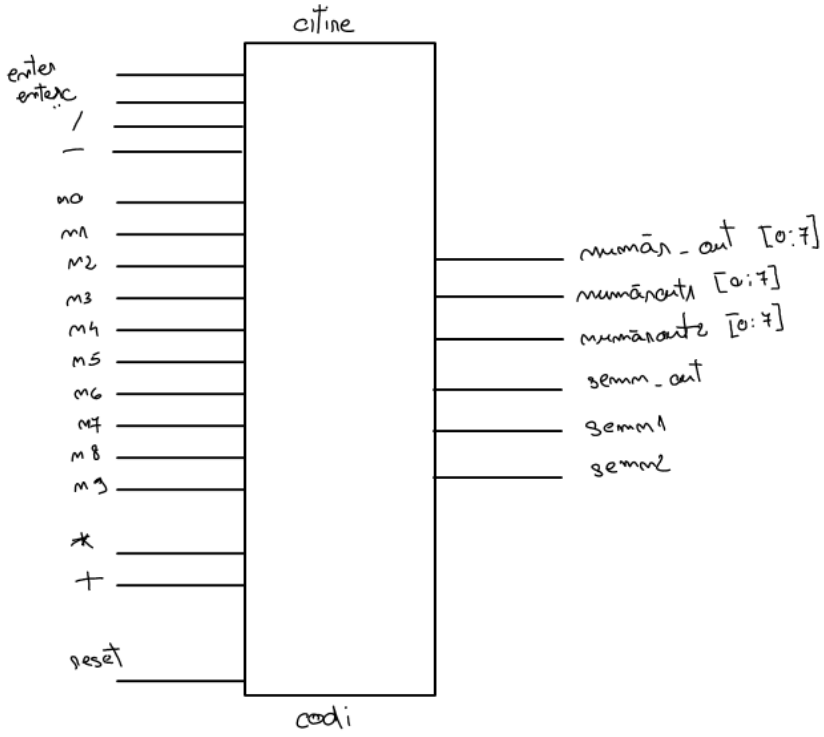
This entire process takes place at high speed; the events repeat themselves and are computed using the values determined before. If there is no impulse applied, the schematic from above is followed until a condition becomes true and the result is modified.

That is the general algorithm and the steps followed in order to do this project.

The main components used in this project are:

a) Encoder

The general schematic:



The **inputs** are represented by switches: digits from 0 to 9 (n0 -> n9),+ , - , * , / **, RESET**, **ENTER** and **ENTERC** (enter_cif).

The **outputs** are: the final result **NUMAR_OUT**, the numbers for the computation of the result **NUMAROUT2** and **NUMAROUT1** on 8 bits, the sign of the result – **SEMN_OUT**; and the signs of the two numbers: **SEMN2** and **SEMN1**; all on one bit.

This component was mostly described behaviorally. Its role is to construct the numbers and to continue the computation after a result is displayed. We have to pay attention at

the results **NUMAROUT1** and **NUMAROUT2** because they are in reversed order. In the following components we will take care of this problem so the numbers are correct.

The geneal algorithm used to construct the numbers consists of muliplying the current number by 10 and then adding the value of the digit active after turning on the switch **ENTERC**. First we check if the **RESET** switch is active. If so, the numbers are initialized with ‚0' (first the variables v1 and v2 for the process, then the signals and numbers). If we want to introduce first a negative number we check whether the sign **MINUS** is active. Next, we check if **ENTER** and **ENTER**C (enter_cif) are both active in order to compute successive operations.

Now, let's see how we construct **NUMAROUT1** and **NUMAROUR2**:

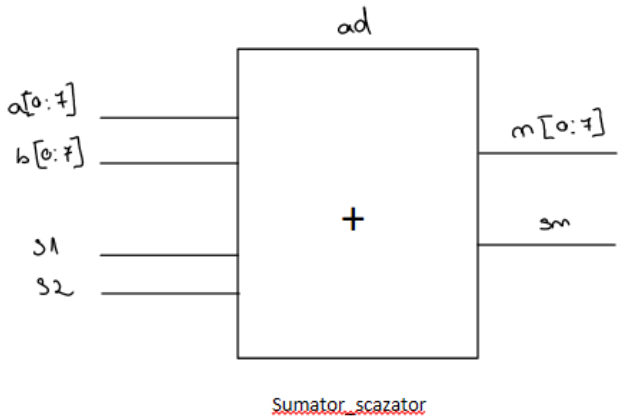
If an operand is active (+,-,*,/) and the first number introduced is greater than 0, that means we want to introduce another number for the computation. The new number is added 10 times in the variable v2 and then we add the digit that was activated. This process repeats if another digit is pressed and the switch **ENTERC** is active.

If an operand is not active it means we want to construct the first number. The algorithm is the same as the one mentioned above, except that the number is constructed in v1.

The procedure is repeated until we reach the maximum capacity of 8 bits. At the end, the signals for the outputs take the values of the variables.

b) Adder/Subtractor

The general schematic:



Sumator_scazator

The inputs for the component are the two 8-bit numbers and their sign (on one bit).

The outputs are represented by the result on 8 bits (the sum/difference of the two numbers) and its sign.

This component is designed using structural description by mapping 1-bit adders. In order to design this component I used thhe following algorithm:

The sum or difference will be computed depending on their sign and magnitude. For 2 numbers A and be we have:

- If A>0 and B>0 => addition, the sign is plus
- If A<0 and B<0 => addition, the sign is minus
- If A>0 and B<0 or if A<0 and B>0 => subtraction, the sign will be the same as the sign of the biggest number between A and B;

In the following image you can see the final functions written down and their demonstration.

Also we have the truth table from which the procedure resulted. On the second column we have the values after comparing the two numbers: If A>B => ‚1' else ‚0'. On the next two columns we have the sign for each number, followed by the new values of A and B.

In the written code, we do a mapping of adders/subtractors on 2 bits using the functions described in the component „sumator". In this new component we implement the functions for Va,Vb, COUT, Sum/Dif, with respect to the computed formulas.

	CMP	Sgn A	Sgn B	V_A	V_B	Sgn	example
A < B	0	(+A) 0	(+B) 0	A	B	(+) 0	4+7
A < B	0	(+A) 0	(-B) 1	A	-B	(-) 1	4-7
A < B	0	(-A) 1	(+B) 0	A	-B	(+) 0	-4+7
A < B	0	(-A) 1	(-B) 1	A	B	(-) 1	-4-7
A > B	1	(+A) 0	(+B) 0	A	B	(+) 0	7+4
A > B	1	(+A) 0	(-B) 1	-A	B	(+) 0	7-4
A > B	1	(-A) 1	(+B) 0	-A	B	(-) 1	-7+4
A > B	1	(-A) 1	(-B) 1	A	B	(-) 1	-7-4

V_A - value of A

V_B - value of B

Sum/Difference = $A \oplus B \oplus CIN$ (or BORROW)

$$V_A = [(\overline{CMP} + Sgn A \odot Sgn B) \cdot A + CMP \cdot \overline{A} \cdot (Sgn A \oplus Sgn B)]$$

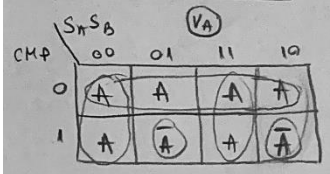
$$V_B = [(CMP + Sgn A \odot Sgn B) \cdot B + \overline{CMP}\ \overline{B} (Sgn A \oplus Sgn B)]$$

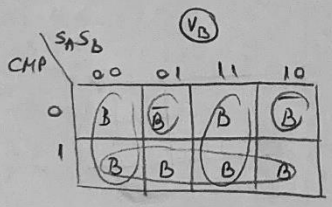
$$\Rightarrow COUT = V_A \cdot V_B + V_A \cdot CIN + V_B \cdot CIN$$

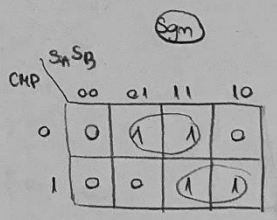
(or BORROW)

$$Sgn = \overline{CMP} \cdot Sgn B + CMP \cdot Sgn A$$

dem. for functions:







$$V_A = \overline{CMP} + \bar{S}_A \bar{S}_B + S_A S_B + CMP \cdot \bar{S}_A S_B \cdot \bar{A} + CMP S_A \bar{S}_B \cdot \bar{A}$$

$$= \overline{CMP} + S_A \odot S_B + CMP (S_A \oplus S_B) \cdot \bar{A}$$

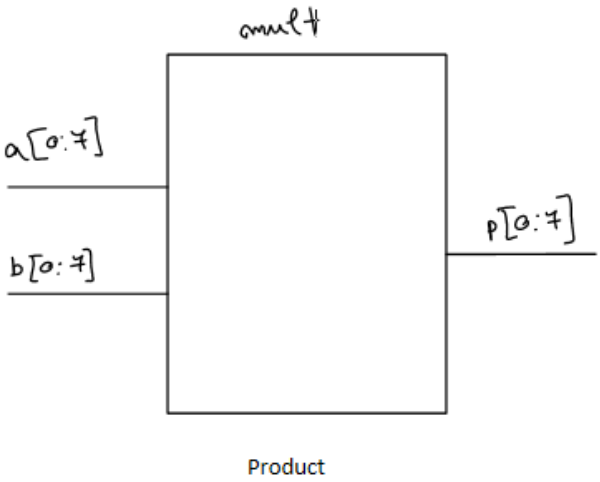
$$V_B = CMP + \bar{S}_A \bar{S}_B + S_A \cdot S_B + \overline{CMP} \cdot \bar{S}_A S_B \cdot \bar{B} + \overline{CMP} \cdot S_A \cdot \bar{S}_B \cdot \bar{B}$$

$$= CMP + S_A \odot S_B + \overline{CMP} (S_A \oplus S_B) \cdot \bar{B}$$

$$Sgn = \overline{CMP} \cdot S_B + CMP \cdot S_A$$

c) Multiplication

The general schematic is:

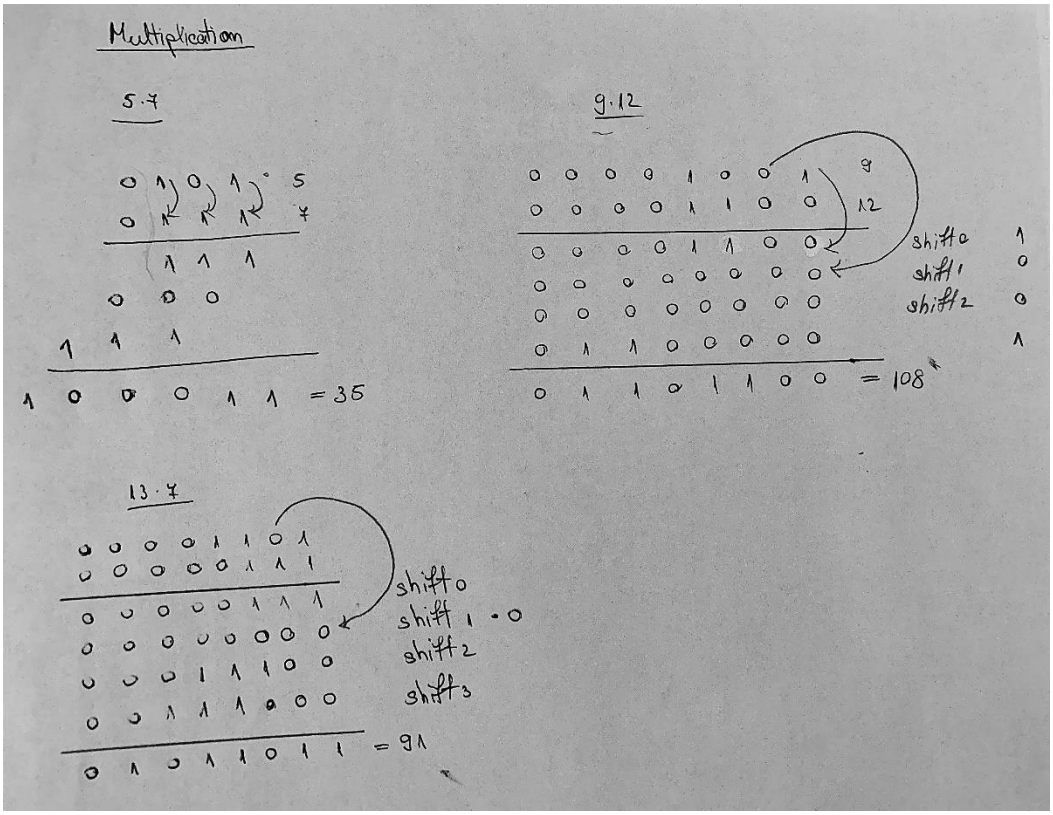


Product

The inputs are the two 2-bit numbers. The output is their product.

We compute the multiplication using the repeated addition method.

For A*B we will add B A-times, depending on A's bits. The rightmost digit of A is multiplied by B and the result is placed for the addition. We go to the next digit and also multiply it by B, but this time we do a left shifting by one bit. This procedure repeats 8 times (beacause we have 8 bits). An example is presented below:




```
A       00010011* (19)
B       00001011  (11)
        --------
1*      00001011 (B shift stanga de 0 ori)
1*      00010110 (B shift stanga 1 data)
0*      00000000 (B al 3-lea bit din stanga al lui A e 0 deci numarul final e
0*      00000000 (B shift de 3 ori)    inmultit cu 0, fiind 0, shift de 2 ori)
1*      10110000 (B shift de 4 ori)
0*        etc
|-----------------------------
        11010001
```

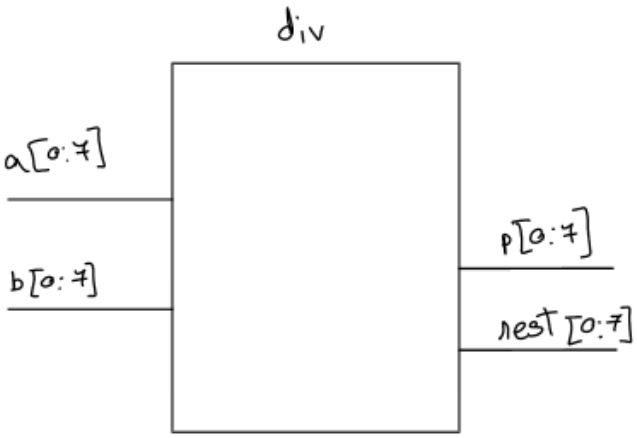
First in this code, every digit of A was checked for deciding how many non-negative numbers will be added. If the binary digit of A is „1” then the number will be B, else ‚0’. Then we do left shiftings depending every digit of A. The rightmost assignment will be a(7) and will remain unchanged. For a(6) it will shift left by one unit. For shifting, the number will be added to itself. For a(5), the current B will be added or two times 0 with the previous value. Ex: 3 becomes 6 then 12 : 0011->0110->1100

Lastly, all of the shiftings are added together in order to compute the final result.

This algorithm uses 8-bit adders constructed by cascading 2-bit adders.

d) Division

The general schematic is :

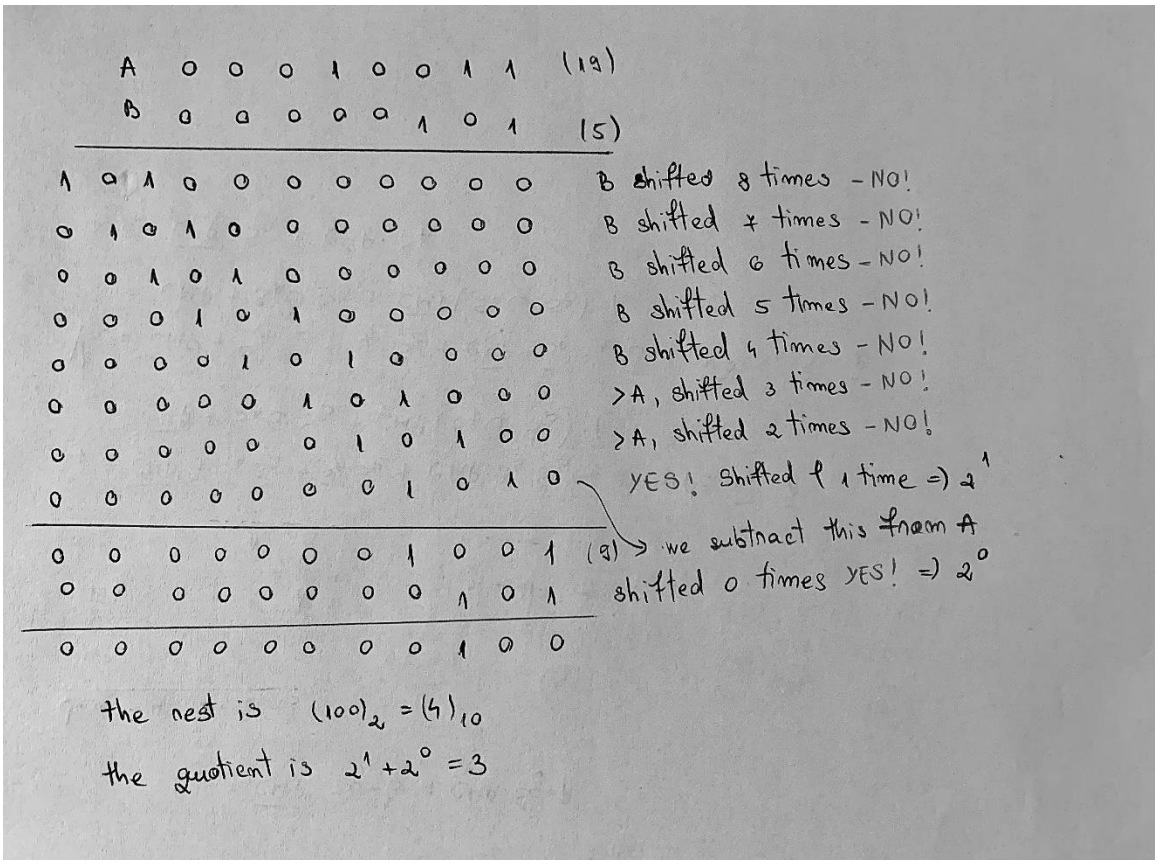


Division

The inputs are the two 8-bit numbers. The outputs are P (the quotient) and rest, both on 8 bits.

If we want to divide A by B, B will be put under A by shifting it to te left in such a way that B is maximum but still smaller than A. The value of the quotient which initially is ‚0' will be increased by 2 at the power of the shifting's rank. Then, the number shifted will be subtracted from A and the procedure repeats until all the cases are verified. The number that remains after subtractions represents the rest and the quotient is computed by adding the powers of 2 that were valid.

Here's an example:



When writing the code I used 16 bits so no bit would be lost during our eight shiftings.

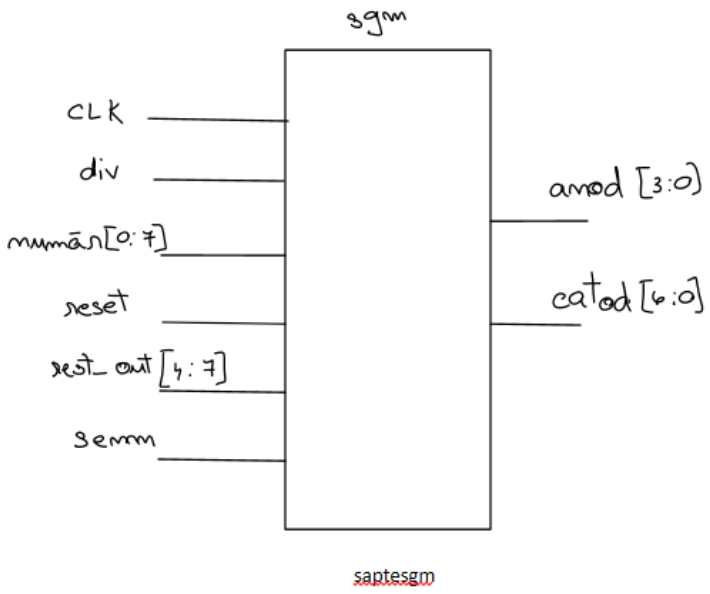
The left shiftings are done in the same manner as for the multiplication, by adding the current number two times, but in this case without adders.

After the shiftings are done, we check (using addition) if the largest shifting, the first one being on 8 bits, is valid. If that happens, from r which takes the value of A we subtract that shifting and then add to the quotient the power of 2^8, 128. If that's not the case, we move on. This procedure is applied at every shifting that is done.

The quotient will be equal to the sum of the powers of 2 (where the condition is completed) and the rest will be what's left after the subtractions.

e) 7 Segment Display (decoder)

The general schematic:



The inputs are: clock (for the board), **div (** the switch for division), **NUMAR_OUT** (the number that we want to display), **RESET**, **REST_OUT** (the rest after division), **SEMN** (the sign of the number).

As outputs we have: **4 anodes** (anod) and **7 cathodes** (catod).

This component is used for displaying on the 7 segments and decoding back the number to decimal.

The first thing we need to do is decide what to display on one segment at some point. Thus, we check the value of **LED_BCD** computed at the end of the component. We will use a „case' in order to decide what value to display. The digits that can be displayed are from 0 to 9 and moreover a,b,c,d (in hexa) for the rest.

For turning on every anode one by one we need to make a selection. First, for the value ‚00', the first anode is activated and the others not; at the same time the output **LED_BCD** takes the part from num_sortat created specially for that anode. This procedure repeats for each anode. The value will be stored on 16 bits, every 4 bits representing the value that have to be displayed on each anode.

Ex: num_sortat- 0010 1010 1111 1001 – 16 bits

| | | |

value -> Anode1 Anode2 Anode3 Anode4

In order to decode the number, it will be divided by 100 so the first digit is obtained. Then, the rest of the result is divided by 10 to obtain the second and last digit. The third digit is the last rest.

The final sign will be displayed on the first anode. The first three bits will be ‚1' and the last one represents the sign. The following step consists in checking the first ‚case" of the component and deciding how the cathodes will be displayed.

f) Decision

The inputs are **ENTER** (e), representing the signal after debouncing (confirms the displaying), +,-,*,/, **SEMN_AD** (the sign of addition/subtraction), **SEMN1**, **SEMN2** (the signs of the two numbers), all on one bit; **SUMA, INM, IMP**, **REST** – the result on 8 bits.

The outputs are **SEMN_OUT** (final sign) on 1 bit, **NUMAR_OUT** , **REST_OUT** -on 4 bits.

The aim of this component is to decide what is stored in the result that will be displayed. If **ENTER** is active and also e, then we check what operation is selected.

- If it is +/- then the final result will be the addition/subtraction. The rest is 0 and the sign the one of the computation;

- The same thing is true for multiplication and division except for the sign. The sign will have the value **SEMN1** xor **SEMN2**.

g) The main function

This is the final component where all the others are linked together.

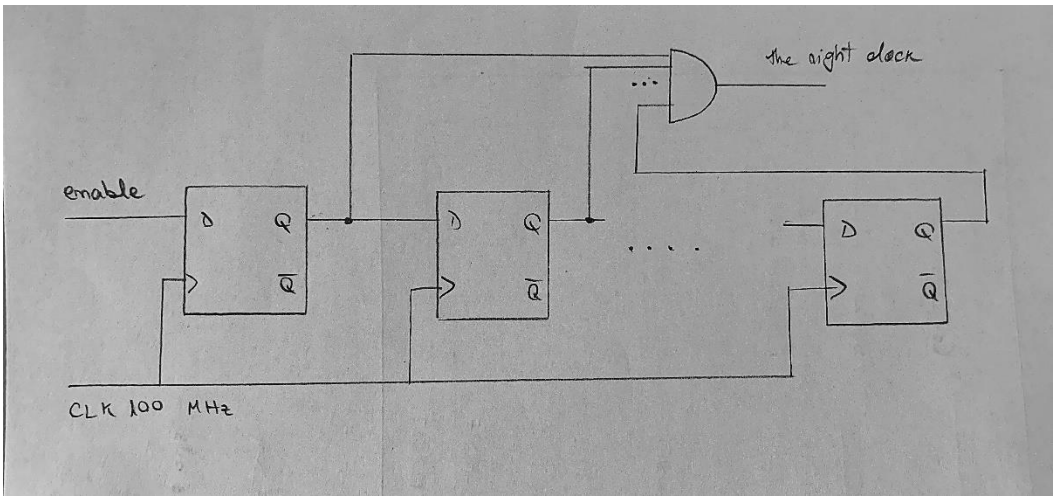
The inputs are the digits 0-9, the operands „+”, „-„ , „*”, ”/”, **ENTER**, **ENTERC**, **RESET**, **CLK** (for the board).

The outputs are the **4 anodes** and **7 cathodes**. Additionally, there will be displayed: the number on 8 bits, the rest on 4 bits and the sign on one bit.

We need to construct a debouncer for the switches **ENTER** and **ENTERC**. Then we put on different signals the operations. According to the operand that is active, the function to be displayed on the 7 segments is chosen.

h) Debouncer

This component is created for avoiding the problem arised by the clock signal. We put 20 D flip-flops to delay the enable signal. After the signal goes through all the flip-flops, we run the outputs through an and gate. The resulted signal is the new clock for the **ENABLE** switch.



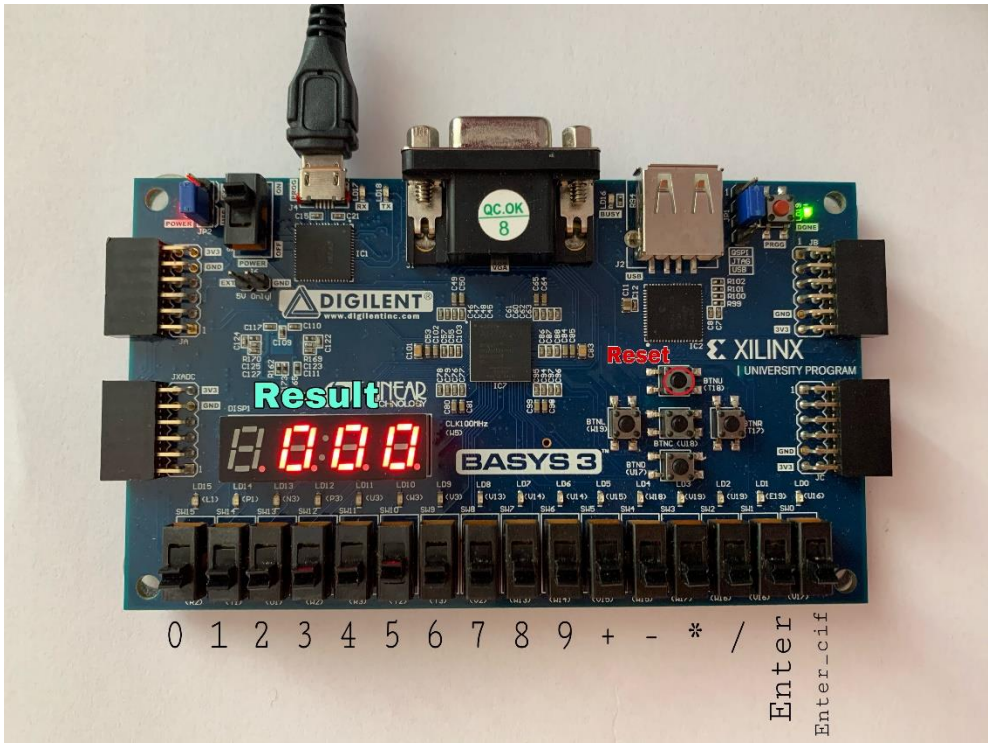
5. Motivations behind the project

I chose this kind of implementation because it seemed the easiest and at the same time the most practical one. First I constructed the main and secondary components and at the end I just linked them with the last component. All of the operations are done permanently. The result remains unchanged until the **ENTER** switch is turned on. It was not needed to implement any states because the signals and results are modified only when external stimulus are applied.

6. Experimental results

The project will be implemented on the Basys 3 board.

First, we push the **RESET** button to initialize the board. Then, we choose a digit, followed by the activation of **ENTERCIF** switch. After the number is constructed, we turn off the switches, select an operation and continue by introducing the second number. At the end, we turn on the **ENTER** switch and the result is displayed.



7. Later improvements

One option would be to eliminate the switch **ENTERCIF**. Thus, the switches for digits would also play the role of the impulse when one is pressed. To continue with another operation we press **ENTER**. Another component is needed: that gives the final result to the first number and sets the second to ‚0'.

8. Conclusion

This project challenged my skills for sure. I think it helped me improve my coding and made me understand VHDL a lot better. When you forget that it is hardware descriptive and treat it is a coding language, the problems you will encounter with your code will intensify, and I learnt that the hard way. Overall, it was a good experince and the satisfaction at the end worth the effort.

9. Bibliography

Most of the information used for this project was learnt from the Digital System Design course and laboratories, but I also read some online courses:

1. http://www.secs.oakland.edu/~llamocca/
2. Active-HDL Student Edition - FPGA Simulation - Products - Aldec – Active HDL for simulation
3. https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vivado-design-tools/archive.html - Vivado for implementing the project on Basys 3