Universitatea Tehnica Cluj-Napoca

Departament Calculatoare-Engleza

Nume:Buda Ioana-Daniela

ALU- Arithmetic Logic Unit

Project documentation

Contents

1.Introduction and main objectives……….....

2.Theoretical Fundamentals…………………….

3.Implementation and Design………………...

4.Algorithms………………………………………….

5.Testing…………………………………………………

6.Conclusions…………………………………………..

7.References ……………………………………………..

**1.Introduction**

Computers are one of the mains tools which have become extremely necessary in the modern living .Because they have such an impact and bring so many benefits to the society it is important to learn details about them. In this project the focus will be on a very important part of their structure ,the ALU .

Alu( Arithmetic Logic Unit) represents one of the main elements in the structure of computer system .It is a major component of the central processing unit and it does all the processes related to arithmetic and logic operations that need to be done on word instructions.

It can be designed by engineers to calculate any operations ,and because of that they are very complex.

-The main purpose and objective of this project is the development ,design and implementation of an ALU( arithmetic logic unit),demonstration of its behavior ,testing and understanding the principles it runs on .

The primary functions and operations allowed by the Alu designed and implemented in this project are:

* -Addition and Subtraction in Two’s complement (C2):
* -Increment, decrement
* -Logical operations :LOGIC AND ,LOGIC OR,LOGIC NOT
* -Rotating left and right
* -Use of an accumulator as an input operand and for the result
* -Use of supplementary circuits for multiplications and division: Multiplication can be performed by treating the multiplier unit as a combinational circuit, multiplicand and multiplier bits as inputs and the bits of the result as outputs.

Division is realized by using an accumulator and the “Subtract and Shift” algorithms and by the use of a finite state machine.

Every operation will be represented by a component instantiated in the main ALU enitity .

# The project will be implemented using VHDL Language in Vivado Hsl Environment and the physical part of the project is represented by the Basys 3 Artix-7 FPGA Trainer Board .

***2.Theoritical Fundamentals***

**ALU** is the main element of a microprocessor which performs the arithmetic and logical operations.

The focus will be on proving a few of the main characteristics of the arithmetic logic unit such as :logical operations such as :AND ,NOT ,OR; addition and subtraction , multiply and division ,all on Two ‘s complement representation.

Binary Number System is one the type of most popular Number Representation techniques that used in digital systems. In the Binary System, there are only two symbols or possible digit values, i.e., 0 (off) and 1 (on). Represented by any device that only 2 operating states or possible conditions .Two's complement is the most common method of representing signed integers on computers and is very used in various arithmetic operations (addition, subtraction) .This way we will represent ALU’s input numbers.

**1.Input and Output**

-Input data consist of two operand namely A and B , both represented in two’s complement representation system .The data will be stored in registers and will have n bits .For one of the operand an accumulator register may be used .Output data consist of the result ,namely X.

For some operation only one input is necessary and for others the input size will be different according to the desired operation and result size.

s

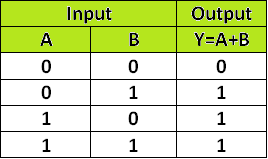
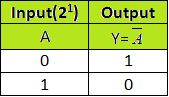


Overall View of the Alu.

**2.Operations**

Logical operations:

-A and B (AND ) A or B (or) NOT A (negation)

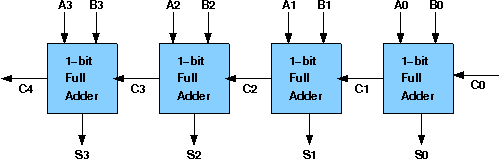
  

-Increment : A= A+1

-Decrement :A=A-1

-shift left ex : A=10100111 Result=01001110

-shift right ex: A=10100111 Result=11010011

-addition 

(example of addition mechanism for two 4-bit inputs)

A+B+Cin=S+Cout

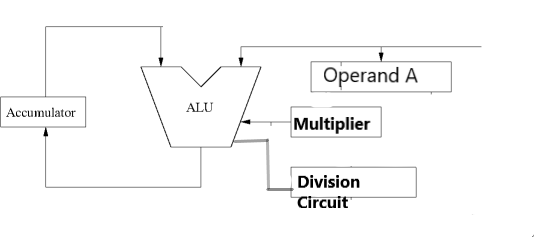
-Subtraction A –B=A+ (not B) +1;

These operations will be included in ALU.

Multiplications and division use more features ,have external circuits and more complex algorithms and will be discussed in the Algorithms section .

Selection of the operation will be made by the user.

**3.Components**



1.Accumulater

-An accumulater is a processor register for storing intermediate results a register .Accumulator work similar to the functionality of counter. The main difference is instead increment the counter value by constant, Accumulator add the input value with the current value.

2.Components included in Alu

-Multiplexer : The multiplexer, shortened to “MUX” or “MPX”, is a combinational logic circuit designed to switch one of several input lines through to a single common output line by the application of a control signal. They will be used mainly to select the operation.

-Adders :circuits that will describe operations such as addition and subtraction.

-Shift registers : circuits that will describe operations such as addition and subtraction.

3.Multiplier circuit

-circuit that will describe and apply the multiplication algorithm.

4.Division Circuit

-circuit that will describe and apply the division algorithm.

***3.Implementation and Design***

The Ide Development Tool chose for the implementation of the Alu prototype is Vivado 2016.4 .The choice was based on its capabilities to work with Basys 3 Artix 7 FPGA Board that will be used to describe Alu s functionalities.

*1.Implementation*

The overall implementation is based on a structural description of the system,as a set of interconnected components. I consider this an being very advantageous as the design can we divided into smaller design units that can be easily taken apart and tested individually .This way debugging in the case of something that goes wrong is much more easy and the project can be build in several steps. Also ,the components can be easily stored and reused.

The present project will have as main components as follows:

-16 bit Ripple carry adder used for addition and also for subtraction ,decrementing , incrementing ,with the right arguments.

- AND ,OR ,NOT , circuits with 16 bit numbers as inputs.

-Shift registers for both right and left directions

-multiplier

-division circuit

-monopulse generator

-accumulater circuit

***MPG(Mono pulse generator)-additional circuit***

The behavior presented when you press a button on the board can sometimes be unpredictable so in order to generate signals like enable from buttons we use a mono pulse generator . The created MPG will be a synchronized 1 clock period mono pulse generator.

The components for it are a 16 bit counter and 3 D latches.

The role of the first register together with the 16-bit counter is to provide a delay necessary to de-bounce the buttons (physically worn out and / or low quality buttons).

**entity MPG is**

**Port ( btn : in STD\_LOGIC;**

**clk : in STD\_LOGIC;**

**enable : out STD\_LOGIC);**

**end MPG;**

The btn input is related to the input buttons from the board we will use.

Enable is the corresponding clock response.

***Accumulator register***

**1. 16 bit Adder**

**SUM=A+B**

Addition of two numbers in C2’s complement is performed by a ripple carry adder. The ripple carry adder is based on the component full adder and it uses cascade of the component to match the number of required bits.

Each full adder is used to add individually one bit from a number the one bit from the other number and each carry output is connected to the input of the other one .The full adders are connected in series .

Vhdl semnificative code:

**ENTITY Adder8 IS PORT (**

**Cin: IN STD\_LOGIC;**

**A, B: IN STD\_LOGIC\_VECTOR(15 DOWNTO 0);**

**Cout: OUT STD\_LOGIC;**

**SUM: OUT STD\_LOGIC\_VECTOR(15 DOWNTO 0));**

**END Adder8**

***COMPONENT add\_1 PORT (***

***cin, a, b: IN STD\_LOGIC;***

***cout, s: OUT STD\_LOGIC);***

***END COMPONENT;***

//code that reflects how the carry output is transmitted in series

***Adder: FOR k IN 15 DOWNTO 0 GENERATE***

***FullAdder: add\_1 PORT MAP (Carryv(k), A(k), B(k), Carryv(k+1), SUM(k));***

***END GENERATE Adder***

***2.Subtractor of two 16 bit numbers***

***SUM=A-B***

Subtraction of two’s complement is done by using addition too .Instead of creating a different component we will use the already designed adder .The number that will be subtracted will be turned into its negative and then the addition will be done .For subtracting a number ,we perform addition with the negated number and we add ‘1’ ( this is the way to negate a number in two s complement) .

Since we already have the adder we will use it with the following instance:

**Adder8 PORT MAP (CIN => '1',**

**A=>A;**

**B=> NOT B;**

**Cout => Count;**

**SUM=> SUM );**

The Sum will actually be the result of subtraction.

***3.Increment***

***A=A+1;***

Incrementation of a 16 bit number is made by using the 16 bit adder described above where the first operand will be the number to be incremented (A) and the second operand will be represented by the number 1 on 16 bites ( 0000000000000001).

The result will be A incremented. The component will be instantiated as it follows:

*Adder8 PORT MAP (CIN => '0',*

*A=>A;*

*B=> “0000000000000001”;*

*Cout => Count;*

*SUM=> DIFFERENCE );*

***4.Decrement***

***A=A-1;***

Decrementation of a 16 bit number is made by using the 16 bit adder described above where the first operand will be the number to be incremented (A) and the second operand will be represented by the number negative 1 in two s complement which is on 16 bites (1111111111111111).

We can use the not operation to negate number 1 in two s complement or we can instantiate the component with the already known value.

The result will be A decremented. The component will be instantiated as it follows:

*Adder8 PORT MAP (CIN => '1',*

*A=>A;*

*B=> “1111111111111111”;*

*Cout => Count;*

*SUM=> DIFFERENCE );*

***4.And gate***

***A and B***

The and operation is one of the simplest operations performed in ALU. On bits 16 bites of the two operands will be applied the AND operation , one by one in pairs and the result will be also a 16 bytes number, corresponding to the result of each pair of bytes.

This is how the entity appears:

**entity AND\_GATE is**

**Port ( A : in STD\_LOGIC\_VECTOR (15 downto 0);**

**B : in STD\_LOGIC\_VECTOR (15 downto 0);**

**C : out STD\_LOGIC\_VECTOR (15 downto 0));**

**end AND\_GATE;**

A for loop will be used in order to have access to all the elements of the arrays , one at a time .Temp will be a temporary signal to hold the intermediate values:

**gen: for i in 0 to 15 generate**

**temp(i) <= A(i) and B(i);**

**end generate;**

***5.OR gate***

***A or B***

The and operation is one of the simplest operations performed in ALU. On bits 16 bites of the two operands will be applied the OR operation , one by one in pairs and the result will be also a 16 bytes number, corresponding to the result of each pair of bytes.

This is how the entity appears:

**entity OR\_GATE is**

**Port ( A : in STD\_LOGIC\_VECTOR (15 downto 0);**

**B : in STD\_LOGIC\_VECTOR (15 downto 0);**

**C : out STD\_LOGIC\_VECTOR (15 downto 0));**

**end OR\_GATE;**

A for loop will be used in order to have access to all the elements of the arrays , one at a time .Temp will be a temporary signal to hold the intermediate values:

**gen: for i in 0 to 15 generate**

**temp(i) <= A(i) OR B(i);**

**end generate;**

**6.NOT OPERAND**

**A=NOT(A);**

The negate operation in vhdl requires only the simple operator NOT .The NOT operator can directly be applied to the whole std\_logic\_vector representing the input so there is no need for another component to be created and instantiated.

This will save resources and time for the whole project.

***7.Rotation left and right***

The component which performs rotation receives as input an 16 byte operand ,the direction denoted R and returns a 16 byte operand representing the input operand rotated left or right one bit in the direction decided by the input R.

A value of input R of 0 represents the direction left and a value of 1 is used to denote the right direction of rotation.

The third input N is the number of bytes used as the argument for rotation.

***entity RotateX is***

***Port ( clk:in std\_logic;***

***A : in STD\_LOGIC\_VECTOR (15 downto 0);***

***B : out STD\_LOGIC\_VECTOR (15 downto 0);***

***N: in std\_logic\_vector(3 downto 0);***

***R: in std\_logic );***

***end RotateX;***

As well as bits shift operation ,the rotate operation can be implemented using concatenation.

A general rotation of n bits formula can be easily deduced as :

* ***Right rotate: A < = A(n downto 0) &A(15 downto n+1)***
* ***Left rotate : A<= A(15-n downto 0) & A(15 downto 15-n+1)***

Because reading an integer from the board and using directly is not possible ,the input N representing the number of rotation bits will be used in a case statement simulating a decoder.

Example:

***Case N is***

***when "0001" => B <= A(14 downto 0) &A(15);***

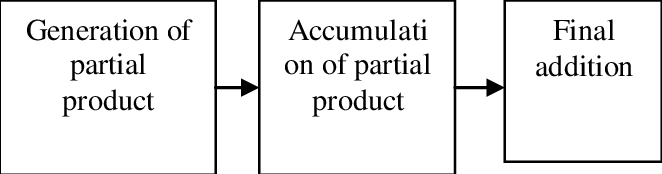
***Depending on the direction D and the input N ,any type of rotation can be simulated.***

***8.Multiplication***

The circuit receives as input two 8 bit operands and will output an 16 bit number representing the result of multiplication between the two input numbers.

The Wallace tree multiplication algorithm is applied with the scope of multiplying two 8 bits input normal ,resulting an 16 bit output product of the two input numbers.

*Block diagram of a Wallace tree multiplier*



The Wallace Tree algorithm will be implemented in a structural way using additional components such as halfadder and fullAder which were described above.

They are used for addition of intermediate term formed after the multiplication of two numbers on each level .

*Components :*

***component add\_2 is***

***port (A,B,CIN: in std\_logic;***

***S,COUT: out std\_logic);***

***end component;***

The component add\_2 is the full adder component .

***component half\_adder is***

***Port ( a : in STD\_LOGIC;***

***b : in STD\_LOGIC;***

***sum : out STD\_LOGIC;***

***cout : out STD\_LOGIC);***

***end component;***

Since there are two 8 bit numbers to be multiplied ,the algorithm will be divided into 4 stages .

To remember each product and sum we use multiple signals :

* S00……….S77 for the first sum signals
* C01…..C68 carry signals
* K01….k68 intermediate signals

***The entity***

***entity MultiplierComp is***

***Port ( A : in STD\_LOGIC\_VECTOR (7 downto 0);***

***B : in STD\_LOGIC\_VECTOR (7 downto 0);***

***prod : out STD\_LOGIC\_VECTOR (15 downto 0));***

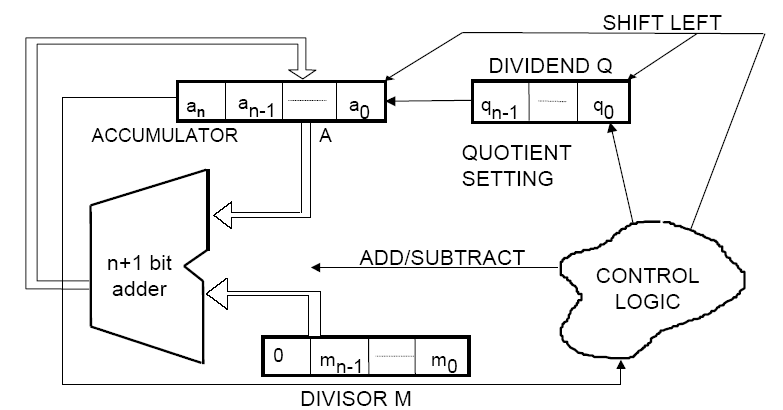
***end MultiplierComp;***

For the first stage we used 12 full adders and 4 half adders .For the second stage we have used 3 halfadders and 13 full adders ,for the third stage have been used 1 halfadder and 8 full adders and in the last stage 3 full adders and 6 half adders.

***9.Division of two numbers***

This is the “paper and-pencil” usual algorithm explained in Algorithms section.

It is the most complex of the Alu’s components and requires the behavior of a finite state machine .



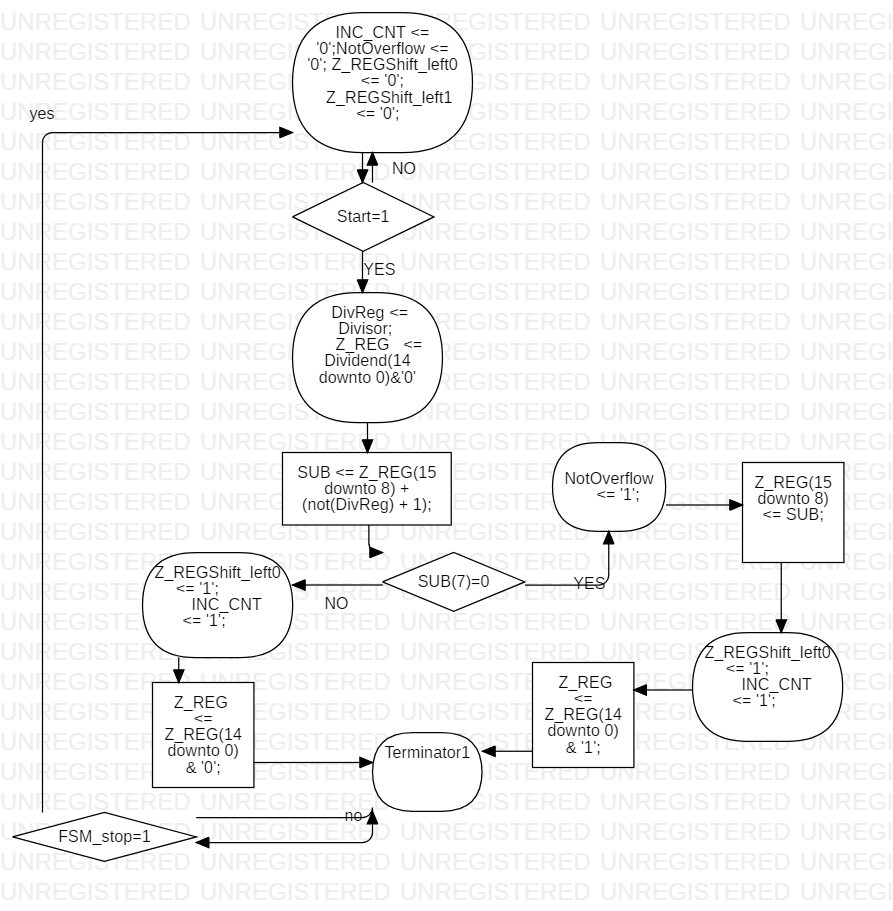
Hardware design of the algorithm.

We will implement the following state diagram which presents the states of the division algorithm.

Shortly,the steps are :

1. Load the divisor and dividend into registers named Z\_REG (role of accumulater,intermediate register) .In the end ,the first bites of the accumulater will represent the remainder and the last 8 bites of the accumulater will represent the quotient.
2. Verify if the difference between the corresponding bites of the dividend and the divisor represent an overflow situation or not and act upon it.
3. If overflow, shift dividend left with one bite. The shift operation will vacate the LSB of the dividend register.
4. If the difference from step 3 is positive then we set the LSB of the result part of the accumulater with 1 ,otherwise with 0.
5. Increase the value of the counter by one. If the counter is equal to eight, end the algorithm otherwise go to step 3.

Finite state machine on which the algorithm is implemented



**Description of the states**

State Idle : State in which we initialize the values that will be used during the others states.

* INC\_CNT –signal used for incrementing the counter, the counter will stop when it reaches 8 .
* NotOverflow – signal that takes the values according to the overflow state of the auxiliary dividend.
* Z\_REGShiftLeft\_0 – the signal will be one when we need to concatenate a 0 to the quotient
* Z\_REGShiftLeft\_1 – the signal will be one when we need to concatenate a 1 to the quotient

**State 0** : If the start bit is 1 we need to initialize the values for the auxiliary dividend and store the initial value of the divisor so we can use it later . The Sub vector will contain the result of the subtraction between the auxiliary dividend and the divisor.

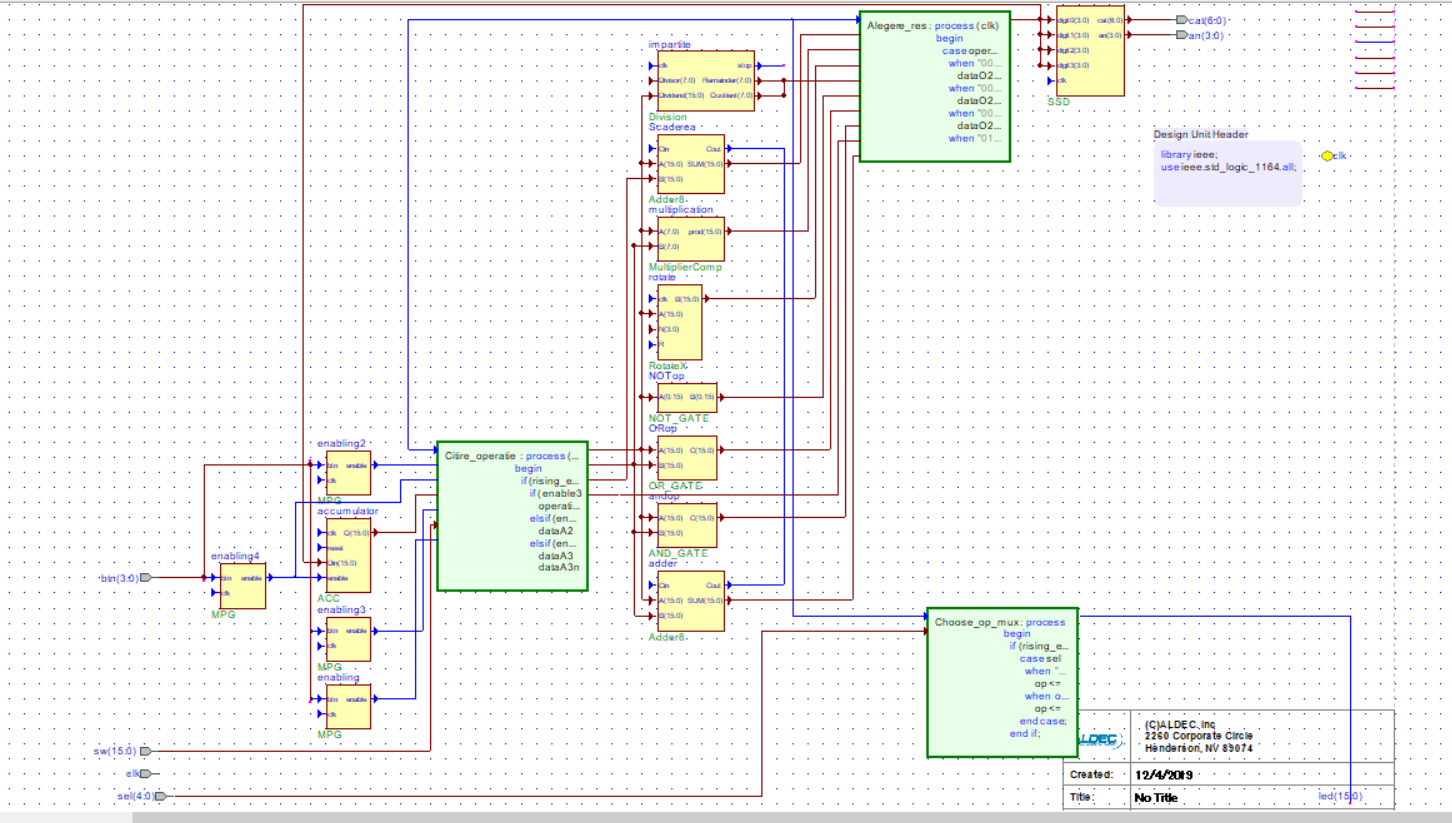
**State 1** : If the last bit of the subtraction is 0 that means we do not have overflow and the Z\_REG can take sub’s value .

**State 2** : If we the last bine of the SUB vector is 1 ,that means we have overflow and the result cannot be used so we will concatenate a 0 to the final Quotient and go the next state to see if it is final. We increment the counter

**State 3:** Since we do not have overflow that means that the subtraction was done correctly and a 1 will be concatenated to the final quotient. We increment the counter.

**State 4 :** This is the last state of the final machine where we verify if the signal FSM\_stop is 1 ,that means that the division is over ,otherwise we continue with the division algorithm on the remaining elements.

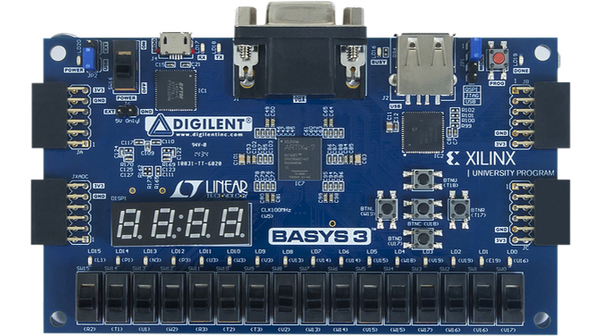
***10.Overall Implementation***



**2.Design**

The design of the required feature will be done using Vivado 2016.4 Developing Environment for developing VHDL language based application .

The VHDL code will be loaded into FPGA BOARD BASYS 3 where it can present the designed features.



After the program is loaded into the fpga board the following steps can be done by the user:

---Choose the operation number represented by the combination of the first 4 switches :

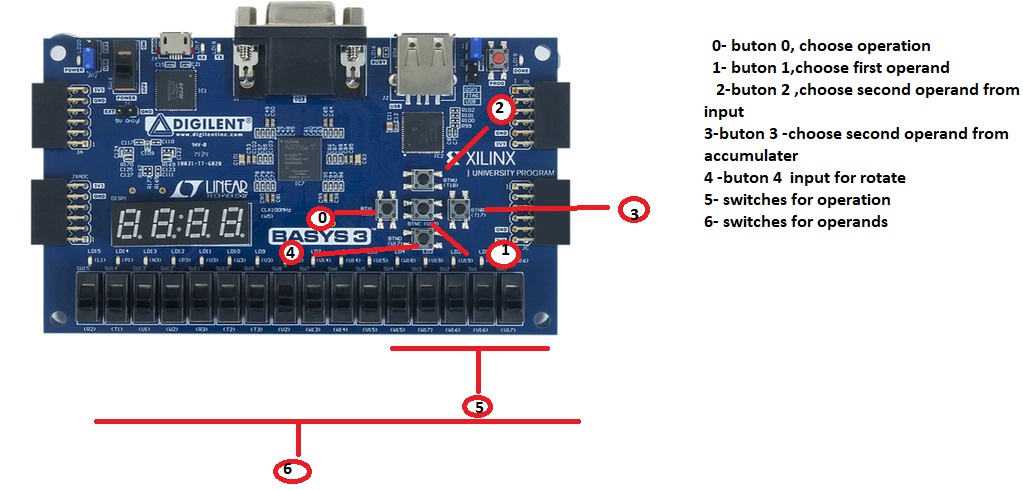
* 0000 – no operation
* 0001 –subtraction
* 0010- addition
* 0011 –and
* 0100 –or
* 0101 – negation
* 0110 –rotate right
* 0111- rotate left
* 1000 – multiplication
* 1001 - division

After the operation is chosen the user needs to press button 0

---Choose first operand represented by the number from the switches .After the number is selected the user needs to press button 2.

---Choose the second operand to be represented by the number from the switches.After the number is selected the user need to press button 3.

---Choose the second operand to be the number in the accumultator by pressing the button 3



***4.Algorithms***

* **Binary division algorithm**

Division of two binary numbers is one of the Alu’s functions to be performed.It is a more complex operation that does require more procedures to be followed ,representing an algorithm.

When we talk about division ,we have four main operands:

* Dividend X–first operand ( number to be divided)
* Divisior Y –second operand
* Quotient Q– main part of the result
* Remainder R

 The division algorithm repeatedly subtracts the divisor (multiplied by one or zero) from appropriate bits of the dividend. Therefore, subtraction and shift operations are the two basic operations to implement the division algorithm, but are executed only if Y R, which results in a quotient digit of 1 (otherwise is 0).

After each subtraction, the divisor (multiplied by one or zero) is shifted to the right by one bit relative to the dividend.

**STEPS**

* **Set** quotient to 0
* Align leftmost digits in dividend and divisor
* **Repeat**
* Align leftmost digits in dividend and divisor
* **Repeat**
  + **If** that portion of the dividend above the divisor is greater than or equal to the divisor
    - **Then** subtract divisor from that portion of the dividend and
    - Concatentate 1 to the right hand end of the quotient
    - **Else** concatentate 0 to the right hand end of the quotient
  + Shift the divisor one place right
* **Until** dividend is less than the divisor
* quotient is correct, dividend is remainder
* **STOP**

At each step in the process the divisor D either divides B into a group of bits or it

does not. The divisor divides a group of bits when the divisor has a value less than or equal to the value of those bits.

Therefore, the quotient is either 1 or 0. The division algorithm performs either an addition or subtraction based on the

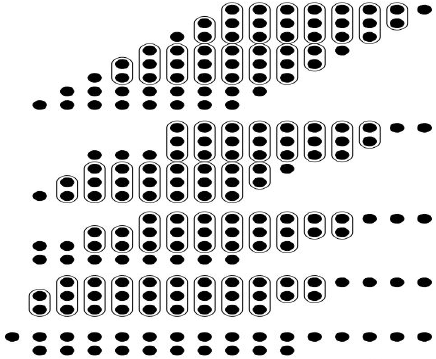
signs of the divisor and the partial remainder.

* ***Wallace tree multiplication algorithm***

Multplication operation is essential in Alu. Multiplication requires larger processing time than addition and subtraction and it plays a key role in arithmetic operations.

Wallace tree consist of three step process :

* The product terms are formed after the multiplication of the bits of the multiplicand and the multiplier
* We reduce using half and full adders
* Final addition done using adders to obtain de result



**5 .Testing**

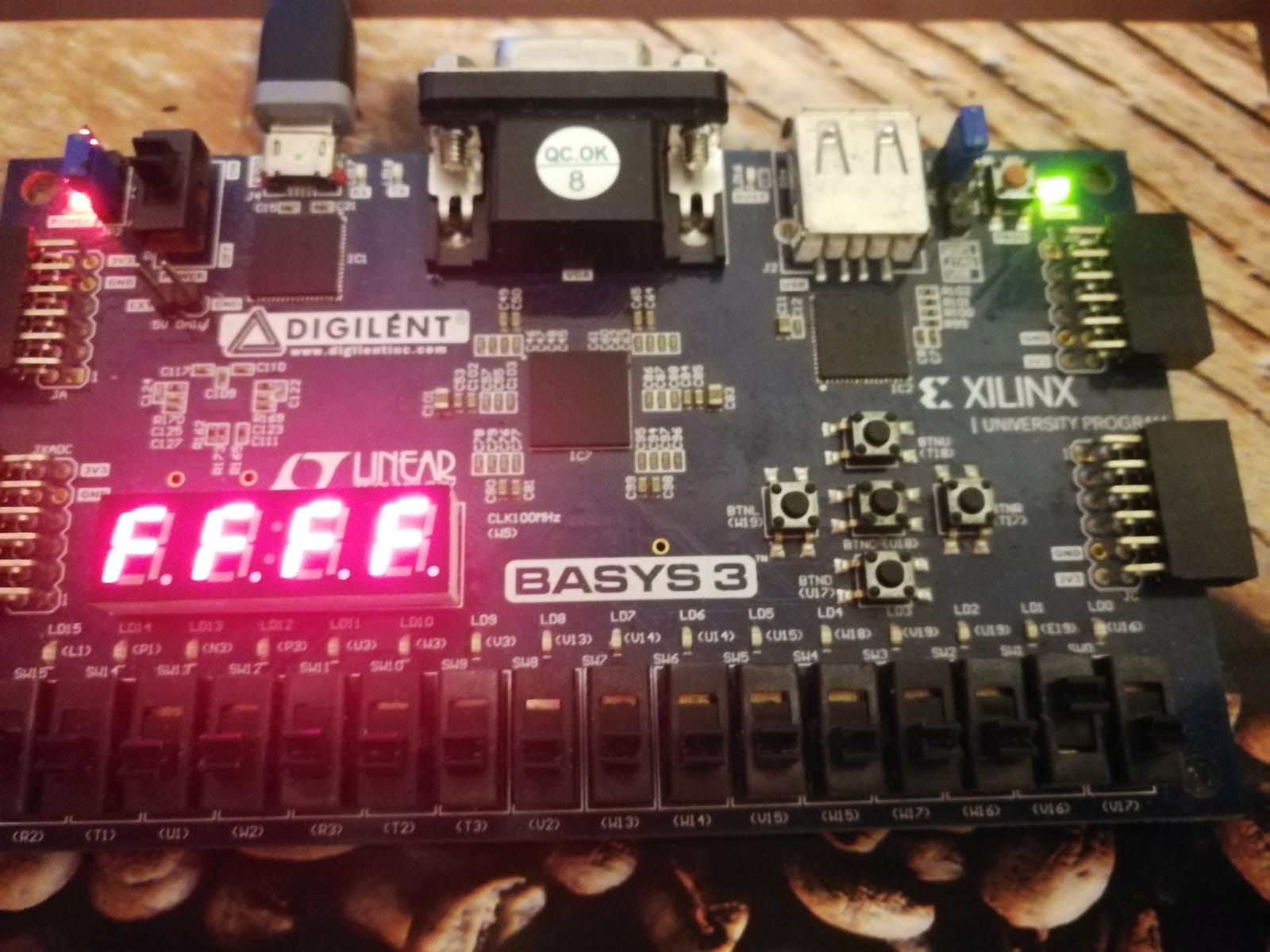
* **5.1 Testing indivividual components**

The Arithmetic and logic unit is build up bottom up using a structural style and components. To assure the well behavior of each of the components , and after that ,of the overall implementation , the components were also tested separately .

For simulating the behavior of the components I have used the **Aldec Product Active HDL Student Edition** and after the result are correct they will be available on the **FPGA BASYS 3 BOARD .**

In order for the simulation results to be easier to follow i have used similar inputs for all the components.

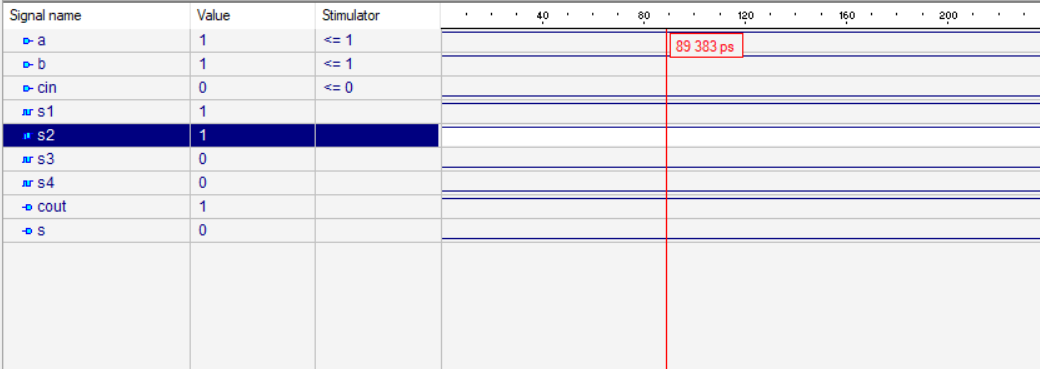
* Initially ,before initialization of the operation or operands on the board the seven segments display will show “0000” or “FFFF” representing that no operation has been done ,so no result will be shown.



* + 1. **FullAdder**

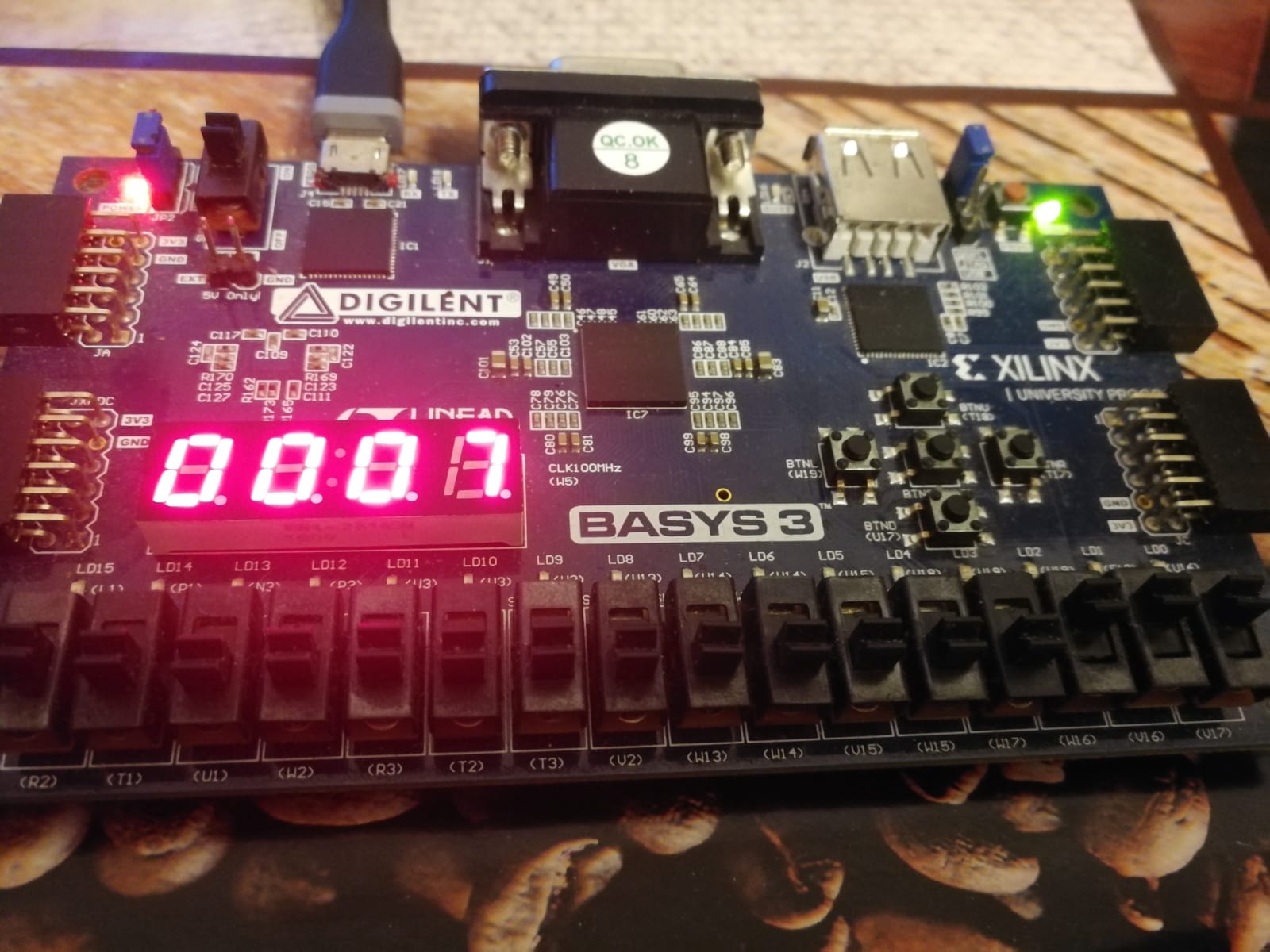
The Input were chosen as follows : A=1 ,B=1 and CIN=0

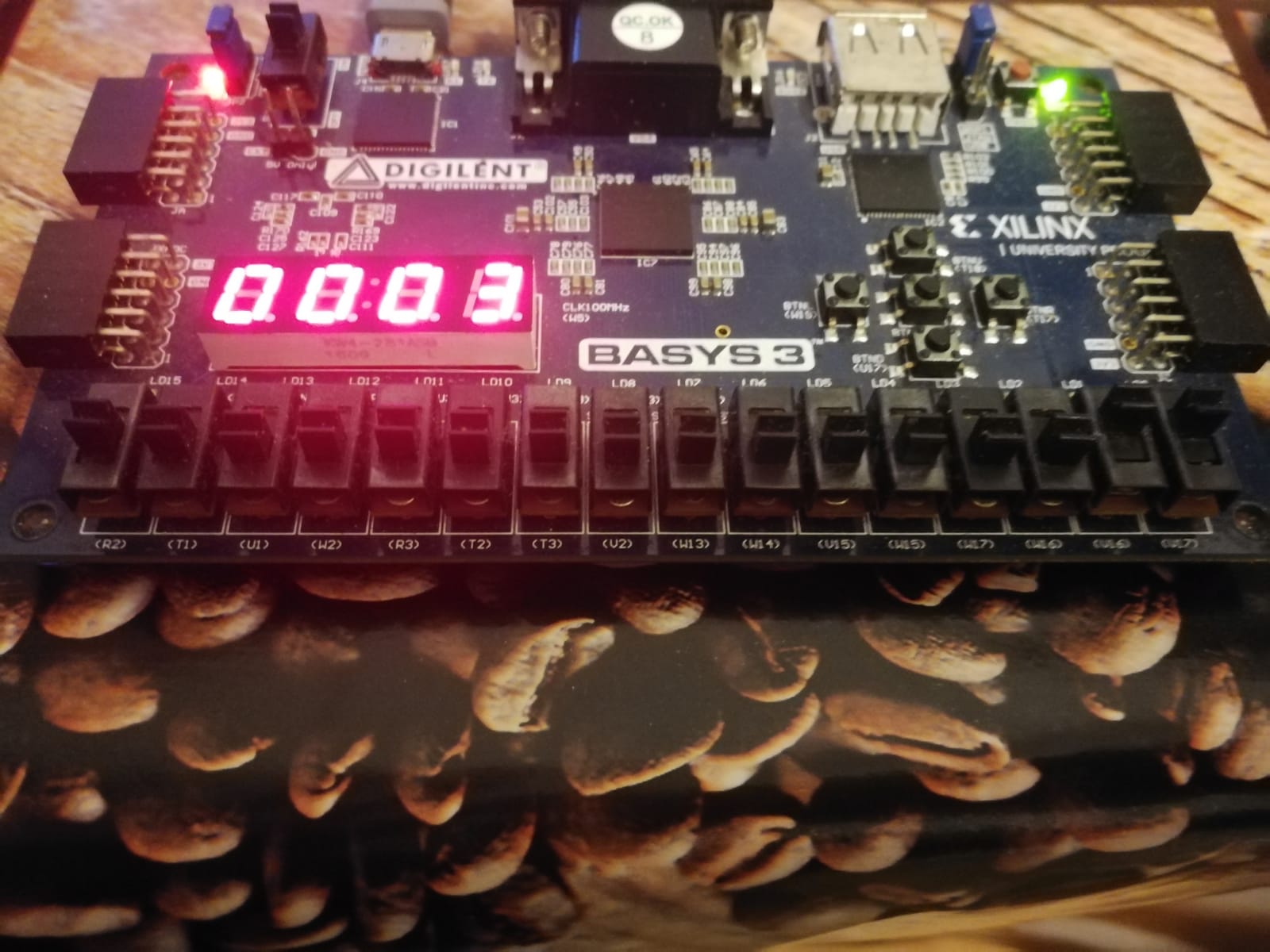
As expected, the sum S=0 and Cout=1.

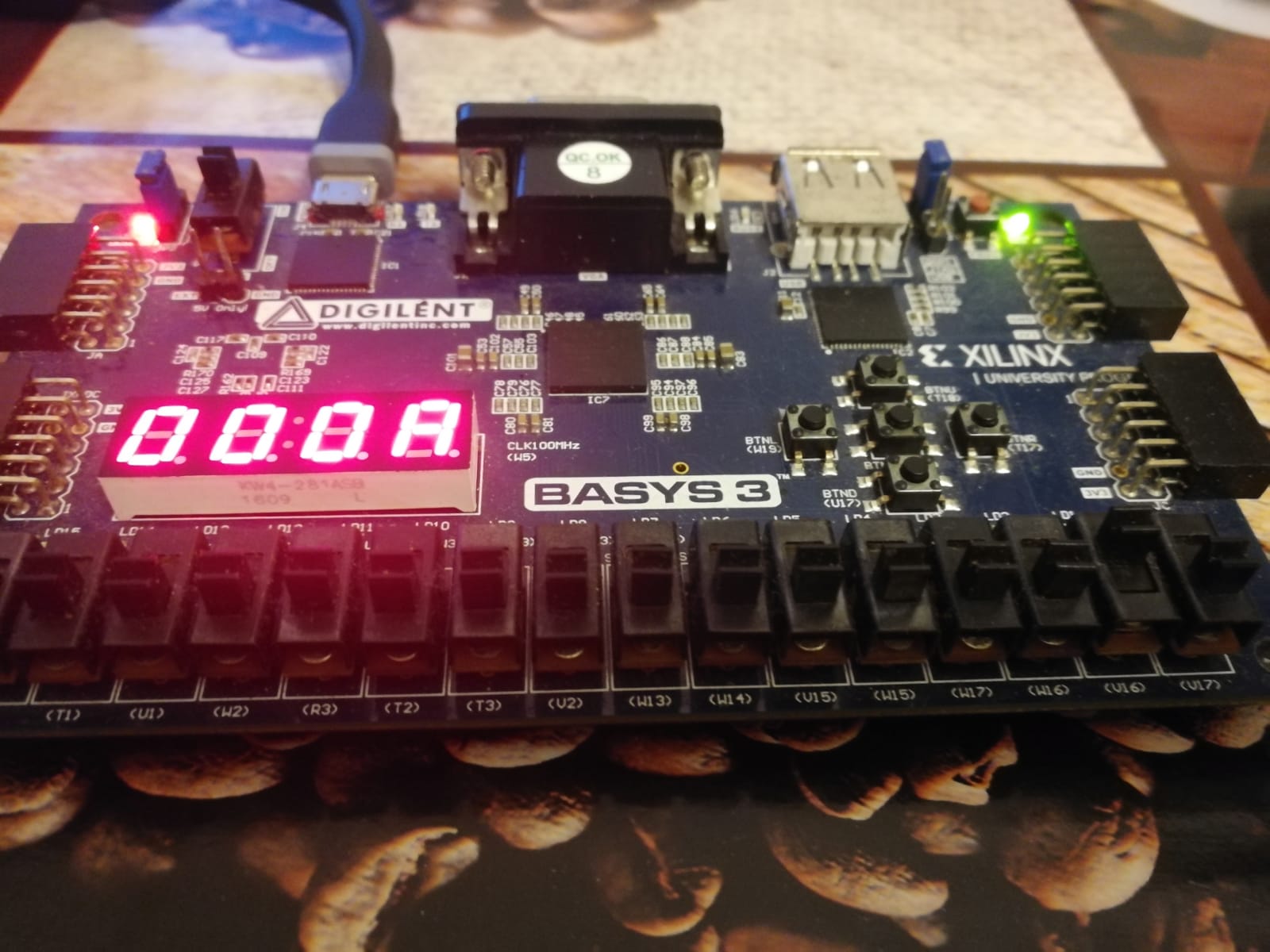
****

* + 1. **Ripple Carry Adder**

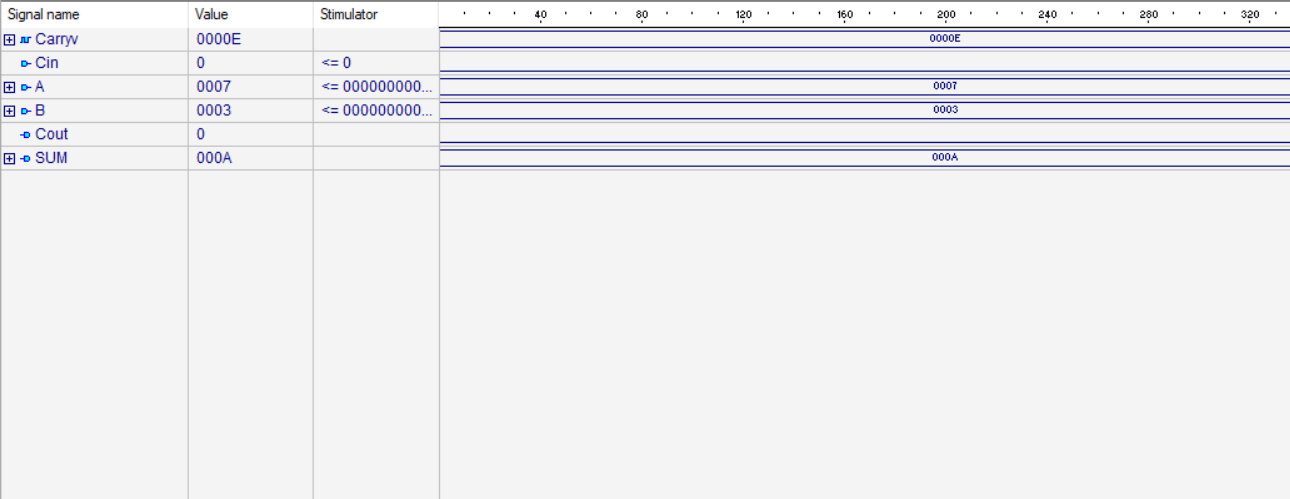
The Input were chosen as follows : A=7 ,B=3 and CIN=0





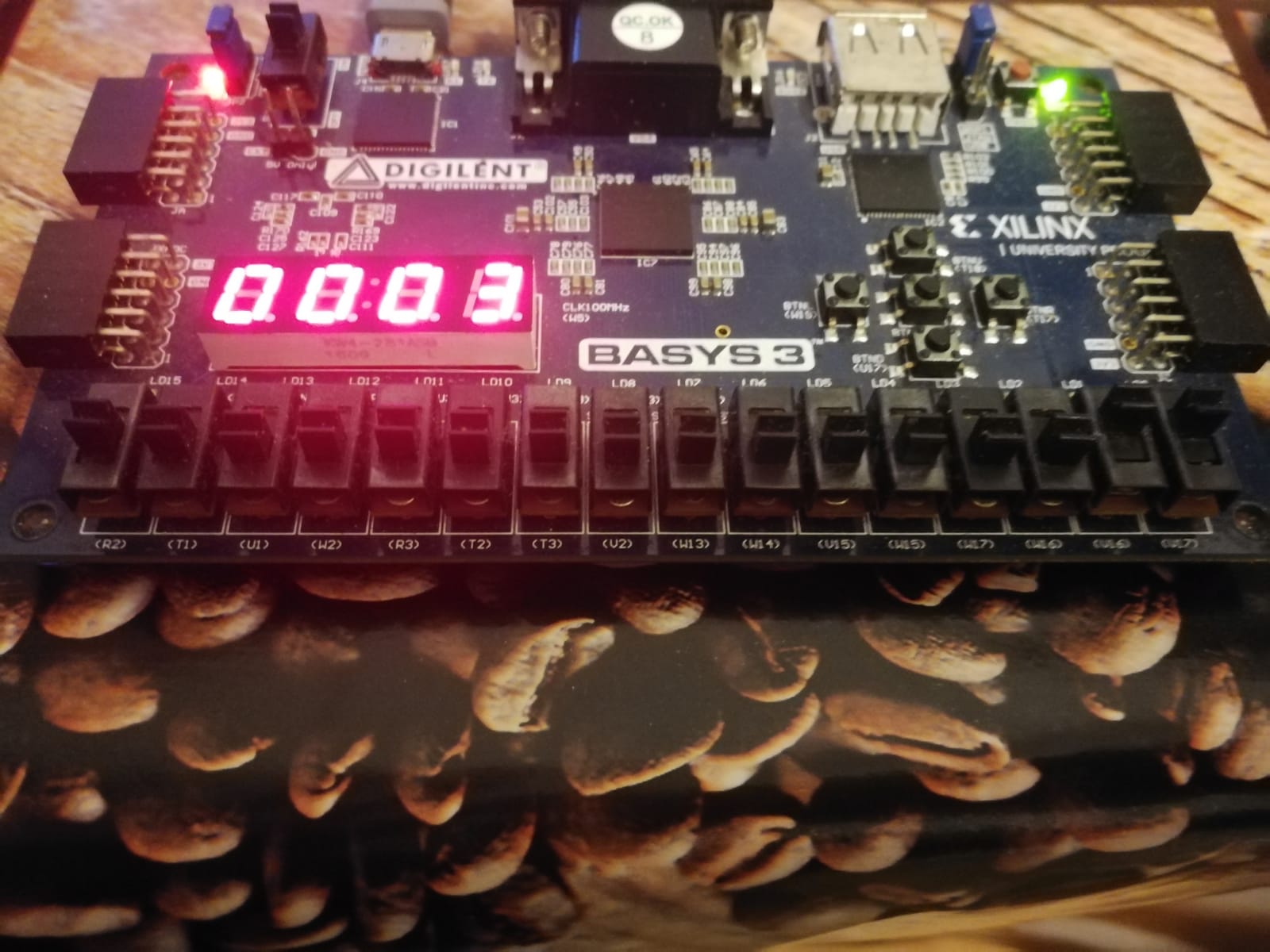


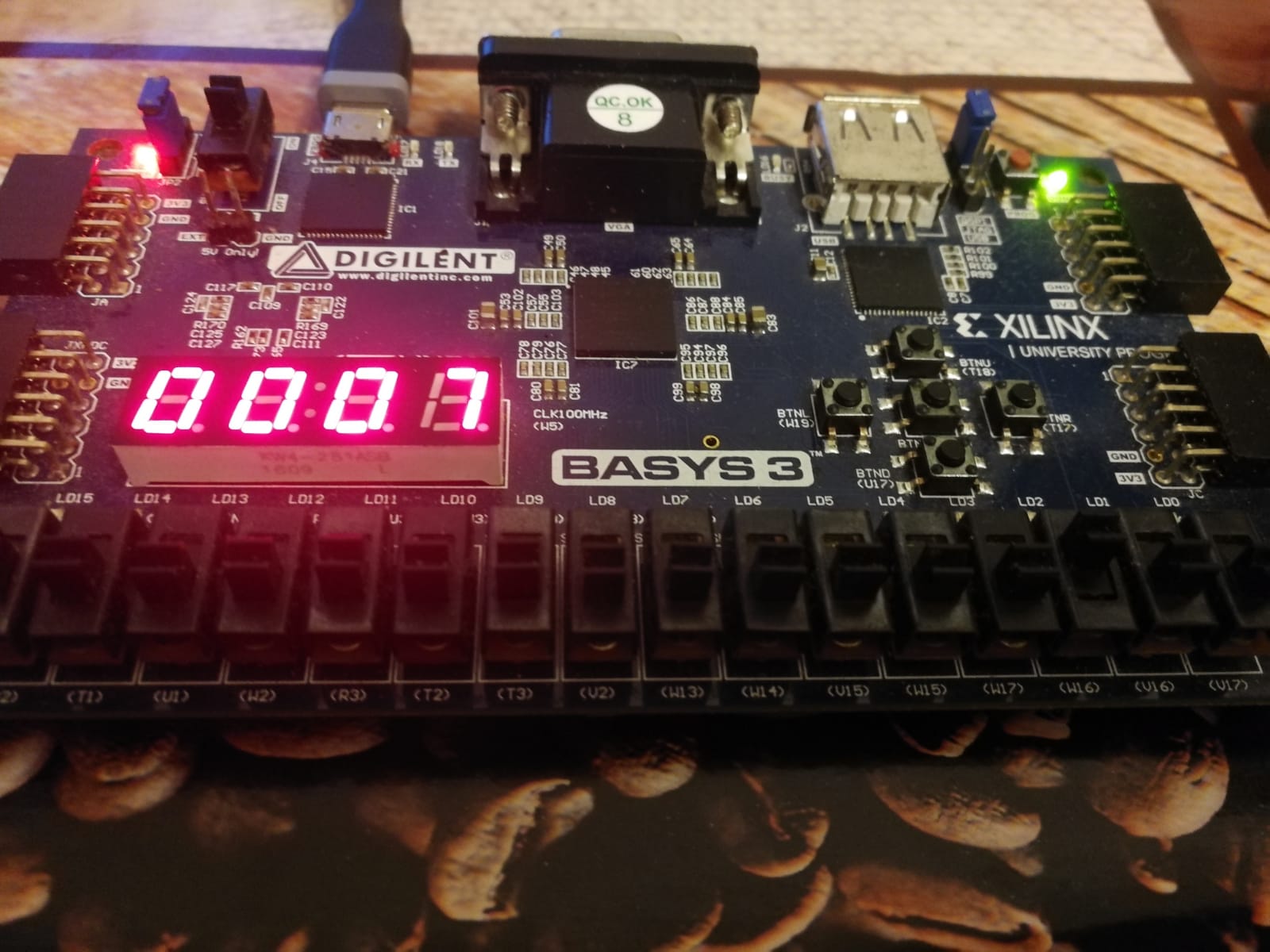
As expected, the sum S=A (10 ) and Cout=0.

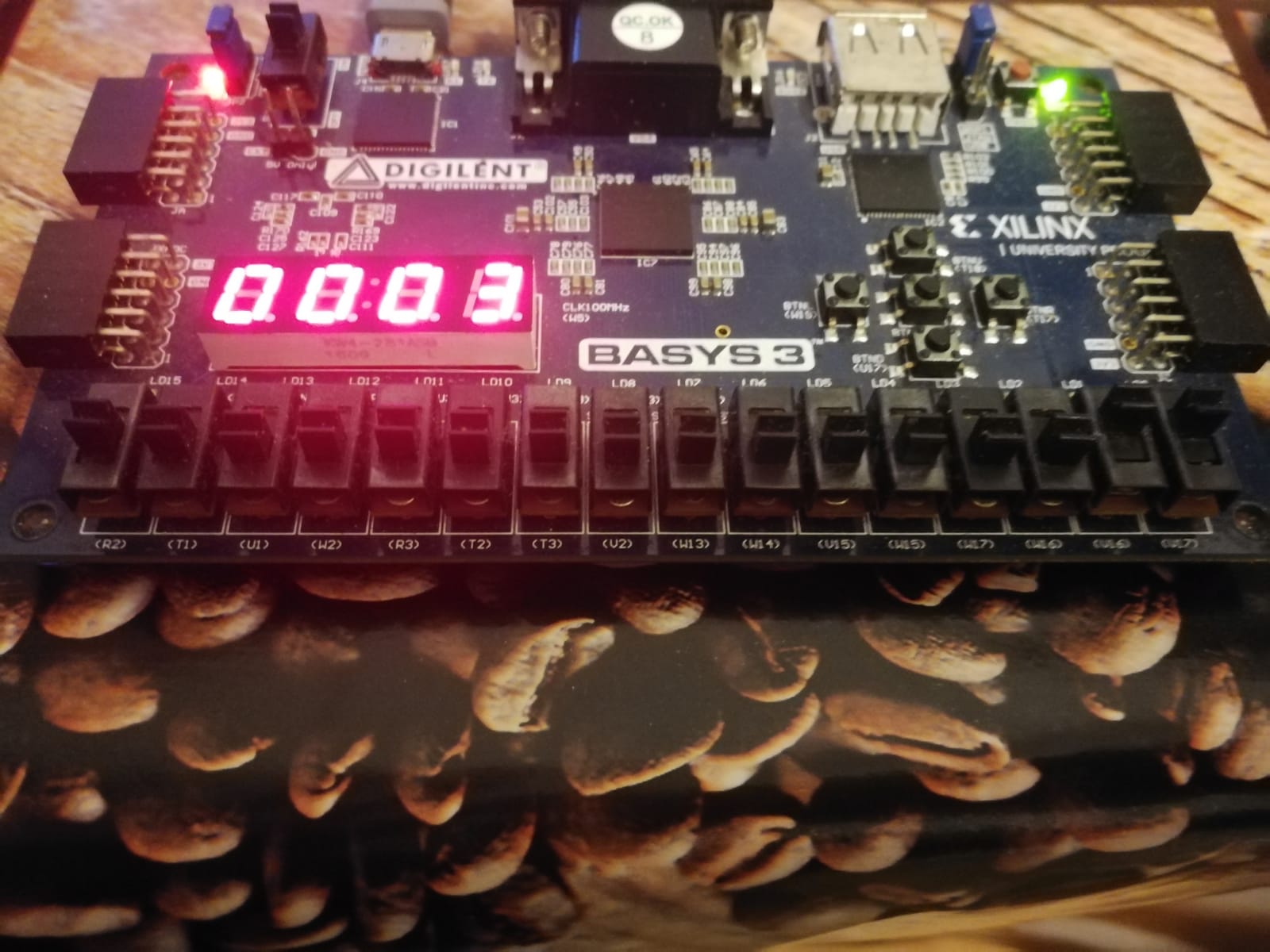


* + 1. **AND GATE**

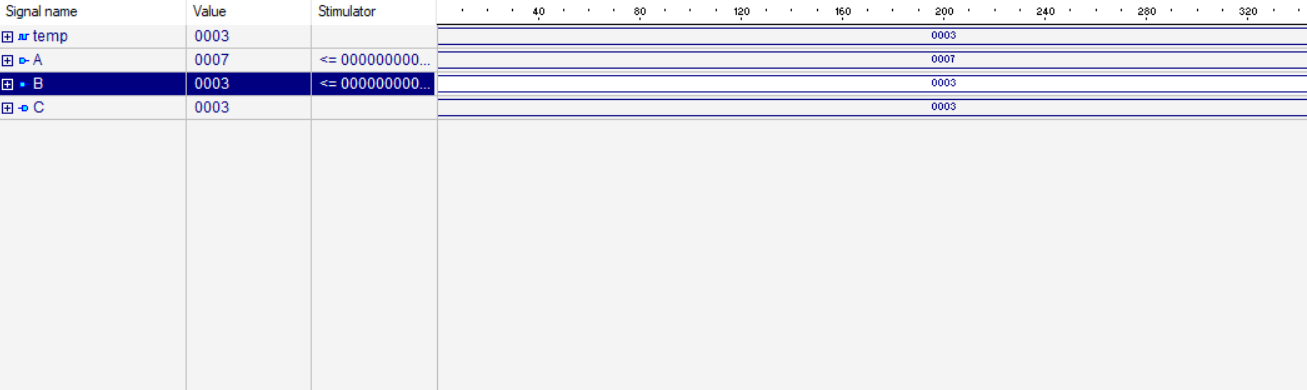
The Input were chosen as follows : A=7 ,B=3





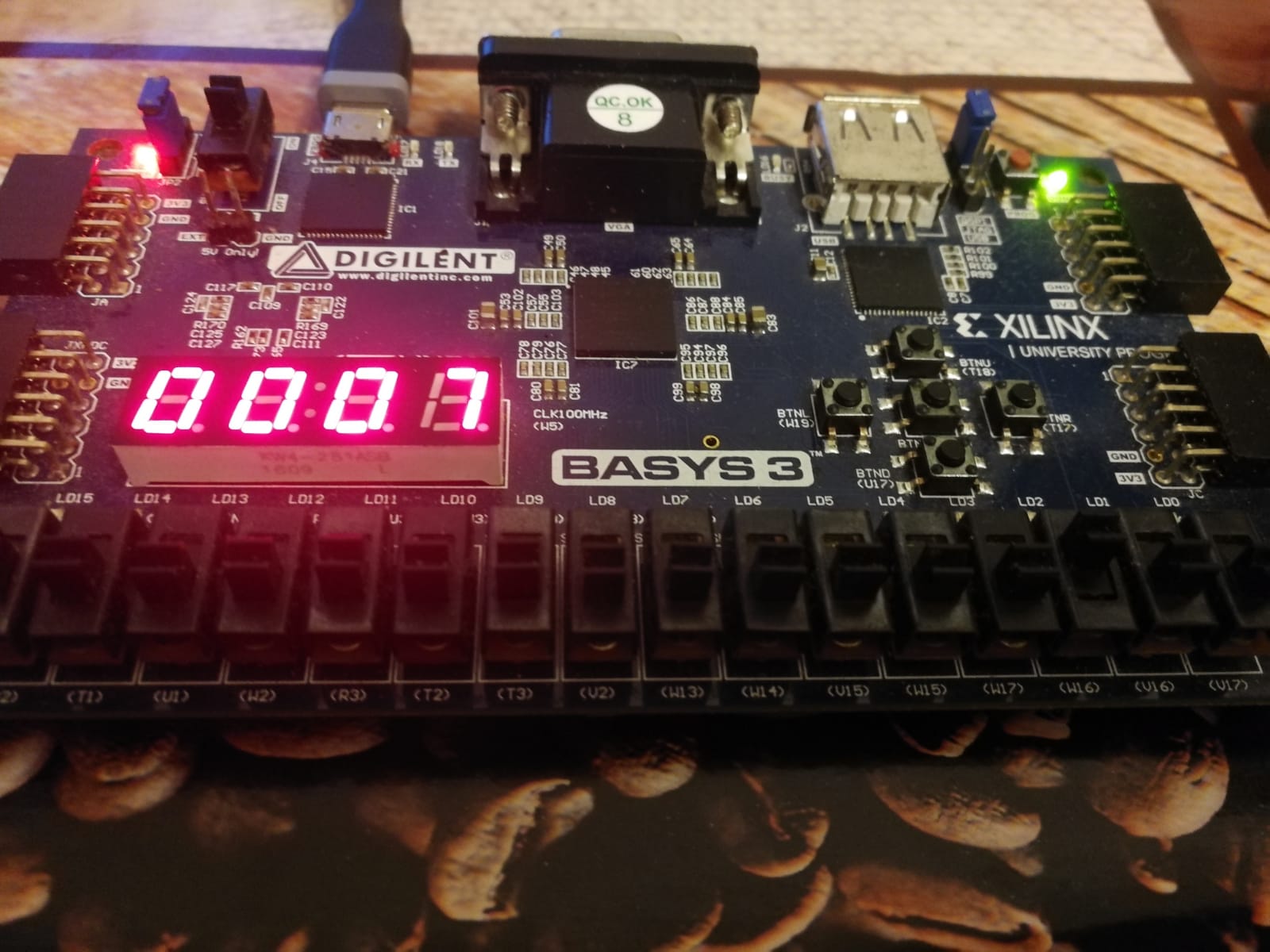


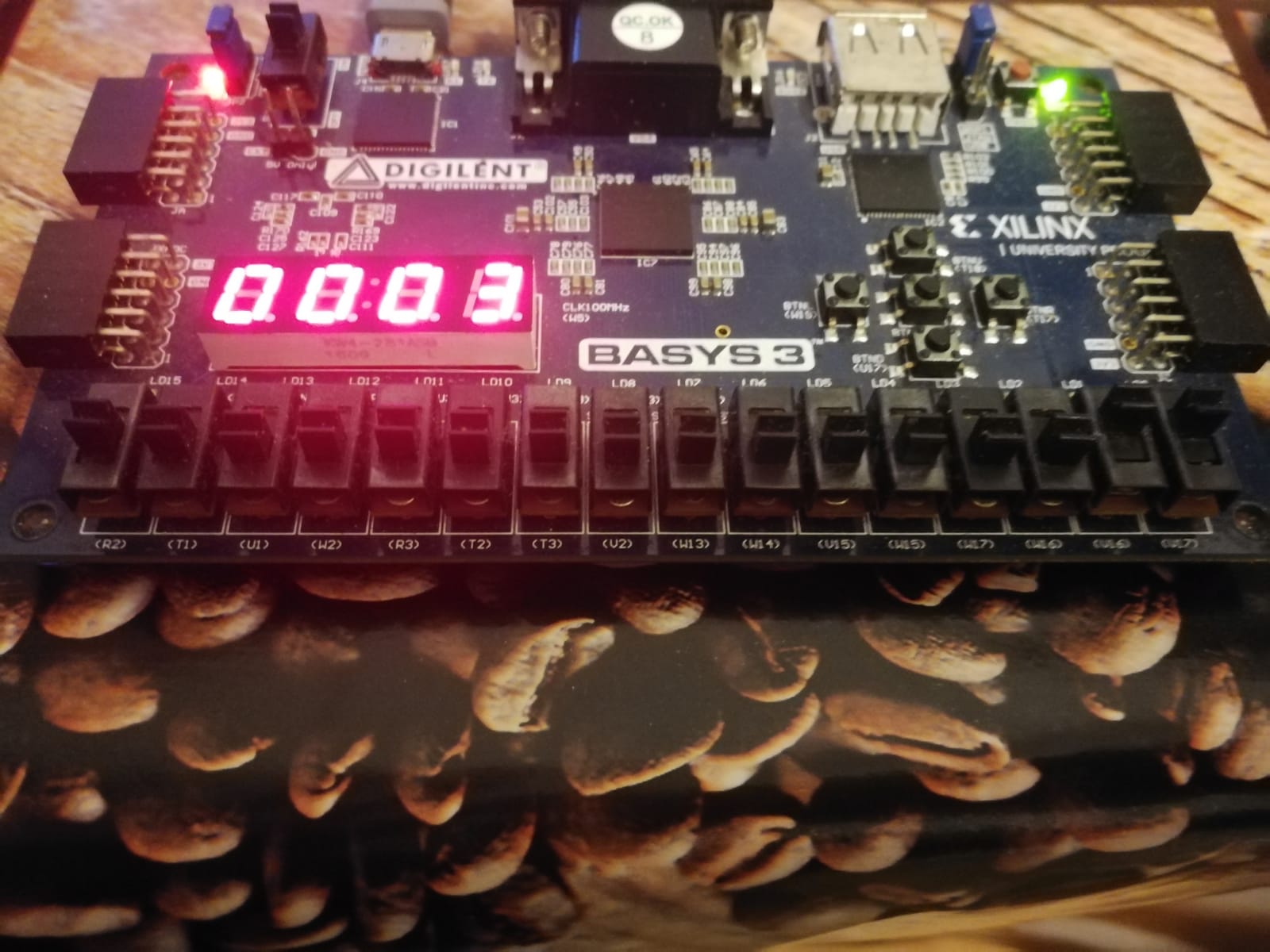
As expected, the result equals C= 000000000000011 (3 in decimal ) .



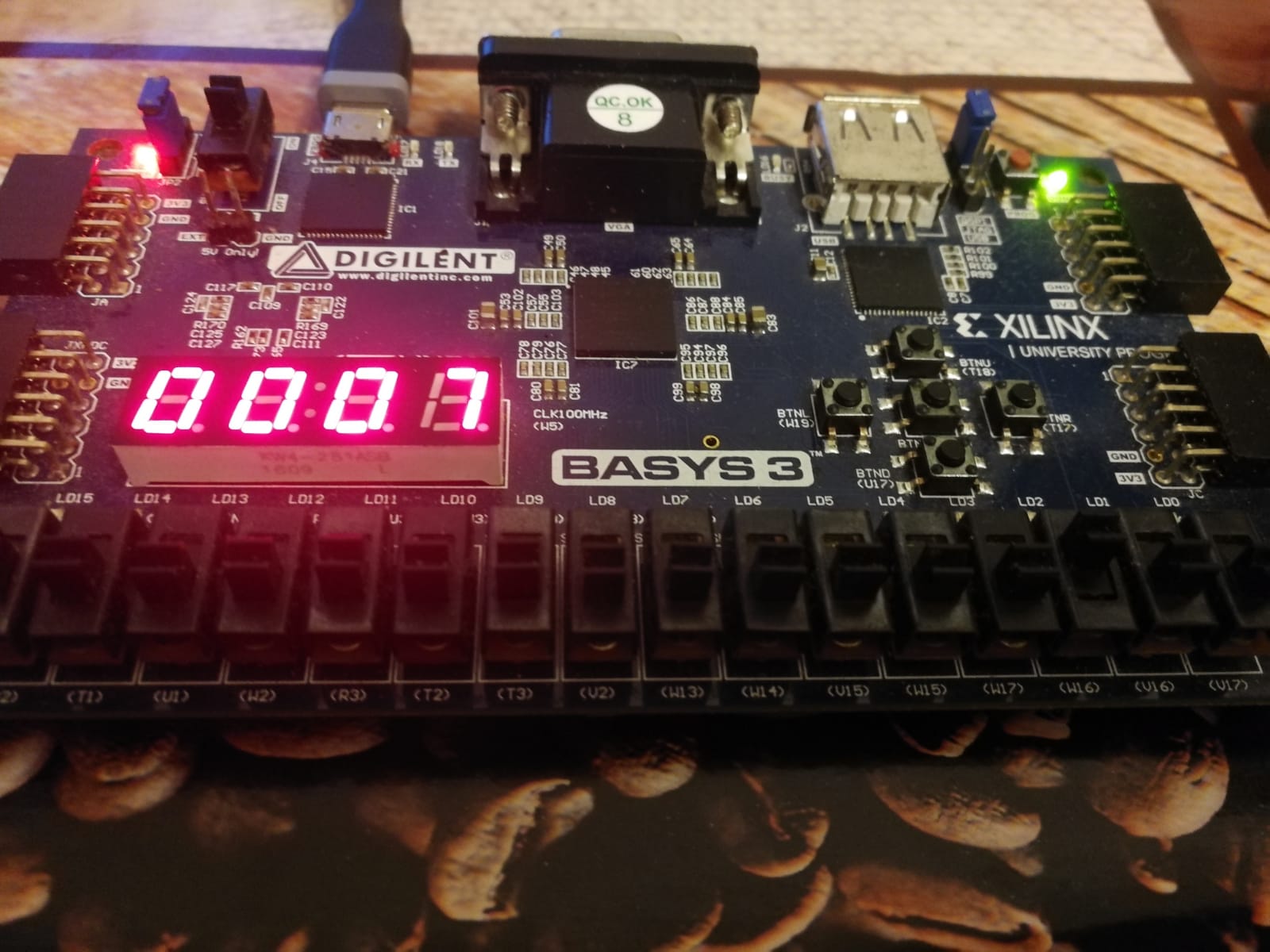
* + 1. **OR GATE**

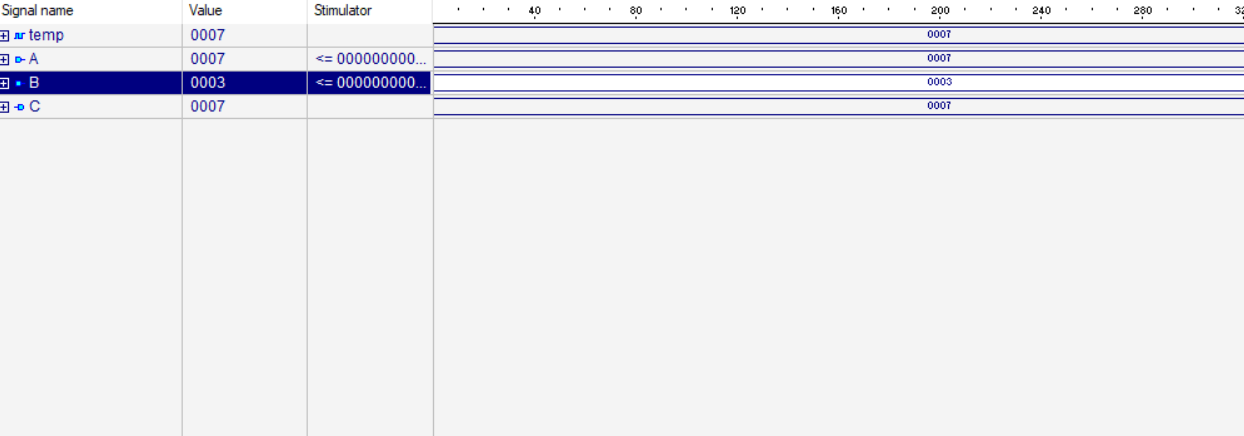
The Input were chosen as follows : A=7 ,B=3





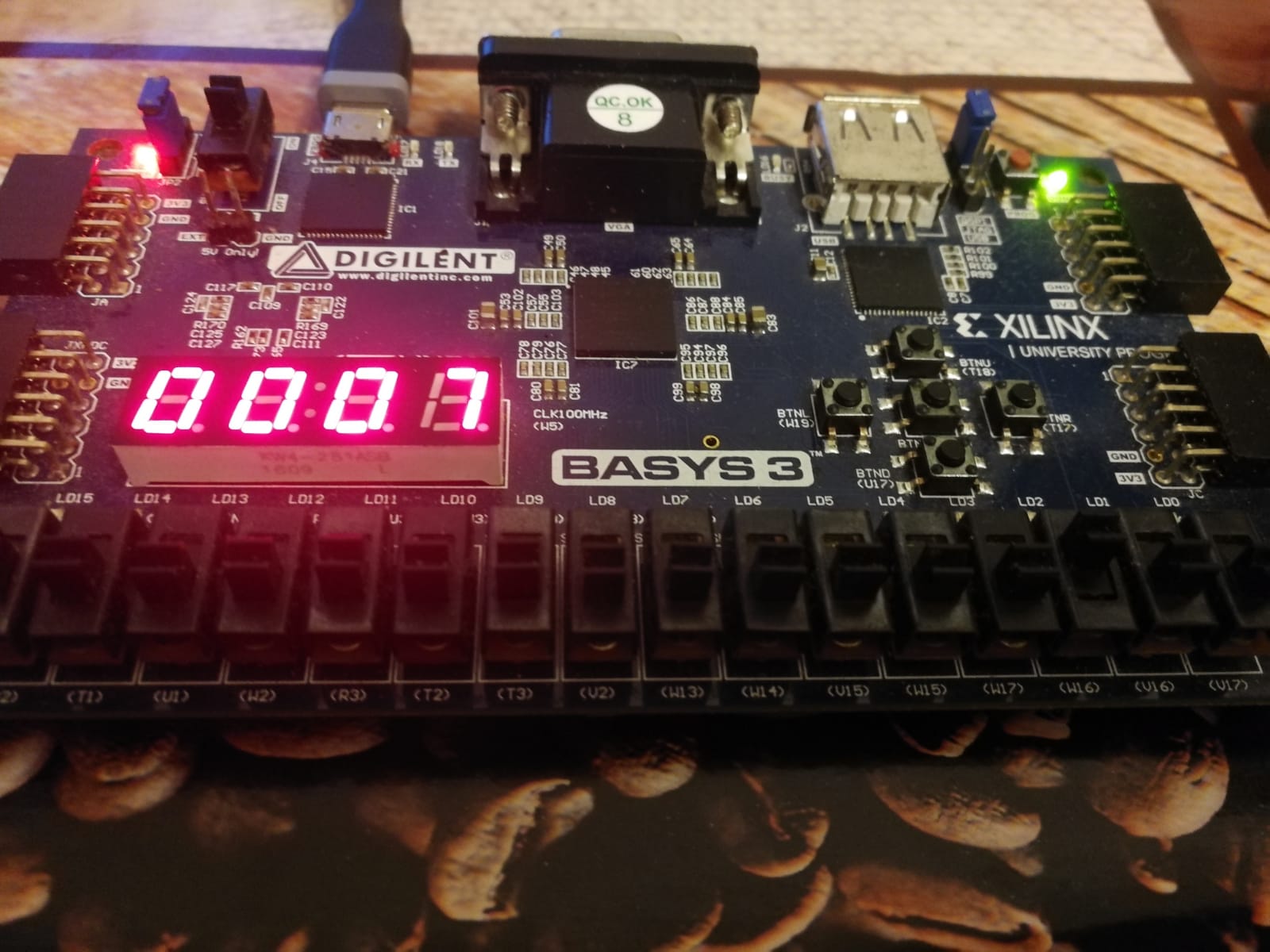
As expected, the result equals C= 000000000000111 (7 in decimal ) .

****

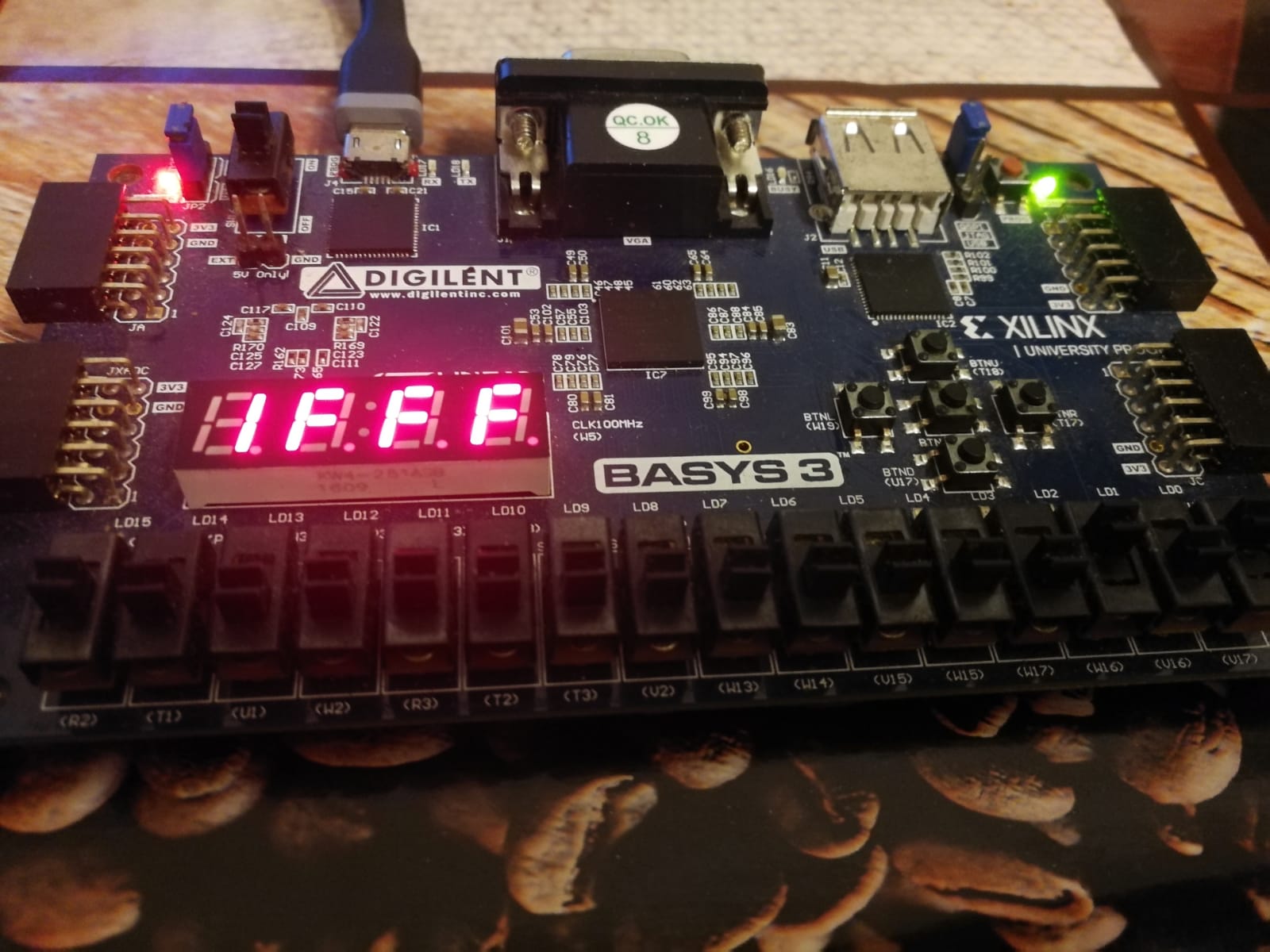


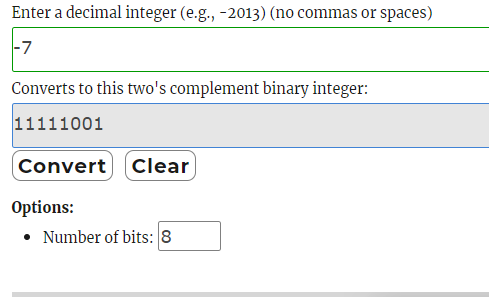
* + 1. **NOT GATE**

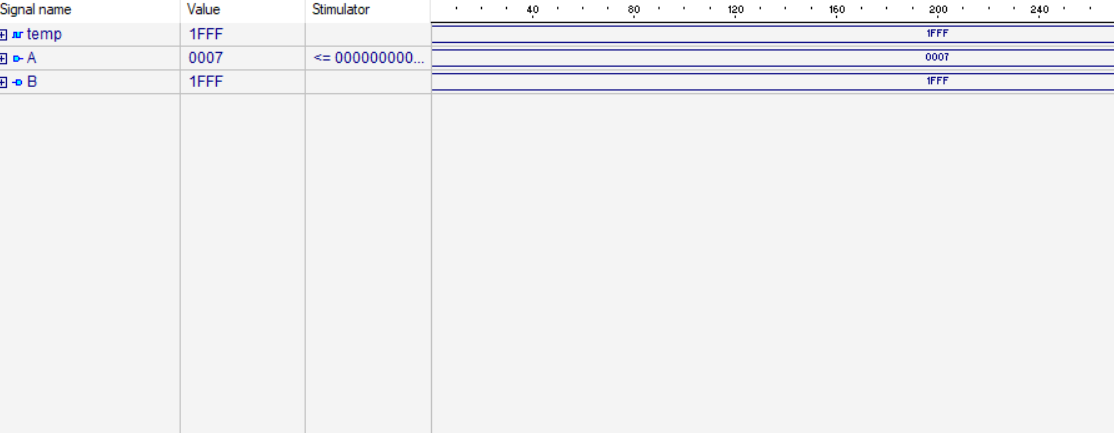
The Input were chosen as follows : A=7



As expected, the result equals B= FFF1 (- 7 in decimal ) .

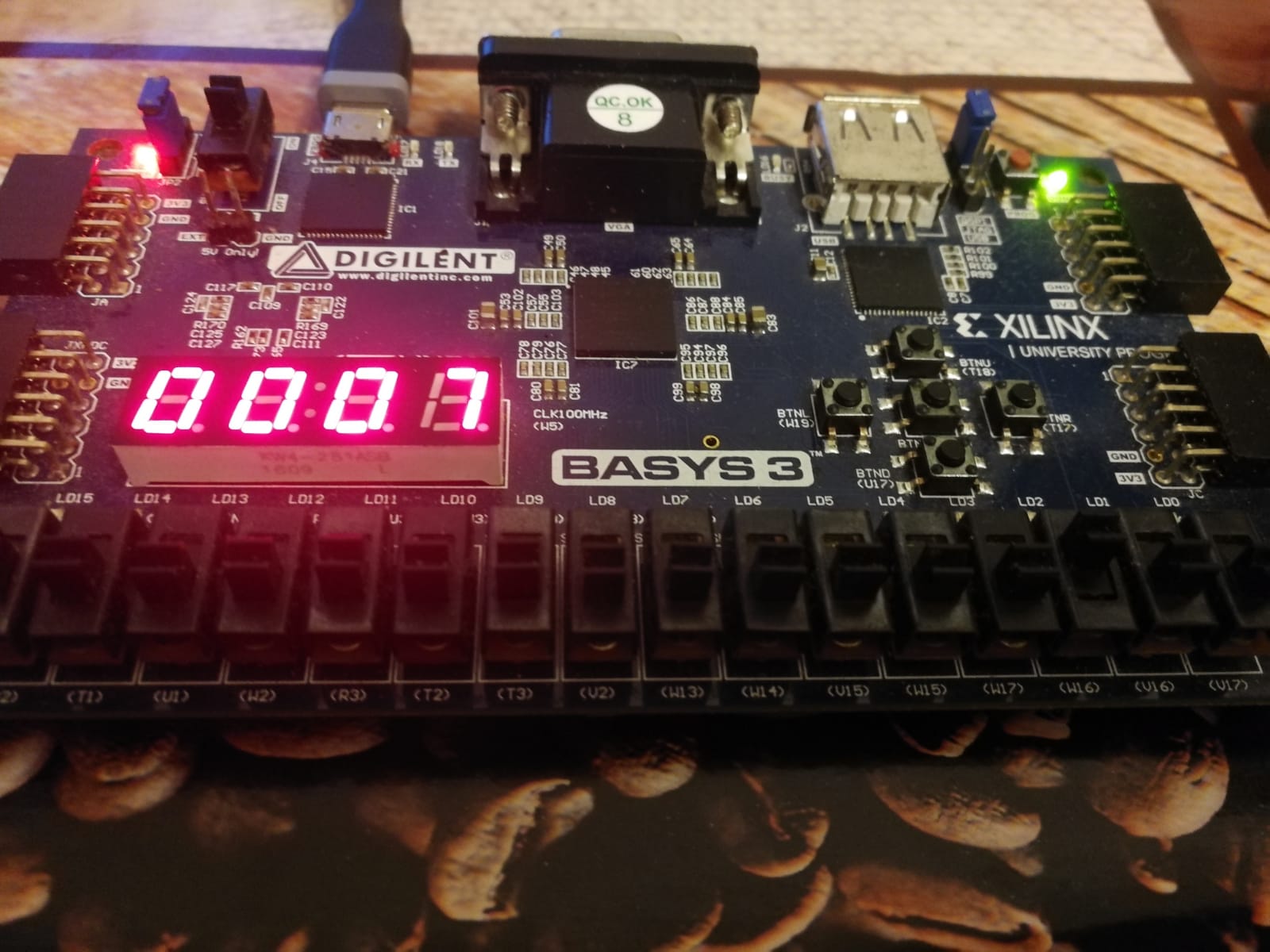




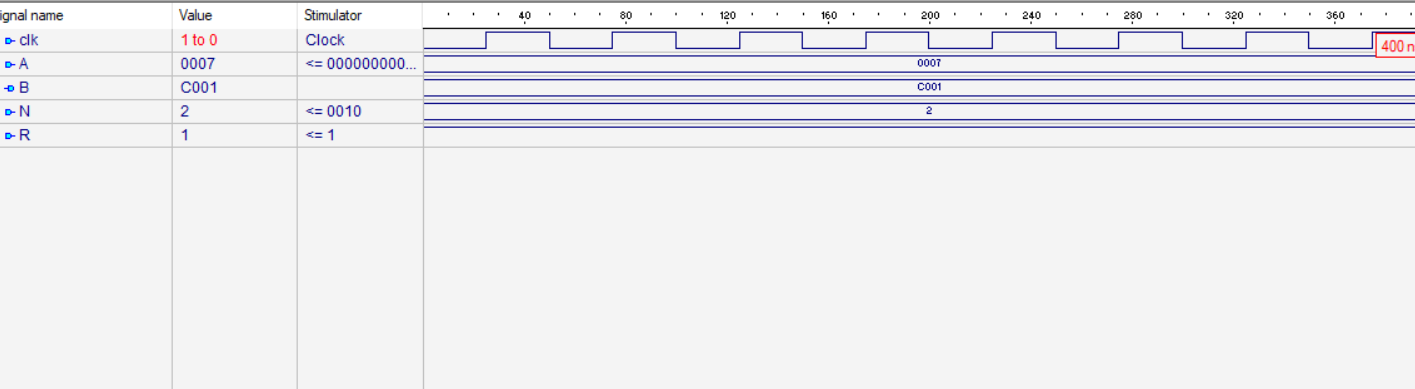


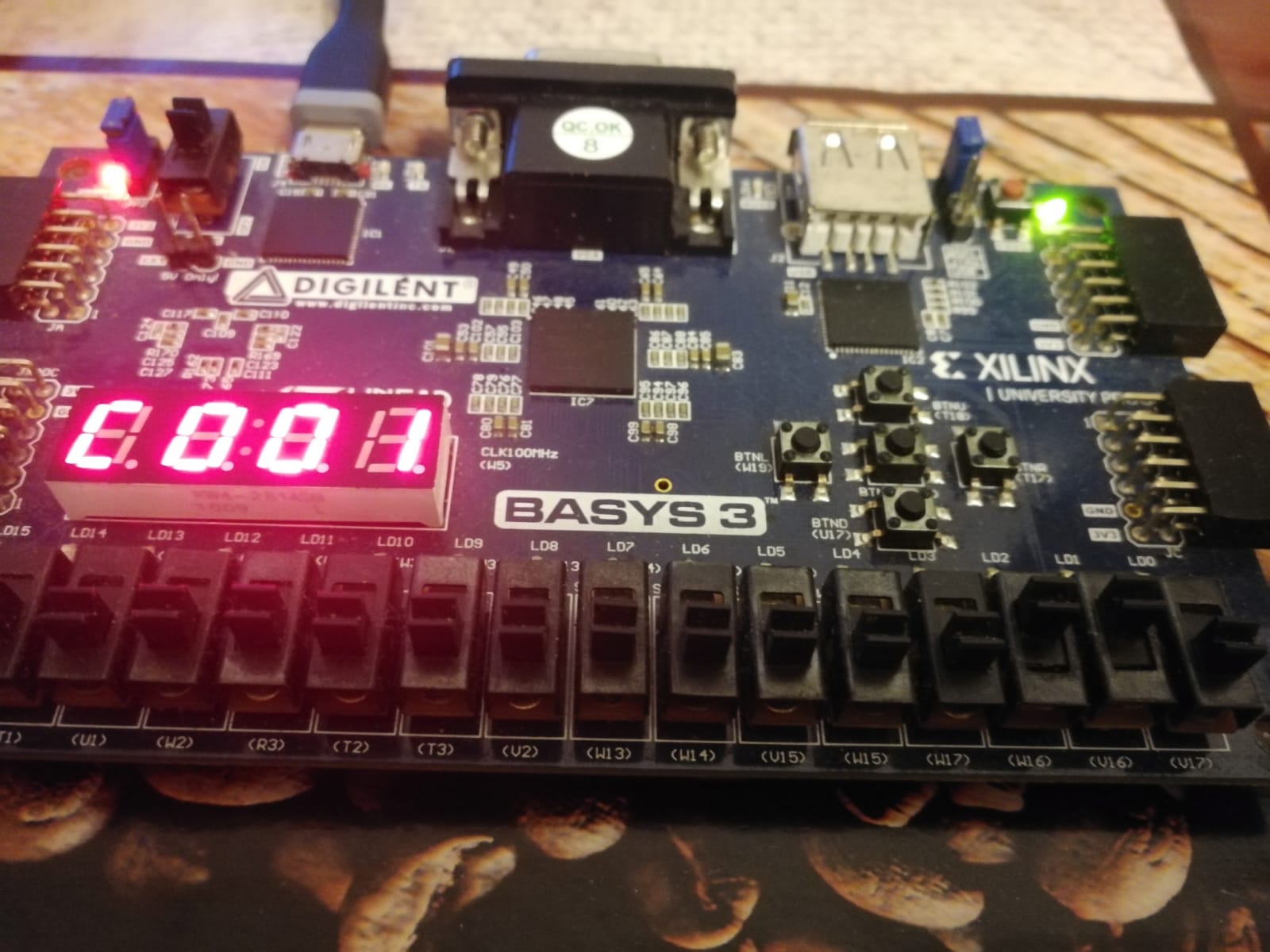
* + 1. **Rotate right**

In order to demonstrate the rotate operation we have assigned the value 0000000000000111 for the input A. We marked the operation as being right rotate by selecting R input to be 1 and we switched 2 bites.



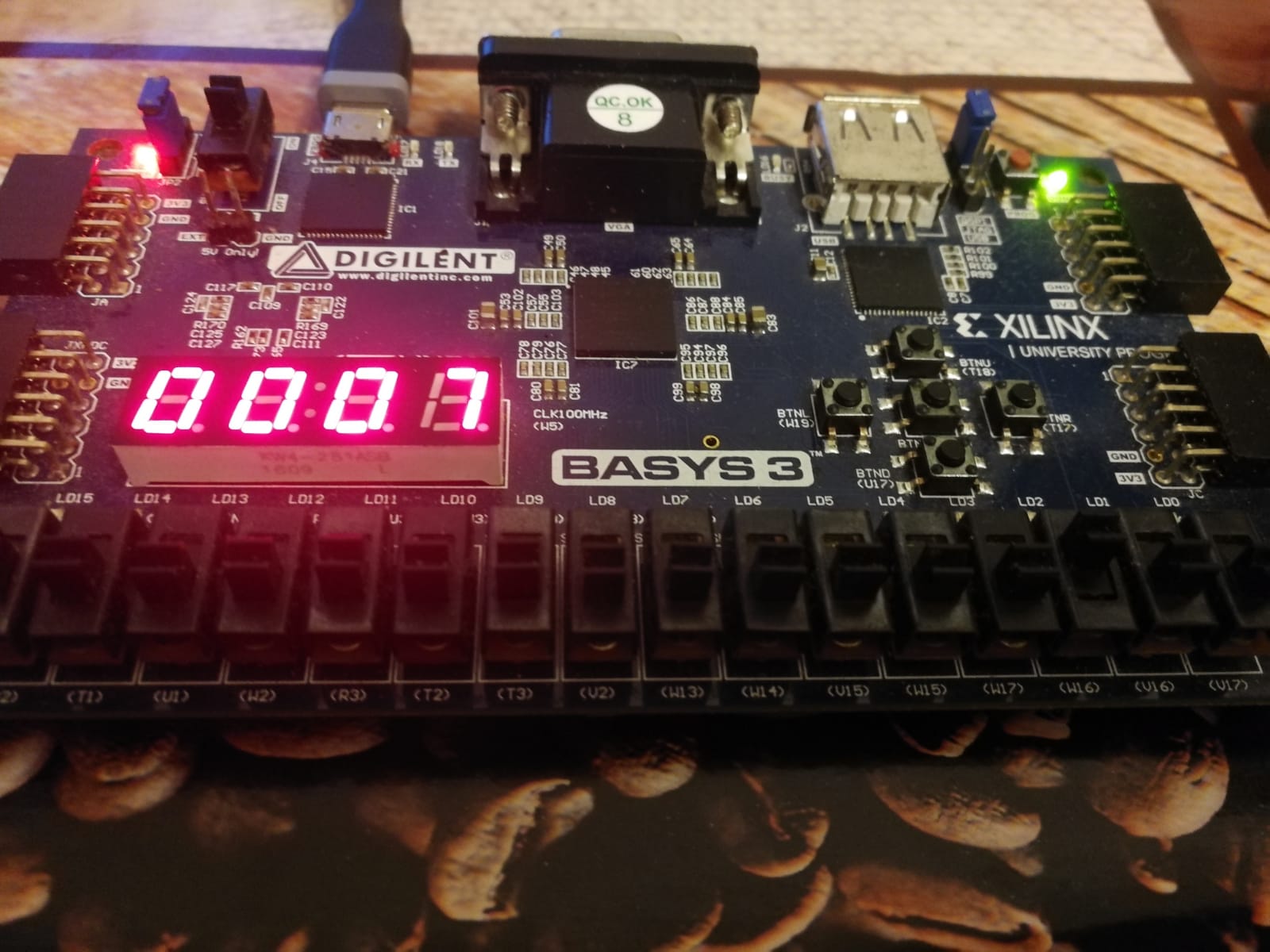
The result is B =1100 0000 000 0001 (C001) .



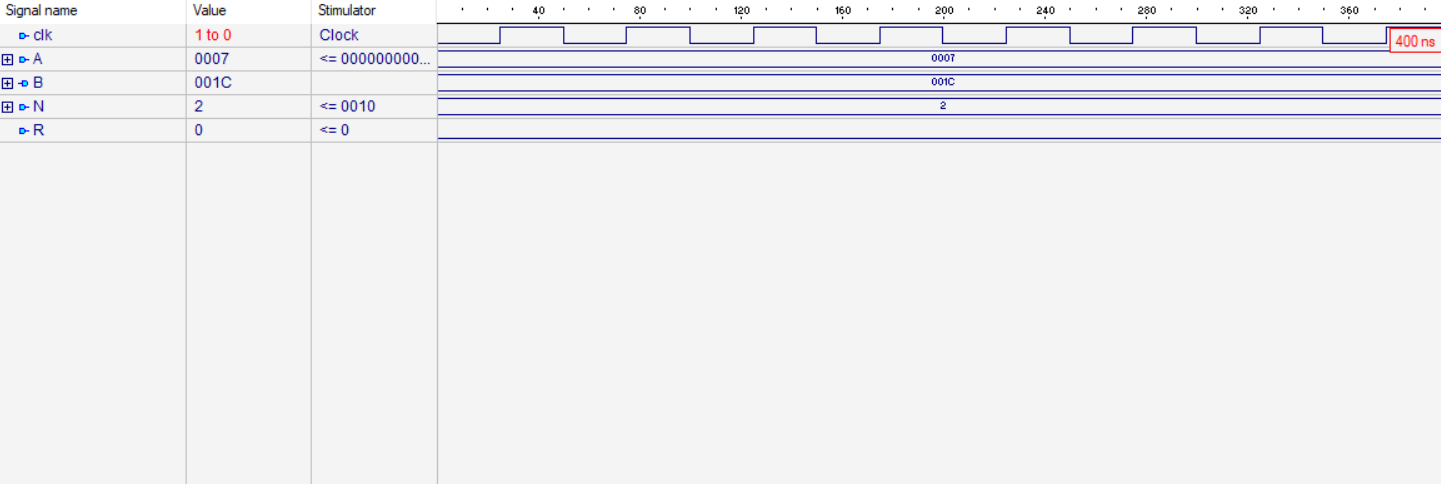
****

* + 1. **Rotate left**

In order to demonstrate the rotate operation we have assigned the value 0000000000000111 for the input A. We marked the operation as being left rotate by selecting R input to be 0 and we switched 2 bites.

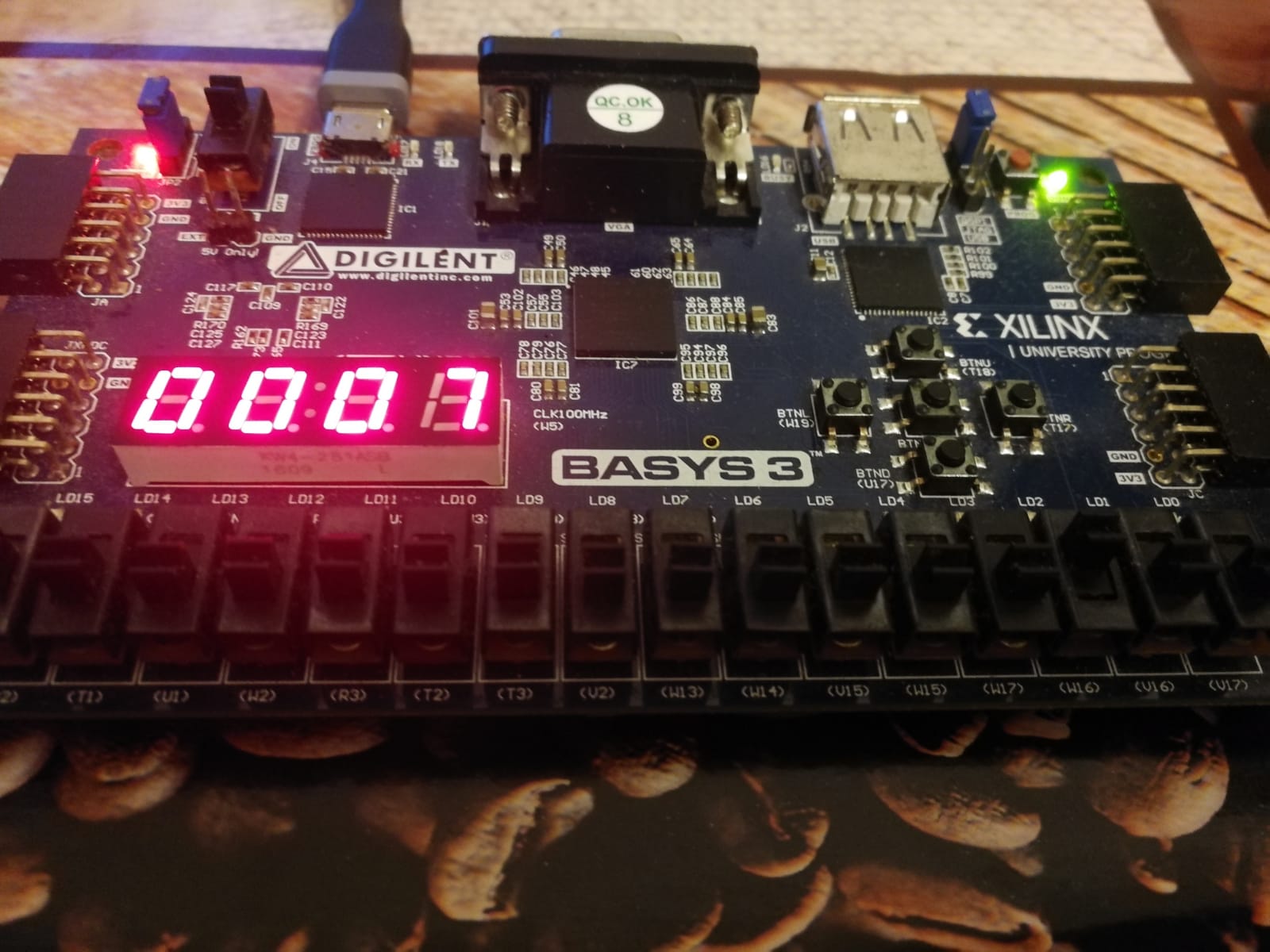


The result is B =0000 0000 001 1100 (001C) .

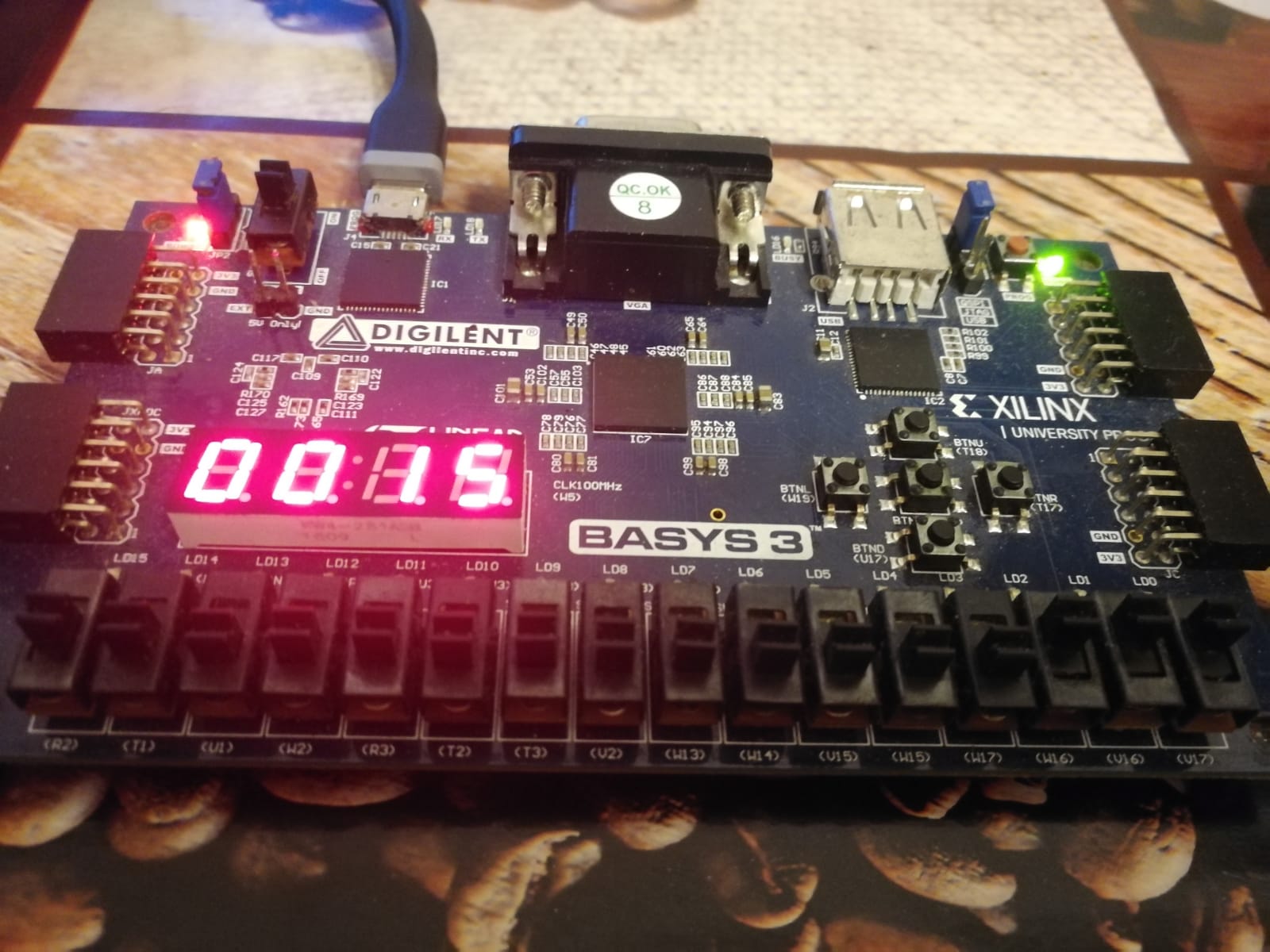


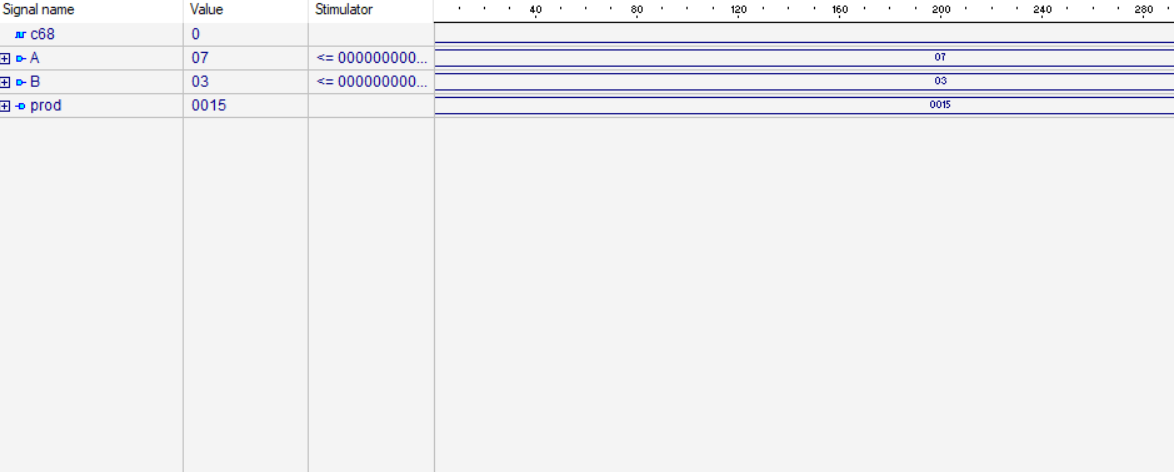
* + 1. ***Multiplication***

The Input were chosen as follows : A=7 ,B=3



As expected, the result equals PROD= 0000 0000 0001 0101 (21 in decimal ) .

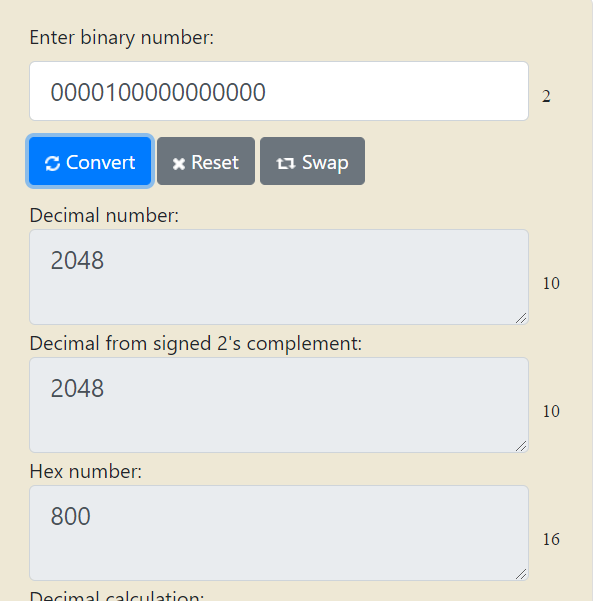
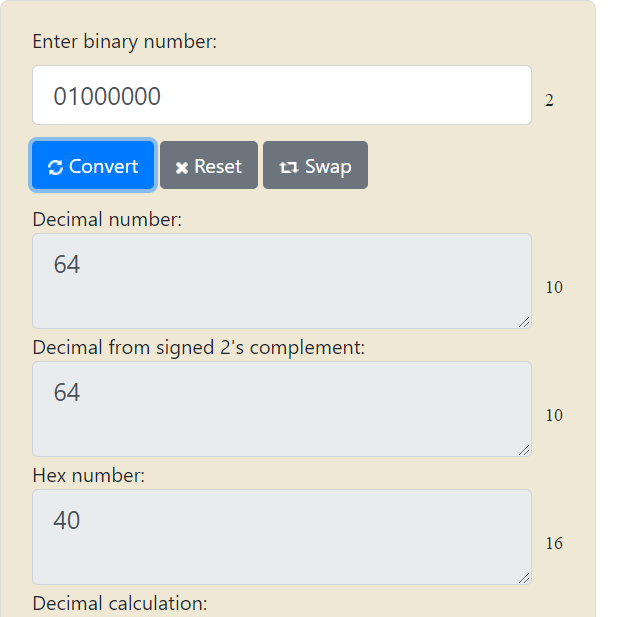




The Input were chosen as follows : A= 0000000001111000 (120),B=0000000000001100(12)

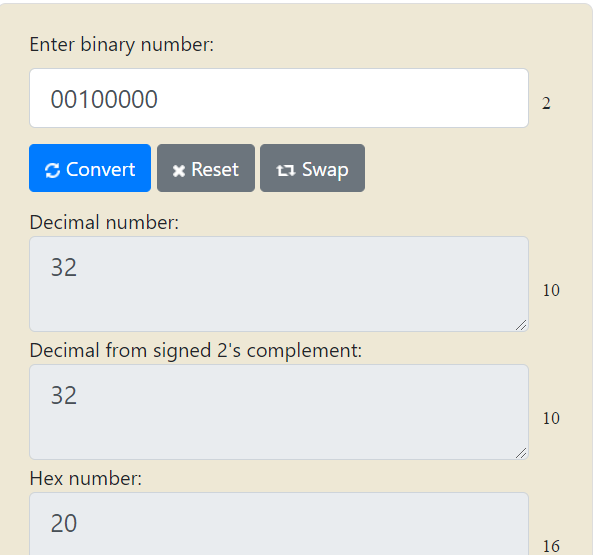
As expected, the result equals PROD= 0000010110100000 (1440 in decimal ) .



* + 1. ***Division***
* The Input were chosen as follows : A=2048 ,B=364
* 
* 

The result is represented by two parts : the quotient and the remainder.

For this input the results will be

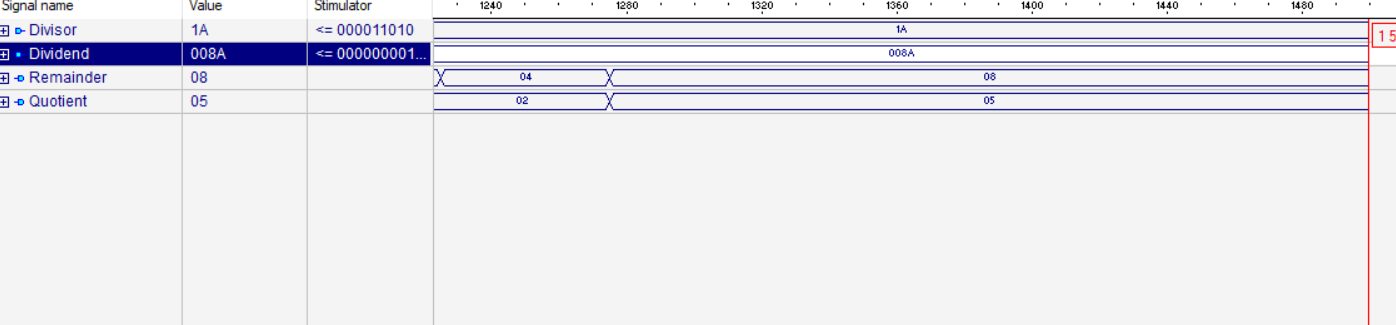


* Quotient = 0100 0000 (32 )
* Remainder = 0000 (0 )
* The divisor is 0 – division by 0

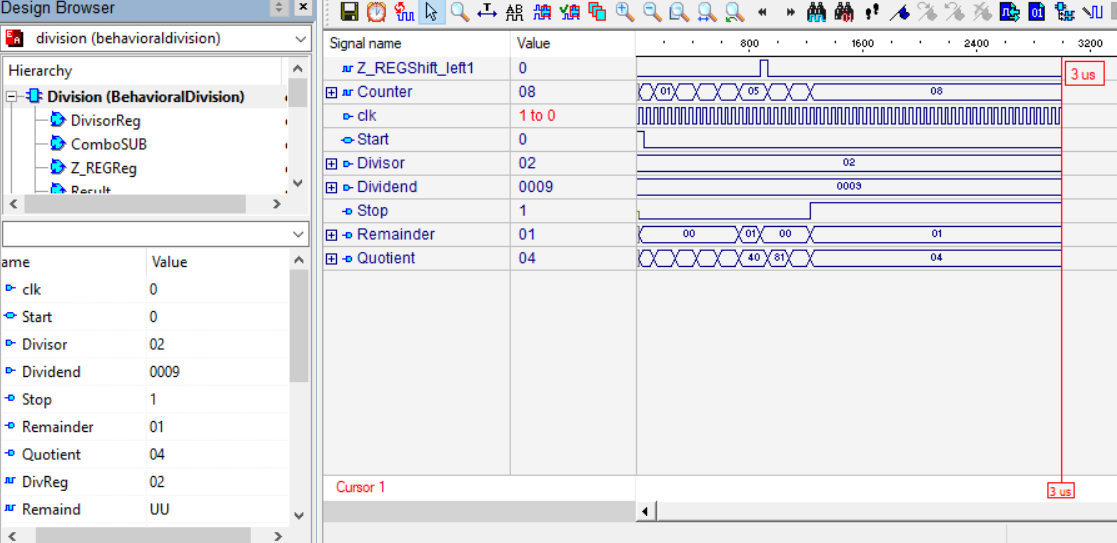
No matter the Dividend ( chosen 10) , if the divisor is 0 the quotient will be 0.

***B***

* A= 000110101 (138) ,B=00011010 (26)
* Quotient =0000 0101(5) Remainder= 0000 1000(8)

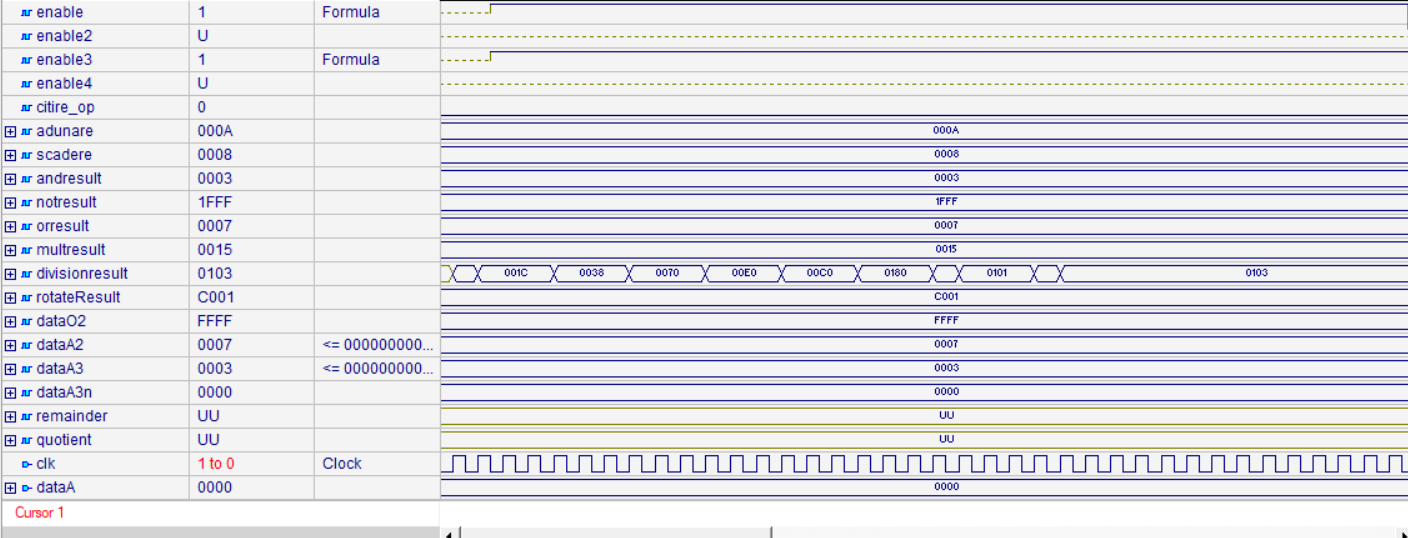
******

* A= 00001001 (9) ,B=00000010 (2)
* Quotient =0000 0100(4) Remainder= 0000 002(1)

******

* + 1. ***Overall results s***

For inputs A=7 and B=3.



***6.Conclusion***

6.1 My experiences

During the time working at this project I have used a lot of knowledge learned in the VHDL lectures and I have achieved a larger and deeper understanding of the language and its domain. I have gained a lot of experience with the simulation and the synthesis tools and it was very interesting to see how real hardware that most people use all day is described and seeing the expected results showing off.

Implementing the ALU s operations was a challenge mostly because the complexity and difficulty was not in designed the separate components and testing them individually but mostly in the whole connection block needed. The synchronization and with the Basys 3 board and between the components it was difficult but I belive that now I gained experience in the hierarchical design with components instantiation in the concurrent space .

6.2 Future Development

Despite the time spend on the project all the features were not implemented for Alu ,but , the main goal of implementing ,designing and testing the main operations was achieved. Of course , for the present Alu further features can be implemented as they are computers such as : operations with floating points .

***7.References***

Mihai Negru Computer Architecture Lectures

Sebestyen-Pal Gheorghe lectures and laboratory works

<https://en.wikipedia.org/wiki/ALU>

<http://www.csit-sun.pub.ro/courses/Masterat/Xilinx%20Synthesis%20Technology/toolbox.xilinx.com/docsan/xilinx4/data/docs/xst/hdlcode7.html>

<https://www.techopedia.com/definition/2849/arithmetic-logic-unit-alu>

<https://arith-matic.com/notebook/arithmetic-logic-unit-introduction>

<https://www.academia.edu/8034521/Structural_VHDL_Implementation_of_Wallace_Multiplier>

<https://www.allaboutcircuits.com/technical-articles/basic-binary-division-the-algorithm-and-the-vhdl-code/>

<https://www.ijareeie.com/upload/2016/april/17_DESIGN.pdf>

<https://www.exploringbinary.com/twos-complement-converter/>

<https://ncalculators.com/digital-computation/binary-division-calculator.htm>