

**DIGITAL SIGNAL FILTERING CIRCUIT**

***STUDENT: Zgavirdici Malina-Andreea***

***GROUP: 30433***

**CONTENT:**

1. Introduction……………………………………………………………3;
2. Objectives……………………………………………………………….4;
3. Theoretical Fundamentation…………………………………..5;
4. Implementation……………………………………………………..11;
5. Testing…………………………………………………………………..15;
6. Conclusions……………………………………………………………18;

***1.Introduction:***

In this section we intend to stress the topic of the project and create a basic description of what it implies. Here, we will defines some of the main terms that must be understood, we will state the final objective and, in the end of the chapter, we will present briefly the history of digital signal filtering circuits.

In order to properly introduce the concept of **digital signal filtering circuit**, one must gain a deep understanding of what both a digital signal and a filtering circuit are.

A digital signal refers to an electrical signal that is converted into a pattern of bits and has a discrete value at each sampling point.

A filter is a circuit capable of passing (or amplifying) certain frequencies while attenuating other frequencies. Thus, a filter can extract important frequencies from signals that also contain undesirable or irrelevant frequencies.

So, knowing that, we can proceed defining a **digital signal filtering circuit**, which  is a system that performs mathematical operations on a sampled, discrete-time signal to reduce or enhance certain aspects of that signal.

A digital filter is characterized by its [transfer function](https://en.wikipedia.org/wiki/Transfer_function), or equivalently, its [difference equation](https://en.wikipedia.org/wiki/Difference_equation). Mathematical analysis of the transfer function can describe how it will respond to any input.

Our final goal is to develop a **digital signal filtering circuit** characterised by the following transfer function:

Y(k) = X(k) \* a1 + X(k-1) \* a2 + X(k-2) \* a3

where : - X(i), i = k, k-1, k-2 represent the input signals

* Y(k) represents the output signal obtained after filtering
* A1, a2, a3 represent fixed coefficients that the user is free to choose

We will make use of VHDL language. **VHDL (VHSIC-HDL)** (**Very High Speed Integrated Circuit Hardware Description Language**) is a [hardware description language](https://en.wikipedia.org/wiki/Hardware_description_language) used in [electronic design automation](https://en.wikipedia.org/wiki/Electronic_design_automation) to describe [digital](https://en.wikipedia.org/wiki/Digital_electronics) and [mixed-signal](https://en.wikipedia.org/wiki/Mixed-signal_integrated_circuit) systems such as [field-programmable gate arrays](https://en.wikipedia.org/wiki/Field-programmable_gate_array) and [integrated circuits](https://en.wikipedia.org/wiki/Integrated_circuit). VHDL can also be used as a general purpose [parallel programming language](https://en.wikipedia.org/wiki/Parallel_programming_language).

***2.Objectives:***

In this section we aim to state clearly our objectives, to specify the steps that need to be followed in order to reach our final goal, to decide what programming language is to be used and briefly describe its advantages and disadvantages and finally, to explain the working mechanism of the application.

The main objective is designing a **digital signal filtering circuit**. To do so we will make a flow diagram, then split the big circuit into components. After that we will implement each component separately and in the end we will connect all the components.

***3.Theoretical Fundamentation:***

In signal processing, the function of a filter is to remove unwanted parts of the signal, such as random noise, or to extract useful parts of the signal, such as the components lying within a certain frequency range. The following block diagram illustrates the basic idea.

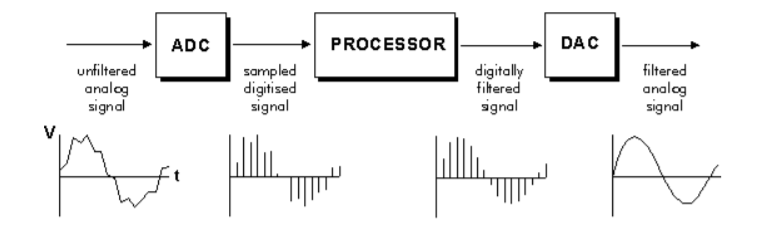


An analog filter uses analog electronic circuits made up from components such as resistors, capacitors and op amps to produce the required filtering effect. Such filter circuits are widely used in such applications as noise reduction, video signal enhancement, graphic equalisers in hi-fi systems, and many other areas.

There are well-established standard techniques for designing an analog filter circuit for a given requirement. At all stages, the signal being filtered is an electrical voltage or current which is the direct analogue of the physical quantity (e.g. a sound or video signal or transducer output) involved.

A digital filter uses a digital processor to perform numerical calculations on sampled values of the signal. The processor may be a general-purpose computer such as a PC, or a specialised DSP (Digital Signal Processor) chip.

The analog input signal must first be sampled and digitised using an ADC (analog to digital converter). The resulting binary numbers, representing successive sampled values of the input signal, are transferred to the processor, which carries out numerical calculations on them. These calculations typically involve multiplying the input values by constants and adding the products together. If necessary, the results of these calculations, which now represent sampled values of the filtered signal, are output through a DAC (digital to analog converter) to convert the signal back to analog form.

Note that in a digital filter, the signal is represented by a sequence of numbers, rather than a voltage or current. The following diagram shows the basic setup of such a system.

A DAC converts an [abstract](https://en.wikipedia.org/wiki/Abstract_object) finite-precision number (usually a [fixed-point](https://en.wikipedia.org/wiki/Fixed-point_arithmetic) [binary number](https://en.wikipedia.org/wiki/Binary_number)) into a physical quantity (e.g., a [voltage](https://en.wikipedia.org/wiki/Voltage) or a [pressure](https://en.wikipedia.org/wiki/Pressure)). In particular, DACs are often used to convert finite-precision [time series](https://en.wikipedia.org/wiki/Time_series) data to a continually varying physical [signal](https://en.wikipedia.org/wiki/Signal).

An *ideal* DAC converts the abstract numbers into a conceptual sequence of [impulses](https://en.wikipedia.org/wiki/Dirac_delta_function) that are then processed by a [reconstruction filter](https://en.wikipedia.org/wiki/Reconstruction_filter) using some form of [interpolation](https://en.wikipedia.org/wiki/Interpolation) to fill in data between the impulses. A conventional *practical* DAC converts the numbers into a [piecewise constant function](https://en.wikipedia.org/wiki/Piecewise_constant_function) made up of a sequence of [rectangular functions](https://en.wikipedia.org/wiki/Rectangular_function) that is modeled with the [zero-order hold](https://en.wikipedia.org/wiki/Zero-order_hold). Other DAC methods (such as those based on [delta-sigma modulation](https://en.wikipedia.org/wiki/Delta-sigma_modulation)) produce a [pulse-density modulated](https://en.wikipedia.org/wiki/Pulse-density_modulation) output that can be similarly filtered to produce a smoothly varying signal.[2]

**Operation of digital filters**

In this section, we will develop the basic theory of the operation of digital filters. This is essential to an understanding of how digital filters are designed and used.

Suppose the "raw" signal which is to be digitally filtered is in the form of a voltage waveform described by the function

V = x(t)

where t is time.

This signal is sampled at time intervals h (the sampling interval). The sampled value at time t = ih is

Xi= x(ih)

Thus the digital values transferred from the ADC to the processor can be represented by the sequence

x0 , x1 , x2 , x3 , ...

corresponding to the values of the signal waveform at

t = 0, h, 2h, 3h, ...

and t = 0 is the instant at which sampling begins.

At time t = nh (where n is some positive integer), the values available to the processor, stored in memory, are

X0 , x1 , x2 , x3 , ... xn

Note that the sampled values xn+1, xn+2 etc. are not available, as they haven't happened yet!

The digital output from the processor to the DAC consists of the sequence of values

Y0 , y1 , y2 , y3 , ... yn

In general, the value of yn is calculated from the values x0, x1, x2, x3, ... , xn. The way in which the y's are calculated from the x's determines the filtering action of the digital filter. [1]

**Digital filter coefficients**

All of the digital filter examples given above can be written in the following general forms:

Zero order: yn = a0 xn

First order: yn = a0 xn + a1 xn-1

Second order: yn = a0 xn + a1 xn-1 + a2xn-2

Similar expressions can be developed for filters of any order.

The constants a0, a1, a2, ... appearing in these expressions are called the filter coefficients. It is the values of these coefficients that determine the characteristics of a particular filter.

Our digital filter is a second order filter.

The Black Box of our design:

X(k)

BLACK BOX

Y(k)

CLK

OK\_BTN

EQ\_BTN

LED

The White Box of our design:

rreg1

Reg1

Mul1

Add1

Add2

Mul2

Reg2

Mul3

Reg3

Main Components:

-REGISTER

- MULIPLIER

- ADDER

**REGISTER:**

A register may hold an [instruction](https://whatis.techtarget.com/definition/instruction), a storage address, or any kind of data (such as a bit sequence or individual characters). Some instructions specify registers as part of the instruction. For example, an instruction may specify that the contents of two defined registers be added together and then placed in a specified register.[3]

**MULTIPLIER:**

A **binary multiplier** is an [electronic circuit](https://en.wikipedia.org/wiki/Electronic_circuit) used in [digital electronics](https://en.wikipedia.org/wiki/Digital_electronics), such as a [computer](https://en.wikipedia.org/wiki/Computer), to [multiply](https://en.wikipedia.org/wiki/Multiplication) two [binary numbers](https://en.wikipedia.org/wiki/Binary_number). It is built using [binary adders](https://en.wikipedia.org/wiki/Binary_adder).

A variety of [computer arithmetic](https://en.wikipedia.org/wiki/Category:Computer_arithmetic) techniques can be used to implement a digital multiplier. Most techniques involve computing a set of *partial products*, and then summing the partial products together. This process is similar to the method taught to primary schoolchildren for conducting long multiplication on base-10 integers, but has been modified here for application to a base-2 ([binary](https://en.wikipedia.org/wiki/Binary_numeral_system)) [numeral system](https://en.wikipedia.org/wiki/Numeral_system).[3]

**ADDER:**

An **adder** is a [digital circuit](https://en.wikipedia.org/wiki/Digital_circuit) that performs [addition](https://en.wikipedia.org/wiki/Addition) of numbers. In many [computers](https://en.wikipedia.org/wiki/Computer) and other kinds of [processors](https://en.wikipedia.org/wiki/Microprocessor) adders are used in the [arithmetic logic units](https://en.wikipedia.org/wiki/Arithmetic_logic_unit) or ALU. They are also used in other parts of the processor, where they are used to calculate [addresses](https://en.wikipedia.org/wiki/Address_(computing)), table indices, [increment and decrement operators](https://en.wikipedia.org/wiki/Increment_and_decrement_operators), and similar operations.[3]

**4.Implementation:**

In this part we will take each module, explain the inputs and outputs and the way it works. We will stress the most important aspects of each module and also we will describe the algorithms used.

**ADDER:**

Inputs : 2 digits (A and B) and the carry in (Cin)

Outputs : the result of the addition (S) and the carry out (Cout).

Formulae used:

* For S : S <= A XOR B XOR Cin ;
* For Cout : Cout <= (A AND B) OR (Cin AND A) OR (Cin AND B) ;

This is an one bit full adder. In order to add numbers of 4 bits we have to cascade 4 such adders.

**MULTIPLIER:**

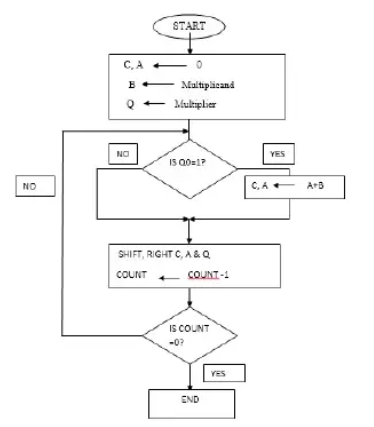
The multiplier works after the Booth’s Multiplication algorithm.

Next, we will describe the algorithm:

**Booth's multiplication algorithm** is a [multiplication algorithm](https://en.wikipedia.org/wiki/Multiplication_algorithm) that multiplies two signed [binary](https://en.wikipedia.org/wiki/Base_2) numbers in [two's complement notation](https://en.wikipedia.org/wiki/Two%27s_complement). Booth's algorithm examines adjacent pairs of [bits](https://en.wikipedia.org/wiki/Bit) of the 'N'-bit multiplier *Y* in signed [two's complement](https://en.wikipedia.org/wiki/Two%27s_complement) representation, including an implicit bit below the [least significant bit](https://en.wikipedia.org/wiki/Least_significant_bit), *y*−1 = 0. For each bit *yi*, for *i* running from 0 to *N* − 1, the bits *yi* and *yi*−1 are considered. Where these two bits are equal, the product accumulator *P* is left unchanged. Where *yi* = 0 and *yi*−1 = 1, the multiplicand times 2*i* is added to *P*; and where *y*i = 1 and *y*i−1 = 0, the multiplicand times 2*i* is subtracted from *P*. The final value of *P* is the signed product.

The representations of the multiplicand and product are not specified; typically, these are both also in two's complement representation, like the multiplier, but any number system that supports addition and subtraction will work as well. As stated here, the order of the steps is not determined. Typically, it proceeds from [LSB](https://en.wikipedia.org/wiki/Least_significant_bit) to [MSB](https://en.wikipedia.org/wiki/Most_significant_bit), starting at *i* = 0; the multiplication by 2*i* is then typically replaced by incremental shifting of the *P* accumulator to the right between steps; low bits can be shifted out, and subsequent additions and subtractions can then be done just on the highest *N* bits of *P*.[[2]](https://en.wikipedia.org/wiki/Booth%27s_multiplication_algorithm#cite_note-Chen_1992-2) There are many variations and optimizations on these details.....

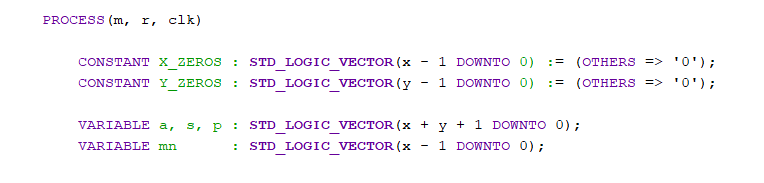
The algorithm is often described as converting strings of 1s in the multiplier to a high-order +1 and a low-order −1 at the ends of the string. When a string runs through the MSB, there is no high-order +1, and the net effect is interpretation as a negative of the appropriate value.



-Inputs : m and r (the numbers to be multiplied) and clk

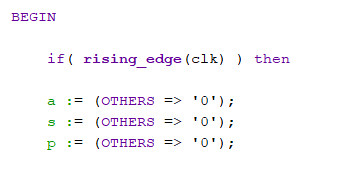
-Output : result (the product)

-Functions:

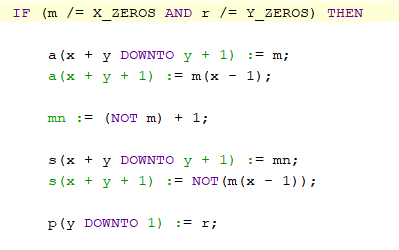


Inside the process, we declare the signals needed in for the realisation of the algorithm.

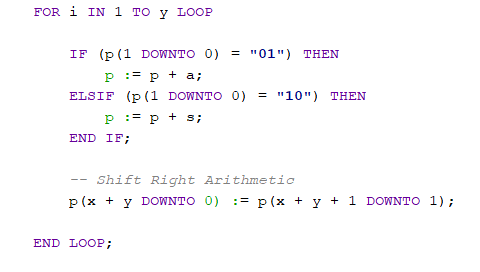
The firs 2 signals are initially 0.



When the process begins, we make give to a, s and p the value 0.



If m is not yet equal to X\_zeros and r is not yet equal to Y\_zeros then changes in a and s are made as seen above and the last y downto 1 bits of the product take the value of r.

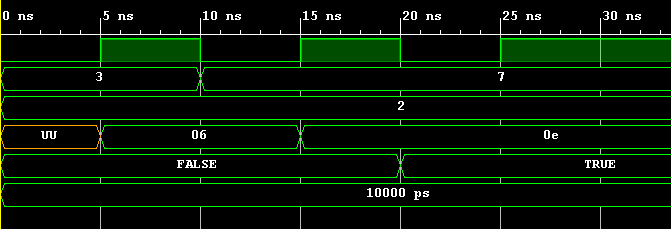


Then, into a loop from 1 to y, according to the last 2 bits of the intermediate product we add to it either a or s. Next we perform a shift to the right and proceed to the next iteration.

**5.Testing:**

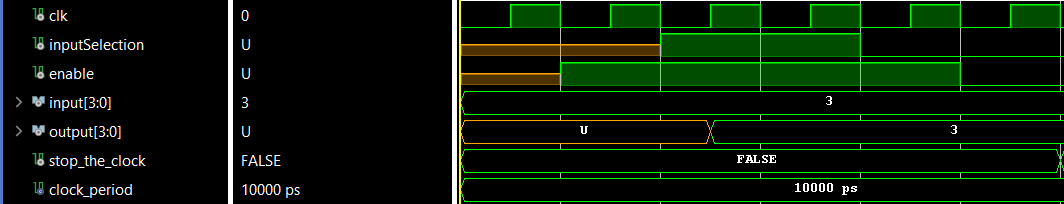
In this chapter we will test some key points of the project (that is the multiplication, the registration and the adder).

Multiplication testbench:



As it can be seen, the multiplication works as desired, every new answer being given at rising\_edge(clk).

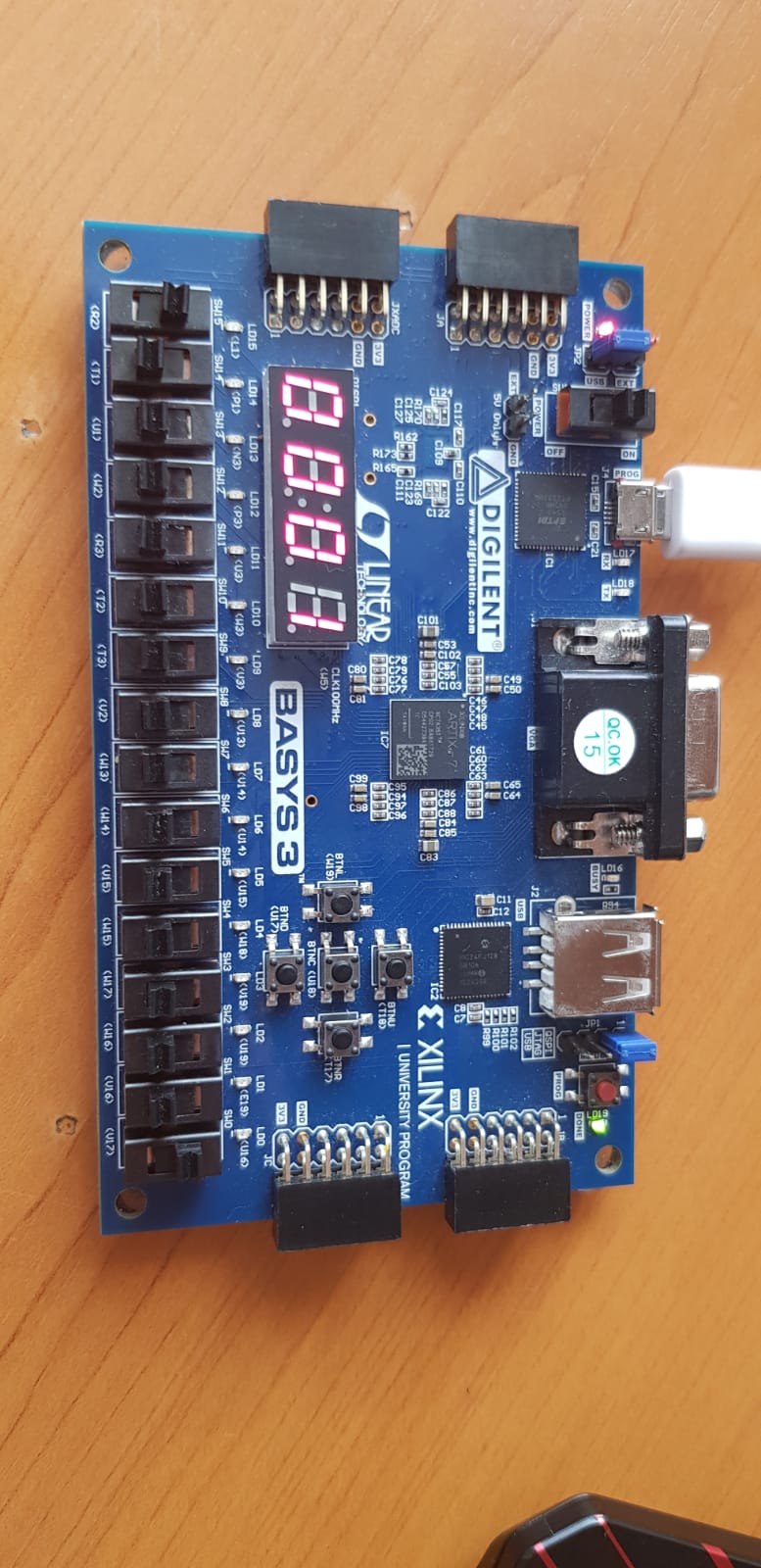
Registration testbench:



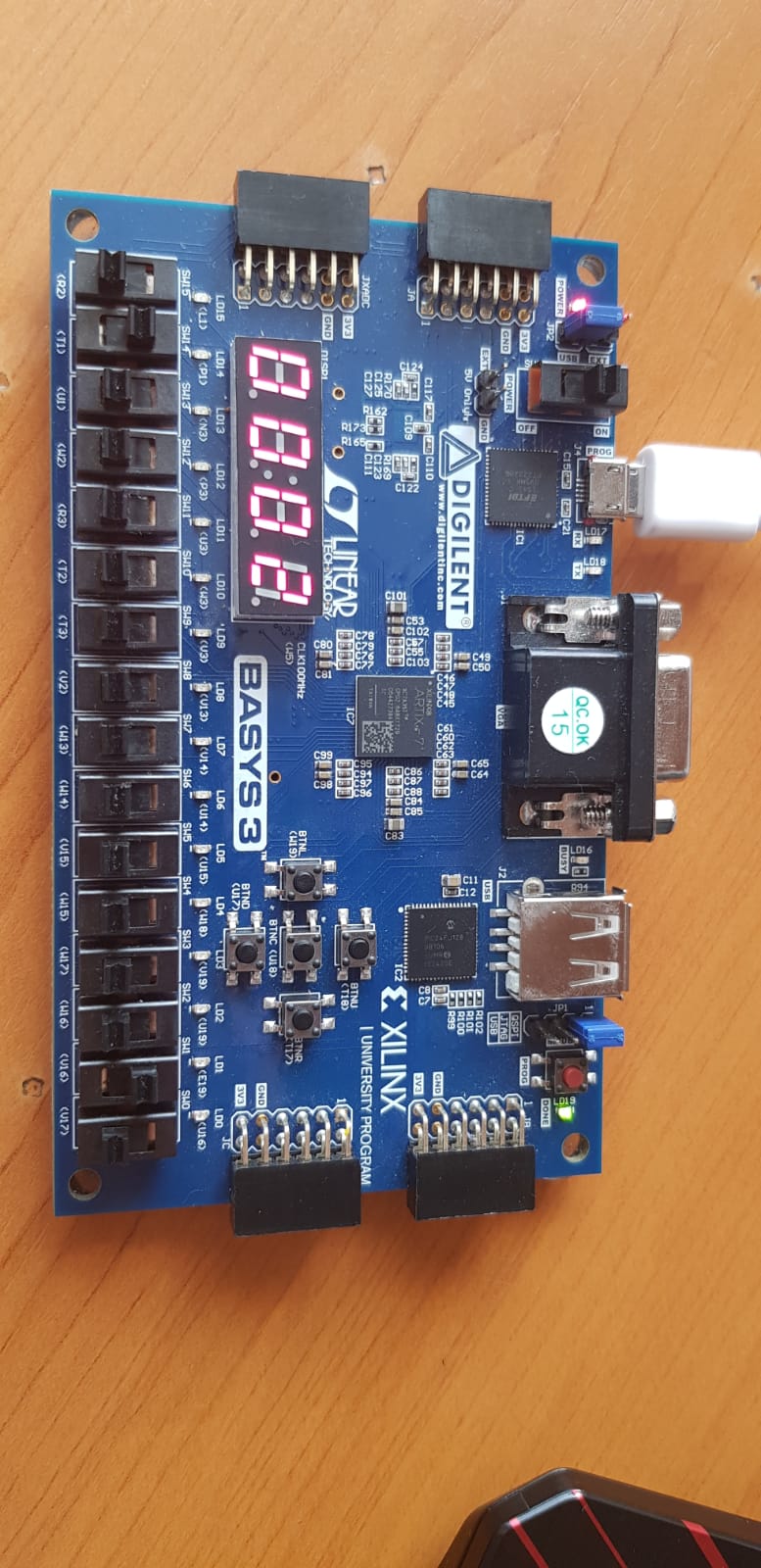
The register works properly, ottput value remains 3 even after the enable signals are gone.

Next, some photos of the testing made on the actual board.

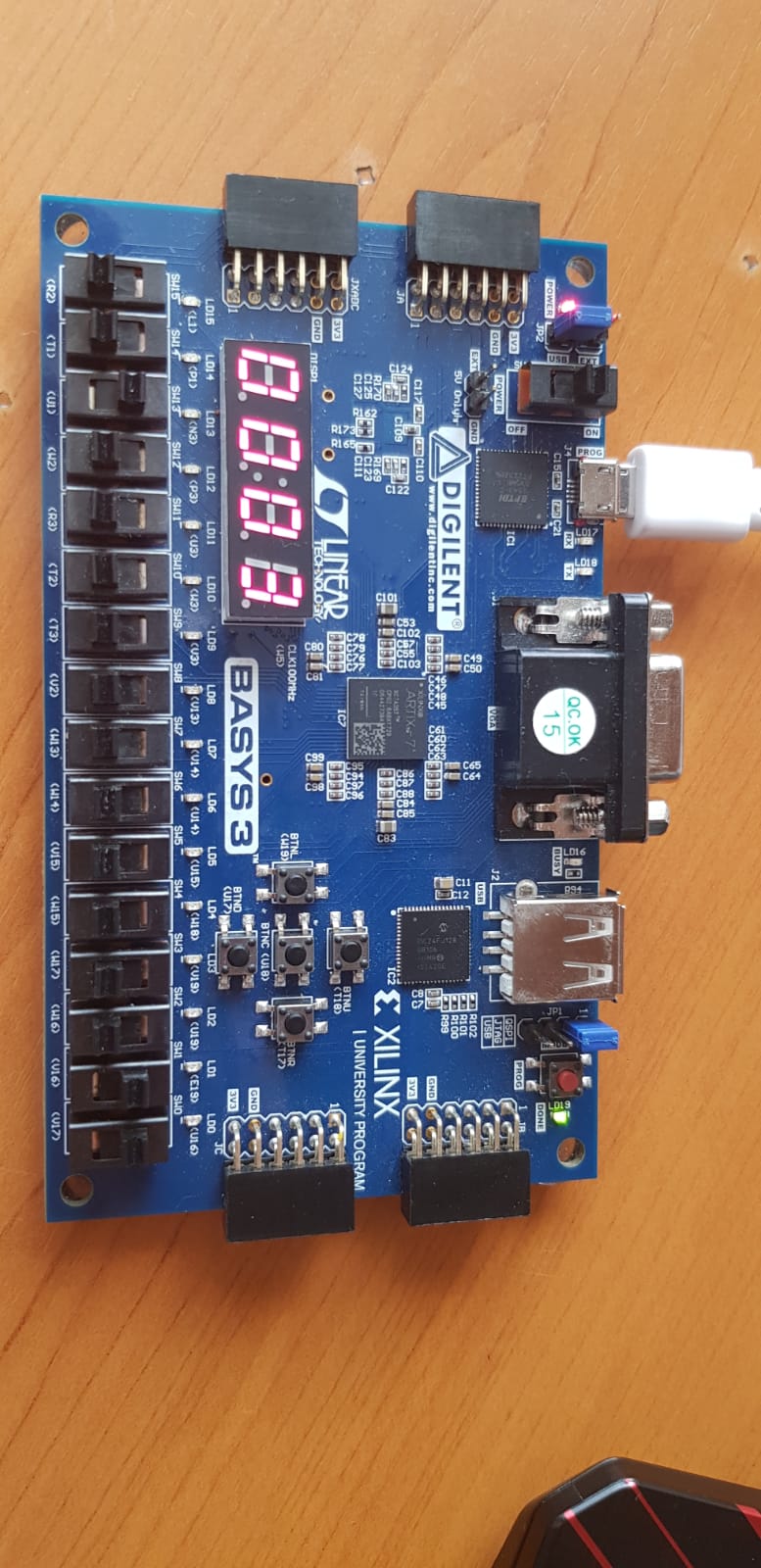
As introduction we mention that the 3 signals that are user-given are: “0001”, “0010”, “0011” and are given using switches 3, 2, 1 and 0

****

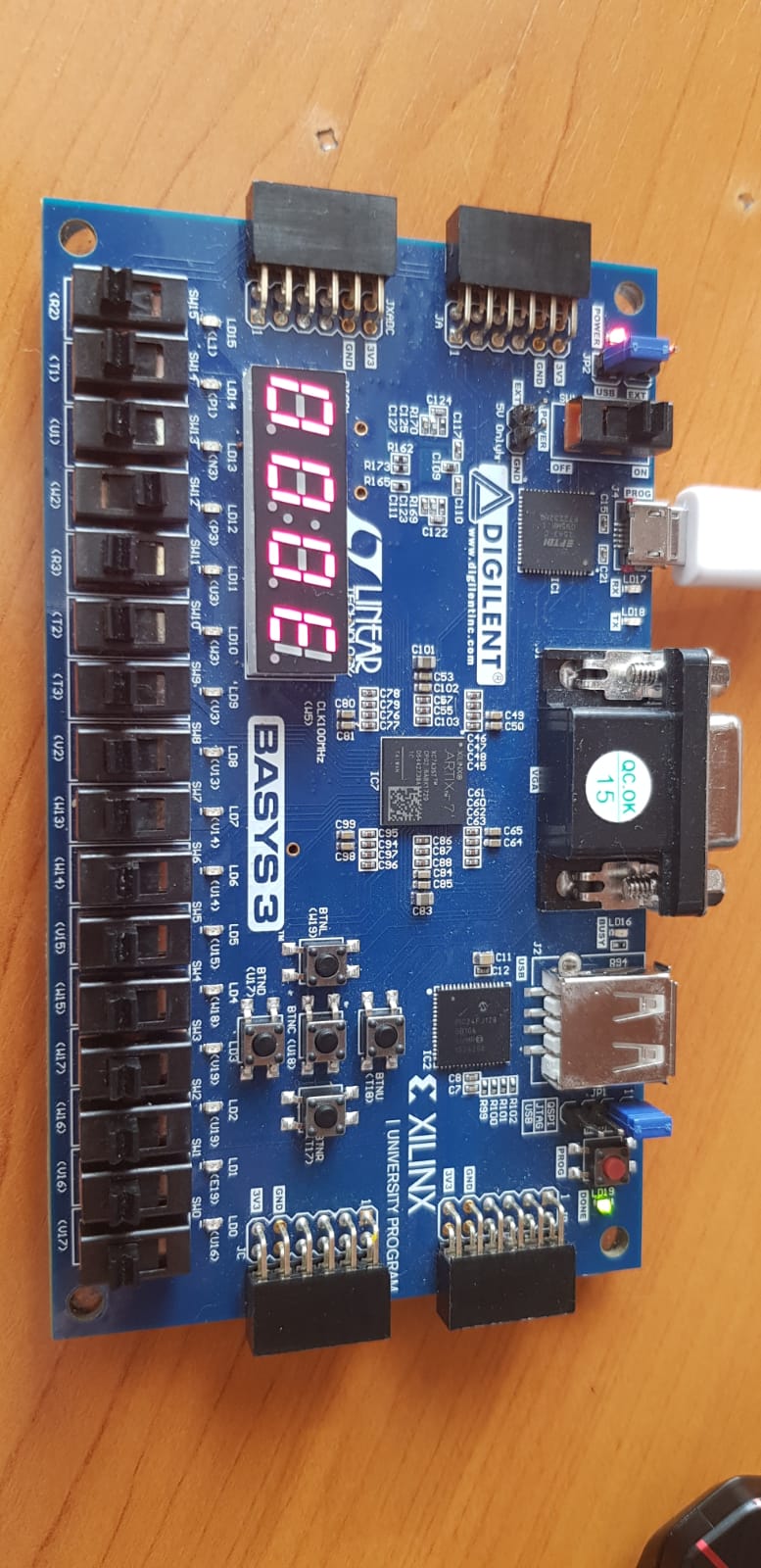
Introducing the first signal.(above)



Introducing the second signal.



Introducing the third signal.The coefficients for filtering are in this exact order 1, 2 and 3. So, doing a simple math equation we come to the conclusion that the result should be 1\*1 + 2\*2 + 3\*3 = 14. In HEX, 14 is translated as **E**.

****

**6.Conclusions:**

In conclusion, filtering is the modification of a measured or calculated signal—using an algorithm and/or logic—to remove undesirable aspects of the signal before it is used in a calculation or a controller. Examples in control are the feedback (or controlled) variable to a PID or APC controller, or the input to a feedforward controller. Calculation examples include computations based on steady-state material/energy balances or process-oriented relationships, as well as process and control metrics.[4]

While developing this project I learned about the internal functioning of the filtering circuits along with some other implementation issues that appeared.

As further development, the filter could be able to compute transfer functions using floating point numbers and also to display some intermediate values.

**7.Refferences**

[1] - <http://123.physics.ucdavis.edu/week_5_files/filters/digital_filter.pdf>

[2] - <https://ro.wikipedia.org/wiki/Convertor_digital-analog>

[3] – Wikipedia

[4] - <https://www.controlglobal.com/articles/2019/signal-filtering-why-and-how/>