Digital signals filtering circuit

Jucan Rares Tudor

Gr. 30432

Content

I. What is a filter?

Networks called filters process signals in a frequency-dependent way. The frequency dependence of the impedance of capacitors and inductors can be used to describe the fundamental idea of a filter. Think about a voltage divider with a reactive impedance on the shunt leg. The reactive impedance's value fluctuates with frequency, and so does the voltage divider ratio. The frequency response, which is defined as the frequency dependent change in the input/output transfer function, is produced by this mechanism.

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 specialized DSP (Digital Signal Processor) chip.

II. The purpose of a filter

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.

Filters have many practical applications. A low-pass filter is often used to stabilize amplifiers by rolling off the gain at higher frequencies where excessive phase shift may cause oscillations.

Note that in a digital filter, the signal is represented by a sequence of numbers, rather than a voltage or current.

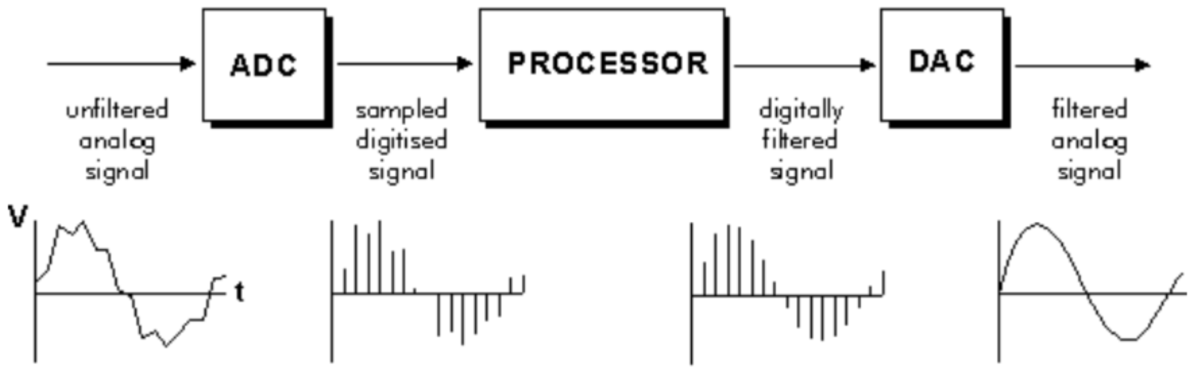


Figure 1

III. Advantages of digital filters

1. A digital filter is programmable, i.e., its operation is determined by a program stored in the processor's memory. This means the digital filter can easily be changed without affecting the circuitry (hardware). An analog filter can only be changed by redesigning the filter circuit.
2. Digital filters are easily designed, tested and implemented on a general-purpose computer or workstation.
3. The characteristics of analog filter circuits (particularly those containing active components) are subject to drift and are dependent on temperature. Digital filters do not suffer from these problems, and so are extremely stable with respect both to time and temperature.
4. Unlike their analog counterparts, digital filters can handle low frequency signals accurately. As the speed of DSP technology continues to increase, digital filters are being applied to high frequency signals in the RF (radio frequency) domain, which in the past was the exclusive preserve of analog technology.
5. Digital filters are very much more versatile in their ability to process signals in a variety of ways; this includes the ability of some types of digital filter to adapt to changes in the characteristics of the signal.
6. Fast DSP processors can handle complex combinations of filters in parallel or cascade (series), making the hardware requirements relatively simple and compact in comparison with the equivalent analog circuitry

IV. Project proposal

This application is designed for the implementation of a digital filter.

The device will be simulated in the Vivado Design Suite. The user will be able to input a sequence of input values and will receive the result, which is a new value calculated based on a certain formula, which takes into consideration what the current and previous value is and uses some predefined constants in order to achieve the needed result, which is the output value. The result will be available in an output file.

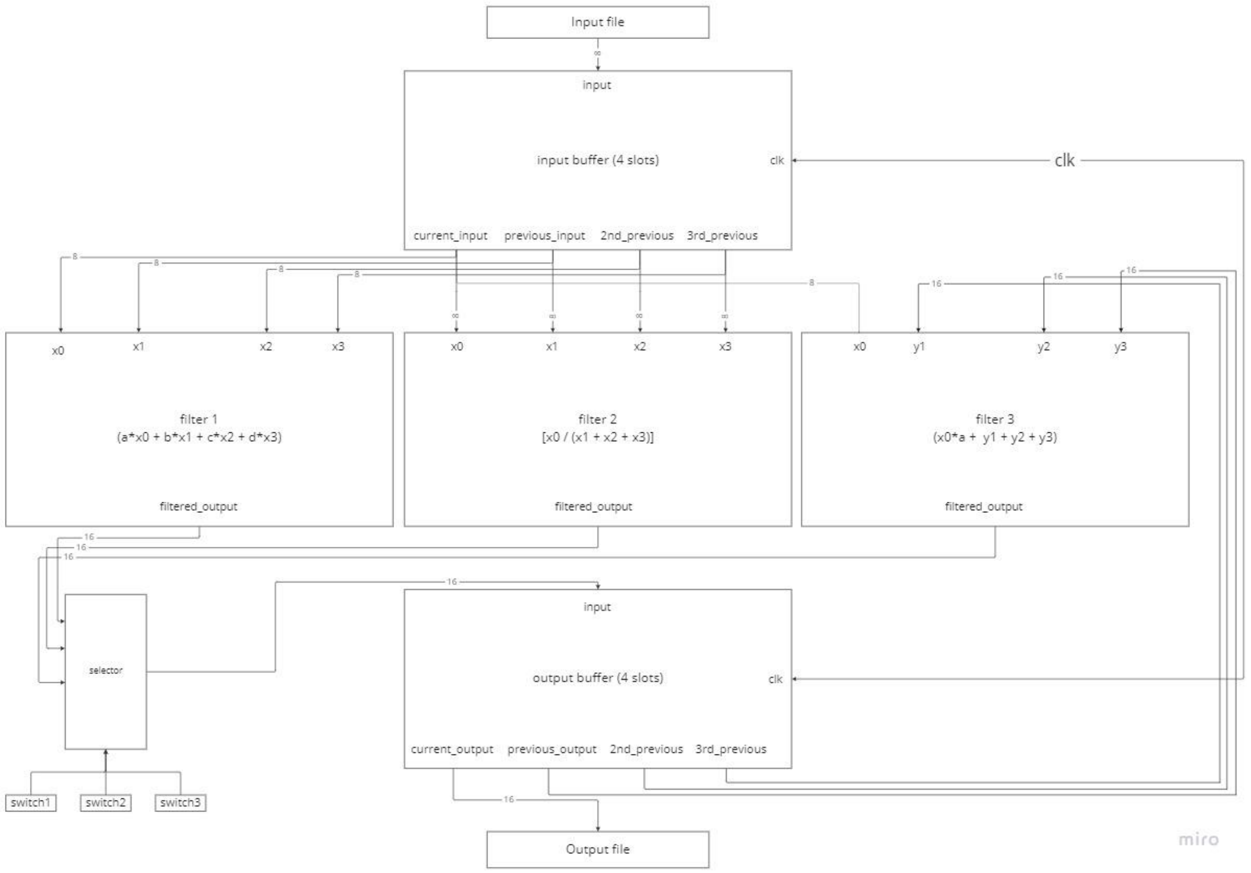
An important aspect that will be chosen is the tap, which represents the number of constants chosen. This is important, as it shows how many previous inputs or outputs will be taken into consideration for the computation of the result.

V. Analysis

The main use case of this project is the user inputting some values after which they will be able to see the filtered results in an output file because the Basys 3 doesn’t have enough 7-segment displays to display the results.

The three main components that I am going to use are: the filters themselves, an input buffer and an output buffer. For the filters I am going to use adders, multipliers and a divider. For the buffers I will use a collection of registers to transfer the data from one to another.

VI. Design


Input file
input
input buffer (4 slots)
clk
clk
current_input
previous_input
2nd_previous
3rd_previous
8
x0
x1
x2
x3
filter 1
(a*x0 + b*x1 + c*x2 + d*x3)
filtered_output
filter 2
[x0 / (x1 + x2 + x3)]
filter 3
(x0*a + y1 + y2 + y3)
y1
y2
y3
16
selector
switch1
switch2
switch3
output buffer (4 slots)
current_output
previous_output
Output file
miro

Firstly, to implement this device I will need to create the basic computational components such as adders and multipliers.

a) The adder

I will use adders to add the results of multiplications and such computing the final values. The adder will be made from full adders, making it a ripple-carry adder.

b) The multiplier

I will use multipliers to compute the multiplication of different inputs with pre-fixed coefficients. For this I will use a shift-and-add multiplier that we studied at the lab (only 8-bit multiplications resulting in 16-bit products).

c) The buffer

I will need to store somewhere the previous inputs and outputs so I can use them later for different computations so for this I will implement a buffer to store the last 3 input and output values using multiple registers.

d) The selector

This is a less important component but ultimately necessary. Using this selector, I will be able to dictate which result from the three filters will be put in the output buffer.

e) The divider

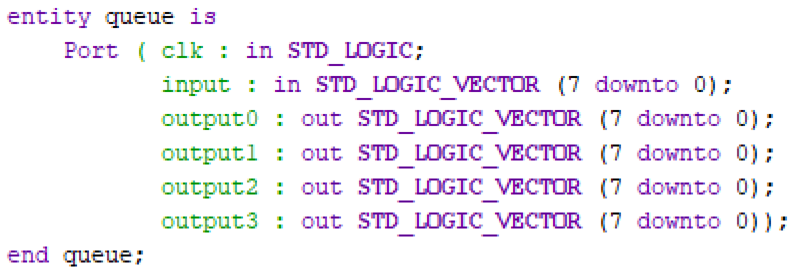
For the implementation of the second filter, I will need to use a divider.

f) The filters themselves

I will implement three filters, each one using a different method. The first filter will use the current input alongside the previous three. The second one will also use only the inputs but this time it will make use of the divider component. And the third one will see the use of the last 3 outputs and the current input.

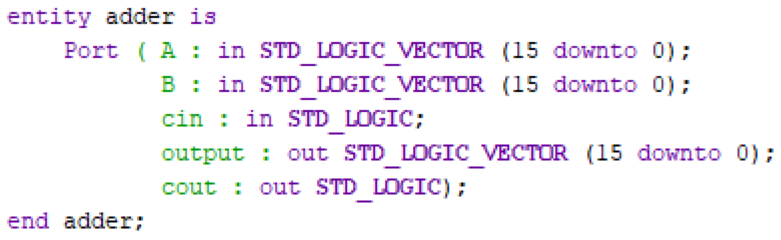
VII. Implementation

The first component that I tackled with is the input buffer, which is basically a group of four 8-bit registers used to store the current and the last three inputs that we are going to use for the filters. It is not something extremely difficult, just four registers connected between them in order to transfer the data from one to another. A tricky part was the sequential part of this component because later I realized that for the right result, we will need to wait an extra clock cycle just for the inputs to get to the correct positions (this happens because the current input needs to firstly get in a register, not directly to the filter).

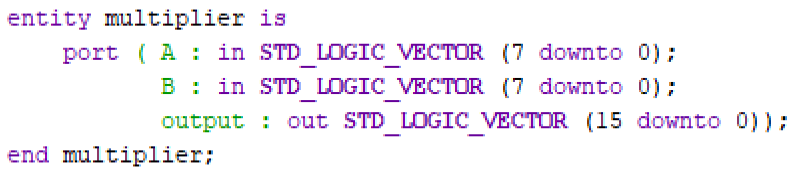


```
entity queue is
    Port ( clk : in STD_LOGIC;
           input : in STD_LOGIC_VECTOR (7 downto 0);
           output0 : out STD_LOGIC_VECTOR (7 downto 0);
           output1 : out STD_LOGIC_VECTOR (7 downto 0);
           output2 : out STD_LOGIC_VECTOR (7 downto 0);
           output3 : out STD_LOGIC_VECTOR (7 downto 0));
end queue;
```

The next components I started working on were the adder and the multiplier. The adder was simple, but the multiplier gave me a bit of a headache. I chose to do a ripple carry adder and a shift and add multiplier (but with a more unorthodox design).

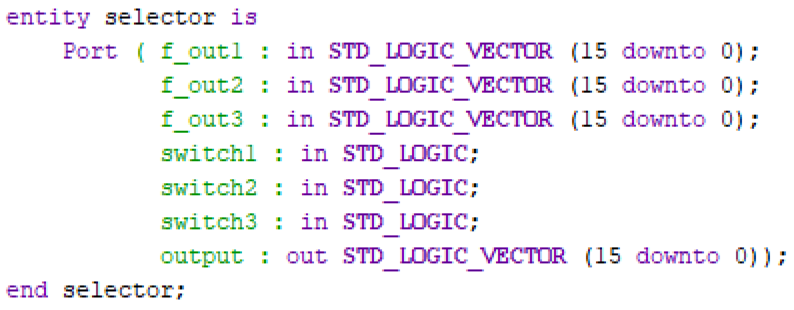


```
entity adder is
    Port ( A : in STD_LOGIC_VECTOR (15 downto 0);
           B : in STD_LOGIC_VECTOR (15 downto 0);
           cin : in STD_LOGIC;
           output : out STD_LOGIC_VECTOR (15 downto 0);
           cout : out STD_LOGIC);
end adder;
```



```
entity multiplier is
    port ( A : in STD_LOGIC_VECTOR (7 downto 0);
           B : in STD_LOGIC_VECTOR (7 downto 0);
           output : out STD_LOGIC_VECTOR (15 downto 0));
end multiplier;
```

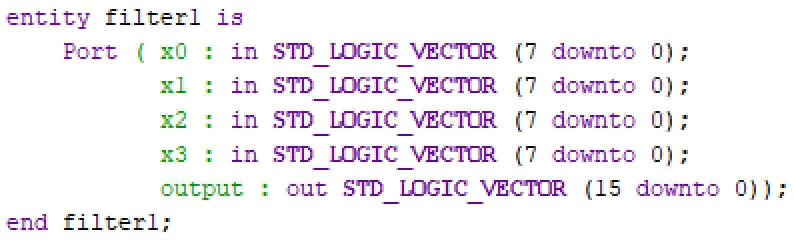
The selector was very straight forward. I needed this component just to decide what filter was going to be used and not let all three outputs go to the output buffer.



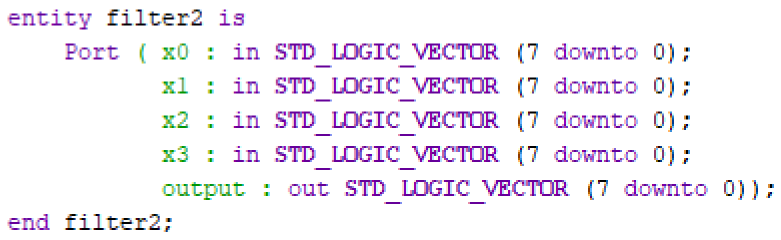
```
entity selector is
    Port ( f_out1 : in STD_LOGIC_VECTOR (15 downto 0);
           f_out2 : in STD_LOGIC_VECTOR (15 downto 0);
           f_out3 : in STD_LOGIC_VECTOR (15 downto 0);
           switch1 : in STD_LOGIC;
           switch2 : in STD_LOGIC;
           switch3 : in STD_LOGIC;
           output : out STD_LOGIC_VECTOR (15 downto 0));
end selector;
```

Another component was the divider. Here I could have taken a harder path and implement a sequential component with a control unit and a shifter and other components, but I chose to do a loop of subtractions until the dividend was smaller than the divisor and so obtaining the quotient.

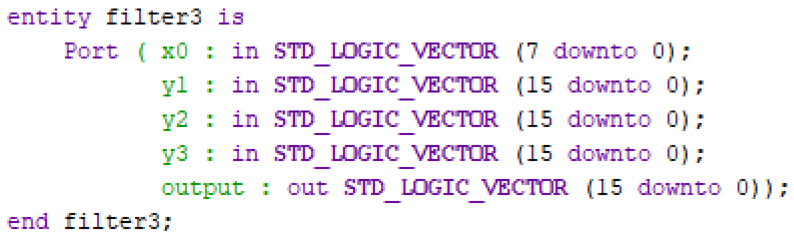
And finally, I implemented the filters. Since I previously implemented all the necessary components, this was just a job of correctly connecting the components between them with the right signals. All three filters work on the same principle: take 4 inputs and return one output. Only the third filter would take one input and three previous outputs.



```
entity filter1 is
    Port ( x0 : in STD_LOGIC_VECTOR (7 downto 0);
           x1 : in STD_LOGIC_VECTOR (7 downto 0);
           x2 : in STD_LOGIC_VECTOR (7 downto 0);
           x3 : in STD_LOGIC_VECTOR (7 downto 0);
           output : out STD_LOGIC_VECTOR (15 downto 0));
end filter1;
```



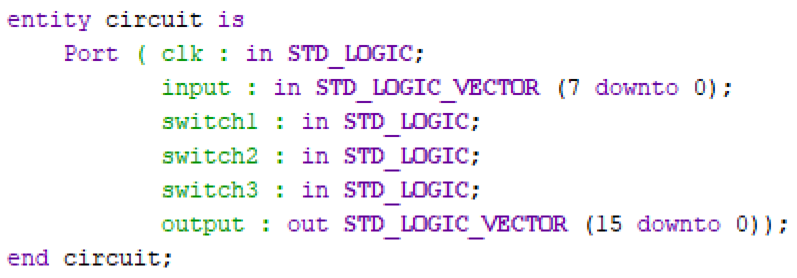
```
entity filter2 is
    Port ( x0 : in STD_LOGIC_VECTOR (7 downto 0);
           x1 : in STD_LOGIC_VECTOR (7 downto 0);
           x2 : in STD_LOGIC_VECTOR (7 downto 0);
           x3 : in STD_LOGIC_VECTOR (7 downto 0);
           output : out STD_LOGIC_VECTOR (7 downto 0));
end filter2;
```



```
entity filter3 is
    Port ( x0 : in STD_LOGIC_VECTOR (7 downto 0);
           y1 : in STD_LOGIC_VECTOR (15 downto 0);
           y2 : in STD_LOGIC_VECTOR (15 downto 0);
           y3 : in STD_LOGIC_VECTOR (15 downto 0);
           output : out STD_LOGIC_VECTOR (15 downto 0));
end filter3;
```

Also, right before I started putting them all together, I noticed that the output from the second filter was on eight bits instead of sixteen so to get past that I added an extender where I concatenated eight zeros with the filtered output and now everything was looking good.

The circuit was ready to be put together. I connected the outputs of the input buffer with the inputs of the filters, the outputs of the filters with the inputs of the selector, and finally the output of the selector with the input of the output buffer.

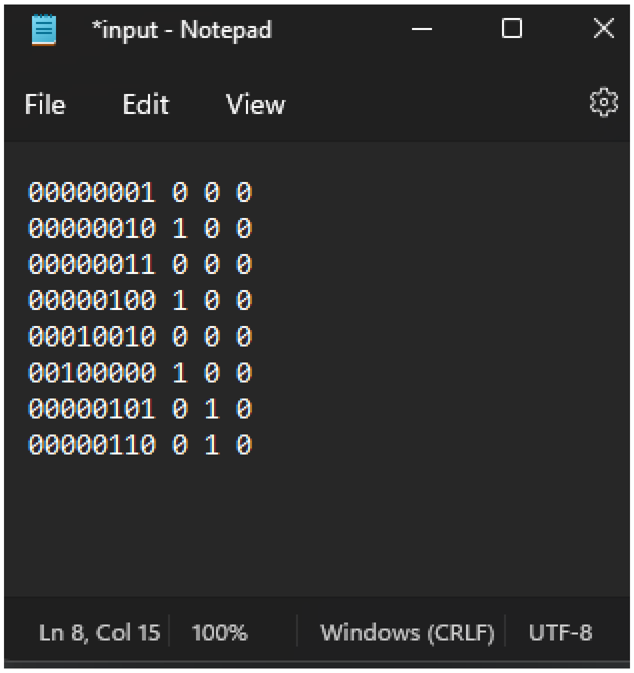


```
entity circuit is
    Port ( clk : in STD_LOGIC;
           input : in STD_LOGIC_VECTOR (7 downto 0);
           switch1 : in STD_LOGIC;
           switch2 : in STD_LOGIC;
           switch3 : in STD_LOGIC;
           output : out STD_LOGIC_VECTOR (15 downto 0));
end circuit;
```

VIII. Testing

For the testing part I decided to work with input and output files. In order to do this, I needed to include the TEXTIO library. It was a bit hard at first but then I quickly got the hang of this.

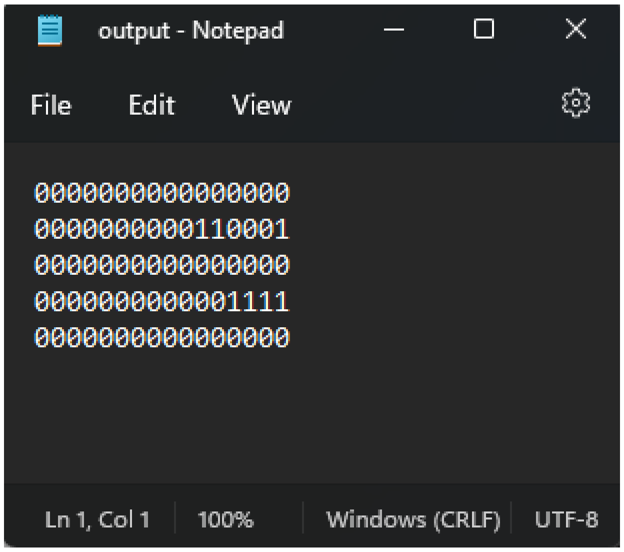
This is how the input file looks like:



```
00000001 0 0 0
00000010 1 0 0
00000011 0 0 0
00000100 1 0 0
00010010 0 0 0
00100000 1 0 0
00000101 0 1 0
00000110 0 1 0
```

Each row contains the following: the input number as an 8-bit number and the status of each switch from 1 to 3 (from left to right). It is important to know that if multiple switches are open then the smallest number switch will be taken into consideration!

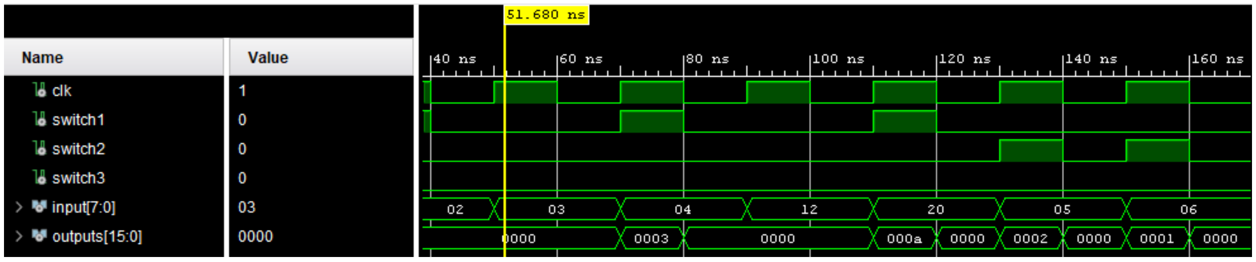
This is how the output file looks like:



```
0000000000000000
0000000000110001
0000000000000000
0000000000001111
0000000000000000
```

The results representing 0 are there because every time we use a switch we need to close it after so we can reuse it later and when no switch is opened then the result passed to the output file will be 0, so between every output there will be a 0.

And this is the waveform of this example:



Name	Value
clk	1
switch1	0
switch2	0
switch3	0
input[7:0]	03
outputs[15:0]	0000

IX. Conclusion

In the end, the design of the digital filter has been reached. The system properly takes the input numbers and computes the output in a file. Except for the third filter because I can not find a way to solve the problem with the output buffer. It was an interesting, yet challenging project. It took me a while to understand exactly what I need to do, but once I figured that out, I started implementing the components easily. A challenging part was the synchronization part.

X. Bibliography

- https://en.wikipedia.org/wiki/Filter_(signal_processing)
- https://123.physics.ucdavis.edu/week_5_files/filters/digital_filter.pdf

Images:

Figure 1 taken from https://www.brainkart.com/article/IIR-Filter-Design_13036/