Implementation and Testing of Digital Filters on STM32 Nucleo-64P

Simona-Daiana Sim
Department of Automation
Tehnical University of Cluj-Napoca
Cluj-Napoca, Romania
sim.simonadaiana@gmail.com

Zsófia Lendek
Department of Automation
Tehnical University of Cluj-Napoca
Cluj-Napoca, Romania
zsofia.lendek@aut.utcluj.ro

Petru Dobra
Department of Automation
Tehnical University of Cluj-Napoca
Cluj-Napoca, Romania
petru.dobra@aut.utcluj.ro

This paper aims to present two ways to design and implement numerical filters. The first is the classic or hand-coded method, which involves the implementation of a computational algorithm in C, while the second is automatically generated code. Both FIR and IIR filters are implemented and based on the results obtained, the differences between them discussed. We highlight the advantages and disadvantages of each method of implementation and the costs in terms of design time, memory, performance. Both implementation variants present satisfactory results and meet the required requirements, the choice of one of them depends on the application used and the programming experience.

Keywords — Digital filters, DSP, FIR, IIR, implementation, testing.

I. INTRODUCTION

A vast theme in the digital era is presented [1] by digital filters. A filter can restrain a system in its minimum form, i.e. any system can be restricted in the form of a filter. Filters are an important part of any system with signal processing. The increasing complexity of digital applications has led to more interest in the implementation of digital filters. Today, we find digital filters everywhere: in phones, TVs, radios, household appliances, but also in computer science, software audio processing, image and videos filtering systems etc. Analog and digital filters remove both unnecessary items or noise and elements of unwanted signal. But these two types of filters (analog or numeric) work differently in their, analog or digital domain. When it comes to managing and processing the signal, the digital filter is a system that performs a mathematical calculation on the signal of discrete time, obtained by sampling the input signal, to reduce or improve certain aspects. Analog filters are those systems that act on a continuous signal [2].

Digital filters can be more expensive [3] than their analog equivalent due to increased complexity and the need for the components (memory, speed, CPU), but they can get results impossible to be transposed by analog filters. Digital filters [4] can be thought as a linear phase pulse response. When used for simulation in real time, the system often has [5] delays due to digital analog conversion or its implementation. Nowadays, reduced complexity architectures have been increasingly investigated [6] [7], in particular for implementation on field-programmable gate arrays.

Integrated circuits [3] provide the ability to design and implement filters with performance characteristics that are difficult to recreate with analog methods. Moreover, these digital filters are not affected by the problems that impact analog implementations, in particular, component shunting and tolerance (for highly reliable applications, in terms of temperature, aging, and radiation). Only a digital signal processor is physically needed to obtain a digital filter. Its output can be programmed through various programs, allowing to design the filter coefficients according to the user's wishes. This widens the range of filters that can be implemented using the same microprocessor and the simplicity of obtaining such a filter.

The model of a digital filter has its own characteristics. The performance increase according to the filter's response is related to the output processing and calculation when using algorithms intended for fixed-point coefficients. We also discuss this aspect. The rounding method used to process the filter coefficients can propagate numerical problems, as these types of rounding errors can accumulate with each iteration, thus, affecting the performance of the filter. There are currently a multitude of methods and principles [8] to minimize rounding errors. It is preferable to use an easy-to-understand rounding method so it can solve problems such as overflow and loss of accuracy.

This paper presents two methods for implementing numerical filters and compares the results by the ease of programming, the applications, and the tools used. Emphasis is placed on the importance of quantization. The first implementation is the classic one, in which the algorithm is hand-coded, and the second implementation is based on the code generated by Simulink, which requires less effort from the programmer.

The paper is structured as follows: Section II presents the setup description, Section III describes the background, Section IV digital filters, Section V the design, Section VI the implementation and Section VII the results. Section VIII concludes the paper.

II. SETUP DESCRIPTION

The simplicity and advantage of digital filters come from the fact that with the help of a development board, filters can be implemented on different sampling frequencies depending on the capabilities of the board. For testing, a signal generator and an oscilloscope are needed.

The Matlab "filterDesigner" toolbox is used to design the filters. The filter coefficients can be exported by various methods. The following implementations are considered: 1.) implementation using STM32 CubeMX and System Workbench Eclipse STM32 and 2.) Keil uVision (MDK Arm), STM32 CubeMX, and Simulink/Matlab were used.

The development board we use is STM32-NUCLEO-L452RE-P. The ports required for the simulation are:

• ADC (Analog to Digital Converter - 12 bits) representing the analog signal input that is to be filtered.

• DAC (Digital to Analog Converter - 12 bits) is use for the output of the digital filter and its transformation into analog signal.

• GPIO (General purpose input / output) to monitor and check the sampling frequency and the period of the interruption routine.

To generate and perform calculations at a certain time interval, it is necessary to activate an internal clock

This work was supported by a grant of the Romanian Ministry of Education and Research, CNCS - UEFISCDI, project number PN-III-P1-1.1-TE-2019-1956, contract number TE 185/2021.

corresponding to the microprocessor. The maximum frequency of the Nucleo 64P board [9] is 80 MHz. The routine of interruptions is set at 10, 20 and 40 kHz as the sampling frequency for the next simulations.

The last step in setting the parameters and preparing for design and implementation is to save the features. For the first implementation, a project that allows programming in C is exported. For the second approach, Toolchain / IDE: MDK-ARM is used for the automatically generated Simulink [10] code.

III. BACKGROUND

A signal is a physical quantity that varies [4] with time, space, or any other variable. Most signals are analog, but to process them digitally a transformation into a digital signal is needed. Digital signals are obtained by sampling [11] continuous signals. To obtain the discrete time signal, it is important to know [8] the sampling period (Ts).

Most applications rely on DSP (Digital Signal Processors), which makes discretization of the continuous signal necessary. This requires [12] an ADC (Analog-to-Digital Converter) and DAC (Digital-to-Analog Converter). The process of transforming the signals is shown in Figure 1.

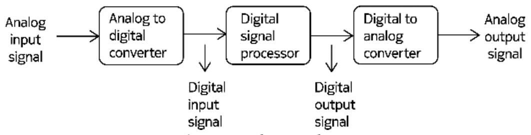

Figure 1. Digital signal processing

A DAC is a data converter [5] that generates an analog output from a digital input. It converts a limited number of discrete codes to an analog output value. The performance of ADCs and DACs is determined by [4] the number of samples it can process and the number of bits used in the conversion process.

The quality and reproduction of a signal has the resolution of the two types of circuits (ADC and DAC), but also depends on the processing time of the controller. The computation time is strongly influenced by the representation [13] i.e., floating point or fixed point, with the disadvantage of limited accuracy and propagating calculation errors. The quantization, also called "amplitude discretization", is a topic of great importance when working with signals. ADC converters (N bits) use a uniform quantization to convert analog signals into digital ones. The limited accuracy of a digitized value is a source of error, called [8] quantization error. The disadvantages of signal quantization are the addition of quantization noise over the obtained signal and the lack of ability to recover the initial signal. Other problems that may occur are calculation errors, such as overflow, which means exceeding the range for variables of a certain type. These aspects are treated and detailed in the following sections.

As mentioned above, the role of filters is to process signals or to eliminate unwanted frequencies. From the point of view of the cut-off frequencies, which determine the signals that are attenuated or rejected, the analog and digital filters are classified as [4]: low pass filters which allow the passage of low frequency signals, i.e. signals that have a frequency between 0 Hz and the cutoff frequency; high pass filters that allow the passage of high frequency signals, i.e. signals that have a frequency higher than the cutoff frequency. There are also bandpass filters and band-stop filters that are complementary and that cut or let a certain frequency band pass. In this paper, due to lack of space, low pass and high pass filters will be presented.

IV. DIGITAL FILTERS

Digital filters can be classified into two main categories: 1) Finite impulse response (FIR) filters and 2) Infinite impulse response (IIR) filters.

A. Finite impulse response filter

FIR can be represented [4] by:

$$y[n] = x[n] * h[n] = \sum_{k=0}^{N-1} h[k]x[n-k] \tag{1}$$

Where: y [n] represents the output signal of the filter, x [n] represents the input signal, h [n] is the Dirac impulse and n is the number of coefficients.

These filters are non-recursive filters and do not have feedback reaction, which makes them more stable to implementation. Their transfer function [4] is:

$$H(z) = \sum_{k=0}^{N-1} b_k z^{-k} \tag{2}$$

where N represents the number of filter coefficients, $b_k\ k = 0, \dots, N-1$, represents the impulse response, i.e., the filter coefficients, and z^{-1} represents the clock delay.

The filter can be represented in several ways. The most used structures are [4]: Direct I shape, Cascade shape, Frequency sampling shape, Lattice shape, etc.

The choice of the form determines how the filter is implemented. The Lattice form is generally used in FIR filters together with an adaptive algorithm. The frequency sampling structure is suitable for filters that have been designed based on frequency response. Each has its own advantages and particularities, but the Direct form I presents the simplest and most practical [4] implementation. Therefore, we use this form.

B. Infinite impulse response filter

IIR filters generate an infinite impulse response. Unlike FIR filters, in which the output is calculated only on the basis of present and past inputs, IIR filters also take into account the filter's past output. This turns the behavior of the filter into a recursive one. The feedback mechanism provides a faster response time. They are intended for programs that do not require a linear phase.

The transfer function of an IIR filter [4] is given in (3).

$$\boldsymbol{H(z) = \frac{\sum_{k=0}^{M} b_k z^{-k}}{1 + \sum_{k=1}^{N} a_k z^{-k}}} \tag{3}$$

The output is computed as:

$$\boldsymbol{y(n) = -\sum_{k=1}^{N} a_k y(n-k) + \sum_{k=0}^{M} b_k x(n-k)} \tag{4}$$

IIR filters can be represented by structures [4] such as Direct form I, Normal and transposed direct form II, Cascade, Lattice, etc. The form we use for the implementation is Direct Form I because it is the simplest one [4].

V. Filter Design

We use "filterDesigner" from Matlab [14] to design the two types of filters. The specifications are:

• Filter type: low-pass and high-pass are considered

• Filter category: IIR (Infinite Impulse Response Filters) and FIR (Finite Impulse Response Filters)

• Filter order: a specific number or a minimum generated by the algorithm based on the other specifications

• Frequency characteristics: the sampling frequency (Fs) used in this paper is between 10 and 40 kHz. Fpass is the frequency of the beginning of a pass band, and Fstop is the frequency of the end of a pass band.

An aspect of major importance is the representation of the filter coefficients, namely: fixed or floating point. Due to the memory and the long time to process arithmetic operations with floating-point representation, the coefficients of the transfer functions are chosen to have a fixed-point representation for the first implementation. The quantization method considered optimal by Matlab was chosen, having a 16-bit representation. For the second implementation, the floating-point implementation was chosen due to the poor export of the coefficients from filterDesigner-Simulink-MDK-ARM.

In the following, both low-pass and high-pass filters will be presented. The sampling frequency for the first implementation is set to 20 kHz. For the second implementation, due to floating point representation, the sampling frequency is set to 10 kHz. It is desired to obtain as small as possible order of the system for an easier and faster processing, but to maintain some performances considered necessary.

A. *FIR design*

For the first implementation, the hand-coded one, the obtained filters were exported as C-headers with an integer vector B of length BL containing the coefficients. For the second implementation using Simulink, they are easily exported to Simulink as a block model directly from filterDesigner.

The low-pass filter aims to attenuate the signal with a frequency higher than the cutoff frequency Fstop and to allow signals to pass unchanged from 0 Hz to Fpass. The high-pass filter allows the passage of signals of a frequency higher than the cutoff frequency Fpass, and the frequencies from 0 to Fstop will be attenuated. The magnitude response characteristics are given by: Astop - attenuation in the stopband and Apass - passband ripple. All FIR filters are designed using the Equiripple method. This method is based on the Remez algorithm [4], which allows [15] for an equiripple response in the assigned filter pass and stop bands. Equiripple filters provide the smallest transition bandwidth possible for a given ripple level.

1) *Hand-coded implementation*

For this implementation, the filters with coefficients having a fixed-point representation. Their sampling frequency is 20 kHz.

The low-pass filter has the following characteristics: Fpass = 200, Fstop = 2000, Apass = 1, Astop = 60 and the order obtained is 21. It can be seen in the Figure 2. The coefficients (signed integer, on 16 bits) are:

BL = {207, 416, 773, 1262, 1874, 577, 3320, 4038, 4660, 5119, 5362, 5362, 5119, 4660, 4038, 3320, 2577, 1874, 1262, 773, 416, 207}.

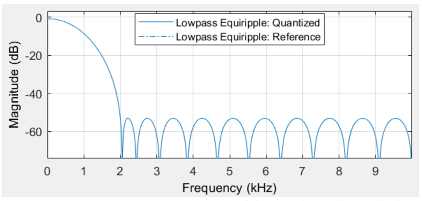

Figure 2. Low-pass FIR filter - hand-coded

The high-pass filter has the following characteristics: Fstop = 1000, Fpass = 3000, Astop = 60, Apass = 3 and the order obtained is 18. It can be seen in the Figure 3. The coefficients (signed integer, on 16 bits) are:

BL = { -895, 210, 1068, 1027, 1024, -578, -2420, -5135, -6792, 24968, -6792, -5135, -2420, -578, 1024, 1027, 1068, 210, -895 }.

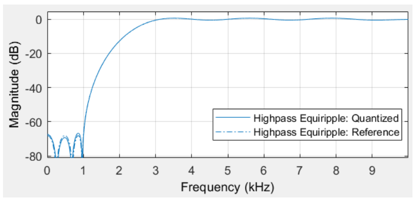

Figure 3. High-pass FIR filter – hand-coded

2) *Automatic implementation*

For the automatic implementation, the sampling frequency was set at 10 kHz and a floating-point representation is used.

The low-pass filter has the following characteristics: Fpass = 500, Fstop = 2000, Apass = 1, Astop = 40 and the order obtained is 8. The magnitude plot can be seen in the Figure 4. The coefficients (real) are:

BL = { 0.02047, 0.07474, 0.13626, 0.19894, 0.21986, 0.19894, 0.13626, 0.07474, 0.02047 }.

The high-pass filter has the following characteristics: Fstop = 500, Fpass = 2000, Astop = 40, Apass = 1 and the order obtained is 10. The filter's response is seen in Figure 5. The coefficients (real) are:

BL = { 0.03709, 0.01608, -0.04184, -0.14059, -0.23637, 0.72381, -0.23637, -0.14059, -0.04184, 0.01608, 0.03709 }.

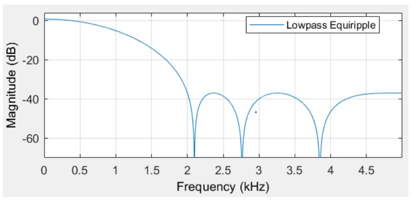

Figure 4. Low-pass FIR filter - automatic

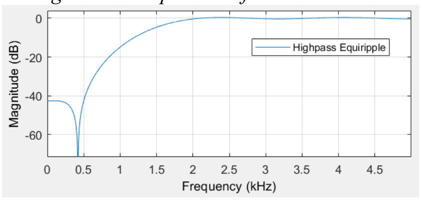

Figure 5. High-pass FIR filter - automatic

B. *IIR design*

Next, we present the design of IIR filters. The quantization of the terms will be applied only for the first implementation and the aim is to obtain an order of the function as small as possible while satisfying the required specifications. Similarly to the FIR filter design, the sampling frequency is 20 kHz for the first implementation and 10 kHz for the second one. To highlight the short computation time of fixed-point implementation, filters with a sampling frequency of 40 kHz were also obtained.

For hand-coded implementation, a C-header file that contains two vectors containing the denominator (DEN) and numerator (NUM) coefficients will be generated. The second implementation involves only creating a Simulink block directly from the filterDesigner. All filters are designed using the Butterworth method. The frequency response of the Butterworth filter approximation function is also often referred to as the "maximally flat" (no ripple) response [16], because the passband is designed to have a frequency response that is mathematically as flat as possible from 0 Hz to the cutoff frequency.

1) *Hand-coded implementation*

The low-pass filter has a sampling frequency of 20 kHz and the following characteristics: Fpass = 1000, Fstop = 5000, Apass = 3, Astop = 60 and order 4. The magnitude plot is shown in Figure 6. The coefficients (signed integer, on 16 bits) are:

NUM = { 5, 21, 31, 21, 5}
DEN = { 8192, -25251, 29863, -15968, 3247}.

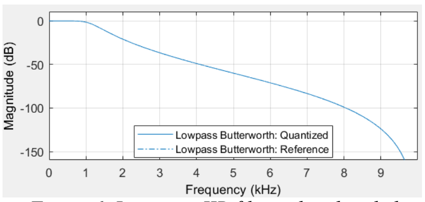

Figure 6. Low-pass IIR filter – hand-coded

The high-pass filter has a sampling frequency of 20 kHz and has the following characteristics: Fstop = 2000, Fpass = 6000, Astop = 40, Apass = 3 and the order is 4. The magnitude plot is shown in Figure 7. The coefficients (signed integer, on 16 bits) are:

NUM = { 1458, -5832, 8748, -5832, 1458 }
DEN = { 16384, 1104, 7986, 232, 291 }.

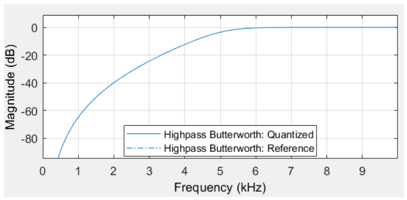

Figure 7. High-pass IIR filter – hand-coded

2) *Automatic implementation*

The low-pass filter has a sampling frequency of 10 kHz and has the following characteristics: Fpass = 500, Fstop = 2000, Apass = 1, Astop = 40 and the order obtained is 4. The filter's response is shown in Figure 8. The coefficients are:

NUM = { 0.00153, 0.00614, 0.00921, 0.00614, 0.00153 }
DEN = { 1, -2.82423, 3.11624, -1.57098, 0.30353 }.

The high-pass filter has a sampling frequency of 10 kHz and has the following characteristics: Fstop = 500, Fpass = 2000, Astop = 40, Apass = 1 and the order obtained is 4. It can be seen in the Figure 9. The coefficients are:

NUM = { 0.28117, -1.12471, 1.68706, -1.12471, 0.28117 }
DEN = { 1, -1.60531, 1.31159, -0.50256, 0.07937 }

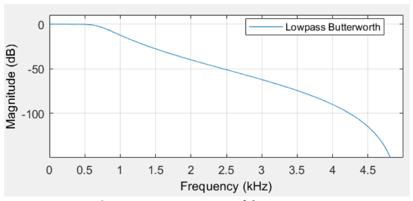

Figure 8. Low-pass IIR filter - automatic

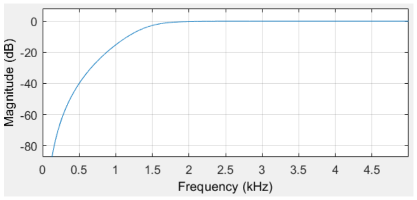

Figure 9. High-pass IIR filter - automatic

VI. IMPLEMENTATION

All methods presented in this paper have the same objective, but the difficulty of programming and the result will be different. The process as a whole is the same regardless of the implementation. The first step is to read the input through the ADC converter, update the input vector, calculate the output and write the scaled output after quantization through the DAC converter. The output signal is displayed on an oscilloscope for validation together with the signal that verifies the sampling period.

The implementation was intended with a frequency of 20 kHz, but in the floating point, this could not be obtained. The automatic generation did not support the connection well (an issue that is left for future work) and thus it was decided to reduce the sampling frequency to 10 kHz. On the other hand, for the hand-coded implementation, results with a sampling frequency of 40 kHz have also been obtained.

A. *Hand-coded implementation*

For this implementation, the algorithm for each type of filter (FIR and IIR) was written in C and is loaded on the development board through System Workbench for STM32. Files directly exported from Matlab were used as headers. The implementation chosen in this paper is based on Direct form I.

The algorithm used to calculate the filter output is based on equations (1) and (4). For all filters, in this paper, the structure that was implemented is the Direct form I using a fixed point representation.

B. *Automatic implementation*

Filters can be directly exported from the filterDesigner to Simulink. In order to create the project that will be loaded on the development board, the pins are set, after which a Simulink file containing the blocks for STM32 is created. All designed filters are exported to the Simulink file and linked between the ADC input and the DAC output as shown in Figure 10.

The time set for the simulation is discrete and has a sampling period of 0.1 ms. After the code is built, the filter can be run by the processor through the Keil µVersion application.

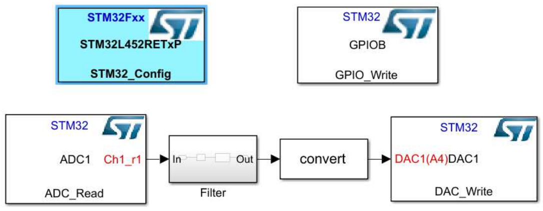

Figure 10. Simulink file

VII. RESULTS AND COMPARISON

This section presents the obtained results for the filters and interpreted with the help of an oscilloscope and a periodic signal generator of certain frequencies depending on the parameters of interest. All the filters have first been simulated and validated in Simulink. The specifications have been satisfied. Afterwards they have been implemented on the microcontroller. The figures in this section are obtained by means of an oscilloscope. The graphs on the following pages show the input signal (blue) and the filter output signal (purple); the PB5 port output (yellow) - which confirms the compliance with the frequency of the interrupt routine and the processing time of the calculations - is only available for certain graphics.

A. Results for hand-coded implementation

1) Low-pass filters

The FIR filter is shown in Figure 11, and the IIR filter, in Figure 12. The input signal is a 0-3 kHz for FIR filter (0-10 kHz for the IIR filter) variable frequency signal. Figure 13 shows the response of an IIR filter with a sampling frequency of 40 kHz. It has the following characteristics: Fpass = 500, Fstop = 2000, Apass = 1, Astop = 40, order 3. The best performance for this frequency is obtained by a filter up to an order of 12.

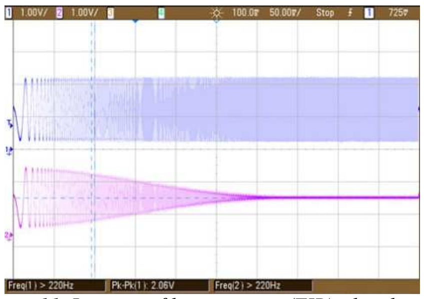

Figure 11. Low-pass filter response (FIR) - hand-coded

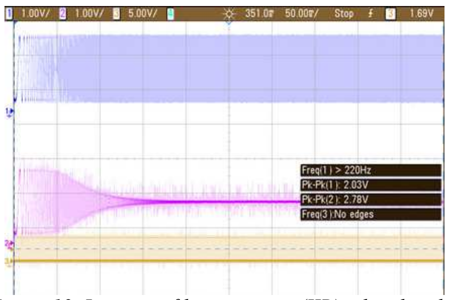

Figure 12. Low-pass filter response (IIR) – hand-coded

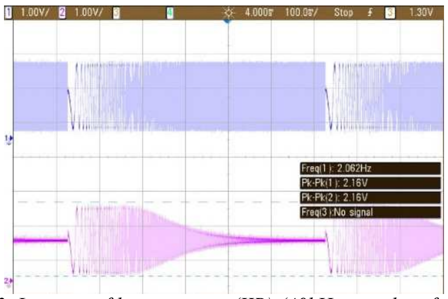

Figure 13. Low-pass filter response (IIR) (40kHz sampling frequency)-hand-coded

2) High-pass filters

It is noted that the filter output maintains the required specifications and the sampling frequency of 20 kHz. The input signal is a variable frequency signal: 0-3 kHz for the first filter and 0-10 kHz for the second. The FIR response can be seen in the Figure 14 and the IIR one in the Figure 15.

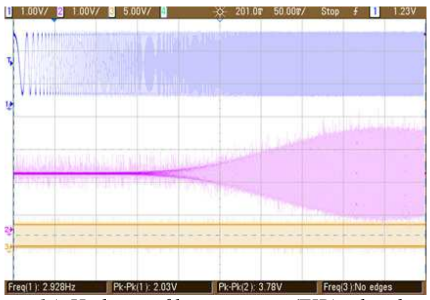

Figure 14. High-pass filter response (FIR) – hand-coded

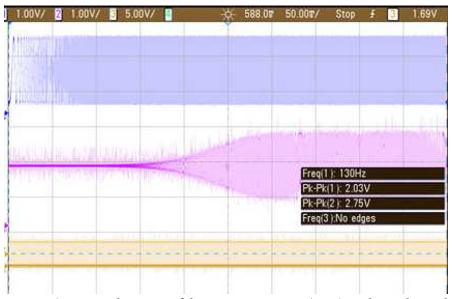

Figure 15. High-pass filter response (IIR) – hand-coded

B. Results for automatic implementation

For this implementation, the input signal is the same for all the filters: a signal with a variable frequency between 0 and 5 kHz.

1) Low-pass filter

The FIR filter response is shown in Figure 16, and the IIR in Figure 17.

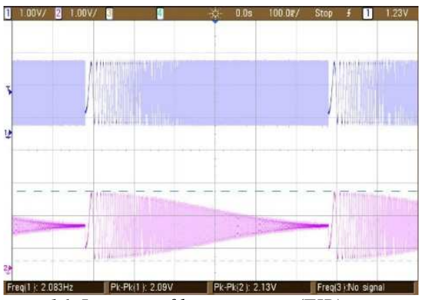

Figure 16. Low-pass filter response (FIR) -automatic

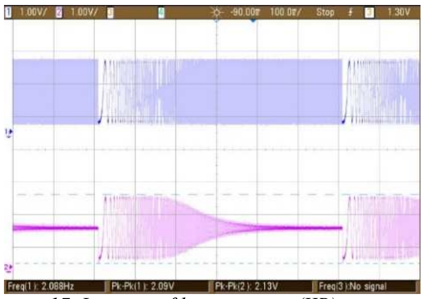
Figure 17. Low-pass filter response (IIR) - automatic

2) *High-pass filter*

The response of the FIR filter design is shown in Figure 18, and the IIR in Figure 19.

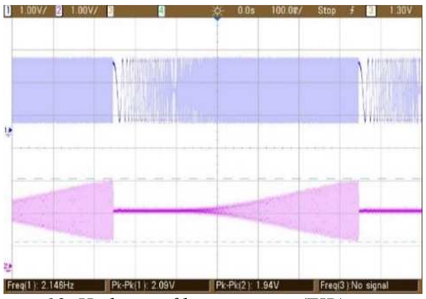
Figure 18. High-pass filter response (FIR) - automatic

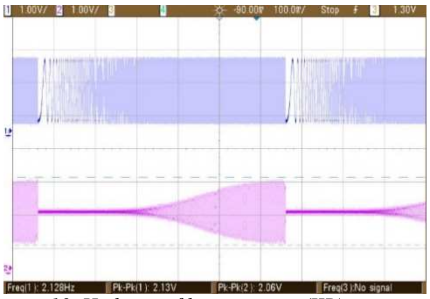
Figure 19. High-pass filter response (IIR) - automatic

VIII. CONCLUSIONS

All filters comply with the requirements and correspond to the simulations performed. A summary of their characteristics can be seen in Table 1.

TABLE I. *DESIGN COMPARISON BETWEEN FILTERS*

Filter	Hand-coded	Automatic

FIR Low-pass	Fs=20kHz
	Fpass = 200
	Fstop = 2000
	Apass = 1
	Astop = 60
	order = 21
		Fs=10kHz
		Fpass = 500
		Fstop = 2000
		Apass = 1
		Astop = 40
		order = 8

FIR High-pass	Fs=20kHz
	Fstop = 1000
	Fpass = 3000
	Astop = 60
	Apass = 3
	order = 18
		Fs=10kHz
		Fstop = 500
		Fpass = 2000
		Astop = 40
		Apass = 1
		order = 10

IIR Low-pass	Fs=20kHz
	Fpass = 1000
	Fstop = 5000
	Apass = 3
	Astop = 60
	order = 4
		Fs=10kHz
		Fpass = 500
		Fstop = 2000
		Apass = 1
		Astop = 40
		order = 4

IIR High-pass	Fs=20kHz
	Fstop = 2000
	Fpass = 6000
	Astop = 40
	Apass = 3
	order = 4
		Fs=10kHz
		Fstop = 500
		Fpass = 2000
		Astop = 40
		Apass = 1
		order = 4

The most obvious conclusions are related to the power of quantization in reducing computational time. By this quantization, satisfactory results were obtained at a switching frequency of 40 kHz. The disadvantage of this quantization is a loss of precision and other problems such as overflowing, instability, etc. The advantage of the automatically generated code is the simplicity and the minimum programming requirements. The downside in this case was the inability to generate a code that works on fixed point representation. This issue needs to be further investigated. The limits tested for hand-coded implementation is at a frequency of 40kHz for a 12th order IIR filter. For automatic implementation, IIR filter was successfully implemented at 20kHz but with a 2nd order.

IX. REFERENCES

[1] C. Dede, "Tendencies for Technology in The Year 2000," *Educational Media International,* vol. 20, no. 1, pp. 23-24, 1983.

[2] S. Rolf, H. Xiao and V. V. Mac E., Design of Analog Filters 2nd edition, Oxford University Press, 2009.

[3] R. Oshana, DSP Software Development Tehniques for Embedded and Real-Time Systems, 2006.

[4] J. G. Proakis and D. G. Manolakis, Digital Signal Processing, Third Edition ed., Prentice-Hall International INC, 1996.

[5] M. J. Pelgrom, Analog-to-Digital Conversion, Springer-Verlag New York, 2013.

[6] R. R. S. Seshadri, "FPGA implementation of fast digital FIR and IIR filters," *Concurrency Computation: Practice and Experience,* vol. 33, p. e5246, 2021.

[7] C. Wang, H. Xi, T. Chen and J. Liu, "Design of IIR Digital Filter," in *International Conference on Cognitive based Information Processing and Applications (CIPA 2021)*, Singapore, 2022.

[8] B. Widrow and I. Kollár , Quantization Noise: Roundoff Error in Digital Computation, Signal Processing, Control, and Communications, Cambridge University Press, 2008.

[9] "STM Product overview Nucleo 64P," [Online]. Available: https://www.st.com/en/evaluation-tools/nucleo-l452re-p.html.

[10] "STMicroelectronics Hardware Support from Simulink," [Online]. Available: https://www.mathworks.com/products/hardware/stmicroelectronics.html.

[11] B. G. Lawrence R. Rabiner, Theory and application of digital signal processing, Prentice-Hall, 1975.

[12] A. V. Oppenheim, R. W. Schafer and J. R. Buck, Discrete-Time Signal Processing, New Jersey: Prentice Hall, 1998.

[13] V. Madisetti, Digital Signal Processing Fundamentals, CRC Press, 2017.

[14] "Introduction to Filter Designer," Mathworks, [Online]. Available: https://www.mathworks.com/help/signal/ug/introduction-to-filter-designer.html.

[15] G. Macchiarella, "“Equi-Ripple” Synthesis of Multiband Prototype Filters Using a Remez-Like Algorithm," *IEEE Microwave and Wireless Components Letters,* vol. 23, no. 5, pp. 231-233, 2013.

[16] G. Ellis, Control System Design Guide, Elsevier Science, 2012.