Digital Signal Processing Simulation Based on Matrix-Vector description

Tibor WÜHRL
Obuda University
Kandó Kálmán Faculty of Electrical Engineering
Budapest, Hungary

wuhrl.tibor@kvk.uni-obuda.hu
orcid.org/0000-0002-7522-3511

Sándor GYÁNYI
Obuda University
Kandó Kálmán Faculty of Electrical Engineering
Budapest, Hungary

gyanyi.sandor@kvk.uni-obuda.hu
orcid.org/0000-0001-9405-3864

Abstract— Even a small error in the design and implementation of a DSP algorithm can cause complete inoperability, finding and fixing these errors is laborious and time-consuming. Of course, the validation of the correct operation of a DSP algorithm must be confirmed by measurement. In case of a deviation, it is important to find the root of the problem, the simulation of the DSP algorithm is a reliable and efficient task which can be performed by a desktop computer running some mathematical program. The simulation – as an operation check – could be performed in the time domain, or in the frequency domain or in both. The system of equations describing the DSP algorithm can also be written in matrix-vector form, therefore these algorithms can be effectively simulated in the Matlab® environment.

Keywords: DSP, simulation, time-domain, frequency-domain

I. Introduction

This paper describes the generation of an efficient matrix-vector shape and a case study. This article is based on and should be considered a continuation of an earlier paper [1].

The design of digital signal processing algorithms needs several steps and any of them could contain possible errors. The transformations are forming a connected system and transformation relationships help to simplify the design steps [2]. The system of transformations is summarized in Fig.1.

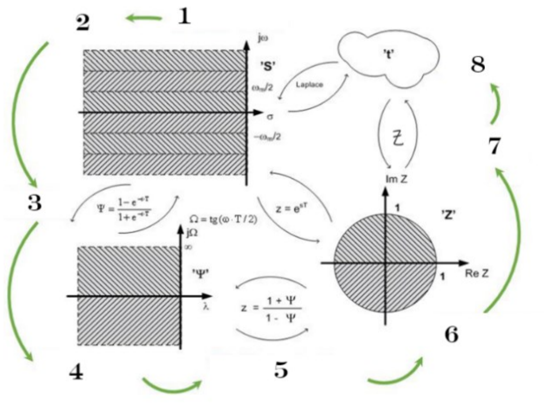

1
2
3
4
5
6
7
8
jω
'S'
σ
Laplace
't'
Z
Im Z
'Z'
Re Z
$\Psi = \frac{1-e^{-sT}}{1+e^{-sT}}$
$\Omega = tg(\omega \cdot T/2)$
$z = e^{sT}$
jΩ
'Ψ'
λ
$z = \frac{1+\Psi}{1-\Psi}$

Fig. 1. System of transformations and design steps

Brief descriptions if the design steps marked on Fig. 1, are the following:

- Firs step: definition of requirement;
- Second step: determination of the sampling frequency (taking into account the spectrum of the signal to be processed);
- Third step: elimination of periodicity;
- Fourth step: Design the passive reference circuit, for example according to wave digital filtering;
- Fifth step: Transform the reference circuit into „Z” plane;
- Sixth step: Transform the flow diagram into time-domain;
- Seventh step: Check the result, simulation;
- Eighth step: Implementation.

Any task performed on a well-chosen plane can usually be solved with fewer design calculations, so the design is simpler and more transparent. The last phase of the design often consists of transforming the algorithm from the "Z" plane to the time domain (sixth step in Fig.1.). The transformation tool is the inverse discrete Laplace transform, which is usually very simple when applied it to a signal flow diagram. The application summary of the inverse "Z" transformation to the individual flow diagram components is shown in Table 1.

DOI: 10.52846/stccj.2024.4.1.58

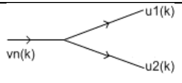
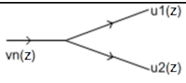
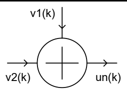
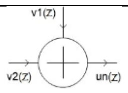
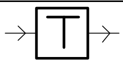
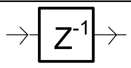
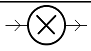
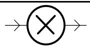
time domain	„Z" domain
	
	
	
	

Table 1. Graphical Inverse Discrete Laplace transformation of FIRS flow diagram components

Of course, the transformation also can be performed on mathematical equations.

In the time-domain column of Table 1. the "k" indicates the takt number. The "T" sampling interval is the reciprocal of sampling frequency of the system.

The implementation of the digital signal processing algorithm, such as creating program code for an embedded controller or hardware definition for a Field Programmable Gate Array (FPGA) circuit, is best suited to an algorithm defined in the time domain. This is why the algorithm check should also take place in the time domain.

II. LINEAR DSP SYSTEM ANALYSIS IN TIME DOMAIN

After the description of the DSP algorithm in matrix-vector form, the time domain simulation can be efficiently performed in the MATLAB environment. The matrix-vector description of the DSP algorithm comes from the difference equations defining the DSP algorithm.

As shown in my previous work [1] as an illustrative simple example for this, we examine a first-order feedback member (Fig.2.). All nodes (y1, y2, y3) of the analyzed algorithm are marked to write the equations.

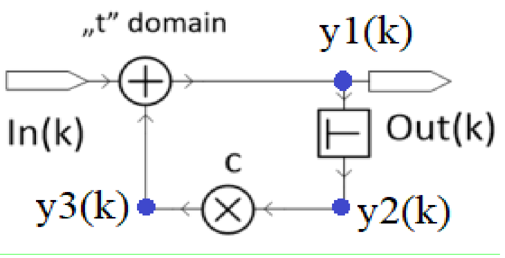


Fig. 2. Tested simple algorithm with marked nodes

By marking all nodes, all sub-calculations of the algorithm become observable.

A. Writing a system of linear equations

Equations (3), (4) and (5) give the connection between nodes with the symbols used in Table 1.

$$y1(k)=In(k)+y3(k) \tag{3}$$

$$y2(k)=y1(k-1) \tag{4}$$

$$y3(k)=c\cdot y2(k) \tag{5}$$

Equations (3), (4) and (5) are formed a system of equations, after rearrangement, all node values interpreted in the "k" step were placed on the left side (6), (7), (8).

$$y1(k)-y3(k)=In(k) \tag{6}$$

$$y2(k)=y1(k-1) \tag{7}$$

$$-c\cdot y2(k)+y3(k)=0 \tag{8}$$

In the following, the (6), (7) and (8) are extended to all nodes (9), (10), (11). Nodes that not included in the equation are interpreted with a zero-multiplication factor.

$$1\cdot y1(k)+0\cdot y2(k)-1\cdot y3(k)=In(k) \tag{9}$$

$$0\cdot y1(k)+1\cdot y2(k)+0\cdot y3(k)=y1(k-1) \tag{10}$$

$$0\cdot y1(k)-c\cdot y2(k)+1\cdot y3(k)=0 \tag{11}$$

The matrix-vector form (12) is written from the system formed by equations (9), (10) and (11).

$$\begin{vmatrix}1 & 0 & -1\\ 0 & 1 & 0\\ 0 & -c & 1\end{vmatrix}\cdot\begin{vmatrix}y1(k)\\ y2(k)\\ y3(k)\end{vmatrix}=\begin{vmatrix}In(k)\\ y1(k-1)\\ 0\end{vmatrix}$$
(12)

The relation in the form (12) can be simplified more (13):

$$M\cdot Y=G \tag{13}$$

The "M" stands for system matrix. This defines and describes the signal processing algorithm. The Y vector is the node vector, which contains the value of the total node in the "k" calculation step. "G" is the excitation vector, which serves the input values of the algorithm in the "k" step. Input values include the input pattern or patterns and state variables. The state variables are the samples calculated in the previous step (k-1) and stored in the "T" storage. The case study above has an input (In(k)) and a state variable (y1(k-1)) only.

Previously, the "M" system matrix was created by using the system of equations. The system matrix can also be created directly from the flow diagram of digital signal processing algorithm. For this, we must use a diagram where all nodes are marked, as seen on Table 1. The system matrix will be a quadratic matrix with the same number of nodes. Nodes are assigned to each row of the matrix as a "source" node, and to each column as a "destination" node. After that, it is examined how we can get from one of the source nodes on the signal flow diagram to the individual target nodes on the signal flow diagram. From the target node, the arrival is examined in the direction opposite to the direction of the signal flow and the result of the arrival is taken into account with a minus one multiplier value.

The calculations to be performed in the "k"-th step are observed in the signal flow diagram. For this reason, the elemental "T" storage building block that stores the state variable is considered broken during the creation of the system matrix.

Examining the main diagonal, here the destination and source nodes are the same, and they are taken into account with a multiplication with one. Fig. 3. illustrates the main diagonal filling of the "M" system matrix.

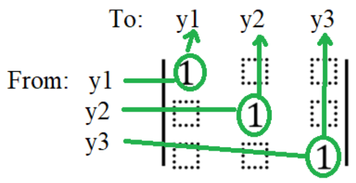


Fig. 3. Fill in strategy of main diagonal of "M".

The filling of the first row of the "M" system matrix is shown in Fig. 4.

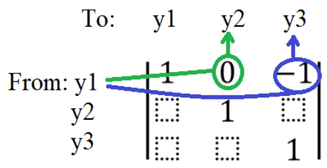


Fig. 4. Filling in the first line of the system matrix

It is not possible to get from the node “y1” to the node “y2”, because in the calculation step (k), the building element "T" (elementary storage) represents a break in the signal flow diagram. The gap between the two nodes is taken into account with a coefficient of 0. At the same time, the value of the state variable calculated in step (k-1) is an excitation signal in step (k), therefore it is taken into account in the vector G, see relation (12).

In the relationship y2 – y1 and y2 – y3, there is a break in the direction opposite to the direction of the signal flow in the algorithm signal flow diagram. This is illustrated in Fig. 5.

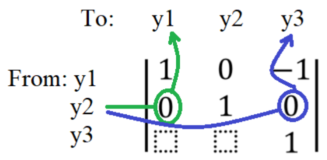


Fig. 5. Filling in the second line of the system matrix

In the third row of the system matrix, the relation y3 – y2 is created by a multiplying term. In the algorithm, the constant of the multiplicative term is denoted by "c". Based on this, the third row, second column element of the matrix will be "-c".

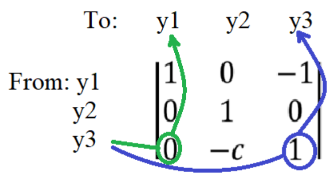


Fig. 6. Filling in the third line of the system matrix

Moving from y3 to node y1 there is no connection, so the first element of the last row of the matrix will be zero.

The system matrix clearly describes the algorithm. The algorithm gives its answers depending on the stimulus signal. According to the description, the samples containing the excitation signals and the state variable values calculated in the previous step are contained in the G vector of relation (12).

B. *Method of solving equations*

In the previous subsection, the description of the algorithm with a system of linear equations (13) and the description methods were given. Now the solution and solvability are being examined. Based on the algorithm, the system matrix M is known from the relationship of the form (13). During the simulation, we look for the response signals of the algorithm, which can be found in the Y vector for each node. The response signal Y will develop as a function of the excitation given by the vector G. It is important here that the initial value of the state variables is known and fixed. In the future, when searching for a solution, the state variables will start from a zero initial value.

To find the solution, relation (13) must be rearranged to (14).

$$M^{-1} \cdot G = Y \qquad (14)$$

From relation (14), it can be seen that the inverse [3] of the system matrix M describing the algorithm must be formed.

Forming the inverse matrix is formally a complex task consisting of many steps. The system matrix (12) of the algorithm that is the subject of our case study contains a symbol "c", which determines the location of the feedback pole. Of course, this constant "c" is a specific numerical value, therefore the inverse matrix formation can be calculated with specific numbers, which is greatly assisted by the MATLAB® program package and its built-in function forming the inverse matrix (15).

$$Mi = inv(M) \qquad (15)$$

The function (15) can be output from the MATLAB® command line interface or inserted into the "m" source code. The result is the "Mi" inverse matrix. The system of equations (14) must be solved "N" times. Based on this, the core of the simulation must be organized into a cycle (16):

```
for k=2:N+1;
 G=[In(k), y1(k-1), 0]';
 Y=Mi*G;
 y1(k)=Y(1);
 y2(k)=Y(2);
 y3(k)=Y(3);
end;
```
(16)

Some explanation for code snippet (16) is important. The loop runs from 2 to N+1, so the loop core will be executed "N" times. For technical reasons, the cycle is run this way and not from 1 to "N", since in this case it would be addressed outside the y1(k-1) vector in the first calculation takt, which would mean an error. In order to prepare the loop, the "N" element length vectors for each node and the "In" vector containing the input test signal must be defined in advance.

To prepare the running, it is advisable to create "empty" vectors, which are actually filled with "0" element values.

```
y=zeros(1,N);
```
(17)

These vectors also automatically set the initial value of the state variables, which in this case is zero. With direct value assignment, the variable value of the state can of course be overwritten and modified if as needed.

The sample sequence connected to the input, i.e. the samples of the excitation signal, is contained in the vector

"In", which also consists of "N" elements. This can be a sample sequence of any arbitrarily chosen signal defined in a time domain. For general tests, it is advisable to use the digital equivalent of the "dirac-delta". The reason for this is that the Dirac pulse for time "dt" contains infinite spectral components with unit amplitude, i.e. its Laplace transform is constant '1' (18) [4].

$$\mathcal{L}\{\delta(t)\} = 1 \qquad (18)$$

The Kronecker delta signal can be considered a sampled signal of the Dirac delta, so it can be produced in the vector "In" with "N" element (19). All its elements will be zero except the first element, which will be set to '1'.

$$In = 1,\ 0,\ 0,\ 0\ \dots\ 0; \qquad (19)$$

For technical reasons, in order for the excitation signal to fit the simulation core (16), the first element must be chosen as zero, and the second element as '1' (20). This is just a "T" time offset.

$$In = 0,\ 1,\ 0,\ 0\ \dots\ 0; \qquad (20)$$

The digital signal processing algorithm will respond to the Kronecker delta excitation with the weight function. The weight function [5] is an extremely important network characteristic function, since the convolution of any other excitation signal and the weight function gives the response [6] to the given signal. Algorithm stability and possible excitation problems [7] can also be observed with the help of a weight function.

Another important property of the weight function is that the spectral image that represents is the same as the transfer function of the signal processing algorithm interpreted in the frequency domain. So we can also check whether the algorithm design frequency criterion requirement is met. The Fourier transform of the weight function gives the complex transfer function, and it absolute value describes the amplitude transfer as a function of frequency.

```
plot(20*log10(abs(fft(Out))));     (21)
```

The relation (21) is actually a nesting of a MATLAB® function, which gives the absolute value of the transfer function in decibel [dB]. The result is displayed on the screen according to (17), but of course the calculated spectrum can also be stored in a vector. When interpreting the result, take into account that a square window was used. Distortions due to the finite number of elements can be influenced by other window functions [8].

III. ANALISYS OF NONLINEAR MODEL

When designing signal processing algorithms, the conditions are considered to be linear in many cases. Algorithms are born in linear conditions.

This simplification can be made as long as the nonlinear effects are negligible. For the real-time operation of the algorithm, we use a high-speed digital signal processing processor (DSP) or a Field Programmable Gate Array (FPGA) device. In the case of these devices, there are usually relatively few bits for the number representation, in the case of lower-priced components, the number representation can even be fixed-point only.

A finite number of bits can cause two nonlinear effects. In some cases, the number no longer fits at certain nodes of the algorithm at the available bit size, therefore it overflows. The other nonlinear effect comes from the lack of necessary precision. Typically, the result of the multiplication operation needs truncation and rounding. The latter adds noise to the signal, but it can also endanger the stability of the algorithm [9].

A. *Overflow problems*

Overflow problems can typically occur at the summation nodes of the algorithm. In the case of summing two or more digital samples (Fig. 7), it may occur that the resulting numerical value exceeds the numerical representation range. In the case of modulo arithmetic, such an overflow causes a gross error and inoperability.

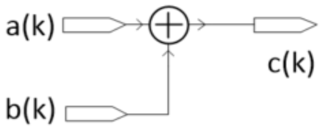


Fig. 7. Typical overflow points in digital signal processing algorithms

The dangerous points of the overflow of the algorithm can be easily detected with the linear analysis described in the previous chapter (it can be considered linear due to the high numerical representation accuracy of MATLAB®).

A vector with "N" elements was assigned to each node of the simulation. During the simulation, all calculated samples are available for the given excitation signal. The samples are very likely to be free of overflow in the simulation environment. By observing the minimum and maximum values of the samples (22) in accordance with a specific implementation environment, problems can be easily detected. This article does not attempt to deal with the problem, it only mentions it.

```
min(y);        max(y);                (22)
```

We can reduce the possibility of overflow by linear transformation of the detected overflow-prone nodes, see Fig. 8. In the example, the value of the constant "c1" is less than 1.

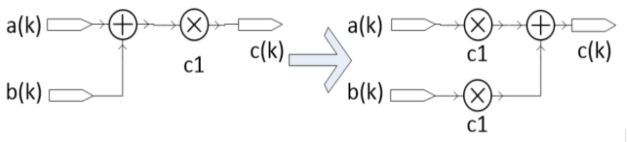


Fig. 8. Reducing the risk of overflow in digital signal processing algorithm

Overflow handling can also be done in the signal processing processor or Field Programmable Gate Array implementation phase of the algorithm by creating a hardware or software saturation method or triangular overflow characteristic in the number representation.

B. *Rounding problems*

Rounding problems typically occur as a result of the multiplication operation. The following signal flow diagram also shows the quantization after multiplication (see Fig. 9) with "Q" marked box.

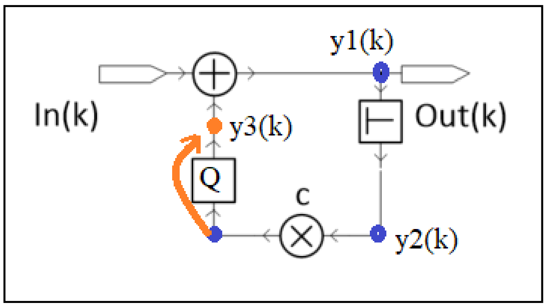


Fig. 9. Quantization point in the algorithm

The system matrix of the case examined during the linear simulation is unchanged, but the core of the simulation (16) must be extended with "Q" function (23).

```
for k=2:N+1;
 G=[In(k), y1(k-1), 0]';
 Y=Mi*G;
 y1(k)=Y(1);
 y2(k)=Y(2);
 y3(k)=Q1(Y(3),nb);
end;                                    (23)
```

In the code fragment (23) the „Q1" is a non-built-in MATLAB® function that we have to create in a file "m" called Q1.m. In the function, we specify the number of bits "nb". The implemented function (24) performs absolute value truncation on the signed, fixed-point fraction number contains "nb" bits.

```
function [q]=q1(x,n)
       q=(floor(x*2^n))/2                (24)
```

Of course, other quantization functions can also be created, such as those performing mathematical rounding. In all cases, the goal is to reproduce the operating environment of the algorithm, as this is how quantization problems arising during implementation can be identified and analyzed.

IV. COMPLEX DSP MODEL SIMULATION IN LINEAR AND NON-LINEAR CASES

In this chapter, a simulation is presented on a wave digital bridge filter. The bridge filter structure is shown in Fig. 10. [2].

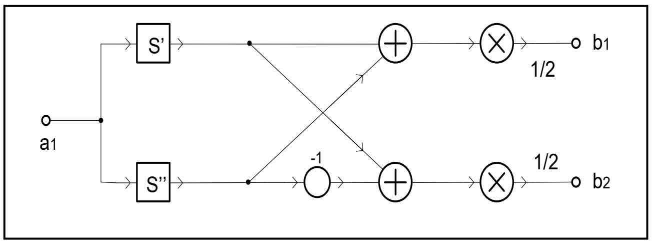


Fig. 10. Sructure of a Lattice Wave Digital Filter

The S' and S" functions can be implemented by using all-pass elements [10].

We use a fifth-order low-pass filter as the subject of the examination. The design of the bridge filter algorithm is not the scope of this article, we only provide the specification and the algorithm that fulfills it.

Filter type: lowpass, LWDF

Attenuation of stop-band:	az = 40 dB
Ripple of pass-band:	aa = 0,2 dB
Upper frequency of pass-band:	fa = 1,8 kHz
Lower frequency of stop-band:	fz = 2,3 kHz
Sampling frequency:	fm= 8 kHz

Fig. 11. shows the algorithm suitable for the specification.

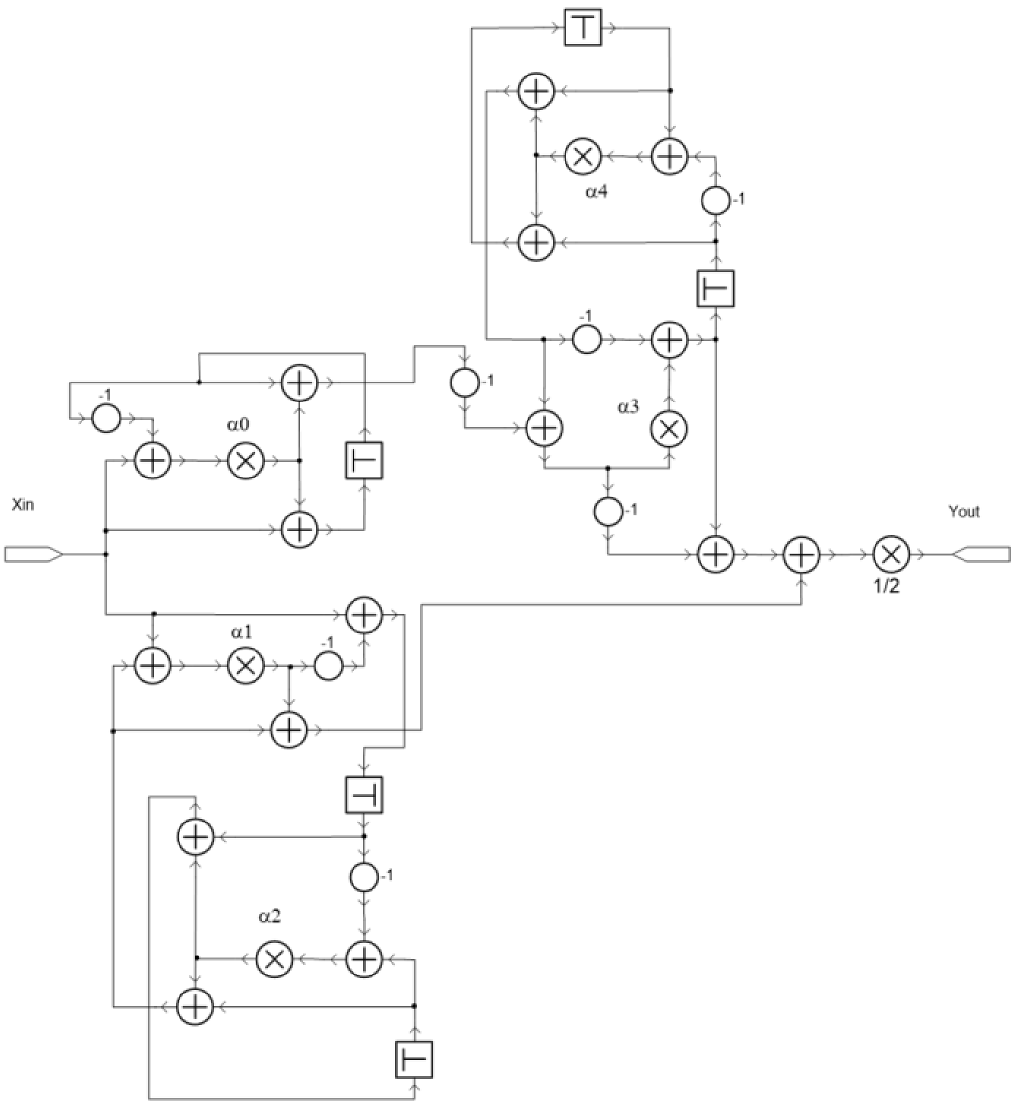


Fig. 11. 5th order Lattice WDF agorithm

The multiplication constants of the algorithm are the following:

α0= 0,3209824

α1 = 0,42110482

α2 = 0,30200688

α3 = 0,1625421

α4 = 0,11045108

This article is not containing the MATLAB® simulation source code in matrix-vector form because of length limitation, but it can be downloaded here: [11]

Based on the linear simulation, the impulse response (weight function) of the algorithm is shown in Fig.12.

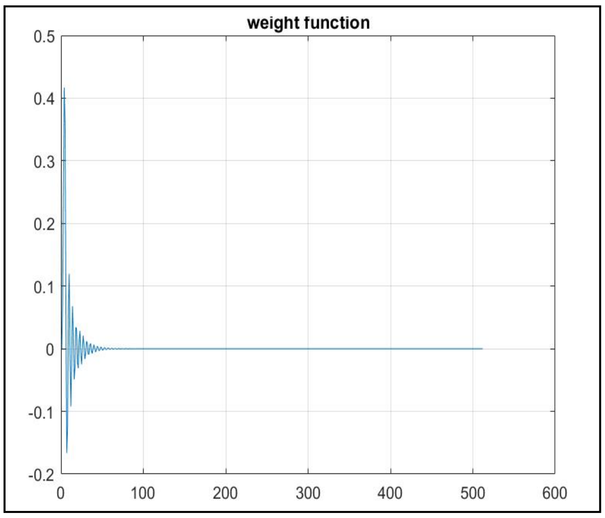


Fig. 12. Impulse response of the algorithm shown in Fig.12.

Fig. 13. shows the absolute value of the Fourier transform of weight function.

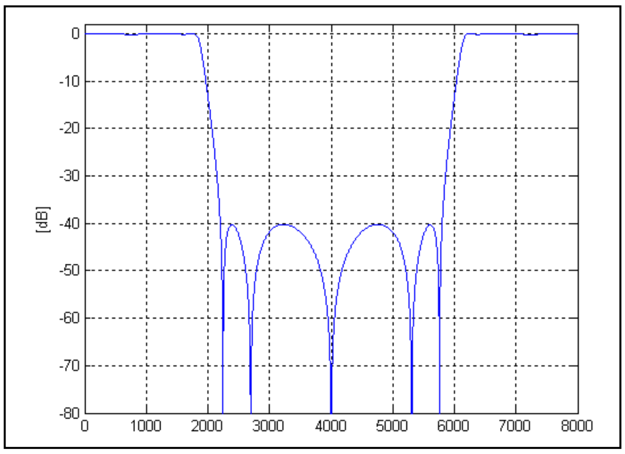


Fig. 13. Fourier trasform of Fig.13.

Fig. 14. shows the simulation results of the quantization effect on the outputs of the individual multiplier components. Fixed-point number representation and 12 bit quantization were performed in the simulation, but this number can be changed by rewriting the constant "nb" in the simulation source code.

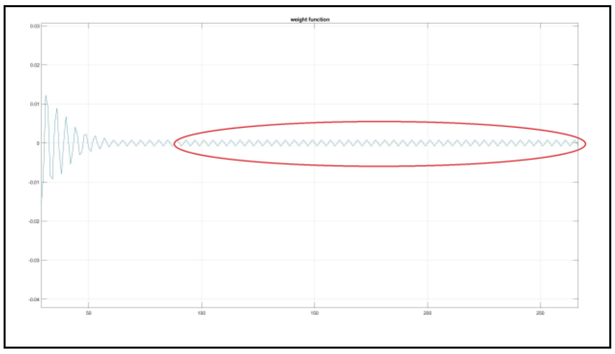


Fig. 14. Limit cycle effect – observed on weight function (fixed point 12 bits – mathematical rounding)

The absolute value of the Fourier transformation of the weight function also shows the quantization noise and the limit cycle phenomenon. In the Fig. 15. the limit cycle observation is marked by red circle, the quantization noise can be observed in the dead band, which is marked in blue.

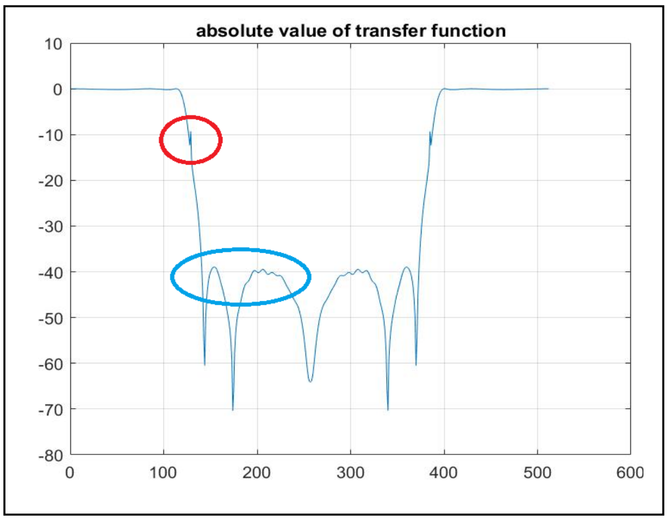


Fig. 15. Fourier transformation of the wave function shown on Fig.15.

By using „floor" function (rounding to the lower number) on 12-bit wide signed fixed numbers, the quantization noise remains, but the limit cycle phenomenon disappears, as shown in Fig.16.

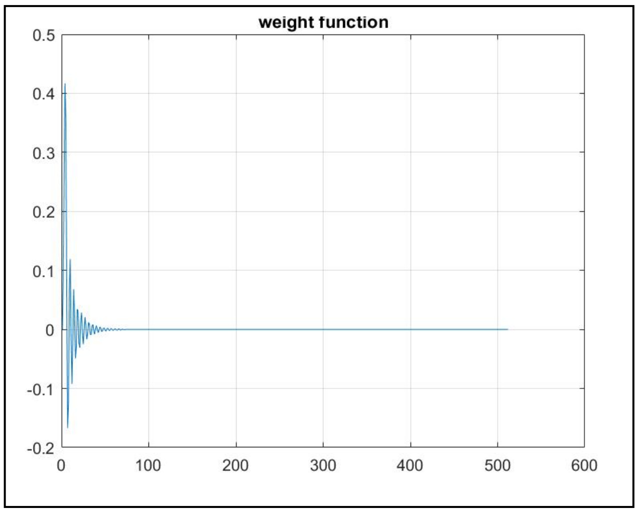


Fig. 16. No limit cycle effect observed on weight function (fixed point 12 bits – "floor" rounding)

The MATLAB® source code of the simulation and the system matrix describing the DSP algorithm are not published in this article because of length limitations, but they can be downloaded from the following location [11]. These files are provided "as-is" and can be used freely.

V. Conclusion

A significant part of DSP algorithms can be described in matrix-vector form. The differential equation system defined in this way can be effectively simulated in the MATLAB® environment. These simulations performed in an environment considered as linear can verify the design result, i.e. the correctness of the signal flow diagram. The MATLAB®

application running in a normal PC environment provides an almost linear environment due to the accuracy of the numerical representation, but in reality, a lower number representation accuracy is available due to the reduced computational power available in real time operation. Relatively few bits (12, 16, 24 depends on the hardware) and often only fixed precision number representation are provided. This causes a clearly visible quantification problem, especially at the output of the multiplier components. These quantization effects can be considered in the linear simulation, so the non-linear behavior of the DSP algorithm caused by truncation and rounding methods can also be well simulated. The quantization noise effect becomes measurable, and the stability of the algorithm can also be examined.

Signal overload may occur at certain points of the implemented DSP algorithm due to the limited number of bits that can be used for number representation. In digital signal processing, an overload - mainly caused by calculation overflow - does not result signal distortion, but the collapse of the entire DSP algorithm operation. With the help of the time domain analysis performed in the MATLAB® environment, the dangerous nodes affected by the overflow of the DSP algorithm can be detected. After detection, the probability of an overflow can be significantly reduced by linear transformation of the algorithm.

A deep analysis of the algorithm is definitely recommended as the last phase of the DSP algorithm design. After successful simulations the verified algorithm can be implemented by using a DSP processor or an FPGA.

Acknowledgment

We would like to thank the Óbuda University for supporting my research.

Special thanks to STCCJ for providing the opportunity to present our research results.

References

[1] Tibor WÜHRL, Simulation of Digital Signal Processing Algorithms in Time Domain; In: Szakál, Anikó (szerk.) IEEE 17th International Symposium on Applied Computational Intelligence and Informatics SACI 2023 : Proceedings Budapest, Magyarország : Óbudai Egyetem, IEEE Hungary Section (2023) 818 p. pp. 767-772. , 6 p.

[2] Tibor WÜHRL: Hullámdigitális jelfeldolgozás alapjai; Budapest, Óbudai Egyetem 2010, 124 p

[3] Mathworks: Compute inverse of Symbolic Matrix; https://www.mathworks.com/help/symbolic/inv.html

[4] Mathematics: Laplace Transform of Dirac Delta function;
https://math.stackexchange.com/questions/1690741/laplace-transform-of-dirac-delta-function

[5] M. Hoffmann: Digital signal processing mathematics
https://cds.cern.ch/record/1100533/files/p10.pdf

[6] Mathematics: Convolutions and the convolution theorem
https://math.stackexchange.com/questions/2770374/convolutions-and-the-convolution-theorem

[7] ALFRED FETTWEIS - KLAUS MEERKÖTTER: Suppression of Parasitic Oscillation in Wave Digital Filters; IEEE VOL. CAS-22, NO. 3, March 1975

[8] Luis Chioye, Art Kay: Fast Fourier Transforms (FFTs) and Windowing, Texas Instrumentrs; https://training.ti.com/sites/default/files/docs/adcs-fast-fourier-transforms-and-windowing-presentation-quiz.pdf

[9] Oleksiy Kuznyetsov: Towards the Control of Multistability, in a PWM-Driven DC Motor Drive by Short Forcing; Acta Polytechnica Hungarica Vol. 19, No. 11, 2022

[10] Lajos GAZSI, Explicit Formulas for Lattice Wave Digital Filters; IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS, VOL. CAS-32, NO. 1, JANUARY 1985

[11] Tibor WÜHRL, Downlodable MATLAB® source codes: https://hti.kvk.uni-obuda.hu/publications-of-dr-habil-wuhrl-tibor/