TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

MIAO ZHAO

DESIGN OF DIGITAL INTEGRATOR FOR ROGOWSKI COIL SENSOR

Master of Science Thesis

Examiner: Pro. Timo D. Hämäläinen

Examiner and topic approved by the Council of the Faculty of Computing and Electrical Engineering on 4 March 2015

ABSTRACT

MIAO ZHAO: Design of Digital Integrator for Rogowski Coil Sensor
Tampere University of Technology
Master of Science Thesis, 94 pages, 5 Appendix pages
April 2015
Master's Degree Programme in Information Technology
Major: Digital and Computer Electronics
Examiner: Professor Timo D. Hämäläinen

Keywords: Rogowski coil sensor, digital integrator design, Newton - Cotes Integration, Genetic Algorithm optimization, finite word length effect.

The goal of this thesis is to create a well performed digital Rogowski coil integrator on PC for later implementation on FPGA, and the final filter is applied with fixed point arithmetic. Integrator's design and optimization based on the specification provided by the company. During the implementation, the constraints of the hardware should be taken into account, and the design method needs to be verified by simulations and practical experiments and tests.

There are two design phases to implementing the filter. The first phase is software implementation, the integrator is realized by creating the MATLAB and C models. The other phase is hardware realization. By software application, the filter could be simulated with targeted test benches. After the software application is verified, hardware implementation could be carried out if it is necessary. In this thesis, RTL model is derived from the C model via translating it with VHDL. Afterward, the integrator is implemented on FPGA board for practical field tests.

From the tests, the validity, practicability of the Rogowski integrator have to be verified from the perspective of both functionality and performance. The software implementation of the integrator is capable of filtering different kinds of the input signals with reasonable and acceptable outputs. Meanwhile, in the practical application, the integrator performed well when dealing with various earth fault cases. All, in brief, this Rogowski integrator has to satisfy the standard of the design specification.

PREFACE

This thesis was made for the Department of Pervasive Computing, Faculty of Computing and Electrical Engineering, Tampere University of Technology, in cooperation with ABB Oy Medium Voltage Products.

I would like to thank my examiner Prof. Timo D. Hämäläinen at Tampere University of Technology for help with my thesis plan and the text. I would like to thank my supervisor Frej Suomi at ABB for the valuable insight during my design of the Rogowski coil integrator. Thank Mika Kauppinen and Juha Ekholm for the opportunity to do this thesis at ABB Medium Voltage Products in Vaasa. At last, I'd like to thank my parents for their unconditional understanding and support for my studying and working in Finland.

Tampere, 16 March 2015

Miao Zhao

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

CT	Current Transformer
AC	Analog Current
PC	Personal Computer.
FPGA	Field-Programmable Gate Array.
BIBO	Bounded-Input Bounded-Output.
ADC	Analog -to-Digital Converter.
VHDL	Verilog Hardware Description Language.
LTI	Linear Time Invariant.
MSB	Most Significant Bit
LSB	Least Significant Bit
GA	Genetic Algorithm.
RMS	Root Mean Square
RTL	Register Transfer Level

i	*Measured current from Rogowski sensor*
V_{out}	*Output signal form Rogowski sensor*
M_R	*Mutual inductance*
$h(k)$	*Impulse response sequence*
Z	*Complex variable*
a	*Denominator of filter coefficients*
b	*Numerator of filter coefficients*
N	*Filter's order*

1. INTRODUCTION

1.1 Background and Motivation

In electrical engineering, including power system, current transformers (CTs) have been traditionally used for protection and measurement applications for the main reason of their ability to produce the high power output. However, the emergence of microprocessor based equipment made high power output unnecessary and introduced other measurements techniques. One such measuring device that equips many advantages over CTs is the Rogowski coil transducers [1].

Rogowski coils are commonly incorporated in measuring and protection systems throughout the industry and in research. Its feature of “air-cored” offers an advantage over iron-cored measuring devices [2]. Thanks to the benefits of Rogowski coils, such as high accuracy, excellent linearity, wide dynamic range, wide band and no magnetic saturation, they can replace CTs and be used for protection, metering and control purpose in the general electric current measurement field [1], [3], [4], [5], [6], [7]. The advantages of Rogowski coil sensor over the traditional current transducers will make it be more extensively used in the future.

Based on the principle of Faraday’s law of electromagnetic induction, Rogowski coil produces a voltage output signal which is proportional to the derivative of the current signal. An integrator is required to restore the measured AC waveform from the voltage signal induced in the Rogowski coil sensor [1]. With a well-designed integrator, the Rogowski coil can achieve excellent performance such as higher accuracy and better linearity than conventional CTs [6]. Thus, an integrator with high performance is the key to guaranteeing the high measuring accuracy of Rogowski coil sensor [3].

This thesis’ goal is to design a well performed digital Rogowski coil integrator on PC for later implementation on FPGA. The idea is to find a proper method to design an integrator with respect to all the specifications and requirements need to be concerned, and improve the performance of the filter as much as possible. Additionally, the constraints of the hardware should be taken into account, and the design method needs to be verified by simulations and practical experiments and tests.

1.2 Requirements and Constraints

This thesis was made by request of ABB Oy, Medium Voltage Products in Vaasa. The main requirements and specifications are listed as below:

1. The target frequency range of the integrator is from 10 to 1250 Hz. The integrator will work in electric power grid with frequency of 50 Hz. Thus, the primary focus should be paid to the frequency around 50 Hz.

2. For the frequency response, the magnitude response should follow the equation Magnitude = 50 / Frequency as close as possible. The phase response should be -90 degree as close as possible.

3. The group delay should be as flat as possible and as small as possible. Additionally, it is better that from 50Hz, the group delay is positive and as close to zero as possible. The max group delay should be no more than two samples when the sampling frequency is 9600Hz.

4. The filter needs to be implemented with fixed point arithmetic.

5. The step response should decay to zero as near as possible.

6. The filter should be BIBO stable.

7. Measurement and ADC quantization noise should not accumulate to disturbingly large DC levels in the integrator.

8. The structure and bit widths of filter implemented on FPGA should be designed to not suffer too many rounding errors.

1.3 Design Platform and Tools

In this thesis, the optimized integrator is finally implemented on the hardware platform, which is an FPGA applied in a protection relay. Figure 1 briefly shows the general workflow, where that Rogowski integrator works.

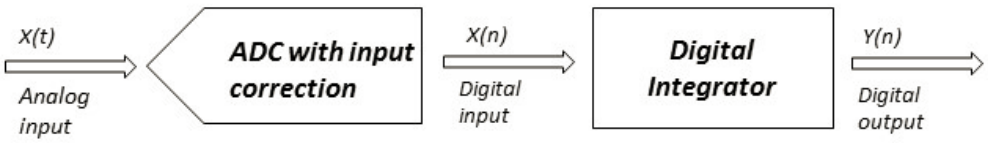


Figure 1. Rogowski integrator workflow

We developed the integrator in a software approach. To guarantee the computability of the filter structure, and as well to verify the accuracy of this structure are of substantial importance in the design [9]. Thus, before the filter algorithm is finally implemented on the FPGA board, the digital filter is designed and simulated on PC with the aid of MATLAB and Visual Studio. All of the early designs and optimizations on the digital integrator are done with MATLAB. After that, Visual Studio is used to create a filter

model in C language. The C model is utilized for making the VHDL, where the advantage is that C is easier for a VHDL designer to understand that MATLAB code.

1.4 Organization of Thesis

In the subsequent part of this thesis, Chapter 2 introduces the theoretical background related to the thesis work. In Chapter 3, specific integrator design algorithms are realized, tested and corresponding comparisons are conducted through simulation with the using of MATLAB. The best design method for the following actual design work is chosen in the end. After deciding the design method of the filter, a detailed explanation of the process of practical design work is introduced in chapter 4. This section contains the distinct approaches to filter's software and hardware implementations. Chapter 5 provides the testing results of the filter, the analysis of filter's performance is made afterward. Chapter 7 consists of the conclusion of the thesis work and possible further work that could be done after this thesis work.

2. THEORETICAL BACKGROUND

2.1 Characteristics of Rogowski Coil Sensor

A Rogowski Coil is an air-cored coil, in which the windings are twined evenly on a non-magnetic skeleton. The Rogowski coil has galvanic isolation [45] with the measured circuit, which is convenient for isolated measurements of high voltage circuits. Thus, it can be utilized as sensing element, such as electronic current transformer. The detailed operation theory of Rogowski coil can be found by referring [9].

The measurement of current is based on Faraday's Law and Ampere's Circuital Law. Figure 2 shows the structure of a Rogowski coil. When the measured current *I* goes through a cable encircled by the coil, the output signal is a voltage V_{out}, which is proportional to the rate of change of the measured current.

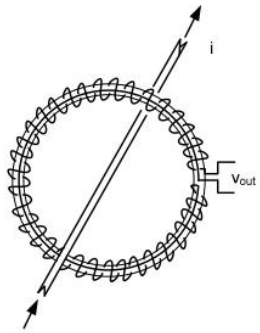


Figure 2. Rogowski Coil's structure [10]

Rogowski coil sensor's current vs. voltage properties is shown in Equation (1):

$$V_{out} = -M_R \frac{di}{dt} \tag{1}$$

In which, *t* is the time and M_R is the mutual inductance between the primary and the secondary windings [10].

The output signal is a so-called transmitted signal, it is proportional to the derivative of the primary current. The phase shift between the input and output signal is 90 degrees [10]. The original signal can be reproduced by integrating the output voltage, as Equation (2) shows in the following [11]:

$$i = \frac{1}{M_R} \int_0^t V_{out} dt \tag{2}$$

Due to the absence of iron in a Rogowski coil sensor, no magnetic saturation occurs. As a result, the output is linear over the whole current range up to the highest current. However, in practical the cabling and connectors will restrict the dynamic range.

Intermittent Earth Fault

An imperative reflection of the integrator's performance is the intermittent earth fault. Intermittent earth fault is a particular type of failure that is encountered mainly in compensated grid with underground cables. According to [12], intermittent earth fault "is characterized by repetitious of short duration self-extinguishing faults. This kind of failure tends to be difficult for conventional directional earth fault protection relays to detect due to highly irregular wave shape of residual current. Whereas residual overvoltage relays used typically as a substation backup protection have better chances for fault detection because of more steady behavior of residual voltage." Due to this fact, intermittent earth fault can often cause non-selective tripping of the substation backup protection and eventually an outage with substantial cost. " That is why in thesis, a particular integrator's performance test for the intermittent earth fault is made. It is necessary to guarantee that the integrator functions well under intermittent earth fault.

2.2 Integrators for Rogowski Coil Sensor

2.2.1 Analog and Digital Integrators

Analog and digital methods can implement the integration part of the Rogowski coil. Analog integrators are usually composed of inertial elements such as amplifier, resistors, and capacitors. Currently, analog integrators are generally at the end of Rogowski coils to transform input current's amplitude and phase. However, analog devices are not temperature and aging stable, which could cause problems, such as leakage, loss of capacity and zero-drift in the analog devices. In addition, it is relatively inflexible to design the feedback and compensation of an analog integrator [13]. All of these factors existing in analog devices are potential to result in integration errors.

Digital integrator, however, possesses the outstanding features which overcome the inherent weak points of analog one. It is based on the sampling algorithm, which samples the signal directly and then restores the original signal by numerical integration methods. Furthermore, the disadvantages of classical analog integrator, such as thermal and time stability, could be entirely avoided [14].

Digital integrator, featuring higher accuracy and stability, good phase response performance as well as flexible structures, are used widely now in Rogowski coil sensors [15]. There are several methods, such as using Data acquisition card and PC to implement digital integrators, the most common method is using ADC and DSP to realize the digital integrator [14].

Therefore, in this paper, a digital integrator of the Rogowski coil was designed, using several integration algorithms.

2.2.2 Digital Filter Design Issues

The design of digital integrators is essentially the design of digital filters. According to [16], "A digital filter is a system that implements a mathematical algorithm in hardware and/or software, in order to achieve the filtering goal in a particular region of a signal spectrum."

Types and Representations

Digital filters can be categorized into several groups depending on the criteria used for classification. The two main types of digital filters are *finite impulse response* (FIR) and *infinite impulse response* (IIR) filters. Both of the filters can be represented by the impulse response sequence, which is often denoted as $h(k)$ $(k = 0, 1, \dots)$ [16]. Since the digital filters we are going to design are linear and time-invariant (LTI), the general forms of FIR filter and IIR filter can be expressed as:

$$y(n) = \sum_{k=0}^{N-1} h(k)x(n-k) \tag{3}$$

$$y(n) = \sum_{k=0}^{\infty} h(k)x(n-k) \tag{4}$$

Where $x(n-k)$ is present input and $y(n)$ is the output sequence. FIR filters are also called as non-recursive digital filters since they do not have the feedback. Sometimes, however, recursive algorithms can be used for the realization of FIR filter. Unlike FIR filters, IIR filters have the feedback and they are therefore known as recursive digital filters. Figure 3 shows the block diagrams of FIR and IIR filters.

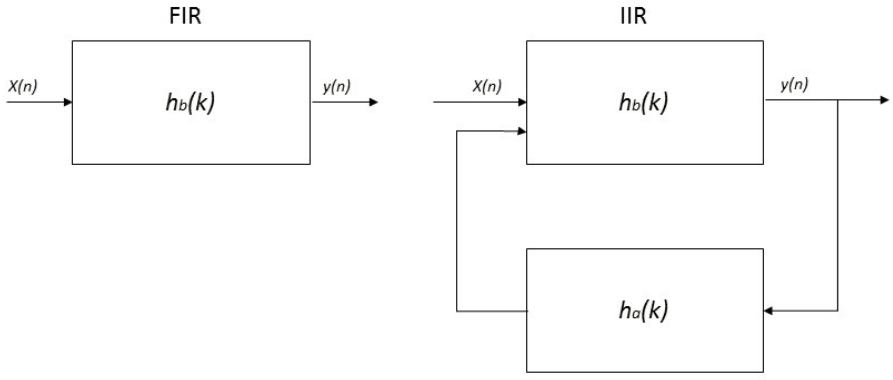


Figure 3. Block diagrams of FIR and IIR filters

In practice, the IIR filtering equation is expressed in a recursive form as equation (5) since it is not feasible to represent the impulse response, which is theoretically infinite.

$$y(n) = \sum_{k=0}^{\infty} h(k)x(n-k) = \sum_{k=0}^{N} b_k x(n-k) - \sum_{k=0}^{M} a_k y(n-k) \quad (5)$$

Where a_k and b_k are the coefficients of the filter. Equation (3) and (4) below are also respectively called as difference equations for FIR and IIR filters. For LTI as well as causal filters, FIR and IIR filters can be expressed as transfer functions in the Z-domain, then the forms could be like the following:

$$H(z) = \sum_{k=0}^{N-1} h(k)z^{-k} \quad (6)$$

$$H(z) = \sum_{k=0}^{N} b_k z^{-k} / (1 + \sum_{k=1}^{M} a_k z^{-k}) \quad (7)$$

Where the filter order equals to the greater one of *M* or *N* (in FIR filter, the order equals to *N*). Transfer functions are of huge importance in evaluating the frequency response for the filters.

FIR and IIR can also be differentiated by visualizing the place of the poles. For FIR Filter, all of the poles of the filters locate at the origin, and, as a result, the shape of the frequency responses are determined only from the locations of the zeroes. On the other hand, IIR filters have the poles to move inside the unit circle, permitting them to contribute more heavily to the shape of the frequency responses.

For FIR filters, they often have linear phase responses, and they are always stable. When comparing with the IIR filters, it frequently needs higher filter order than FIR filters to achieve the same response for fixed specifications [16]. As a price, for IIR filters, they always have non-linear phase responses and have big potentials to be unstable.

Digital Filter Design Issues

Selection of Filter Type

It is always a trade-off between the choices of IIR and FIR filters. However, the decision is mostly dependent on the specific design conditions. It is much easier to choose FIR filter when there is a need for linear phase, and there is no strict requirement on the filter order. However, when considering the filter order or when particular phase response is needed, it is better to choose IIR filter. Even though, there is a big risk to get unstable filter during the design. In many applications, the linearity of the phase response is not as important as the computational efficiency [8]. However, in this thesis, specific phase response is required, and with the resource consumption of hardware taken into account [18], which makes IIR filter a better choice.

General Design Steps

Following are the design steps for an IIR filter in this thesis:

1. Filter Specification. Filter specification determines the characteristics of the desired filter and indicates the requirements we should meet during the design process. The objective of designing a digital filter is to develop a causal transfer function $H(z)$ which can satisfy the filter specifications.

2. Coefficients Calculation. When the specification is made, an essential step is to decide the transfer function, which could realize the filter frequency response specifications as well as possible. In other words, it means to find out the optimal filter coefficients. The filter's order needs to be estimated after the selection of the digital filter type. There are several approaches to developing a filter, the particular method is based on the transfer function and the filter specifications. Additionally, it is necessary to guarantee the filter to be stable when an IIR filter is desired.

3. Structure Realization. For IIR and FIR filters, there are different kinds of structures. IIR filters often use direct, cascade and parallel forms, for FIR filters, direct form is widely used. When considering the finite word length problems, cascade and parallel structures are better choices than direct form in IIR filter design, as cascade and parallel structures have fewer coefficient sensitivity problems than direct forms, especially when they are with high filter order. In many applications, a higher order IIR filter is implemented either in a presentation of cascade of second-order sections or in the form of a parallel connection of second-order sections. Poles are more sensitive to the quantized coefficients and other finite word length effects [8].

4. Implementation. The practical filter application is either done in software or hardware, but neither can provide infinite precision for the coefficients as well as the signal variables. It turns out that direct realization of a digital filter may not satisfy the designer's expectation for the performance due to the finite precision arithmetic [9]. Thus, it is necessary to develop new suitable filter structures when converted from transfer function, and to avoid significant finite word-length effects.

Fixed Point Consideration

In digital filters, the arithmetic operations can be coarsely divided into three broad categories: *floating-point, fixed-point, and block floating point* representations [18]. We assume floating-point arithmetic of the calculation and approximation steps during the design process. Then fixed point arithmetic is employed by replacing the floating point arithmetic. It is evident that finite word length effects will degrade the filter's performance, and the extent of the effects have to be confirmed before the IIR filter is

finally implemented. In addition, it is necessary to find a remedy if the degradation is not acceptable [16].

In the practical filter implementation, it is often necessary to represent the coefficients by specific number of bits. The number is represented in the way that a binary point is used to separate the integer part from the fractional part [8]. In addition, a sign bit is placed in the leading position for the signed fixed point number. The fixed point format varies depending on the way negative number are represented. In this case, the fixed point negative numbers are represented in *two's complement* representation. For example, a positive number has the sign bit 0 while a negative number has the sign bit 1 [20]. In general, the decimal equivalent of a binary number consisting of B_1 integer bits and B_2 fractional bits, which could be represented as following form [21]:

$$\sum_{i=-b}^{B1-1} A_i 2^i \tag{8}$$

From above we get:

$$A_{B1-1}A_{B1-2} \dots A_1 A_{0\Delta} A_{-1}A_{-2} \dots A_{-B2} \tag{9}$$

Where each bit A_i is either a 0 or a 1, the leftmost bit $A_{B1-1}$is called the most significant bit (MSB), and the rightmost bit, $A_{-\mathrm{B2}}$ is called the least significant bit (LSB). Δ denotes the binary point or radix point, which is fixed.

Radix point of a number is used to separate its integer and fractional numeric fields [18]. For description convenience, a fixed point number is represented in a form of **s B.B2** in this thesis, where B is the sum of B_1 and B_2. In other words, B is the total bit width of the number, and B_2 is the bit width of the fractional part.

The basic arithmetic operations in the implementation of digital filtering algorithms are addition (subtraction) and multiplication. It should be carefully treated that the arithmetic of addition and multiplication of two fixed-point binary numbers may result in more bits than those in the two numbers [22]. As a consequence, when the result is stored back in the memory, the result must either be truncated or rounded to fit the memory word-length. That is one of the word-length effects caused by the finite word length.

The use of finite precision arithmetic leads to three types of finite word-length effects [16].

1. Quantization. The quantization includes input/output quantization and the coefficient quantization. We mainly focus on the coefficient quantization in this thesis. Quantization of the filter coefficients is likely to disturb the locations of the filter' poles and zeroes. Then, the corresponding filter response is different from the ideal filter response. This deterministic frequency response error is referred to as coefficient quantization error.

2. Arithmetic round off errors. Quantization makes it necessary to quantize filter calculations by rounding or truncation. Round off noise is like low-level noise in the filter output, which comes from rounding or truncating calculations during the filtering process.
3. Overflow. With fixed point arithmetic, it is possible for filter calculations to overflow. The overflow occurs when the result of an addition exceeds permissible word-length.

Structure Realization

Structure realization is based on the known transfer function. The structural representation plays a crucial role in filter implementations, for it provides the relations between intermediate variables with the input and the output [8].

For one filter or one transfer function, several equivalent structures are available. However, in practice, when fixed point arithmetic is employed, a particular realization behaves differently from its other equivalent realizations. Hence, under a fixed point arithmetic, it is necessary to choose a structure which has good quantization properties carefully. Here in our case, we only consider the realization problem of IIR filters. We outline several common forms that are used in the realization of IIR filters.

Direct Form I Structures

As Figure 4 shows, Direct Form I structure on the left can be regarded as an all-zero filter section followed in the series by an all-pole filter section. In general, it is always possible to implement a N^{th}-order filter using only N delay elements (Here we assume M equals to N). Direct Form I structure needs twice as many delays as are necessary.

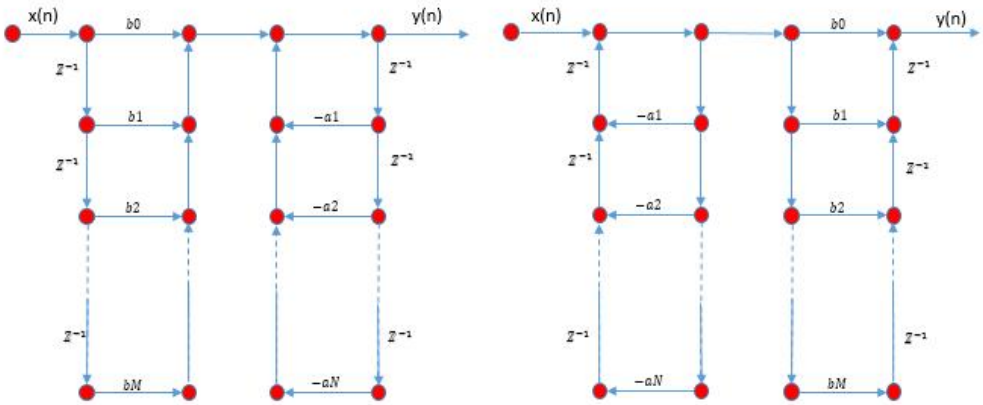


Figure 4. Direct and Transposed Direct Form I Structures

The right side of the figure shows transposed Direct Form I structure. The difference between direct and direct transpose realizations is not significant. Both structures have the same multiplication coefficients. The position of delays determines the main difference. Similar to direct realization structure, the direct transpose realization structure uses 2*N* delays, (2*N*+1) multiplications and 2*N* additions.

One advantage of the direct form I implementation is that it cannot overflow internally in two's complement fixed-point arithmetic while most IIR filter implementations do not have this property [23]. The disadvantages of this realization includes the greatest sensitivity to the accuracy of coefficients, and the biggest complexity due to implementation.

Direct Form II Structures

The direct form II structure is shown on the left side of figure 5[15]. It can be regarded as an all-pole filter section followed by an all-zero filter section. It is canonical with respect to delays, since the delay elements associated with the all-pole, and all-zero sections are shared with each other. As a result, direct form II realization structure has reduced number of delays to the minimum, that is, *N* delays. That is the main advantage over direct form I realization structures.

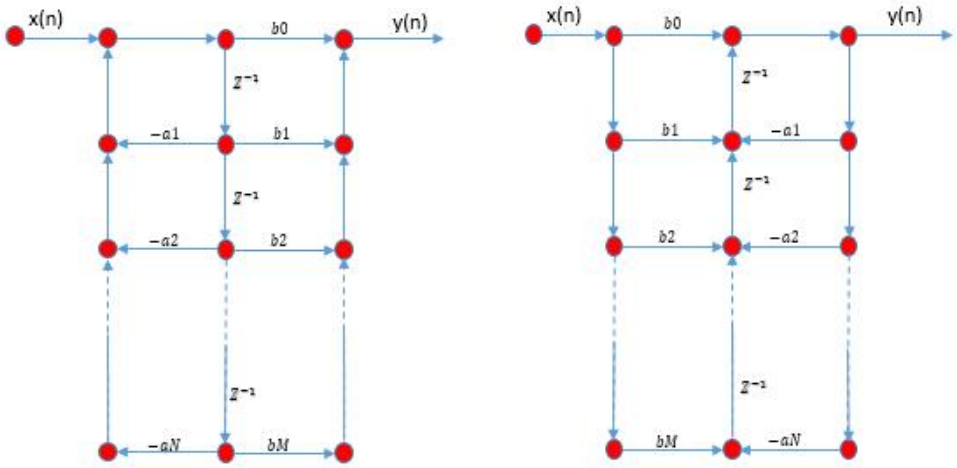


Figure 5. Direct and Transposed Direct Form II Structures

On the right side of figure 5 is the transposed direct form II structure It uses *N* delay elements, (2*N*+1) multiplications and only (*N* +1) additions, while direct form II needs (*N* +1) additions.

Unlike direct form I structure, overflow can occur at the delays in direct form II structure with fixed-point arithmetic. In general, all direct-form structures are very sensitive to round-off errors in the coefficients, especially for filter with high orders. For this reason, series low-order sections are applied in filter realization to get lower quantization sensitivity [23].

Cascaded Second-Order

As indicated before, under fixed point arithmetic, equivalent structures have different performance. Especially when the filter order is high, direct realization of the filter is very sensitive to finite word-length effects and should be avoided in these cases [16]. In practice, the transfer function is broken down into smaller sections, typically second and/or first order blocks, which are connected in cascade. For a cascaded second order filter, there are two 2nd order filters connected in the cascaded form. Figure 6 and Figure 7 describe the differences of these two structures.

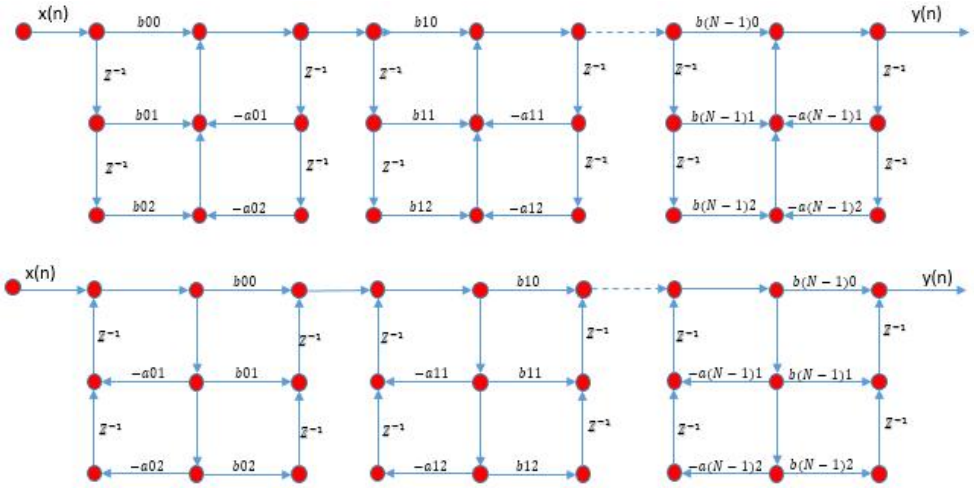


Figure 6. Cascaded second *order filter with* Direct *and Transposed Direct Form I structures*

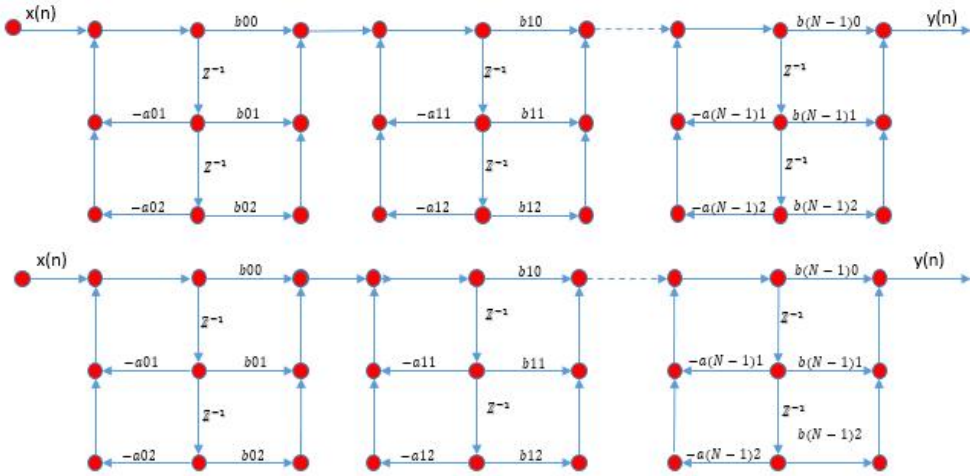


Figure 7. Cascaded second order filter with Direct and Transposed Direct Form II structures

2.3 Digital Integration Algorithms

This chapter presents several known integrator design practices with general theories. The specific design for a starting point filter will be explained later with details in chapter 3.

2.3.1 Direct Design Method Based On the Frequency Response

By analyzing the design specification, there is an explicit requirement for magnitude and phase response in the particular frequency range of interest. With the help of MATLAB Signal Processing Toolbox, we could use the function "***invfreqz***" to generate corresponding filter coefficients. This function finds a discrete-time transfer function that corresponds to a given complex frequency response. From a laboratory analysis standpoint, "***invfreqz***" can be used to convert magnitude and phase data into transfer functions [24, 25]. This function performs like [26]:

```
[b,a] = invfreqz(h,w,n,m,wt,iter,tol,'trace')
```

Where the parameters are explained in the following:

- *h* is the vector that stores the complex frequency response
- *w* is the vector specifying the frequency points
- *m* is the desired order of the numerator
- *n* is the desired order of the denominator
- *wt* is a weighting factors vector of the same length as w
- *iter* is the iteration bounds
- *tol* is the convergence parameter

The function returns the real numerator and denominator coefficients in vectors ***b*** and ***a*** of the transfer function:

$$H(z) = \sum_{k=0}^{n} b_k z^{-k} / \left(1 + \sum_{k=1}^{m} a_k z^{-k}\right) \tag{10}$$

Based on the frequency response specification, it is easy to specify the input parameters. It is a very fast way to design a filter. However, the resulted filter performance needs to be tested since specifying comprehensive frequency points for the filter is difficult to some extinct, and it could not describe the frequency response in an exact way with good precision. As a result, the filter we obtained with have deviations from the specifications. It is necessary to specify the input parameter carefully during the design, and several iterations may do help for getting better coefficients.

2.3.2 Newton-Cotes Integration Rules

The transfer function of digital IIR integrators could be derived by the simple application of z-transform to the difference equations defined by the various numerical integration rules [27, 28]. There are several methods to implement the basic integrators which are based on the Newton-Cotes integration rules, such as Al-Alaoui's first-, second- and third-order integrators and [27, 30], Ngo's third-order integrator [31]. Newton-cotes interpolation formula is a technique that calculates a definite integral/curve by replacing that curve by a more integrable and simpler curve [32].

In numerical analysis, the Newton-Cotes integration rules are carried out by evaluating the integrand at equally spaced points [47]. A related good starting point filter is necessary to guarantee the convergence to the optimal solution. A few of the classical integrators using newton-cotes rules are worth mentioning, like Trapezoidal Integrator [27], Simpson Integrator [33], Rectangular Integrator, which are among the most popular methods for approximating the evaluation of the definite integrals [18]. In this section, these three integration rules are introduced.

Trapezoidal algorithm

The numerical integration rule of Trapezoidal algorithm [27] could be represented as the following function:

$$\int_c^d f(t)dt \approx \frac{h}{2}[f(c) + f(c+d)]; \quad h = (d-c); \tag{11}$$

Where, $f(t)$ is the function that needs to be integrated between the interval c to d, and c is smaller or equal to d. The transfer function of the trapezoidal integrator to Z domain is given by:

$$H_T = \frac{T}{2}\frac{1+z^{-1}}{1-z^{-1}} \tag{12}$$

Where T is constant, that is obtained from the transformation. Based on the transfer function, the coefficients of the integrator are directly obtained in (13). However, the derived filter is not stable, this issue will be discussed later.

Numerator coefficients: $b_0 = 1, \quad b_1 = 1;$

Denominator coefficients: $a_0 = 2, \quad a_1 = -2.$ (13)

Simpson algorithm

The Simpson algorithm [27, 33, 34] could be represented in the following formation:

$$\int_c^d f(t)dt \approx \frac{h}{3}\left[f(c) + 4f\left(\frac{c+d}{2}\right) + f(d)\right]; \quad h = (d-c); \tag{14}$$

The transfer function of the Simpson 1/3 integrator is given by:

$$H_S = \frac{T}{3}\frac{z^{-2}+4z^{-1}+1}{1-z^{-2}} \tag{15}$$

This filter is a third order filter. The coefficients are listed as following:

Numerator coefficients: $b_0 = 1$, $b_1 = 4$, $b_2 = 1$;

Denominator coefficients: $a_0 = 3$, $a_1 = 0$, $a_2 = -3$. (16)

Except for the Simpson 1/3 rule, there is another Simpson 3/8 rule which could yield a 4th order filter, which we will not discuss in our case.

Rectangular Algorithm

The Rectangular algorithm [27, 35, 36] could be represented as following:

$$\int_c^d f(t)dt \approx \mathrm{h} * \mathrm{f(c)}; \quad h = (d-c); \tag{17}$$

The transfer function of the Simpson integrator is given by:

$$H_R = T\frac{1}{1-z^{-1}} \tag{18}$$

This filter is a third order filter. The coefficients are listed as following:

Numerator coefficients: $b_0 = 0$, $b_1 = 1$;

Denominator coefficients: $a_0 = 1$, $a_1 = -1$. (19)

2.3.3 Pole-Zero Placement

In general, the frequency response of an ideal digital integrator [32, 37] is given by:

$$H(w) = \frac{1}{jw} \tag{20}$$

Where $j = \sqrt{-1}$ and w is the angular frequency in radians.

The filter can be directly derived since the transfer function of this filter is known. Then by analyzing the specification, new desired filter with higher order is designed by adding extra zeroes and poles to the primary filter. This approach is also known as the ***pole-zero placement*** method. When a zero is placed at a given point on the z-plane, the frequency response will be zero at the corresponding point. On the other hand, a pole

produces a peak at the relevant frequency point. In order to make the filter stable, all of the poles need to be inside the unit circle.

2.3.4 Optimization Method

Since the design of digital filters involves multiple design specifications, which are often conflicting, generally there is no easy way to an optimal design. Optimization based methods are presented to design digital filters that would satisfy the prescribed specification. However, optimization methods often require more computation work than basic design methods and are not so time efficient.

After generating the starting point filter, global optimization method is employed. ***Genetic Algorithm (GA)*** [38] is a good method to improve on a good but not perfect filter.

Genetic Algorithms (GAs) is one type of evolutionary algorithms, which are based on the mechanics of natural selection and genetics [39]. As an optimization method, GAs are flexible, generic and robust. GAs operate on a population of several individuals in parallel in each generation/iteration, where each individual, notated as the chromosome, represents one candidate solution to the problem. GAs search the solution space of a function through the use of simulation and evolution, during which the fittest one survives. According to the heuristic characteristic of GA, the most qualified individuals of the population are prior to survive and reproduce the next generation, thus improving the successive generations, and the most fitting characteristics are imposed in the next generation. On the other hand, inferior individuals can just survive and reproduce by chance [40, 41]. A simplified GA process is presented in following figure:

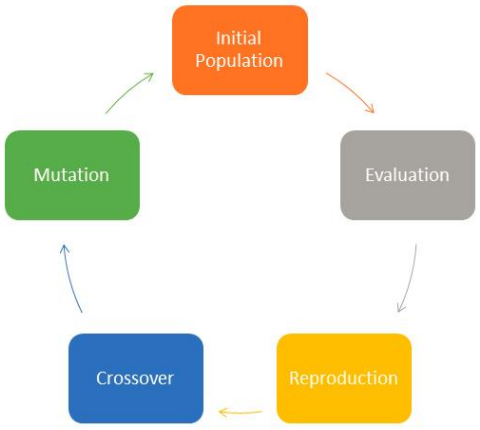


Figure 8. A simple Genetic Algorithm Cycle [39]

A complete GA entailing four fundamental steps is introduced as follows [40]:

Step 1: Initializing a random population.

Step 2: Evaluating the fitness of the chromosomes by precisely defined criteria, which is based on the design requirements for searching the best matching solutions.

Step 3: If the chromosome with best fitness value satisfies the requirements sufficiently, producing that chromosome as the expected solution and stop. Otherwise, continue to Step 4.

Step 4: Applying crossover and mutation among the chromosomes to generate more chromosomes, and then restart at Step 2.

Then the derived algorithm is summarized with specific notations in the following [40]:

1 Supply a population P0 of N individuals and corresponding fitness value for each individual in one generation.

2 Iteration from 1 to i

3 Choose individuals P_i' from selection function $(P_i - 1)$ for reproduction.

4 New generation P_i is got from reproduction function P_i'

5 Evaluate (P_i) by evaluation function

6 Jump to next generation $i=i+1$

7 Repeat Step (3) until termination

8 Print out best solution found

Six fundamental issues are used in a GA algorithm: **chromosome representation, selection function**, **genetic operators, initial population**, **termination criteria**, and the **fitness function**. These issues will be discussed separately later in this section [41].

Chromosome Representation

In a Genetic Algorithm, chromosome representation is used for describing the individuals who will be manipulated by other functions. According to [40], “The chromosome representation is crucial for a GA since the representation scheme determines how the problem is prototyped in GA and affects the determination of other genetic operators. Each chromosome consists of a sequence of genes from a particular alphabet. An alphabet could be made of binary digits, floating point numbers, integers, symbols, matrices, etc.”

Selection Function

The purpose of individual selection is to create new successive generations. Selection plays a crucial role in a GA. A probabilistic selection is performed based on the individual's fitness value so that the better individuals have an increased chance of being selected. An individual in the population can be chosen more than once with all individuals in the population having a chance of being selected. There are several strategies for the selection process: roulette wheel selection, scaling techniques, tournament, elitist models and ranking methods [38, 40].

Roulette wheel, to be as the first selection method, was developed by Holland [42]. The probability, P_l, for each individual is defined by:

$$\mathrm{P[\ Individual\ } l \mathrm{\ is\ chosen\ ]} = \frac{F_l}{\sum_{j=1}^{PopSize} F_j} \tag{21}$$

Where F_l equals the fitness of individual l.

Ranking methods only require the evaluation function to map the solutions to a partially ordered set, thus allowing for minimization and negativity. Ranking methods assign P_i based on the rank of solution i when all solutions are sorted. Normalized geometric ranking, [43], defines P_l for each individual by:

$$\mathrm{P[\ Individual\ } l \mathrm{\ is\ chosen\ ]} = q'(1-q)^{r-1} \tag{22}$$

Where:

- q = the probability of selecting the best individual,
- r = the rank of the individual, where 1 is the best.
- P = the population size
- $q' = \frac{q}{1-(1-q)^P}$

Genetic Operators

Genetic operators provide the underlying search mechanism of the GA. The operators are used to create new solutions based on existing solutions in the population. There are two fundamental types of operators: mutation and crossover. Crossover takes two individuals and produces two new individuals while mutation alters one individual to produce a single new solution. The application of these two fundamental types of operators and their derivatives depends on the chromosome representation used.

Initialization, Termination, Evaluation Function

An initial population must be created for the GA at the beginning. The most common method is to generate individuals for the entire population randomly. However, since GA can iteratively improve existing individuals, the beginning population can be stored as potentially useful solutions, with the remaining randomly generated individuals in the population.

According to [44], “The GA moves from generation to generation selecting and reproducing parents until a termination criterion is met. The most often used stopping criterion is a specified maximum number of generation. Another termination strategy involves population convergence criteria. In general, as will force much of the entire population to converge to a single solution.” Alternatively, a target value for the evaluation measure can be established based on some arbitrary threshold.

Fitness functions of many forms can be used in a GA, subject to the minimal requirements that the function can map the population into a partially ordered set.

3. DIGITAL INTEGRATOR DESIGN

In this thesis, an optimal digital integrator is designed step by step. This filter is expected approximately to fulfill all the specifications mentioned in chapter 1. It is a multi-criterion design problem as there are specific requirements for both magnitude and phase responses. To this end, several design methods are combined.

Firstly, the choice of starting point filter has a great influence to the further optimization. Its design strategy varies depending on what integration algorithms are employed. The starting point filter is carefully selected by referring to the design specification and constraints mentioned in Section 1.2. There are three proposed approaches that we could choose to create a starting point filter.

Secondly, optimized filter is obtained by applying optimization algorithms to the starting point filter, GA optimization method is used in this thesis.

After the optimal filter is derived, several filter structures are proposed to illustrate the suitability and stability of the desired filter with fixed point arithmetic. The frequency responses, as well as the corresponding filter structures, will be presented graphically. Group delay and time domain responses are compared with the specified criteria.

In this case, the design flow is explained in Figure 9 as following:

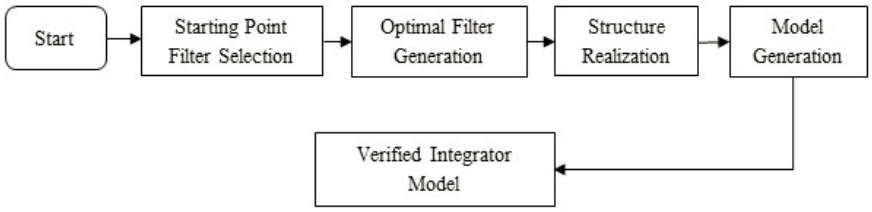


Figure 9. Simple Genetic Algorithm Cycle

3.1 Coefficients Calculation

Three different methods are proposed in this chapter. The methods introduced here can be used as suitable candidates for finding proper starting point digital integrators. Both the amplitude and the phase responses are taken into consideration during the design. The best starting point filter is carefully chosen by comparing the filter's performance, which is based on the specification mentioned in chapter 1. This design strategy is relatively fast and straightforward to find a close neighborhood of the desired filter. The whole design process used floating point arithmetic with aid of MATLAB.

3.1.1 Starting Point Filter Design

In this section, several integrators are designed. All the filters' coefficients use floating point arithmetic, the finite word length effects are not taken into consideration at this design stage.

Direct Designed Integrators

For direct design method, the most important thing is to find out proper input parameters. It is mainly a process of trial and try. In the beginning, very simple data are used, then according to the output coefficients and filter performance, corresponding modifications can be done.

According to the filter specification, the filter operates with frequency of 9.6 KHz. The interesting frequency range (w) is from 10 to 1250 Hz, which is specified in radiance between 0 to 2π. The magnitude (mag) at 50 Hz should be exactly 1. For expected filter performance, magnitude at 0 Hz should be 0. The phase response (phase) should be -90 degree as close as possible. Here "wt" is initial for weight. These parameters are set in the following table. Then the first experiment is carried out by applying the above parameters to the function ***invfreqz***. It yields a 3rd order filter, obtained coefficients are shown as well in the table:

Table 1. Example settings of using direct design method

Parameters	Value
w	[0,pi*25/4800,pi*50/4800,pi*100/4800,pi*200/4800, pi*400/4800,pi*800/4800,pi*1250/4800,pi*4800/4800]
mag	[0,2,1,0.5,0.25,0.1,0.06,0.03,0.00001]
phase	[0/180*pi,-89/180*pi,-89.75/180*pi,-89.90/180*pi, -89.94/180*pi,-89.97/180*pi,-89.98/180*pi,-89.99/180*pi, -89.999/180*pi]
wt	[1,1,10,10,10,10,10,10]
Function: [b,a] = invfreqz(mag.*exp(1i*phase),w,3,3,wt,1000,0.01,'trace');	
Result	
b	[-0.04035058,0.058111759,0.041267960,-0.05772230]
a	[1,-0.97516931,-0.99410984, 0.970325597]

Figure 10 shows the frequency response of this 3rd order filter:

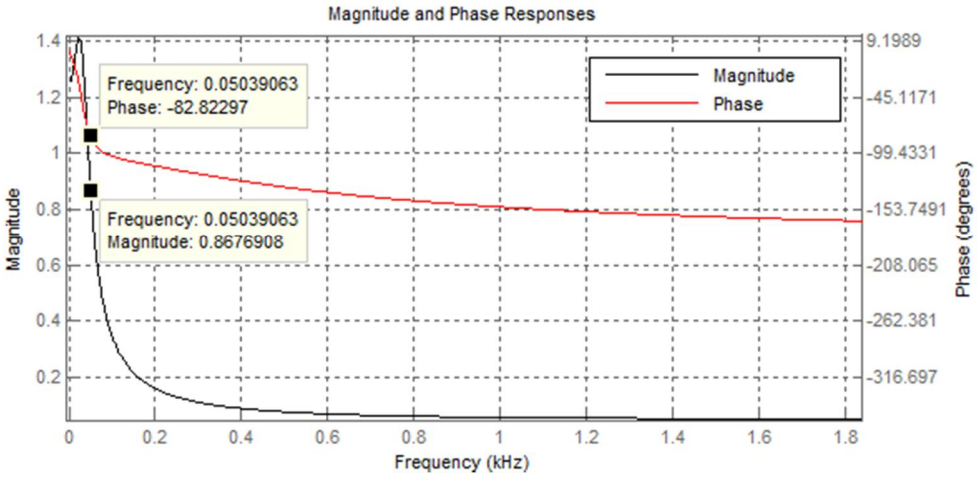


Figure 10. Frequency response of the resulted filter

Comment: Form the figure, the magnitude barely match the requirements, and the phase response of this filter is not good. As a result, this filter could not be taken as a starting point filter. If this method is further used, it undoubtfully will take a lot of time for tuning the parameters. However, the performance of the filter cannot be guaranteed. Thus, it is better to use other methods.

Newton-cotes Based Digital Integrators

Three different digital integrators based on Newton-cotes rules are simulated in this section: ***trapezoidal digital integrator, Simpson digital integrator*** and ***rectangular digital integrator***. Here the simulation configuration and the results of the simulation are given. The example codes of simulators can be found in Appendix. It is evident that the proposed integrators have different orders, and correspondingly have different frequency responses.

Trapezoidal Digital Integrator: Based on the transfer function which is mentioned in section 2.3, the coefficients is directly obtained (i.e., ***b*** is [1 1] and ***a*** is [2 -2]). The coefficients are applied into the MATLAB function ***tf2zp***. Then to stabilize the system as optimization, specifically, all of the poles are kept inside the unit circle by multiplying the previous pole's value with 0.9995. Even though the multiplier of 0.9995 is near to 1, the performance of the filter will change to some extent because IIR filters are very sensitive to the change of poles.

Simpson Digital Integrator: Here Simpson 1/3 rule is chosen. The same configuration is done as above, the example simulator can be found in Appendix.

Rectangular Digital Integrator: By analyzing the transfer function, we get the coefficients as following and by the same means to apply the coefficients to MATLAB functions.

The magnitude and phase responses of the above filters are depicted separately in following figures.

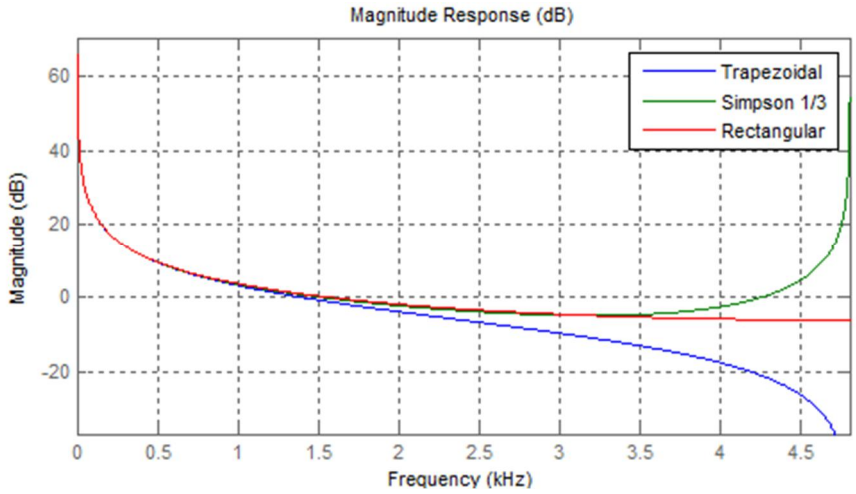


Figure 11. Magnitude responses of three filters

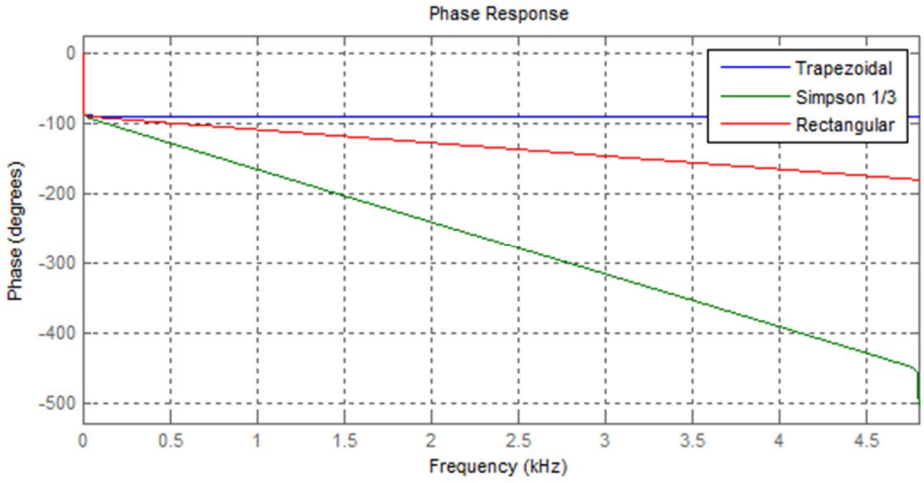


Figure 12. Phase responses of three filters

Comment: According to the figures, the trapezoidal integrator has better performance in terms of the phase response. When it comes to magnitude response, rectangular integrator has more similarities with the ideal filter, and the trapezoidal integrator has better performance than the Simpson filter. Since the phase response is more complicated to be tuned in the optimization process, it is more reasonable to choose trapezoidal filter for further optimization.

Next, in order to get a better filter, either optimization methods needs to be applied, or the filter's order needs to be increased if the trapezoidal integrator was chosen as the starting point filter. Based on this conclusion, we increased the order of trapezoidal integrator by adding a notch near 0 Hz. The example simulator is given in Appendix. and the corresponding frequency response showed in Figure 11 proved our choice was

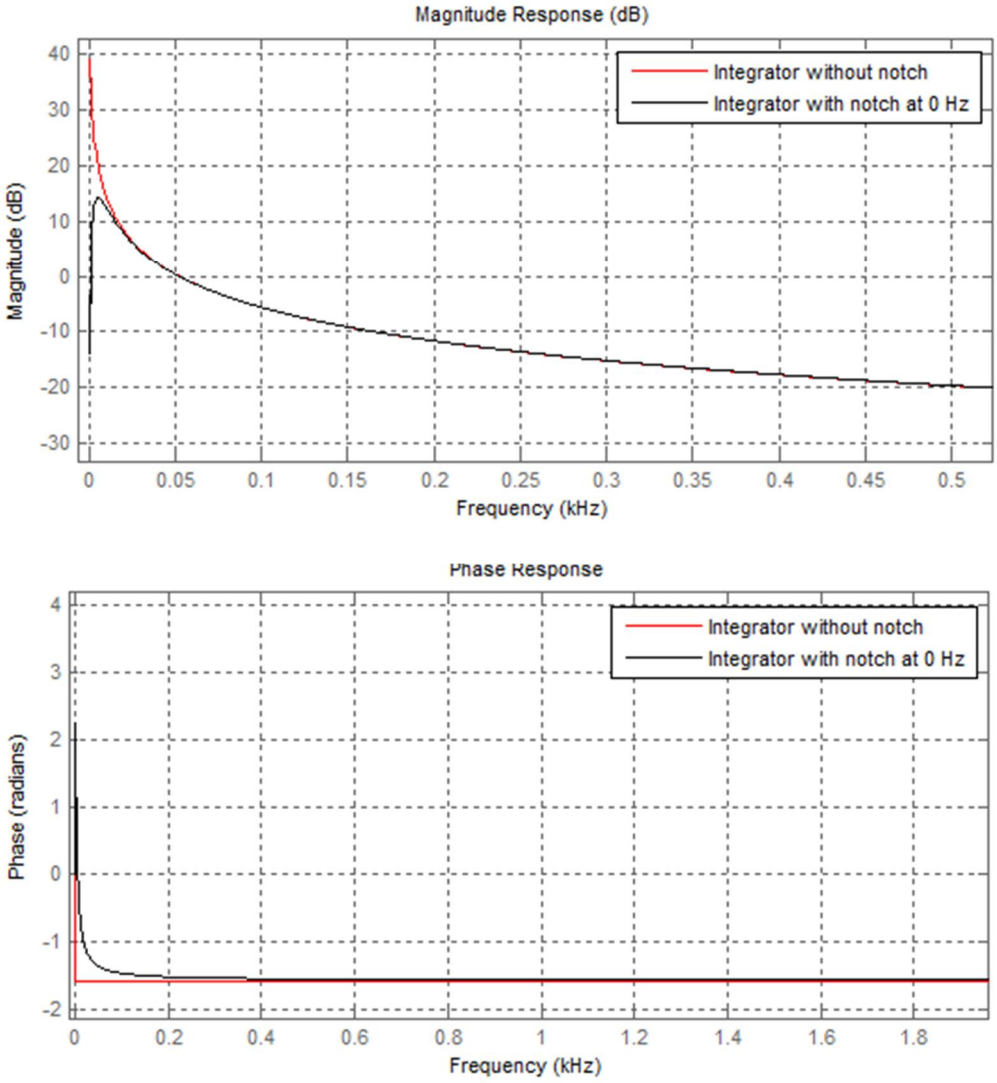


Figure 13. Frequency response of trapezoidal integrator after adding a notch

right.

Comment: After adding a notch, here this modified filter is called as ***candidate 1*** filter. This filter has not only good phase response but also excellent magnitude response. As a result, this filter could be a superb choice for the starting point integrator.

Direct Design Based On the Frequency Response

Another fundamental digital integrator was derived by directly identifying the transfer function, which corresponds to the given frequency response. In this case, the expected magnitude and phase response ***h*** and ***w*** are defined at first in specific frequency points.

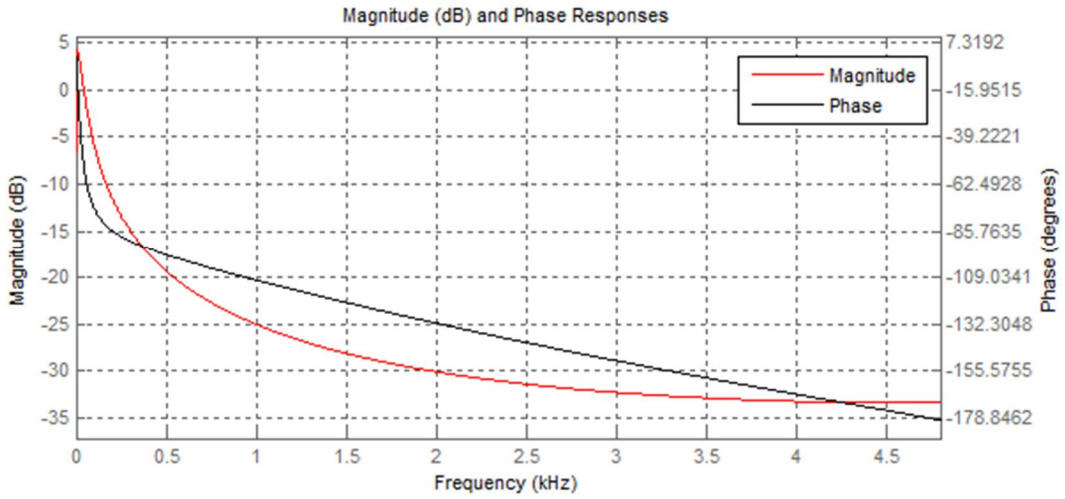


Figure 14. Frequency response of directly designed filter

Then the MATLAB function ***invfreqz*** is used to convert data into transfer function coefficients. A simple case is introduced in Appendix to explain how it works:

Comment: By analyzing the above figure, it is evident that both magnitude and phase response of the integrator is far away from the expectation of the desired filter. As a conclusion, even though further optimization methods could be used on the filter, it would cost more time to choose this method to design the starting point filter than to select the ***candidate 1 filter***.

A comparison between the ***directly designed filter*** and ***candidate 1 filter*** is given in Figure 15. It is evident that ***candidate 1 filter*** has better performance than the ***directly designed filter***.

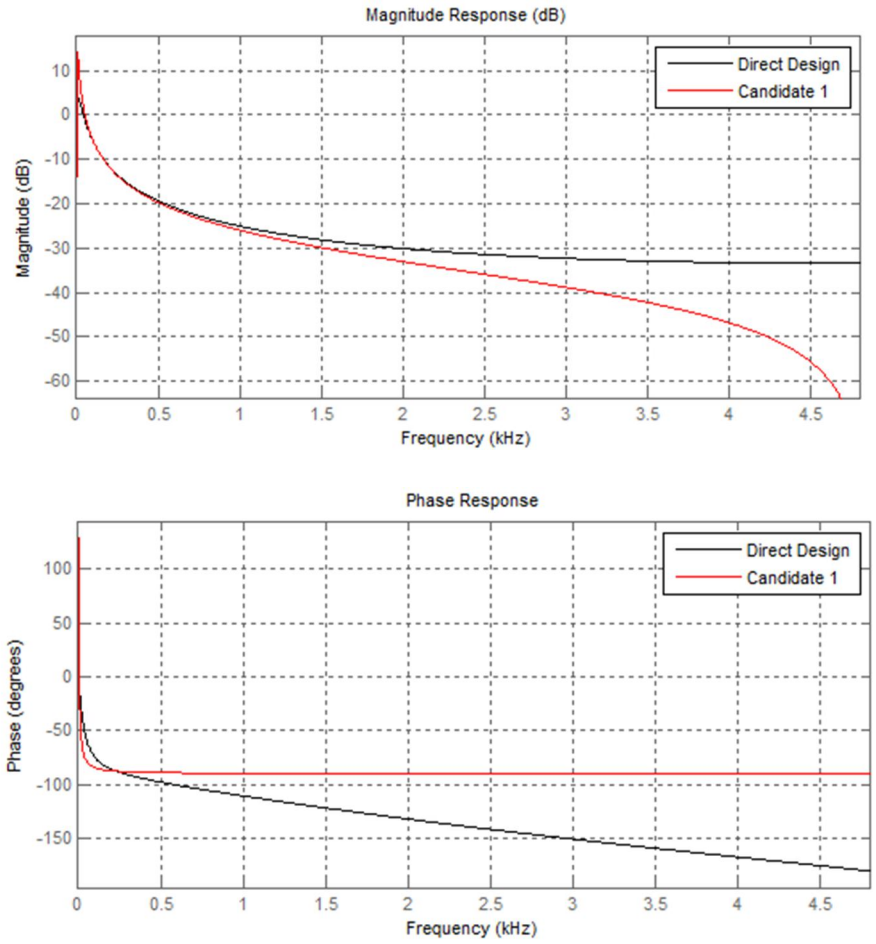


Figure 15. Magnitude and Phase responses of directly designed filter and candidate 1 filter

Pole-Zero Placement

In order to obtain the desired frequency response, poles and zeros can be placed strategically on the ideal integrator to generate new integrators. This method can as well be used to increase the filter's order because the higher the order, the better the performance of the filter.

Design Instance 1

An example is shown to explain how to guarantee the designed filter be stable and how to adjust the magnitude response by means of tuning gain parameter.

Here the original trapezoidal integrator is employed to illustrate how the pole-zero placement method works. Figure 14 shows the position of pole and zero of the trapezoidal integrator on z-plane.

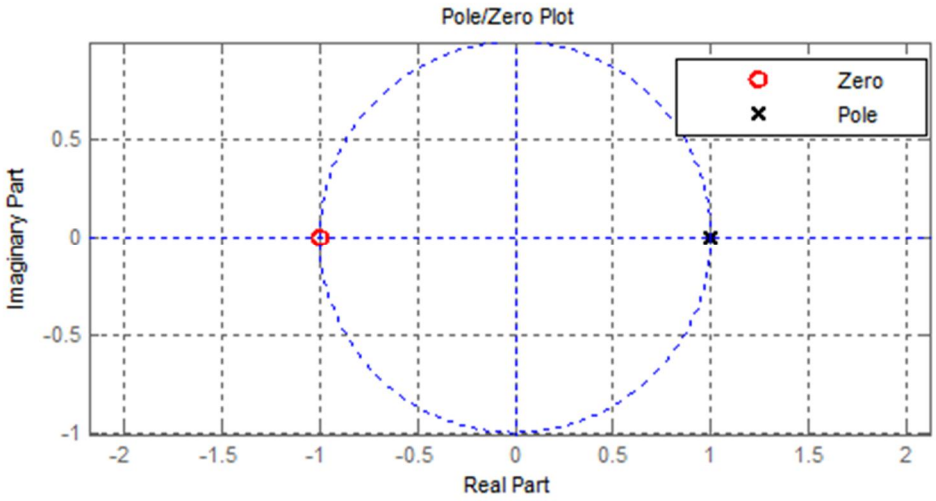


Figure 16. Pole- zero position of the original trapezoidal integrator

Comment: As it is known, the filter is stable when all of the poles are inside the unit circle. However, from the above figure, it 's hard to tell whether the poles are inside the unit circle or not. In order to guarantee the stability of the filter, the pole's position needs to be adjusted. The same method mentioned in the section for trapezoidal integrator design (**program 1**) in Appendix is utilized to make sure the position of poles.

Now the filter is stable. By adjusting the obtained gain value ***k***, the magnitude response of the filter at 50 Hz is tuned to be 1 (i.e., 0 dB). Figure 17 is the magnitude and phase response of the modified first-order trapezoidal integrator obtained above. As the figure shows, it is clear that the magnitude response at 50 Hz is very close to 0 dB.

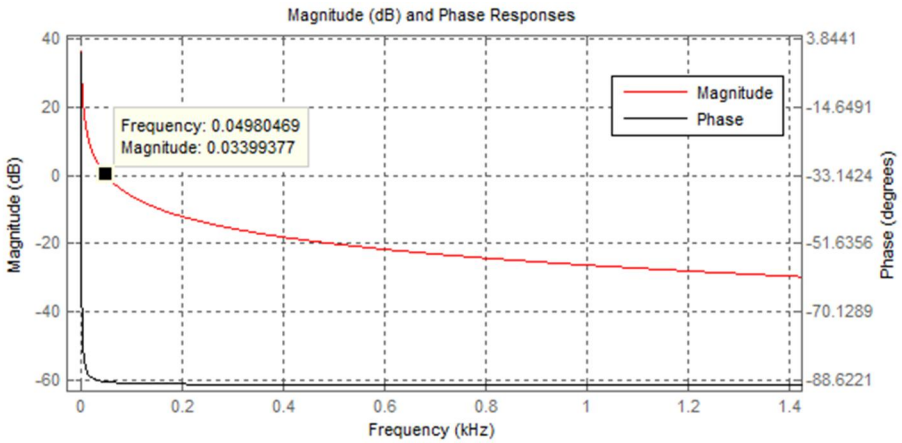


Figure 17. Frequency response of the modified first-order trapezoidal integrator

Design Instance 2

This example is for explaining how to utilize pole-zero placement to add a notch for a filter. According to Figure 17, the phase response of the first order trapezoidal integrator performs well. However the magnitude response at 0 Hz is huge, which means that if there is a DC input, the output value of the filter will be saturated and would never decay. This result is not expected. Additionally, based on the specification, it is needed to minimize the magnitude response at 0 Hz. Therefore, it is necessary to add a notch around 0 Hz, where the pole-zero placement method is utilized again.

A conjunct pair of zero and pole is added around 0 Hz, which will yield a 3^{rd}-order filter. What needs to be mentioned here is that the width of the notch affects the stability, impulse response and other performance of the filter. A concrete example is given in Appendix.

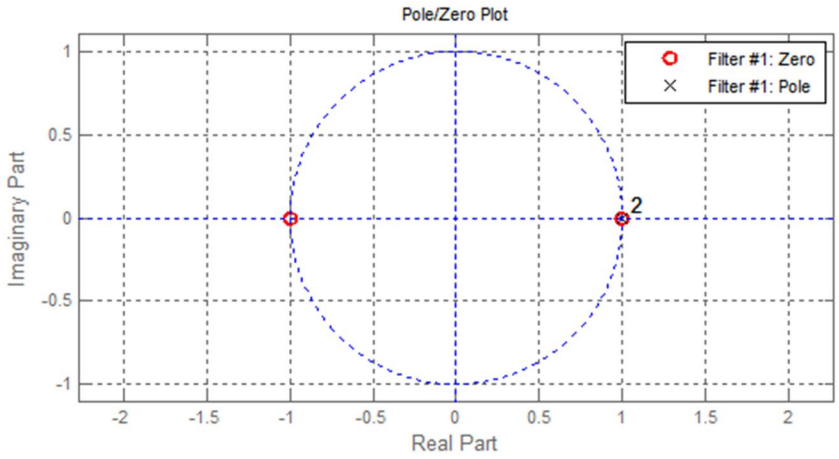


Figure 18. Pole- zero position of the 3rd-order integrator

The new filter's zero-pole-gain value are obtained after these operations. As is shown in Figure 18, all of the poles and zeros are quite near to the boundary of the unit circle. In other words, even though tiny changes of the coefficients of the filter, the poles will have great chances to be outside the unit circle, and the filter will be unstable correspondingly. As a result, in order to get a better filter, the coefficients or the positions of poles and zeros need to be adjusted carefully.

Figure 19 gives a comparison between this 3rd-order integrator and the previous *candidate 1 filter*.

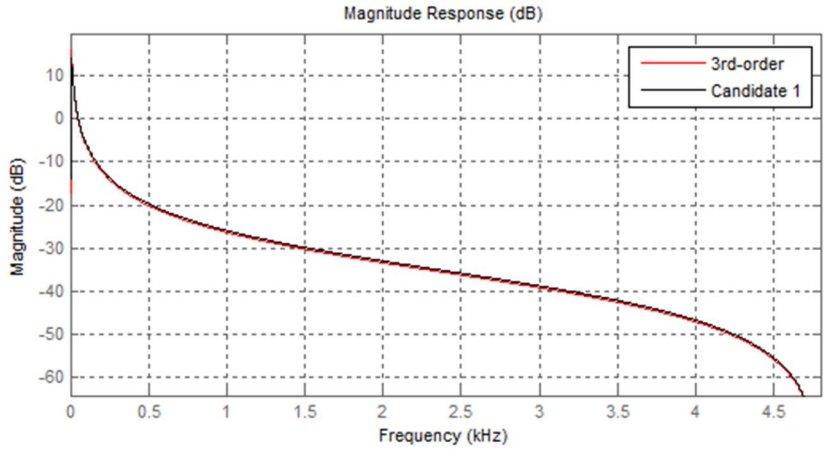


Figure 19. Magnitude response of 3rd-order integrator and the Candidate 1 filter

Summary: this 3rd order filter is very similar to *candidate 1 filter*. In brief, this filter is developed from the basic trapezoidal filter, by means of adjusting the gain value and

adding a notch (or increase the order of the filter). That is, we used the basic design method as well as pole-zero placement to develop a 3rd order starting point filter. As a result, it is not very easy to develop a well-performed filter by purely using only one design method. We should take all of the specifications into account and take advantage of all of the methods we have to make the filter as better as possible.

3.1.2 Coefficients Optimization

Above section mentioned that the higher of the filter's order, the greater possibility of getting a better performance. In this section, a genetic algorithm is proposed to solve the optimization problems in search of better 3rd and 4th order filters. Both the 3rd and 4th order filters are derived by using the pole-zero placement method above. The 4th order filter is obtained by adding a new notch around 0 Hz on the GA optimized 3rd order *candidate 1 filter*, which has very good frequency response and impulse response if the fitness function is well defined.

The GA optimization was implemented with the aid of MATLAB. In this case, 5 user-defined MATLAB functions are created, which are: **GA_optimization, Selection, Fit_Evaluation, Mutation,** and **Crossover**. For confidential consideration, this thesis would not show the whole subroutines. But in the rest of the section, the main aspects associated with the algorithm are discussed with details.

GA_optimization Function

This is the primary function of GA optimization method in this thesis, which runs the simulated evolution. The primary call to this function is given by the following MATLAB command:

```
[BestValue,best,bestfit]=GA_optimization(pSize,pGen,N,Fs,pCross,pMutation,
mutStr)
```

Output parameters

- *BestValue* is the best solution string, i.e. final solution,
- *Best* is group of each BestValue from every generation during optimization,
- *Bestfit* is group of fitness value from every generation.

Input parameters

- *pSize* is the size of the population
- *pGen* is number of iterations
- *N* is the order of the filter
- *Fs* is the sampling frequency
- *pCross* is crossover possibility

- *pMutation* is a mutation possibility
- *mutStr* is strength of mutation

Population Initialization

The purpose of *population initialization* is to provide the **GA_optimization** a starting point filter. It is executed by using the method of pole-zero placement. The particular operation to place the poles and zeros is based on the observation and analysis of pole-zero distribution (Figure 18), which is from a basic 3rd order filter.

Based on the pole-zero distribution graph, there are 3 steps to be done to determine the strategy of corresponding pole-zero placement.

Step 1: Find the general distributions of all zeros. From the graph we can see, there is one zero on the negative real axis, and other two on the positive real axis. The absolute values are all smaller than one but very near to the unit boundary. MATLAB function ***rand*** can be used to generate the initial values of zeros.

Step 2: Find the general distributions of all poles. There is one pole right on the positive real axis, which is quite near to the unit circle but smaller than 1. There are two poles with very small angles, which is quite near to the real axis.

Step 3: Determine the variation interval for each variable (or the search space), including all zeros and poles as well as the scale value.

An example of how to generate new poles and zeros can be found in Appendix.

Termination Condition

The termination algorithm determines when to stop the simulated evolution and return the result population. **GA_optimization** checks the termination condition every generation. The function will terminate either at specified generation or the optimal or max generation when best individual case is found.

Selection Function

The *selection function* determines which of the individuals will survive and continue to the next generation. The **GA_optimization** function calls the selection function each generation after new population is created from the old one. The basic function call used is:

```
[selectpop] = Selection(pop)
```

Where *selectpop* is the new population selected, input *pop* is the current population. In our selection algorithm, both *Ranking method* and *Roulette wheel* are used. *The former*

one is to decide the choosing probability, the latter one is to choose the individual from the current population based on the choosing probability.

Fitness Function

The function is called from GA main fucntion to determine the fitness of each solution string generated during the search process. It is important to decide which performance of the filter should be taken into account and how to evaluate it, that is, how to translate the performance into quantifiable data. For example, the filter's magnitude responses at interesting frequency points are necessary to be represented and evaluated. In addition, phase response, group delay, impulse response and stability of the filter have to be taken into consideration.

After the evaluated performance is decided, the priority of these factors needs to be determined. Furthermore, the metric standard, which determines what kind of data is useful and what kind of data is bad has to be made. There should be significant differences in the fitness values between the well performed and the badly performed filters.

Fitness function is called by **GA_optimization** twice during every generation. One is at the beginning, and the other is at the end after the new generation was derived. Here is the basic call of this function:

```
[fitness] = Fit_Evaluation(pop)
```

Where *fitness* is the fitness value for every individual of the current generation *pop*, which is the input to the function.

Mutation and Crossover Functions

Mutation is one of the operator functions of the GA, the other one is *Crossover*, both of which provide the search mechanism for GA and create new solutions based on existing solutions in the population. Mutation changes one individual's value to produce a new solution, which could be tuned by the mutation strength. Crossover takes two individuals' value and provides two new individuals after corresponding operations.

The function call for *Mutation* is as follows:

```
[NewPop]=Mutation(OldPop,pMutation,mutStr)
```

Where `OldPop` is the current generation, `pMutation` is the mutation possibility for the generation and `mutStr` is the mutation strength for one picked individual. The `newPop` is the new generation with mutated individuals.

The *Crossover* function is similar:

```
[NewPop]=CrossOver(OldPop,pCross,opts)
```

Where `OldPop` is the current generation, `pCross` is the crossover possibility for the generation and `opts` is the option for different crossover strategy. In this thesis, we use multi-point and equally crossover algorithms. The `newPop` is the new generation of mutated individuals.

3.1.3 Design Example

In this section, a detailed explanation of the designing and optimization process is given. It is based on the real experiments by optimizing the 3rd and 4th ordered integrators.

Starting Point Filter Generating

At first, a starting point filter has to be designed for GA optimization. This filter is taken as the best individual among all the population at the first generation. Then, when the optimization process starts, the starting point filter will be replaced by the new one which has better fitness value. Here, *candidate 1 filter* is chosen for GA optimization. The primary 3rd order filter's *zero-pole-gain* value is listed at Table 2 as following:

Table 2. Example of setting the individual and population for GA optimization

Parameters	Value
zeros	[-1 1 1]
poles	[0.999500000000000 0.998363753826255 0.998363753826255]
gain	0.016361741209689
Function: Individual=[zeros poles gain]; Population=[Individual 1; Individual 2; … ; Individual d];	

The coefficients of one individual or filter are expressed in a row vector in function (1). The whole population in one generation are stored in a matrix, which is shown in function (2), here *d* is the size of one population.

Before optimization, the chosen starting point filter is stored as one individual at the end of the first-generation matrix. The other individuals are generated by using *Pole-Zero Placement* method.

In the following parts of this section, a 3rd and a 4th order integrators are generated. The whole optimization processes of the two filters are almost the same. They use the same operators and same basic function parameters. The only difference is the individual generating method because they have different filter orders.

In order to create a new 3rd order filter, the first step is to observe the pole and zero distributions. According to Figure 18, all of the three poles are very close to the positive axis of the unit circle. Two zeros are near or on the positive real axis, and the rest one is on or close to the negative real axis of the unit circle. From above analysis, a new random 3rd order integrator is generated by *Pole-Zero Placement* method, and the code is shown in **Program 8** in Appendix.

That's how new individuals are created in GA optimization. After this optimization, better 3rd order integrator can be obtained with well-defined fitness function. If the optimization results are not satisfactory, more times of iterations are need for the optimization until a best filter is found.

Based on the optimized 3rd order integrator, a 4th order integrator is generated by adding a notch exactly at 0 Hz. An example is shown in Program 9 in Appendix:

Since a notch is directly added at 0 Hz, the notch angle is 0, there is no need to think about the conjunct pair of the new zero and pole. That's why the filter's order is only increased from 3 to 4 but not 5. All of the poles are at the positive part of the unit circle, and they are very close to the unit border. For the zeros, there are 3 of them at the positive part on the unit circle, the other one is on the negative side. According to this conclusion, new filters for 4th order filter optimization can be created, the code is given in Program 10 in Appendix.

When the starting point filter and the population for optimization are ready, the GA optimization Process can be started.

3rd order filter optimization

The starting point filter for optimization is ready from the last section. Then, the input parameters for the optimization function need to be decided.

```
[BestValue, best, bestfit] = GA_optimization (pSize, pGen, N, Fs,pCross, pMutation, mutStr, options)
```

Except for pSize and pGen, the rest of parameters are set as following:

```
N         = 3;  % the order of the filter
pCross    = 0.85; %crossover
pMutation = 0.10; % mutation
mutStr    = 1;% strength of mutation
Fs        = 9600;
options   = 0;
```

The most relevant parameters are pSize and pGen since they have high impacts on the optimized results. There is a trade-off between the computation efficiency and the total

optimization time. If pSize and pGen are huge, even though more optimization results could be found, the time that the whole optimization process will take will be longer correspondingly. That's why different parameter pairs of pSize and pGen are used for testing in this case.

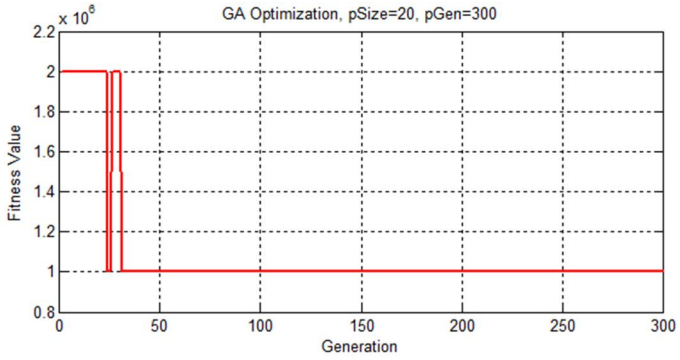
Table X shows the optimized individuals obtained during the optimization process with different input values of pSize and pGen. It is obvious that the bigger pSize and pGen are, the longer the whole optimization process is. It is necessary to select a proper pair of pSize and pGen for the further optimization. That's also the point of this experiment.

Table 3. Optimization results for different input pairs of pSize and pGen

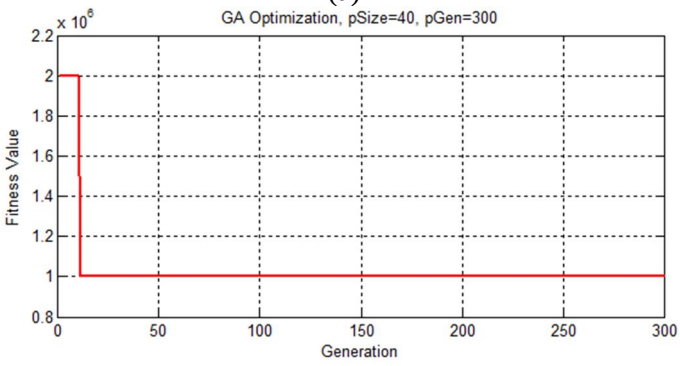
pSize	***pGen***	***Optimized Individuals***
20	100	7
30	100	6
40	100	4
20	300	14
30	300	13
40	300	19
20	500	13
30	500	12
40	500	11

Comment: Finally, the number of pGen is set to be 300. When pGen is too small (pGen=100), there are not enough optimized results. When pGen is big (pGen=500), the optimization does not converge any longer. When pGen=300, reasonable number of optimized individuals are obtained, and it takes shorter optimization time than pGen=500.

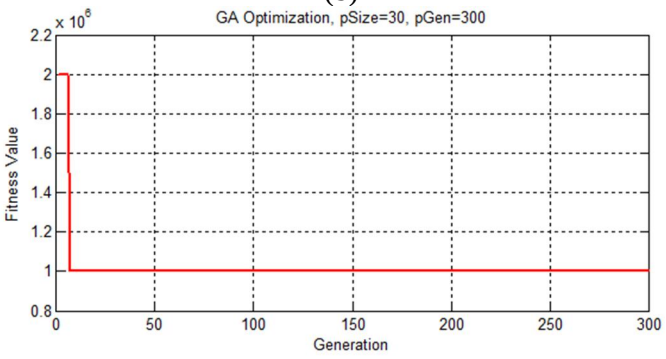
The optimization process is plotted to show the variations in fitness values when the generations are evolving. Figure 20 depicts the differences in the fitness values tuning process with different pSize numbers.



(a)



(b)



(c)

Figure 20. Variation in fitness values through generations with pSize numbers of a) 20, b) 30 and c) 40

Comment: According to above figures, the optimization results were not stable when pSize was small. When pSize changed to be bigger, the optimization results tended to be more reasonable. In account to optimization efficiency, it is a better choice to set pSize as 30.

Performance Test

Even though the optimization method has been chosen, it is not very easy to select a suitable filter by simply adopting the GA optimization functions. One reason is that the input parameters of the GA functions should be appropriately chosen, for the purpose of getting good optimization result without taking too much computation time. On the other hand, not all of the optimization results are valid since the fitness function could not take all of the performance factors into account for evaluation. And sometimes the optimized results deviate from the primary filter coefficients too much that the resulted new filter is not an integrator any more. That's why additional functional tests are still needed after optimization.

The following is an explanation about how to validate the optimization results.

According to the GA functions, all of the results are stored in a matrix, in which every row is a set of coefficients consist of zeros, poles and the gain factor of a filter. The latest optimized result is stored in the first row of the matrix. For most cases, the first row of the matrix could be taken as the best-optimized result among all of the results. However, sometimes it doesn't work since the fitness function could not evaluate the filter adequately. Then the following tests are needed for further selection of the best filter coefficients.

The optimized results are obtained from the above experiments when pSize=30 and pGen=300, there are 13 optimized results obtained from this optimization case. The first row, the last row and the middle row (7^{th} row) are selected for the comparison of their performance. These three filters are separately named as *Filter1, Filter2, and Filter3*. In order to describe the filter in a clear way, here the coefficients are only present in the first four digits after the decimal point, and all the coefficients are in floating point representation. Table 4 shows the results:

Table 4. Filters' pole-zero-grain value from the GA optimization

Filter 1
Filter1=[-0.6557 0.9929 1.6168 0.1308+0.0001i 0.4153-0.0003i 0.6672 0.0107]
Z1=[-0.6557 0.9929 1.6168]
P1=[0.1308+0.0001i 0.4153-0.0003i 0.6672]
K1=0.0107
Filter 2
Filter2=[-0.9999 0.9934 1.0454 0.9978+0.0004i 0.9970-0.0002i 0.9921 0.0163]
Z2=[-0.9999 0.9934 1.0454]
P2=[0.9978+0.0004i 0.9970-0.0002i 0.9921]

```
K2=0.0163
```

```
                                Filter 3
```

```
Filter3=[-0.9999   0.9937   0.9993   0.9993+0.0006i 0.9977-0.0005i 0.9959
0.0162]

Z3=[-0.9999         0.9937          0.9993]

P3=[0.9993+0.0006i  0.9977-0.0005i  0.9959]

K3=0.0162
```

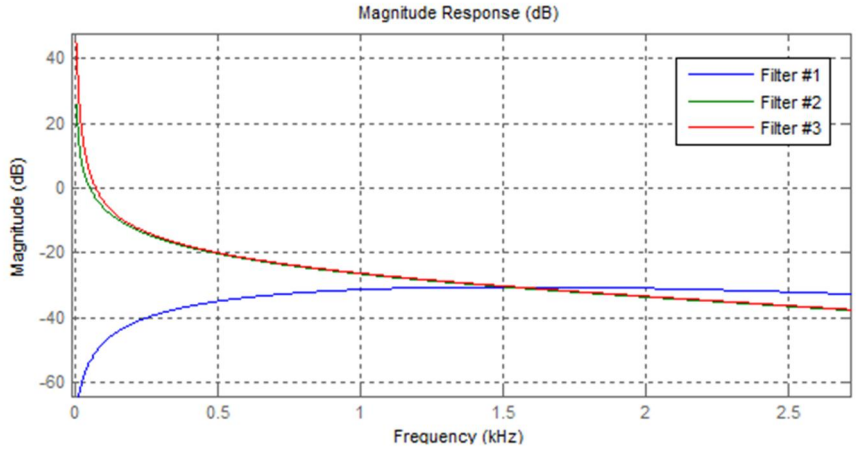
For the performance test, firstly the stability of every filter need to be tested. After that, it is necessary to examine the filter's responses in frequency and time domains. Then the zero-pole-gain values are transformed into filter objects with specific filter structures. Here *Direct form II transposed* filter structure is used for all of the filters.

The following codes test the filter's stability:

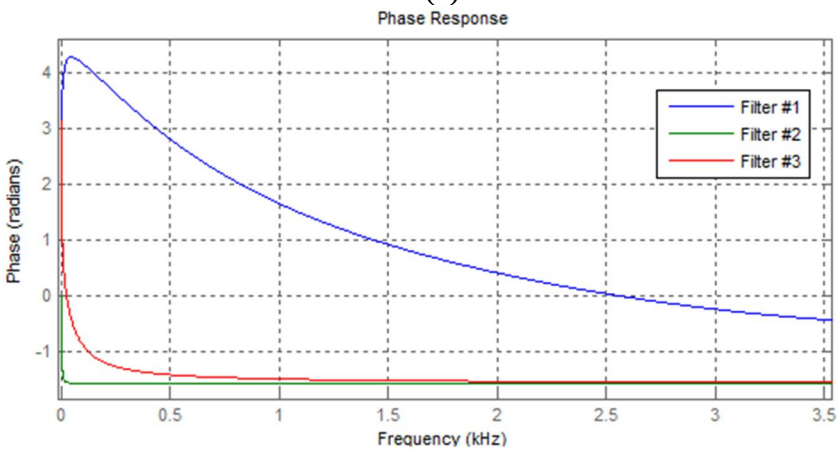
```
1     [b,a]=zp2tf(z',p',k);
2     HD=dfilt.df2t(b,a);
3     flag=isstable(hd);
```

If the filter is stable, flag equals to 1. Otherwise flag equals to 0. In this case, these three filters are stable. If a filter is found unstable, it should be abandoned immediately.

In the next step, the filters' frequency response is tested. The frequency response of *Filter1, Filter2, and Filter3* are shown as following:



(a)



(b)

Figure 21. Frequency responses of Filter1, Filter2 and Filter3, a) Magnitude Responses, b) Phase Responses

Comment: From Figure 21, Filter1 should be abandoned since both magnitude and phase responses of it are far worse than the other two filters. It also proves that a filter with best fitness value is not the best choice.

Then Filter 2 and Filter 3 are compared again. Form above figures, it is obvious that Filter 3 has better performance than Filter 2. But in some cases, it is very difficult to distinguish the two filters only by the frequency responses if the filters have very similar coefficients, or even some coefficients of the filters are the same. Then it is necessary to compare filters' performance in particular frequency points. The following codes are shown as an example in Table 5:

Table 5. Example codes showing how to test filter's performance

Test magnitude response at 0 Hz and 50 Hz:	
1	[h,w]=freqz(hd,9600,'whole');
2	resps=[abs(h)];
3	angles=angle(h)*180/pi;
4	w=w*4800/pi;
5	mag0=resps(1)% magnitude at 0 Hz
6	mag50=resps(51)% magnitude at 50 Hz
Test phase response at 50 Hz and 1000 Hz:	
1	ang50=angles(51)% phase at 50 Hz
2	ang1000=angles(1001)% phase at 1000 Hz
Test group delay from 10 Hz to 1300 Hz with interval 10 Hz:	
1	[grpd]=grpdelay(hd,[10:10:1300],9600);

All of the above tests are about filter performance in the frequency domain. The filter's performance in the time domain can be compared as well by checking out its impulse and step responses. An example is shown in Table 6:

Table 6. Example codes showing how to test the filter in the time domain

Test its impulse and step responses	
1	Ts=1/9600;
2	sys=tf(b,a,Ts);
	% impulse response
3	[impy,impt]=impulse(sys);
	% step response
4	[stpy,stpt]=step(sys);

The results are visualized by plotting the response variances with time changing. Figure 22 shows the impulse and step responses of Filter2 and Filter3.

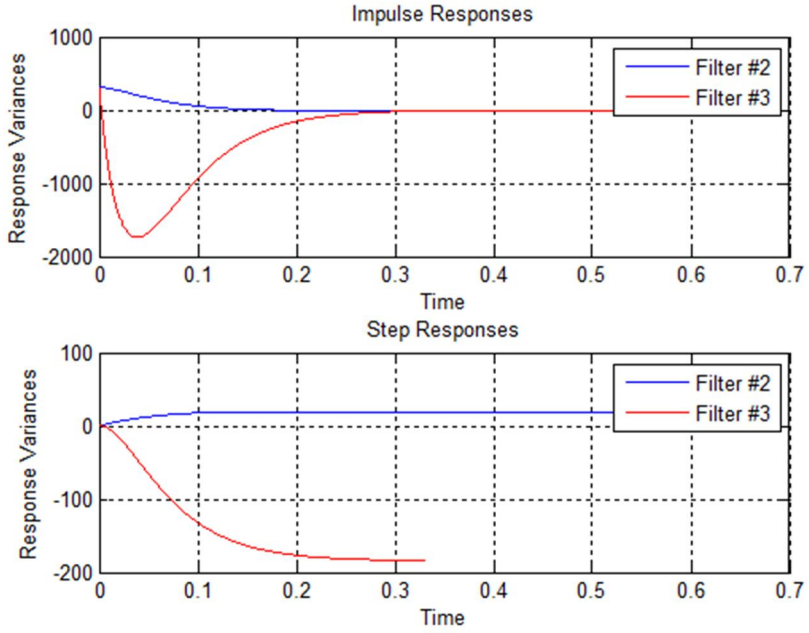


Figure 22. Impulse and step responses of Filter2 and Filter3

Comment: According to the above figure, Filter3 has better performance than Filter2 in both Impulse response and Step response. From the previous test, Filter3 also has excellent performance in the frequency domain. Thus, Filter3 is an excellent proof of the effectiveness of the GA optimization functions. In order to get even better filter, Filter3 can be further optimized by using the GA optimization again, or by increasing the filter's order and applying it to GA optimization.

In the following section, a 4th-order filter's optimization is discussed.

4th order filter optimization

From last section, a very good 3rd order filter is obtained. Next a 4th order filter is generated by adding a notch at 0 Hz. The newly generated filter has the following coefficients:

```
z = [-0.9999 0.9937 0.9992 1.0000]
p = [0.9992+0.0006i 0.9977-.0005i 0.9958 0.9940]
k = 0.01623
```

This filter is taken to be starting point filter for the 4th order filter optimization. Then, the next optimization process is exactly the same as the 3rd order filter. According to previous experience, pSize and pGen are separately set to be 30 and 300. Performance

tests are necessary after the optimization. After picking out the best result, it can be chosen to be the starting point filter for the next optimization round. After 3-5 times' iterations, a 4th order integrator with very good performance is obtained.

3.2 Structure Realization

After the optimization, the next step is to realize the filter with a suitable filter structure. Several available structures have been introduced in Chapter 2. This section mainly discusses important issues about how to select a right filter structure.

3.2.1 Available Structures

When fixed point arithmetic is employed, carefully choosing a suitable structure is critical. Based on the structures mentioned in Chapter 2, basic filter structures can be taken into consideration at first. The following four figures show Direct Form structures of this 4th order IIR filter. All of the structures are described in block diagrams and derived by the Simulink toolbox of MATLAB.

Direct Form I Structure:

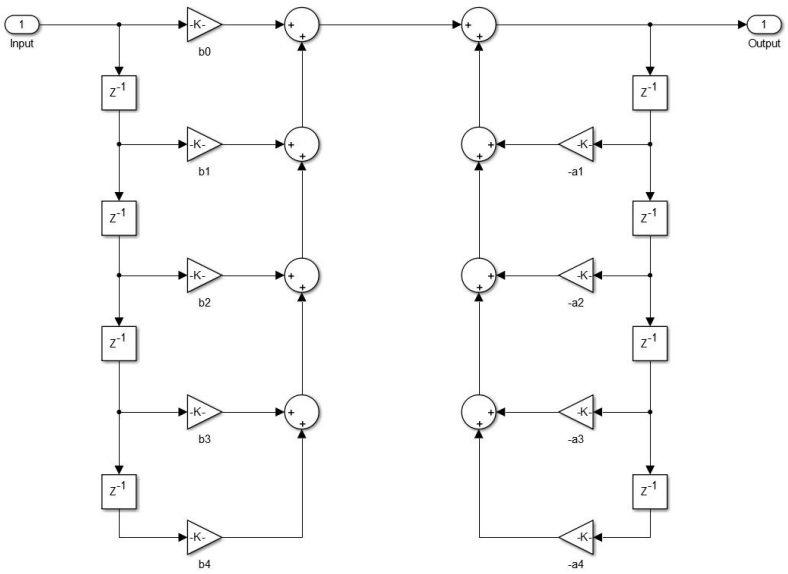


Figure 23. Direct Form I of the 4th order IIR filter

Direct Form II Structure:

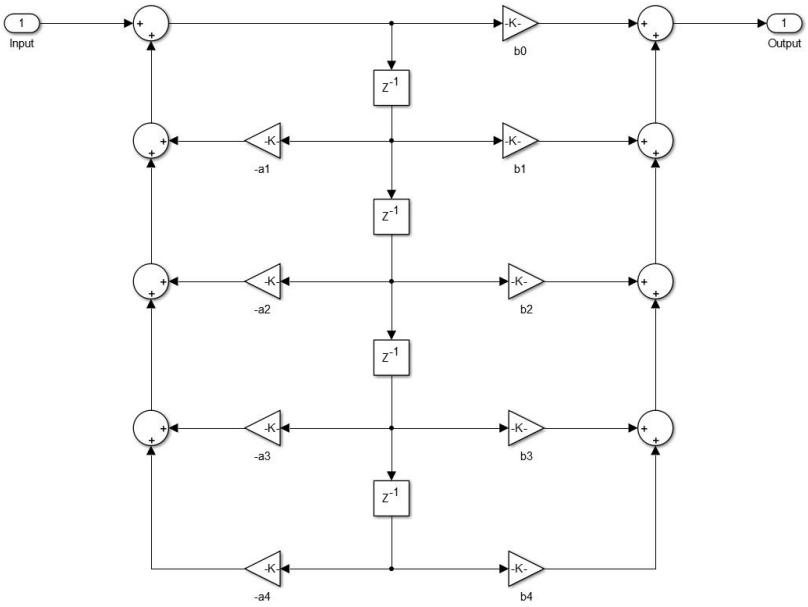


Figure 24. Direct Form II of the 4th order IIR filter

Transposed Direct Form I Structure:

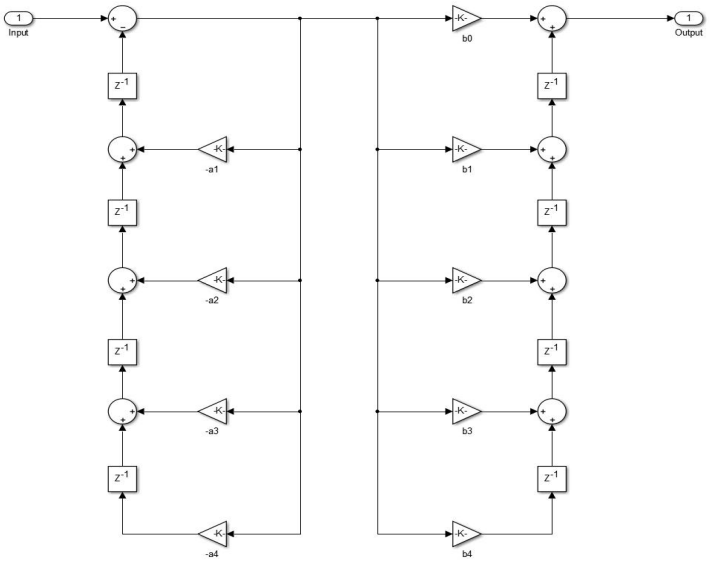


Figure 25. Transposed Direct Form I of the 4th order IIR filter

Transposed Direct Form II Structure:

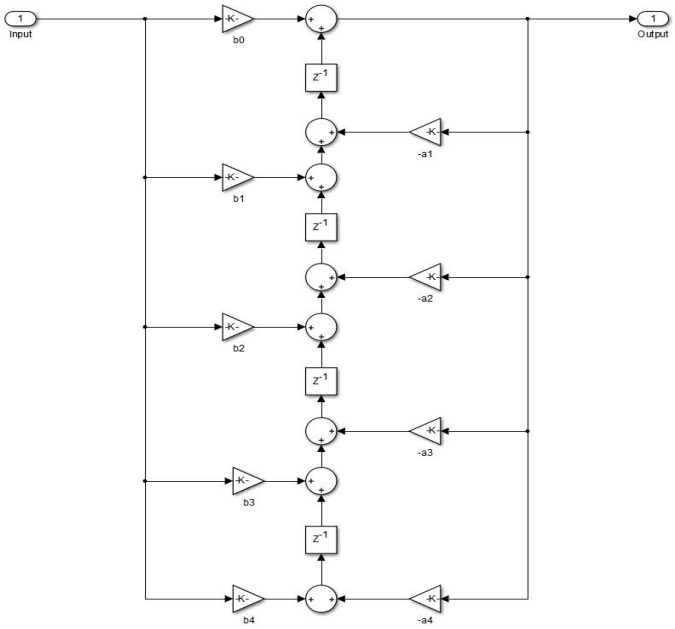


Figure 26. Transposed Direct Form II of 4th order IIR filter

By analyzing these four structures, the implementation costs that each structure takes are list in Table 7 shows the details:

Table 7. Implementation Costs for Different Filter Structures

Filter Structure	**Number of Multipliers**	**Number of Adders**	**Number of State**
Direct I	9	8	8
Direct II	9	8	4
Transposed Direct I	9	8	8
Transposed Direct II	9	8	4

Comment: Direct II structures have obvious advantages on saving the recourses. As this filter will be implemented on FPGA, Direct Form II structure is chosen for the further analysis. As is the case with all direct-form filter structures, the filter' poles and

zeros can be very sensitive to round-off errors, especially for higher order direct form filters. But for a simple second-order section, this problem could be settled. Thus, to minimize this sensitivity, it is common to factor filter transfer functions into series and/or parallel second-order sections. In the following, two cascaded structures with direct form II format.

Cascaded Direct Form II Structure:

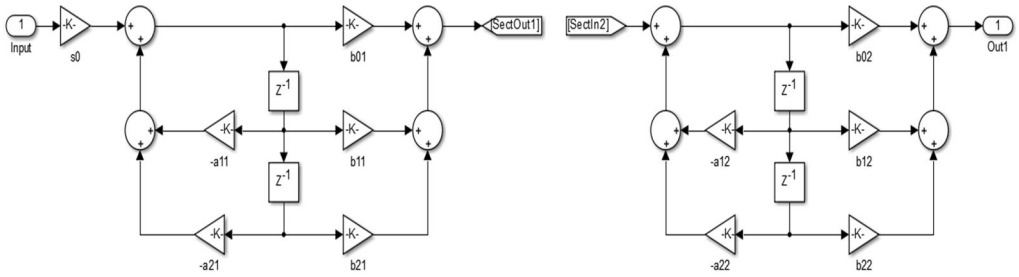


Figure 27. Cascaded second order filter with Direct Form II

Cascaded Transposed Direct Form II Structure:

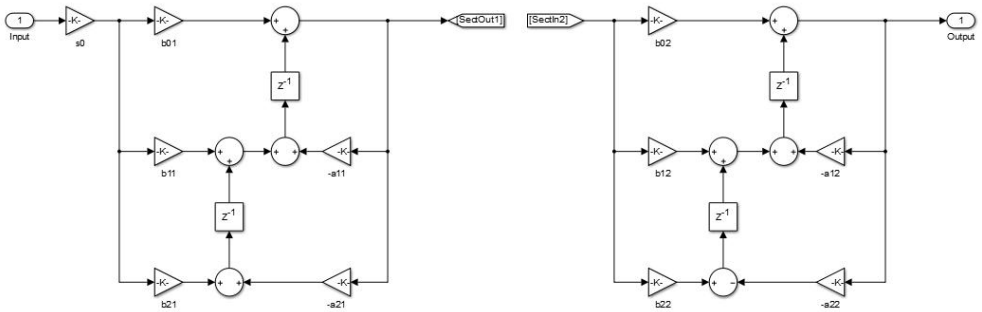


Figure 28. Cascaded second order filter with Direct Form II transposed

3.2.2 Finite Word Length Effects Analysis

When the filter is using floating point arithmetic, the filter's performance remains unaffected with different structures. That's why Transposed Direct Form II structure can be used for all filters during the coefficients calculation. In practice, fixed point arithmetic is employed when MATLAB is used to simulate the filter with different structures.

In the following section, the 4^{th} order filter which is obtained in section 3.1is taken as a reference filter. Because this filter is designed with floating point arithmetic, it will not suffer from the performance degradation that caused by finite word length effects.

Fixed Point Implementation

It is necessary to implement fixed point arithmetic on this 4th order IIR filter for filter realization. When deciding fixed point format of the coefficients, the software and hardware limitations need to be taken into account. Technically, there is no limitation when using MATLAB to simulate with different fixed point formats. In practice, in order to save the resources of the hardware, the best choice is to use as a short bit width as possible. As the goal is to use reasonable bits to achieve good and acceptable filter performance when compared with the original filter in floating point format. To this end, the bit width of the coefficients is set to be 32 bits, which is the same length as the input data.

After the bit width is chosen, another important issue is how to distribute of the coefficients' integer and fractional bits. The fractional bits have a decisive effect on the data precision, and during calculation, proper integer bits could avoid overflow errors [reference].

Taking the 4th order direct form II structure as example, Table 8 explains how to set the filter's fixed point format with specific structures with MATLAB:

Table 8. Example showing how to set filter's fixed point attributes

Design direct form II filter object HD
`HD=dfilt.df2(b,a);`
Specify fixed point attributes and Word and Fractional length properties

`HD.Arithmetic = 'fixed'; % Set fixed point precision`
`HD.Roundmode = 'floor';  % Set rounding strategy`
`HD.AccumWordLength = 64;`
`HD.StateWordLength = 64;`
`HD.CoeffWordLength = 32;`
`HD.InputWordLength = 32;`
`HD.OutputWordLength = 32;`
`HD.InputfracLength = 31;`
`HD.OutputfracLength = 31;`

To ensure satisfactory fixed-point operation of the 4th order IIR filter, the following issues are necessary to be examined:

- Stability
- Coefficient Quantization
- Round-off Noise

Stability

The easiest way to check this attribute is using MATLAB function "***isstable***":

```
flag = isstable(HD);
```

If the flag has value 1, the filter is stable. Otherwise, the filter is unstable, and the fixed point version is not suitable for the filter.

Then the stability of the Cascaded and Direct Form II structures are tested with 32 bits coefficients format. These four filters are found stable.

There is another way to test the filter's stability, which is related to the poles' position on pole/zero plot. Next section will discuss this issue.

Coefficient Quantization

The effect of quantization can be seen in the pole/zero plot. In fact, the issue of filter's stability has to do with the denominator coefficients and correspondingly the poles, on the pole/zero plot.

Pole-zero Deviation

For a quantized filter, the quantized coefficients will cause the displacement of poles and zeroes of the filter. Since the poles are very sensitive to a fixed point arithmetic, and a small change in the coefficients will cause the poles' position changes, which may yield significant performance changes correspondingly. In the following part, the displacement of poles and zeroes are discussed and analyzed by comparing different filter structures with the ideal filter.

In Figure 29, this is the pole/zero plot of the filter in Direct Form II structure. The real part of z-domain is zoomed in to show clearly how the poles and zeros' positions deviate from the reference filter.

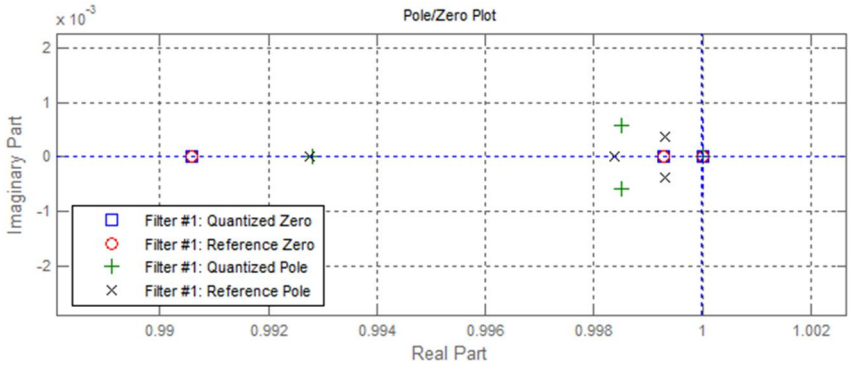


Figure 29. Pole-zero deviations

Comment: It is easy to see that the poles' positions have visible changes at the resolution of 10^-3. The zeros' positions didn't change too much. It proves that poles are more sensitive to coefficients quantization effects. For stability, only if any one of the poles is out of the unit circle, this filter with corresponding structure is unstable and invalid. When plotting the pole/zero diagram for the transposed Direct form II structure as well as the other 2 cascade structures, it turns out that there is no big difference in the pole and zero positions between each of these four filters. As a result, it is very difficult to define which filter structure has better performance with fixed point arithmetic by observing the pole-zero deviations in our case.

Round-off Noise Analysis

Figure 30 depicts the round-off noise spectrum of the four filter structures. It is evident that cascade transposed direct form II structure has the best performance when compared to the other three structures. The second best filter structure is transposed direct form II. The other two filters have very huge round off noise, these two structures have to be abandoned.

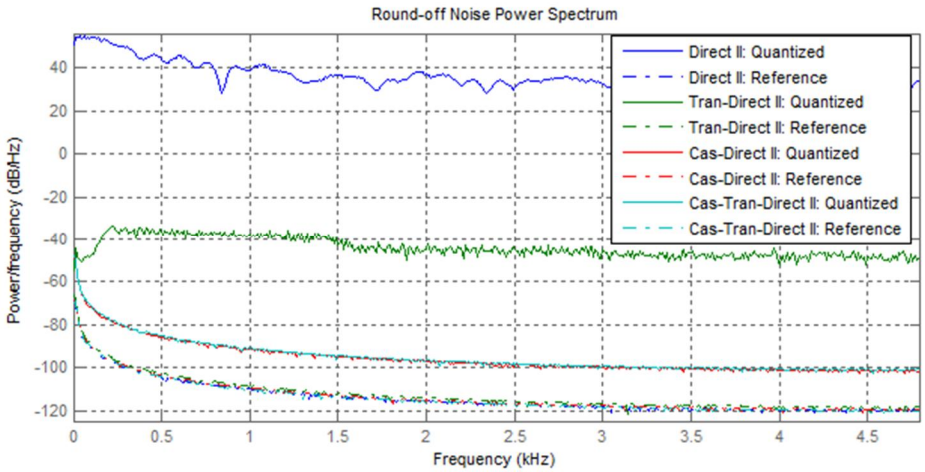


Figure 30. Round-off Noise Power Spectrum

Filtering Functionality Simulation

From the finite word length effect analysis, two appropriate filter structures are decided: **Transposed Direct II** and **Cascade Second Order Section with Transposed Direct II** structures. Finite word length effects are not the only issues that have to be carefully treated when choosing the filter's structure. The filter's functionality is also an essential factor that needs to be checked.

In this section, the filtering functionality of these filters is simulated with the MATLAB function "***filter***":

```
Y=filter(HD,X);
```

It filters a vector of real or complex input data X through a fixed-point filter HD, producing filtered output data Y. The vectors X and Y have the same length. In this case, sinusoidal input X is generated in fixed point format s32.31, this signal is applied to the filter function together with these two filter objects. Then the outputs ***Y*** is compared with the reference signal, which should be the ideal output result. Figure 31 shows the filtering results.

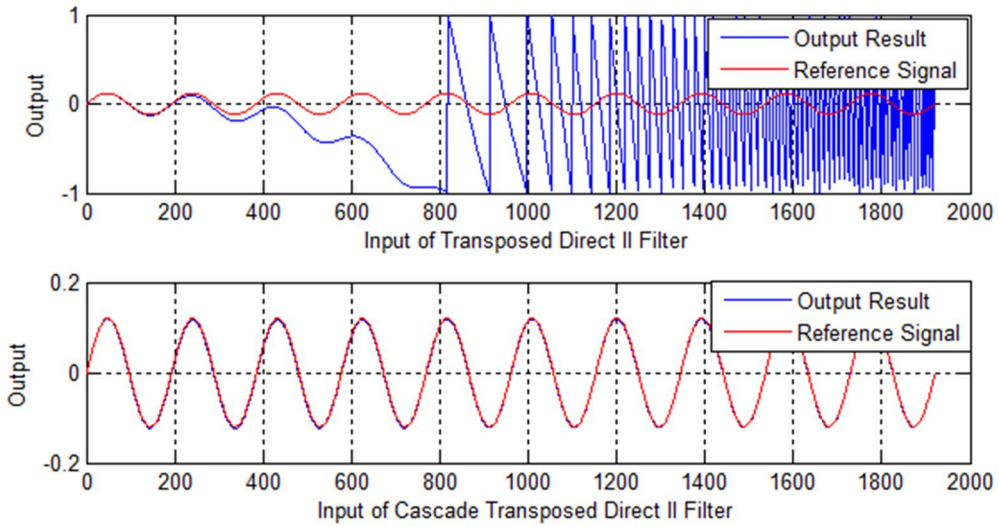


Figure 31. Filtering Functionality Simulation

Comment: It is obvious that filter with cascade structure has excellent output result while the transposed direct II filter has very bad output results. Its output signal deviates from the reference signal significantly.

This chapter gives a complete explanation of the process of how to design an integrator with MATLAB. Based on all the facts, the filter structure can be decided to be Cascade Second Order Section Transposed Direct II.

4. FILTER IMPLEMENTATION

In modern real-time DSP, it is an efficient way to operate filtering cooperated with DSP processors. There are basic blocks onboard, including inbuilt hardware multipliers. This approach is called as hardware implementation [reference]. In some cases, the filter is implemented in a higher level language, such as C or MATLAB. The filter is then run on the computer. This approach is described as software implementation. In the following part, both two design strategies are explained in details with the filter derived above.

The filter is created with cascade realization using the second-order direct II form. The filter's structure is as following. This original filter is called as HD0, using the floating point arithmetic.

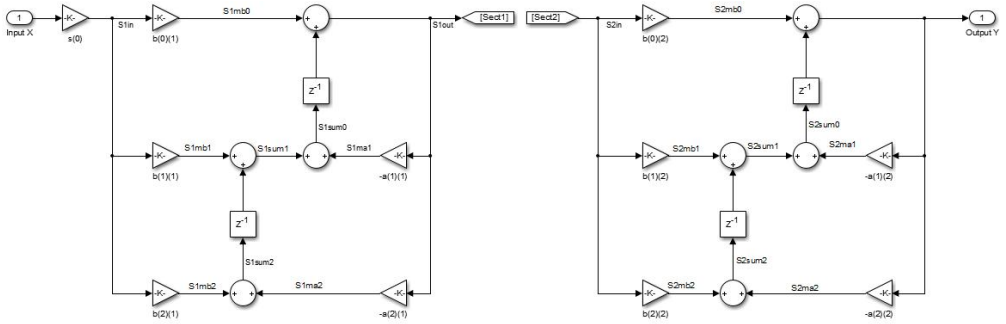


Figure 32. The 4th-order filter Cascade Form structure

4.1 Software Implementation

It is necessary to verify that the designed filter indeed meet the design specification. Software implementation is an effective method to test the functionality of the filter before the hardware implementation. In this thesis, MATLAB and C language are employed in the filter implementation for computer simulation, which contains the FPGA test and the earth fault test. After the simulation, there is a discussion of the feasibility, reliability and effectiveness of MATLAB and C models.

4.1.1 MATLAB Model

The first approach is describing the filter in the form of MATLAB equations. A valid computational algorithm is based on the right ordered equations, which can characterize the filter's structure correctly.

This model must be capable of filtering both floating point and fixed point data. It is an easy way to design firstly the floating point filtering part, and then add fixed point arithmetic to the model. Since fixed point precision affects a lot of the filtering performance. It is necessary to guarantee that the model with floating point can work properly before fixed point arithmetic is implemented.

Floating Point Filtering

This section mainly focuses on how to create right ordered equations for the MATLAB model. Before deciding the equations, the filter's coefficients are initialized accordingly.

Figure 32 shows the filter' block diagram, with each section realized using a standard bi-quad structure. Here, section 1 is taken as an example to illustrate the algorithm for generating the right equations. The corresponding sets of difference equations are as follows. For section2, the equations should be the same order as in section1 .

```
1     S1in=x*s(0);
2     S1out= S1in+S1sum0;
3     S1ma1=S1out*(-S1a1);
4     S1mb1=S1in*S1b1;
5     S1sum1=S1ma1+S1mb1;
6     S1sum0=S1sum1+S1sum2;
7     S1ma2=S1out*(-S1a2);
8     S1mb2=S1in*S1b2;
9     S1sum2=S1mb2+S1ma2;
```

Here a proposed function named as "filter_matlab" is used to represent the MATLAB model. This model is a m-file function with input and output parameters. The basic way that this model deals with signals is like the following, (details cannot be narrated)

```
y(n) = filter_matlab (x(n), option)
```

Where y(n) is the filtering output vector or matrix, x(n) is the input signal in a matrix form. The option is used for choosing the mode of the filter, which means using filtering operation or not, and it has values of either 1 or 0.

Computability Test

The values of all the coefficients in this model now are in floating points. For the original filter HD0, MATLAB function "***filter***" is used to execute the filtering operations. While for the MATLAB model, the filtering function is realized by directly applying the input signal to the function. The testing strategy is to use the same input signal for both filters and then compare the outputs of them. The operations are following:

```
Out1= filter(HD0, X);
```

```
Out2= filter_matlab (X, 1);
```

Where Out1 and Out2 represent the filtering outputs, X(notation) is the sinusoidal input. It turns out that Out1 exactly equals to Out2, which proves that the MATLAB model is computable and functions well.

Fixed Point Filtering

After the computability test, the basic MATLAB model is verified to be functional. The next step is to add the fixed point arithmetic on the basic model. Appropriate data precision of the filter needs to be decided by means of several tests and analysis. It is obvious that if more bits are used when implementing fixed point arithmetic, the more precise the filter would be. However, with the hardware resource limitation under consideration, the input and output precision are decided to be s32.31.

In this case, the word-length of all coefficients are set to be 32 bits, as same as the input and output data. Since the value of the coefficients varies in relatively broad range, correspondingly the fractional part length of coefficients varies as well. Then, how to choose the proper fixed point representation for the intermediate variables becomes a primary problem. An appropriate method is to assess the effects of finite word length on the filter performance and tune the fixed point precision.

The filter's performance is evaluated by means of noise simulation. In general, integrators can have stability problems with zero input with some noise, which will result in output with DC levels due to rounding errors. The DC levels sometimes even go quite high. This happens mainly when there is no signal from the Rogowski coil. The main reason for doing noise test is to check whether the integrator generates DC output or not, and if it does, how much the DC output appears to be.

During the test, noise input is applied as a training signal to simulate the filter with different coefficient precisions. By comparing and analyzing the output data, the fixed point representation of the coefficients is finally decided. It is necessary to tune carefully and select the coefficients' precision, a very good choice would yield smallest or even no DC output when small noise input signal is processed by the filter. Figure 33 shows the DC output with the final precision that be chosen. It is evident that the filter can avoid visible noise interrupts. There will not be significant DC level in the output, for both noise signals with huge or tiny amplitudes.

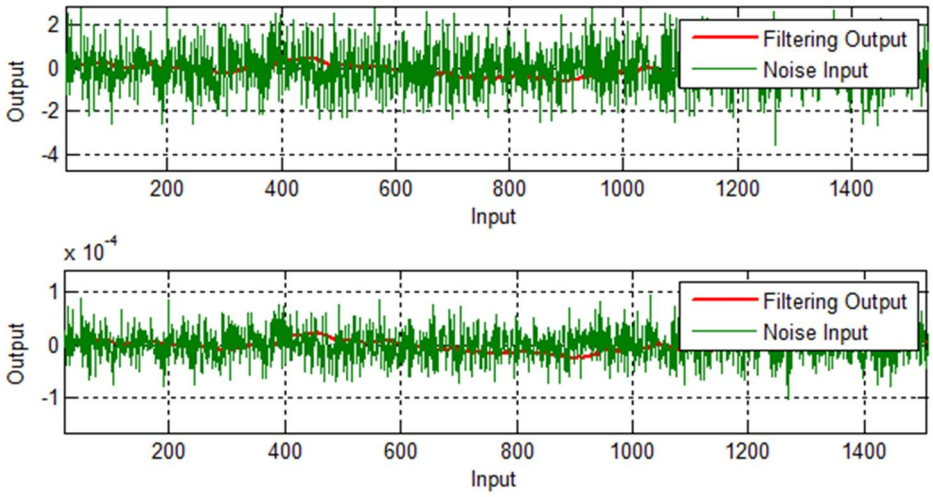


Figure 33. Filtering results of noise inputs with different amplitudes

Dynamic Range

When it comes to the DC fluctuations, the dynamic range needs to be mentioned, which is dependent on the system. In this thesis, the required dynamic range is 60x*In* in RMS level for current measurements.

On the FPGA board, after the AD converter, the first operation is to scale the input to an intermediate signal level. This means for the current input, +/-1 s32.31 representation indicates the whole dynamic range of 60**In*, where *In* is configurable by the user. For confidential consideration, the thesis would not explain in detail about how to decide the signal level. According to the company's requirement, the signal's RMS (Root Mean Square) level is set to be 4.1667e-05. The following is an example when using the MATLAB sine function to create a signal:

```
signal=4.1667e-05* sin(2*pi*f*t)
```

Where we obtain the signal's RMS value of 4.1667e-05 and peak value of 4.1667e-05*sqrt (2) = 5.8926e-05. As a result, the expected output level should be less than 4.1667e-05.

Noise Simulation

After settling the problem of input signal range, noise tests can be executed afterwards. The tests include random noise input with different amplitudes: *0.005xIn, 0.5xIn* and *60xIn*.

MATLAB function is used to generate these noise inputs. Function "***rand***" can be used to generate random numbers and matrices with elements uniformly distributed in the

interval (0,1). Likewise, function "***randn***" can be used to generate random numbers and matrices with items that are usually distributed with zero mean and unity variance []. And all of the input data are in the fixed point format of S32.31. Then it comes to the specific tests.

Test Case 1

At first, a comparison is done between two different data precisions: **S32.30** and **S64.60**. They are separately the shortest and longest word length that can be used in the hardware implementation. Figure 34 to 36 depict the simulation results of noise inputs with the two filters in S32.30 and S64.60. These noise inputs are with different amplitudes levels ranging from very small to regular.

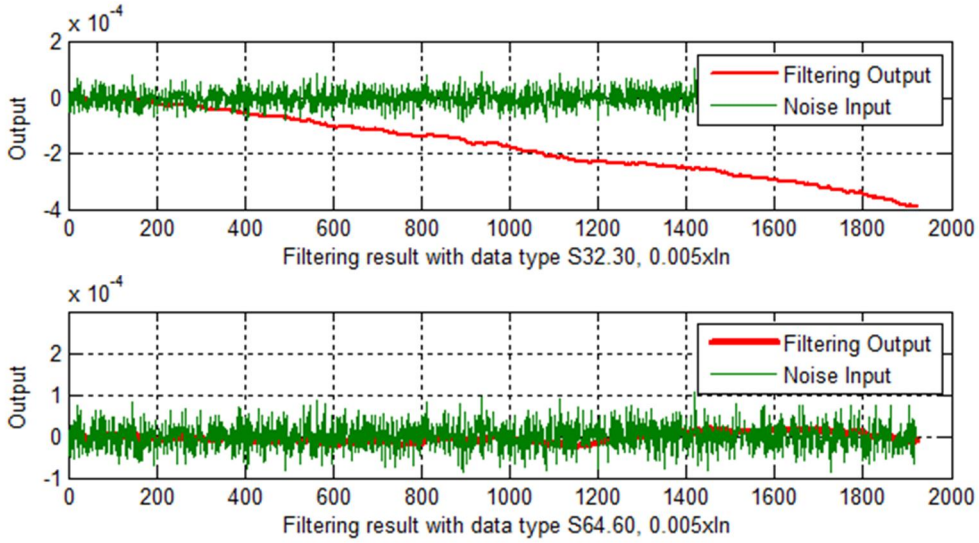


Figure 34. Filtering results of noise inputs (0.005xIn) with different data types

From Figure 34, it is evident that filter with data type S32.30 generates significant DC level in the output while filter with data type S64. 60 works normally.

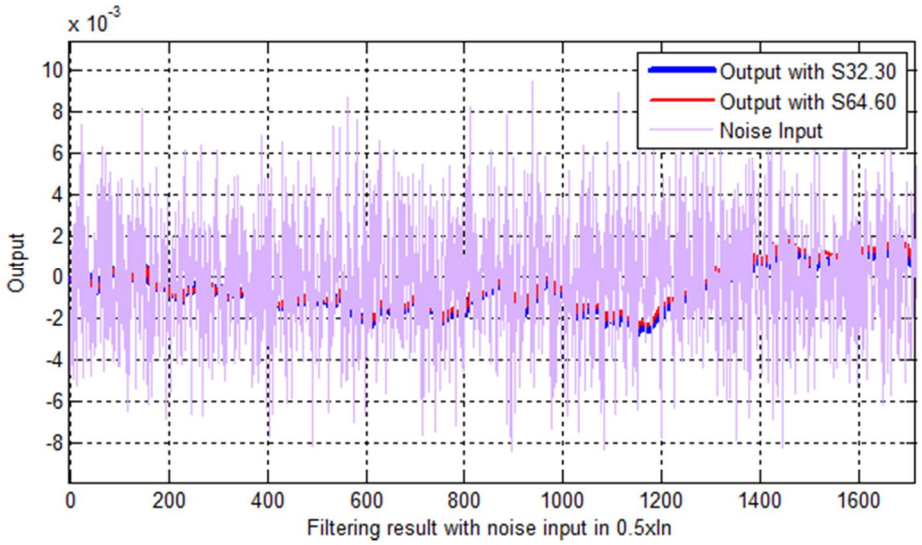


Figure 35. Filtering results of noise inputs (0.5xIn) with different data types

From Figure 35, it proves that when the noise input is normal, the filtering outputs of both filters are normal as well. For this reason, it shows the filter's feasibility when dealing with normal input signals. On the other hand, this filtering result reflects that it is necessary to test the filter with small input to guarantee its stability.

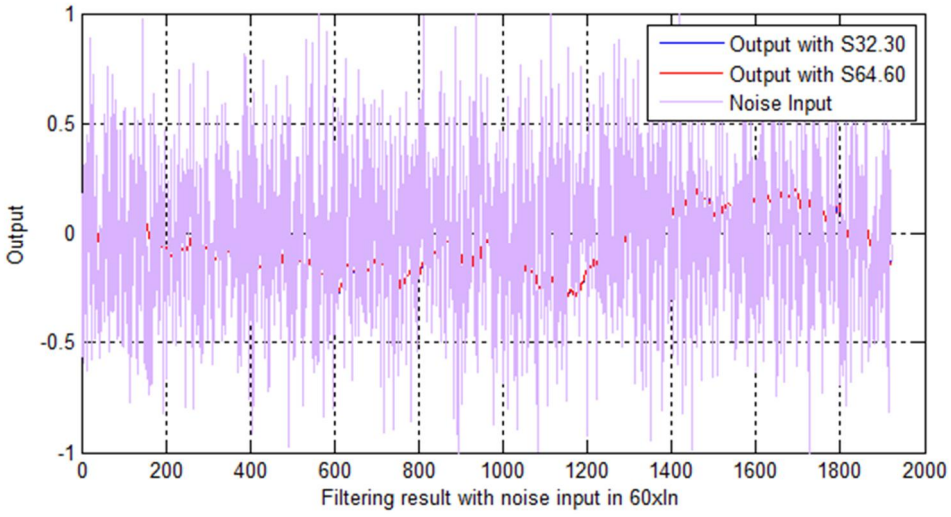


Figure 36. Filtering results of noise inputs (60xIn) with different data types

From Figure 35, it is not easy to distinguish the outputs of the filters. Figure 36 proves again that when the input signal grows to nomal amplitude, both of the filters can work normally.

Based on the above results, it is obvious that filter with data type S64.60 is capable of dealing different noise inputs, even when the signal is very small. Then it is necessary to test whether this filter can also deal with tiny sinusoidal input signal. Figure 37 gives the simulation result, when the input signal is in amplitude of 0.005xIn.

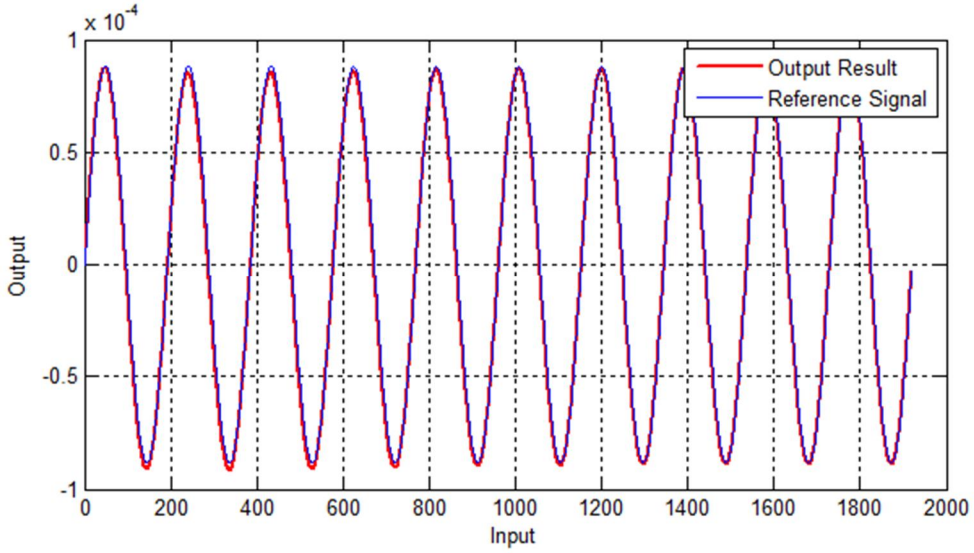


Figure 37. Filtering results of sinusoidal input (0.005xIn)

Comment: The filter with different word-length has quite different output results of the same noise inputs. When using numeric type S64.60, there is no DC fluctuation accumulation in the output, and there is no apparent degradation in the performance with the typical input signal. On the contrary, when using numeric type S32. 30, the noise output is quite great and it's even greater than the noise input values. It is evident that more bits are needed in the integrator.

Test Case 2

From Test Case 1, numeric type S64. 60 is proved sufficient to guarantee good performance. However, more bits means the expense of increased cost. The goal of this test is to find a filter with shortest bit width but still can perform well. More specifically, the new filter does not degrade normal operations too much or make the filter unstable. Here the following numeric types are proposed for testing: **S60. 56, S54. 50, S50, 40**. In order to test and guarantee the performance of the new filter with shorter bit width, input with very small amplitude (0.005xIn) is used in this test. In order to compare the performance differences, each new data type is compared with data type S64.60, the output errors based on the reference signal are depicted in Figure 38.

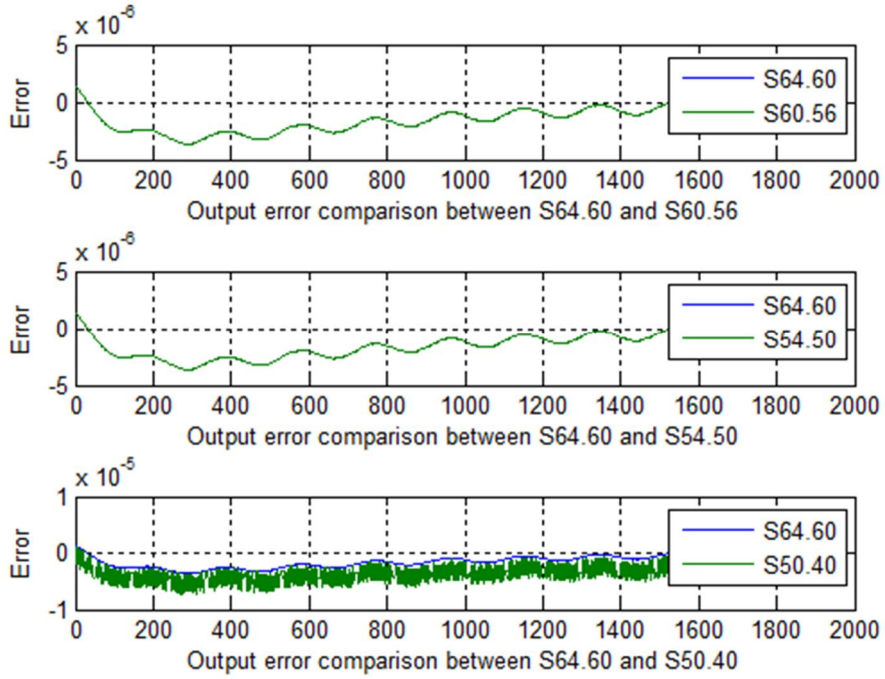


Figure 38. Output error comparisons of different data types

Figure 38 shows that except for the filter with data type of S50. 40, other filters performed well. This means that data type S50. 40 is not sufficient, in another word, either the whole data bits or the fractional bits are not long enough. Then new numeric types are proposed: **S50. 45, S54. 43**. The outputs of these three filters are presented in Figure

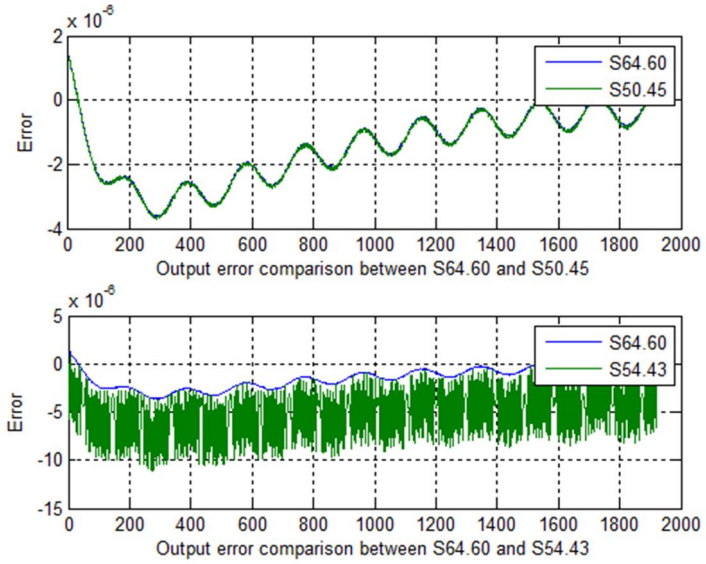


Figure 39. Output error comparison of different data types

39.

From the figure above, it is evident that numeric type S54.43 has awful filtering performance, even though with a longer bit width. This fact indicates that the primary reason for performance degradation is that the fractional bits are not long enough. From this, the numeric type S50.45 could be a real choice. For more precision consideration and several more functional experiments, the filter's numeric type is finally decided to be S53.45.

4.1.2 C-Language Model

The second proposed approach is to realize the filter by C language. C model is beneficial than MATLAB in the sense to directly make VHDL from C, without the aid of Mathworks C to VHDL tool.

The fixed-point representation of the filter has been carefully chosen for filter realization with MATLAB. However, the descriptions of the same coefficients and intermediate data of the filter are different in C model. In MATLAB model, the fixed-point algorithms are designed by using the built-in MATLAB *Fixed-Point Toolbox*. MATLAB functions such as "***fi***", "***fimath***" are used to construct and characterize the fixed-point numeric objects directly from the floating-point numeric type. When referring to the C language model, the data is in the appearance of integer numeric type, which needs to be transformed again from the fixed-point data. For a fixed point number ***A,*** the corresponding integer type ***A_int*** is derived by following operation:

```
A_int=A.int;
```

Multi-Precision Library -- GMP Implementation

In C language model, there are several variables that need more than 64 bits after the arithmetic operations (mainly multiplication). It is obvious basic integer types in C language are not sufficient for this case, a new method to represent the filter's variables is needed. For this purpose, GNU Multiple Precision (GMP) Arithmetic Library is introduced. It is a free library for arbitrary precision arithmetic, operating on signed integers, rational numbers and floating-point numbers [43].

The integer arithmetic functions provided by GMP library is also called as "***mpz***". There are about 150 arithmetic and logic functions in this category [43]. GMP integers are stored in objects of type *mpz_t*, and the GMP integer functions start with the prefix *mpz_*. In Table 9, integer A_int and GMP integer A_mpz are used to explain the basic operations of GMP integer functions.

Table 9. Basic GMP integer functions

Data Initialization

mpz_init_set_si(A_mpz, A_int); // Assign the value of A_int to A_mpz
or
mpz_init(a_mpz); // A_mpz is initialized to with default value 0

Data Conversion
B_int= mpz_get_si(A_mpz); // Convert A_mpz (mpz int) to B_int (int)
Integer Arithmetic

mpz_add(C_mpz, A_mpz ,B_mpz); // C_mpz= A_mpz+B_mpz
mpz_mul(C_mpz, A_mpz, B_mpz); // C_mpz= A_mpz+B_mpz
mpz_fdiv_q_2exp(m, m, 22); // C_mpz= A_mpz+B_mpz

4.1.3 Performance Evaluation

Since MATLAB and C Language models represent the same digital filter, the outputs of these two models must be exactly the same. The equivalence test is done by comparing the output of the two models with the same input. It turns out that the only difference between these two models is the filtering time when the input is in fixed point arithmetic. The C model is obviously faster than MATLAB model. That's another advantage of C model.

Either model can be used for the following FPGA test and Earth-Fault test. Even though the C model is much faster, in some cases it is still more convenient to use MATLAB model. Since MATLAB has built-in analysis and debugging visualization tools while C does not have.

FPGA Test

This test is based on the company's FPGA model, which is applied to modeling the real-time FPGA analog signal processing. MATLAB's *Fixed-Point Toolbox* is used to simulate the fixed point arithmetic operations for the real FPGA hardware. The filter model is one integrated part of this FPGA model.

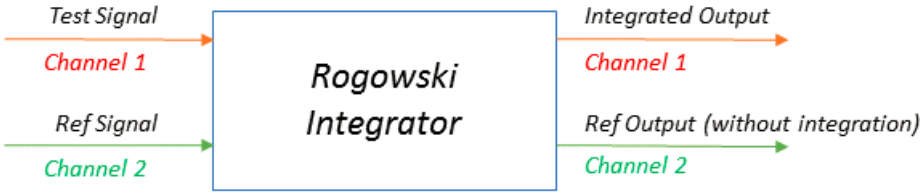


Figure 40. Block diagram of filter functioning

A test bench is made for this model. Test cases are simulated in this test bench, some are especially designed for the Rogowski integration functionality testing. Two fundamental sine waves are employed for the functionality tests. One is used as test signal, and the other is used for testing reference. The brief function diagram is shown in Figure 40.

The two sine waves are all the same except for the phase difference, the test signal is 90-degree phase lead to the reference signal. Theoretically if the filter works correctly, the output of the two signals should be exactly the same. Here two test cases under different frequency conditions are listed for analyzing the integrator's performance.

Test Case 1

This test is done in 9.6 kHz. After the scaling work, the input signal is applied to the FPGA model for a series of operations. Figure 41 shows the two scaled input signals. Except for the phase difference, the two signals are exactly the same.

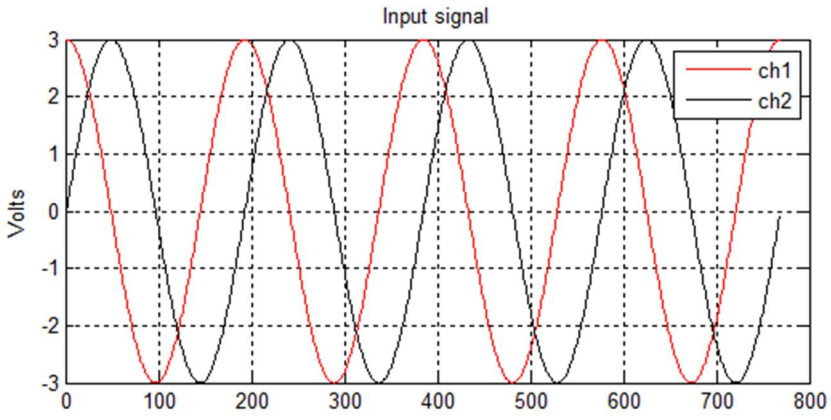


Figure 41. Filtering results of test case 1

Comment: After the input correction, the signal in channel 1 is integrated by the integrator while the signal in channel 2 stays the same. Figure 41 shows output signals that after integration, signal in channel 1 changed to have the same phase as signal in

channel 2. And the deviation in amplitude is very small. This proves that this filter is equipped with the integration functionality

When the filter's functionality is verified, the next step is to examine its performance. The proposed method is to check how much difference there is between the output signal and the reference one. Figure 42 plots the error between the integrated and reference signals. The maximum value of the error is about 0.02 while the amplitude of the signal is 0.5. The error percentage is 4%, which is acceptable.

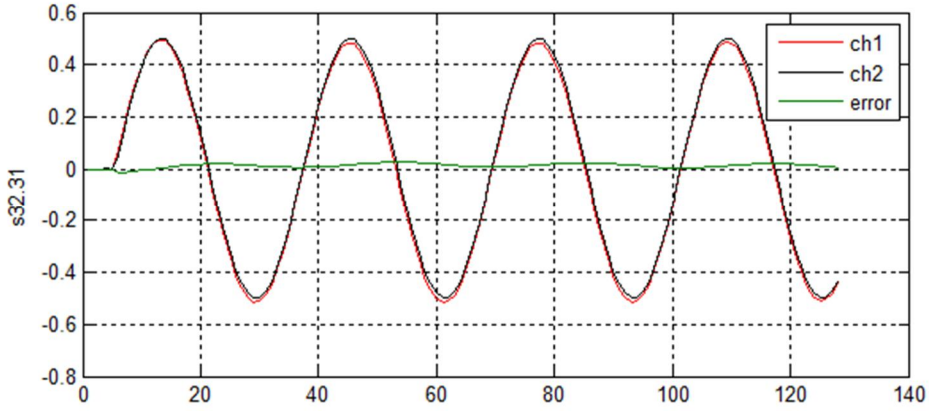


Figure 42. Output signals after integration and the error between two signals

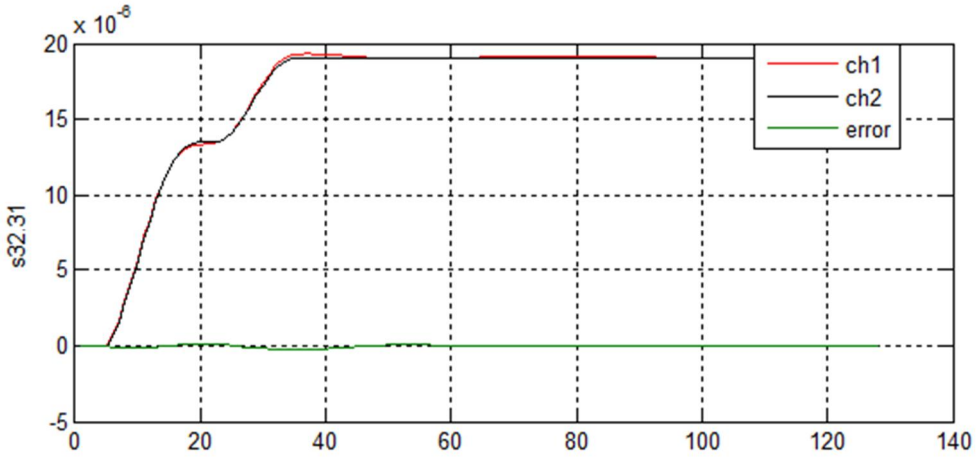


Figure 43. Output results after one operation on FPGA

The output data are applied to several other operations after the integrator. This provides another idea for testing the quality of the output signal. That is, to check the output of the integrated signal, and then compare the deviation from the reference output without integration. Since the purpose of Rogowski integrator is to restore the current and apply it for further processing. Figure 43 shows the output results after one operation on the

FPGA board. The entire signal shapes are nearly the same in these two signals, while there are visional but acceptable deviations/error between them. And the deviations become very close to zero with time passing by. This declares that even though the integrator can works perfectly, the performance is still very good.

Test Case 2

In this case, input signals are tested at frequency of 12.5 Hz, the input signals are the same as test case 1. The idea of this test the integrator functionality with small frequency. Figure 44 shows the result of the integration. This integrator works as expected considering the frequency response figures of the filter for 12.5 Hz. Since there is no apparent phase deviations and the shape of the two signals are similar. This test case proves that even though in very small frequency, the integrator will not suffer un-

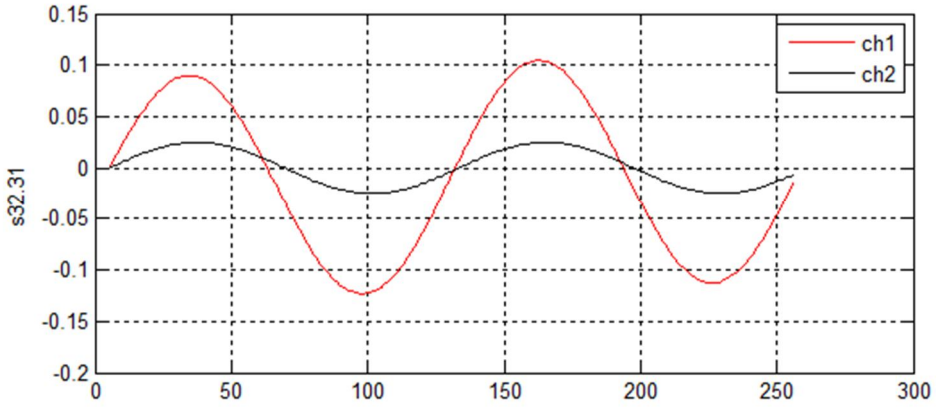


Figure 44. Output signals after integration

expected deviations.

Earth-Fault Test

The main purpose of this test is to guarantee that the integrator can normally work under circumstances when some intermittent earth faults happen. The specific test bench is designed for simulating the earth-fault on the output signal of the integrator. The test bench will generate the simulation result with corresponding text messages.

The test data used for testing came from the practical field test recordings, which is composed of three current channels and three voltage channels. The three phase current channels are I_1, I_2, I_3, with different phase. It has very high sampling frequency up to 100 KHz and relatively significant amplitudes for the integrators. It is necessary to operate the data with downsampling and rescaling operations before applying it to the integrator. After being resampled and rescaled, the three channel currents are used to the

MATLAB model. After integration, these three channels are combined into one channel by taking the average operation, which yields I_0. I_0 is downsampled again from 9.6 kHz to 1.6 kHz, which is the working frequency in the earth fault test bench.

The same operations are executed on the three voltage channels, the new voltage signal V_0 is finally obtained. I_0 and V_0 are stored in one file, which is taken as an input parameter or test case in the earth fault test.

It is not convincing to say that the integrator is good enough by taking only one test case for this proving. The number of test cases should be sufficient to demonstrate this argument. There are a lot of test data in the recordings. As a result, every file storing different I_0 and V_0 values can be considered as a test case. These test cases are applied to the simulation.

For confidential consideration, the simulation messages are not shown here. The result shows that these test case are passed. It proves that the integrator work well under the conditions where earth-fault occurs. This is very important in the practical application.

4.2 Hardware Implementation

After the functionality and the performance of the integrator have been verified, this integrator should be realized with hardware implementation, which is built on FPGA. The design of the hardware model is based on the C model developed above. Here for confidential consideration, only the topology of this integrator is shown.

4.2.1 Topology

The Rogowski integrator is composed of two cascaded second order filters. The two filters are almost the same in the structure. Topology of the first cascaded section is depicted in Figure 45.

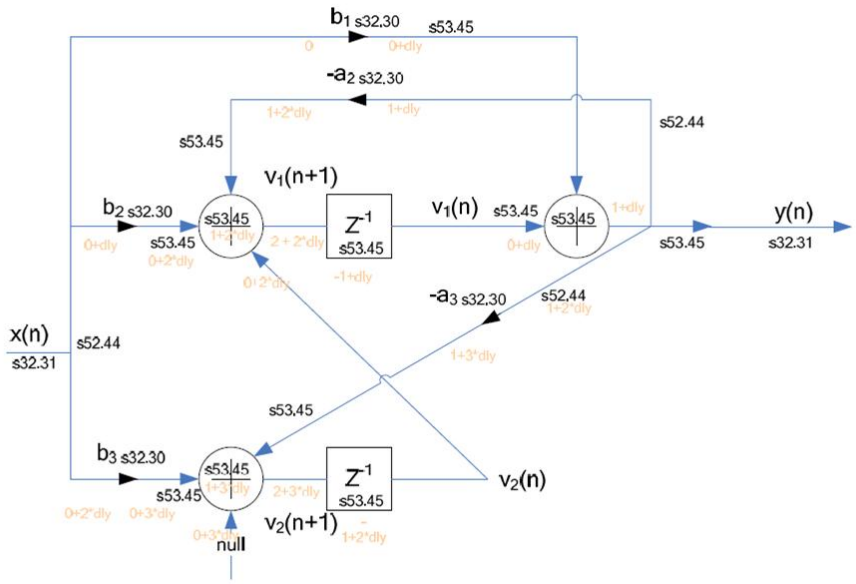


Figure 45. Filter topology of one section

The topology gives particular bit width of each coefficient as well as the internal variables. Red markings in the figure are corresponding delays in each stage. The implementation is realized by means of state machine operation, using a single multiplier and a single wide summing block (another sum operation gets inferred into output saturation logic). The delay for the calculated output data is at a minimum only 4 clock cycles. Delay is always generated to match the delay in group delay filter, which is especially design for the Rogowski integrator.

5. TESTING RESULTS AND ANALYSIS

For the purpose of testing, the integrator designed above is applied to both computer simulation and practical field tests. This chapter mainly discusses and analyze the essential functionality and performance of the filter that required by the design specification.

5.1 Functional Verification

The integration functionality test is conducted by analyzing the outputs of the integrator, which is applied to different typical type of inputs. Five primary test cases are carried out in this section.

In the necessary tests, typical signals such as *DC input*, *sinusoidal inputs*, *unit step inputs*, etc. are generated at first. These tests also involve the conversion of data from floating point to fixed point format. And the data is in the numeric type of **S32.31**. The input data is real number within the particular scaling range. When test data are ready, reference signals are generated according to the outputs of an ideal integrator or integrator in floating point format. In the following, each test case with different input will be discussed.

5.1.1 Sinusoidal Input

As the smaller the signal, the more possibility that the accuracy will get worse. And it is necessary to make sure the integrator can normally work with very small inputs. Thus, the sinusoidal input is with RMS value of 0.005xIn, and it is in 50 Hz frequency. The

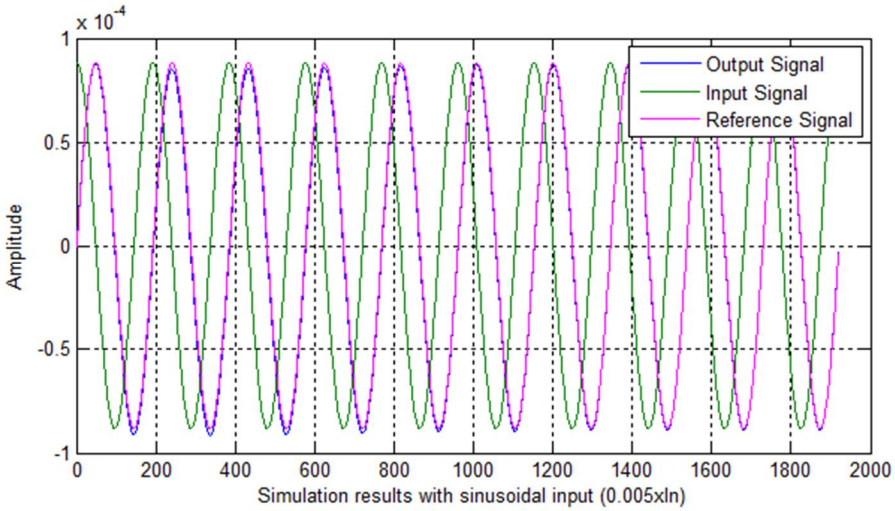


Figure 46. Simulation results with sinusoidal input

filtering result is compared with the original input signal as well the reference signal. Figure 46 depicts the comparison.

The figure shows that the filter integrated the input signal with -90-degree shift and almost the same amplitude of the previous signal is kept. As in the specification, when the input signal is 50 Hz, the integrated result should exactly 1. The figure also proves that the integrated signal is almost the same as the reference signal, even though, the input is relatively small. This fact indicates that this integrator normally works even with tiny input signals.

5.1.2 DC Input

According to the specification, the integrator output should be zero with the DC input to prevent the integrator windup. When designing the integrator in chapter 3, notches were explicitly added correctly at 0 Hz for this purpose. However in a practical implementation, it is very difficult to realize excellent performance for quantization effects. Additionally, the filter's magnitude response at 50 Hz should be 1, and then there will be a very steep peak between these two frequency points. As a conclusion, the expected output for a DC input may have fluctuates at the beginning, but it will decay to zero with time passing.

In this test case, the input is still in the range of 0.005xIn to ensure the filter's functionality. Figure 47 shows the filtering result.

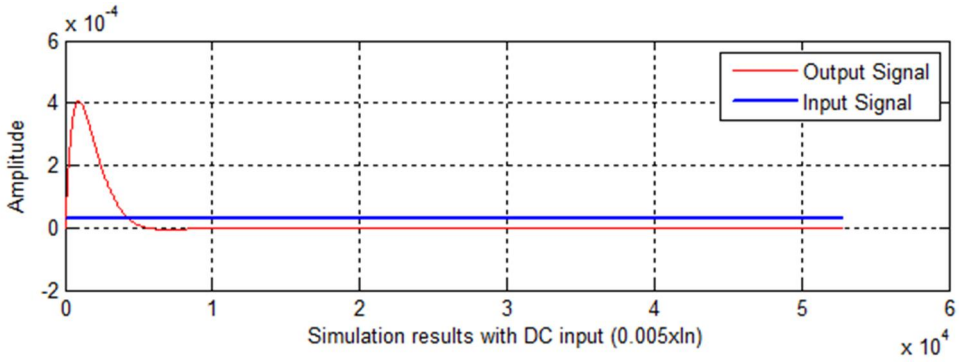


Figure 47. Simulation results with DC input

From the figure, the output has relative but acceptable fluctuate in the beginning. With time passing by, the output tended to be zero. It is very good to see that after 10000 samples (9.6 kHz sampling frequency is used so that the period is about 1 second), the output is very close to 0. This test result proves again that the integrator normally works .

5.1.3 Impulse Input

The impulse signal is used as an input to check the filter's stability when dealing with signal which has immediate changes in the amplitude. It is very necessary for the filter to be steady when such kind of condition happens. Figure 48 shows the result.

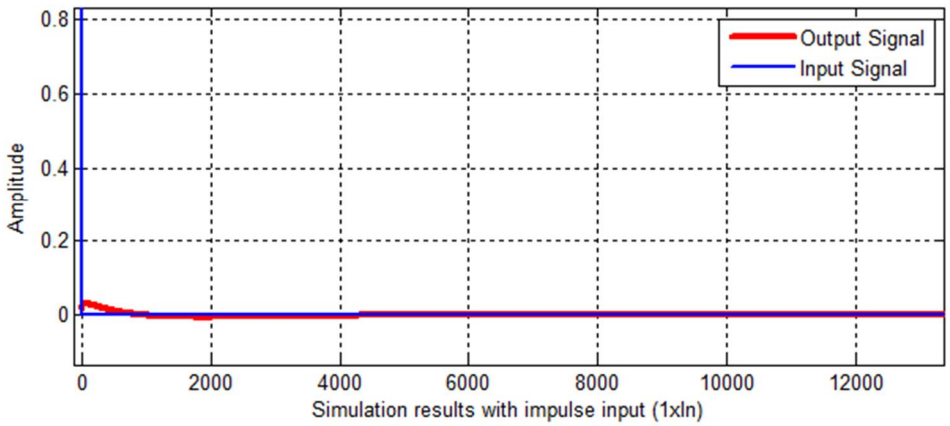


Figure 48. Simulation results with impulse input

As an input, the impulse signal has the value of 1xIn in the first sample, then the signal turns to be 0 afterward. From the figure, the integrated output is very small and decays to zero very quickly.

Derivative of impulse Input

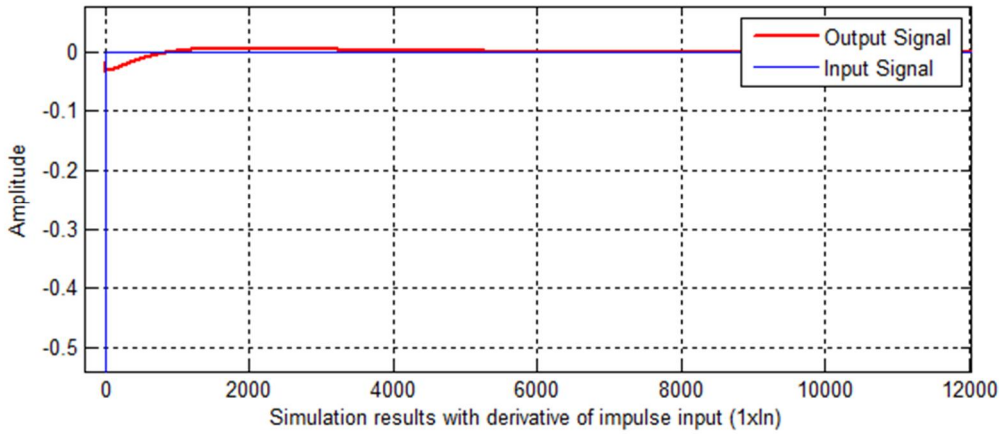


Figure 49. Simulation results with derivative of impulse input

Figure 49 shows the output of a derivative impulse signal. This test simulates the derivative of a real current impulse, which should result in an impulse output without many residual fluctuations. The filtering result proves that the filter works well.

5.1.4 Noise Input

The purpose of taking noise as input is to exam that whether the integrator would generate or cumulate DC level in the output signal. In practical application, it is unavoidable

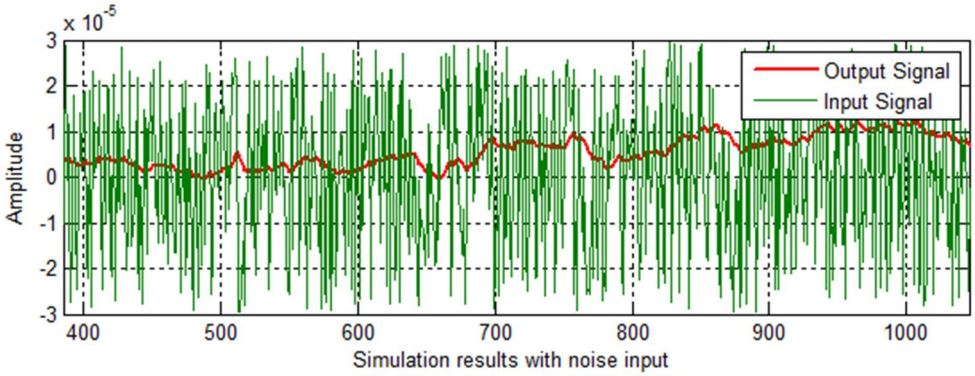


Figure 50. Simulation results with random noise input

that the input under operation will be mixed with some extra noises. A well-performed filter has to deal this properly, that is the output should stably stay in acceptable amplitude range without obvious value changes. Figure 50 shows the integrating result.

Based on the figure, the DC level of the output is very small. And, even though, the input noise is very small, the output of the integrator is far smaller than the noise signal. It is necessary to guarantee. The input is simulated with 5 seconds, the output stayed with similar small values. In a brief, the integrator handles very well with noise input problems.

5.1.5 Complex Input

In practice, the normal input signal is mixed with noises. In this test case, we will test the functionality of the integrator by applying AC signal with noises. The AC signal is standard sinusoidal data with frequency of 50 Hz. The simulation result is presented in.

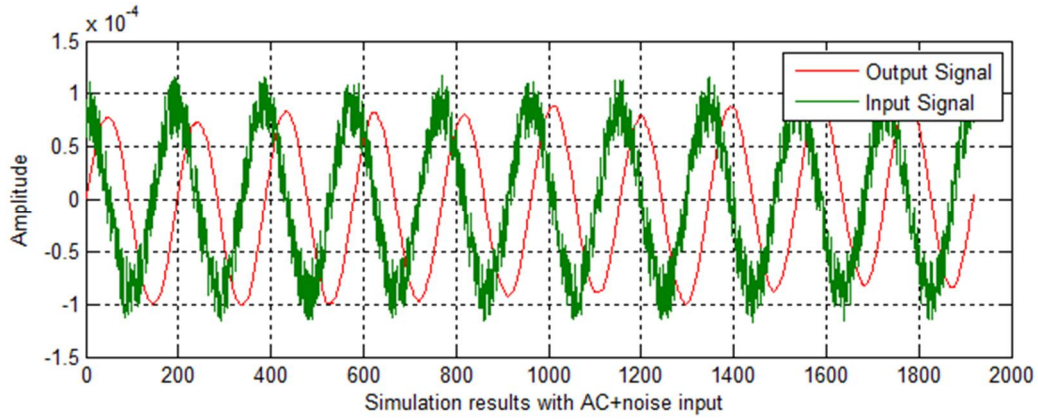


Figure 51. Simulation results with complex input

Here, in the above figure, the integrated signal has very small fluctuates and degraded the noise in the output. There is no apparent fluctuate in the output, and it remains a general sinusoidal signal shape.

5.2 Performance Verification

From the functionality verification section, we confirm that the integrator could normally work. For industry application, the requirements for an integrator will be even stricter. That's why we conduct the following performance tests to verify that the designed integrator could filter signal with excellent performance in the specified fields.

The performance of the integrator is reflected by the current which is calculated by integrating the signals from Rogowski sensor. The current data is from the latest field test recordings with the integrator embedded on the FPGA board. Since the filter's performance has been simulated with test bench already, the main idea of this section is to verify that the practical field tests of the integrator match the simulation results.

When we simulated the data for earth fault tests, the value of calculated I0 was derived from the sum of three phase current channels, which at first passed through the Rogowski sensor and then were filtered by the MATLAB or C integrator model. In practical tests, the calculated I0 value is stored in the recording of the analog channels. Meanwhile there is also a reference I0, which is from a core balance current transformer. Thus, it is very easy to extract the I0 and calculated I0 for comparison and analysis directly from the recordings.

There are different test cases since there are hundreds of recording files, and the field tests covered different kind of faults, we will mainly list several test cases with various earth faults introduced. For each test case, the calculated I0 is analyzed by the following three performance issues:

- Accuracy
- Transient State
- Stable State

5.2.1 Solid Earth-Fault

At first, we analyze the test case where the solid earth-fault was interrupted. From Figure 52, it is evident that the accuracy of the calculated I0 is very good since the two current signals overlap on each other relatively well.

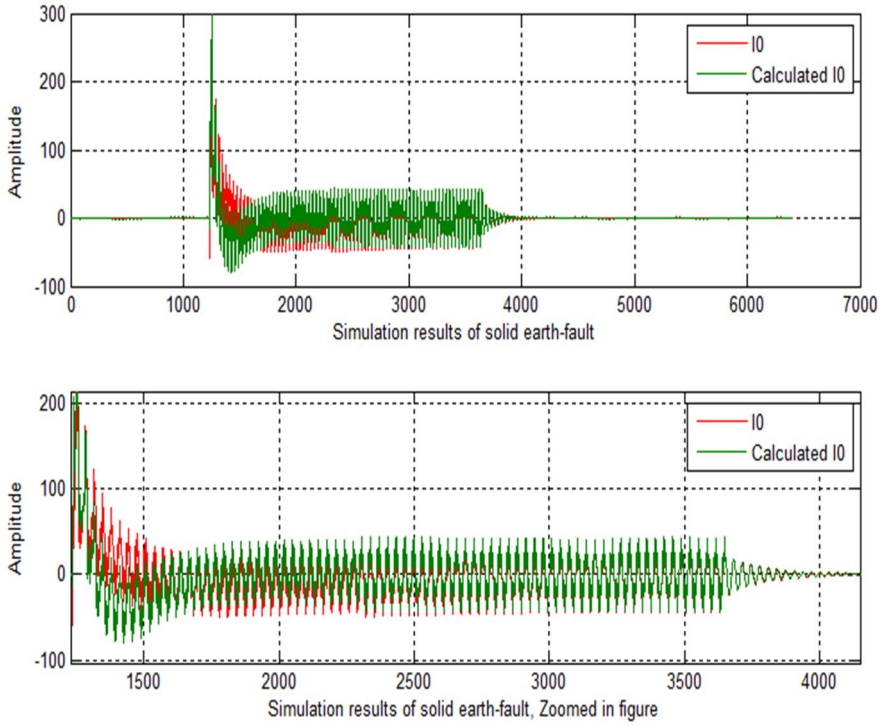


Figure 52. Simulation results with solid earth-fault

Then the figure is zoomed in for more details about the transient and stable states. The diagram in the below of Figure 52 depicts the transient state after the earth fault was introduced. As we could see, there are fluctuates in the calculated I0 signal but the amplitude and phase of the current are almost the same as the reference signal. We could refer this condition to the Impulse Response simulation in section 5.1.3 since the transient process is similar to the impulse response. In that simulation, the response at first

went up immediately with very high amplitude and then went down to a lower amplitude level. Afterwards the response returns to zero. Here, in this case, the calculated I0 has a similar response when the fault happened, which was like an impulse interrupted in the typical signal.

After the transient state, the current went to a stable state, in the zoomed in diagram, we could see that the calculated I0 has very great matching degree to the reference I0. As a conclusion, the practical performance of the integrator in the solid earth-fault is excellent.

5.2.2 Low-ohmic Earth-Fault

In this section, a similar analysis is done for the integrator when the low-ohmic earth-fault happens. The following figure is derived from one of the test recordings.

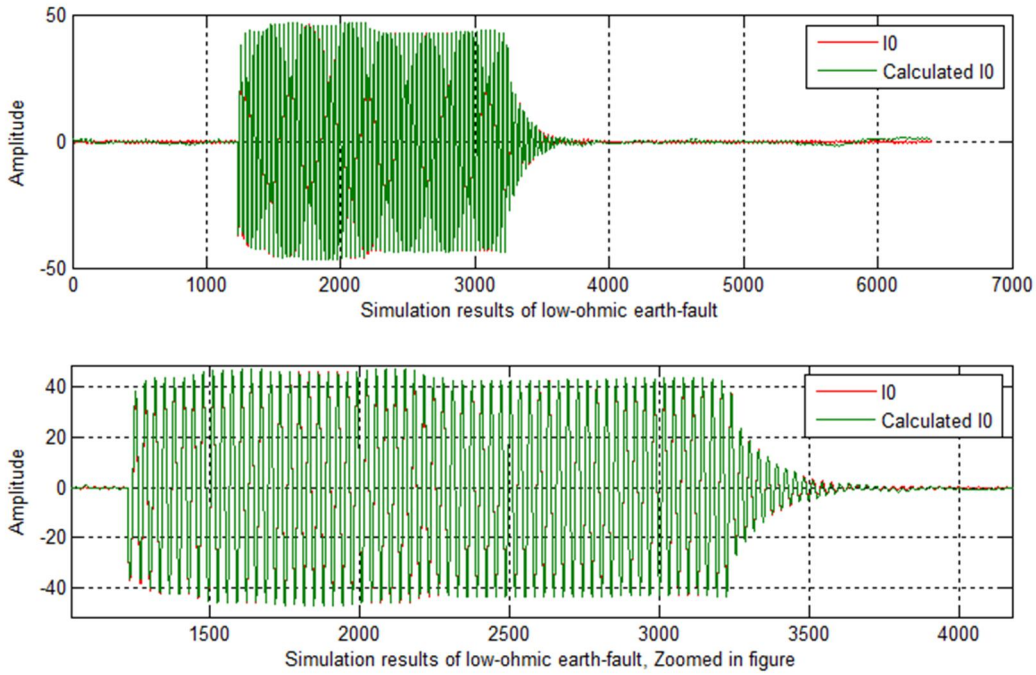


Figure 53. Simulation results with low-ohmic earth-fault

This test case is a very good proof of the integrator's great performance. From the above figure, even in the transient state, the calculated I0 has very small deviations from the reference I0. The accuracy of calculated I0 is at a very high level.

5.2.3 High-ohmic Earth-Fault

Figure 54 shows the calculated I0 and corresponding reference I0 when a high-ohmic earth-fault happens. This test case is similar to the low-ohmic earth fault case. For both

cases, they have very good performances in the transient state, one problem is that there are small current fluctuations in the stable state. But in the general, the accuracy of the calculated I0 are very high in both amplitude and phase responses.

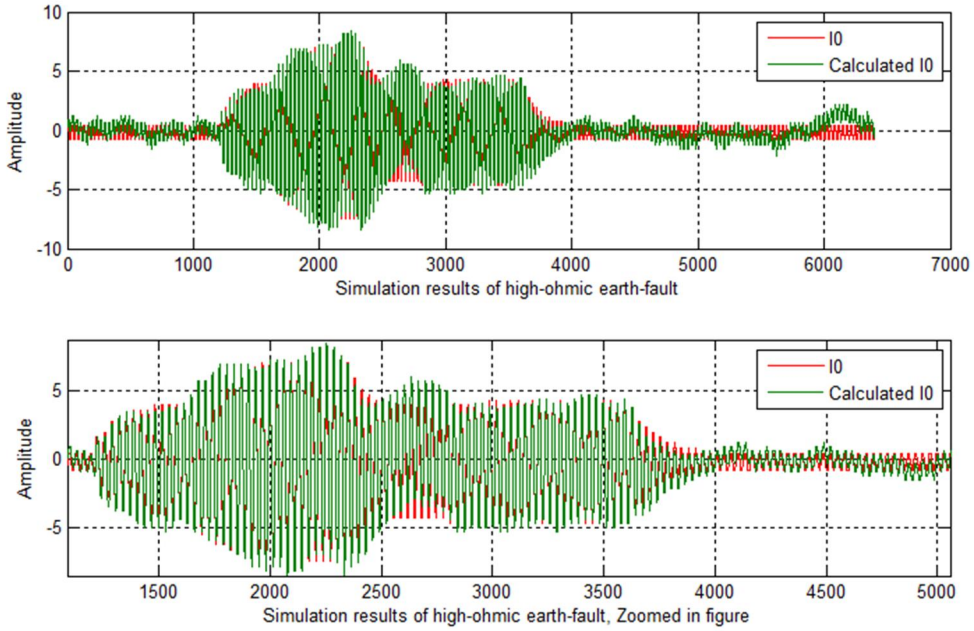


Figure 54. Simulation results with high-ohmic earth-fault

5.2.4 Intermittent Earth-Fault

The last test case is about the performance of the integrator when the intermittent earth-fault happens. Figure 55 displays the test results.

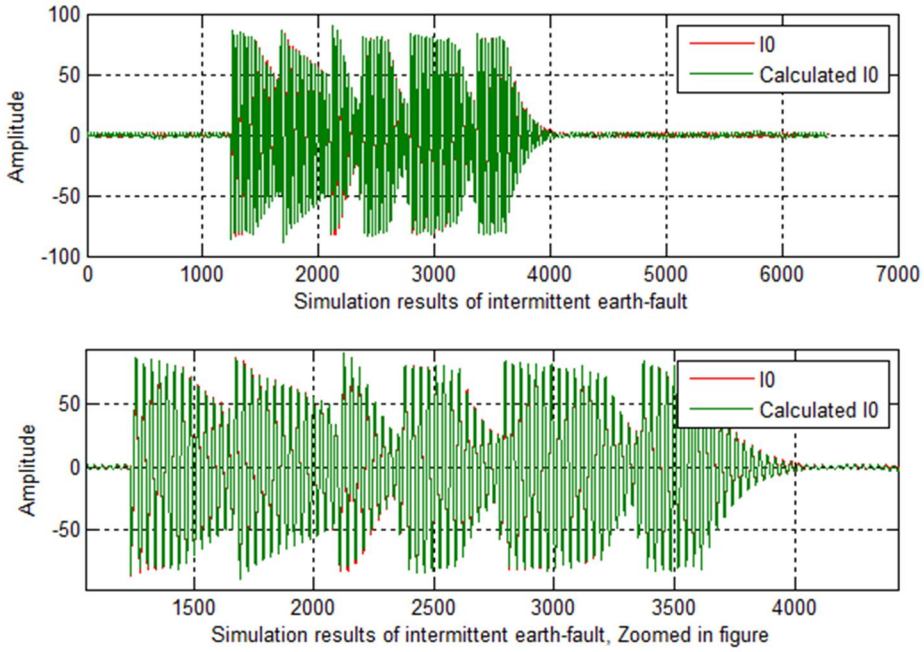


Figure 55. Simulation results with intermittent earth-fault

The integrator performed very well in this test case, it is evident that the calculated I0 is in very high accuracy when compared to the reference current. There are several interrupts during the transient state, the calculated I0 has almost the same amplitude as well as phase value with the reference I0. There are no noticeable fluctuations in the current.

5.3 Summary of the Tests

From the above tests, the validity, practicability of the Rogowski integrator have been verified from the perspective of both functionality and performance. The software implementation of the integrator is capable of filtering different kinds of the input signals with reasonable and acceptable outputs. Meanwhile, in the practical application, the integrator performed well when dealing with various earth fault cases. All, in brief, this Rogowski integrator is within the standard of the design specification.

6. CONCLUSIONS

6.1 Discussion

As an imperative part of the Rogowski sensor, the integrator plays a significant role in the accuracy of the measured current. A lot of issues need to be considered when designing an integrator.

Choosing a good starting point filter for further improvements is necessary at the first stage. Several methods can be utilized during the design. In our design, the fundamental prototype adopted is a trapezoidal integrator. Then we optimize the filter by increasing the filter's order with pole-zero placement. Pole-zero placement is a very efficient way to increase or insert a particular notch/peak. The filter's performance regularly turns better after increasing its order. However the increase of the order comes along with the price of more resource costs, a good designer should find a proper balance between the performance and the costs.

Genetic Algorithm is used for the filter's optimization. There are also other computational algorithms for optimization, such as simulated annealing algorithm, differential evolution algorithm, particle swarm algorithm, colony optimization algorithm, and Tabu search algorithm, etc. [34]. The efficiency of Genetic Algorithm depends on how well defined of the fitness function. The fitness function evaluates the filter's performance and gives corresponding fitness values. The evaluation standard refers a lot to the design specification. Thus, a proper definition and analysis of the specification is very important for the designer.

For a high order IIR filter, the general choice for the structure realization is the second order sections in cascade or parallel form. The decision also lies on the finite word-length effects if fixed point arithmetic is employed in the filter. The coefficients and intermediate variables' accuracy will be degraded by the quantization effects, which will cause the filter's performance deviation. Other finite word-length effects also affect the performance of the filter. A good structure could effectively minify the effects to some extent.

There are two approaches to implementing the filter. One is software implementation, the other is hardware realization. Software approach in our case is realized by creating the MATLAB and C model of the integrator. By software implementation, the filter could be simulated with targeted test benches. It is very efficient and valuable method to exam the filter's functionality before implementing it in the hardware. When the software implementation is verified, hardware implementation could be carried out if it is necessary. In our case, RTL model is derived from the C model by translating it with

VHDL. Afterward, the integrator is implemented on FPGA board for practical field tests.

Based on the testing results, the performance of design integrator is very good when referring to the design specification. The basic magnitude and phase response of the calculated I0 are of very high accuracy. Its ability of dealing with different earth faults is also very good.

6.2 Challenge

There is always improvement remaining to be done. For this integrator, what could be considered to improve is the DC fluctuations caused by the noise in small currents. One possible solution is to increase the filter's order, which may have a positive effect in the frequency range around 0 Hz. The other method under consideration is to use new optimization method for searching even better filter coefficients.

REFERENCES

[1] Kojovic, Ljubomir. "Rogowski coils suit relay protection and measurement [of power systems]." *Computer Applications in Power, IEEE* 10.3 (1997): 47-52.

[2] Ward, David A., and J. La T. Exon. "Using Rogowski coils for transient current measurements." *Engineering Science & Education Journal* 2.3 (1993): 105-113.

[3] Luo, Pandian, Hong-bin Li, and Zhen-Hua Li. "Two high accuracy digital integrators for Rogowski current transducers." *Review of Scientific Instruments* 85.1 (2014): 015102.

[4] Zhang, Gang, et al. "A new electro-optic hybrid current-sensing scheme for current measurement at high voltage." *Review of scientific instruments* 70.9 (1999): 3755-3758.

[5] Tong, Yue, and Bin Hong Li. "An accurate continuous calibration system for high voltage current transformer." *Review of Scientific Instruments* 82.2 (2011): 025107.

[6] Becherini, G., et al. "Critical parameters for mutual inductance between Rogowski coil and primary conductor." *Instrumentation and Measurement Technology Conference, 2009. I2MTC'09. IEEE*. IEEE, 2009.

[7] Ljubomir Kojovic, Martin Bishop. Rogowski Coil Designs. Available: https://www.pacw.org/issue/autumn_2007_issue/protection_rogowski/rogowski_coil_designs/complete_article/1.html

[8] Mitra S K, Kuo Y. Digital signal processing: a computer-based approach [M]. New York: McGraw-Hill, 2006.

[9] IEEE Standard C37.235-2007, IEEE Guide for the Application of Rogowski Coils Used for Protective Relaying Purposes.

[10] V. Skendzic, B. Hughes. Using Rogowski coils inside protective relays[C]//Protective Relay Engineers, 2013 66th Annual Conference for. IEEE, 2013: 1-10.

[11] Ramboz, John D. "Machinable Rogowski coil, design and calibration."Instrumentation and Measurement Technology Conference, 1995. IMTC/95. Proceedings. Integrating Intelligent Instrumentation and Control., IEEE. IEEE, 1995.

[12] Sensor Technology, Applications for medium voltage, ABB Medium Voltage Products. Available: http://www05.abb.com/global/scot/scot229.nsf/veritydisplay/576c37ea10824a4bc1257b12002a396d/$file/NewAndInnovativeWay_article_757842_ENa.pdf

[13] Xin Z, Longhua M. Design for digital integrator of Rogowski coil based on pipeline structure[C]//Power and Energy Engineering Conference (APPEEC), 2010 Asia-Pacific. IEEE, 2010: 1-4.

[14] D'Antona, G., et al. "AC current-to-voltage transducer based on digital processing of Rogowski coils signal." *Sensors for Industry* Conference, *2002. 2nd ISA/IEEE*. IEEE, 2002.

[15] 高迎霞, et al. "基于 Rogowski 线圈的电流互感器信号处理中积分算法的研究."*电测与仪表* 43.11 (2007): 1-5.

[16] Ifeachor E C, Jervis B W. Digital signal processing: a practical approach[M]. Pearson Education, 2002. P318-33, p455-62, 509-30,

[17] Hatem Bchir. Simultaneous Amplitude and Phase Optimization of IIR Filters. Master Thesis of Tampere University of Technology, 2009

[18] Hwang, Kai. Computer arithmetic: principles, architecture and design. John Wiley & Sons, Inc., 1979.

[19] Cofer, R. C., and Ben Harding. "Fixed-Point DSP Algorithm Implementation."*EE Times-Ind*

[20] Juha Yli Kaakinen. Optimization of Recursive Digital Filters for Practical Implementations. Master Thesis of Tampere University of Technology, 2009

[21] Hussain, Zahir M., Amin Z. Sadik, and Peter O'Shea. Digital signal processing: an introduction with MATLAB and applications. Springer Science & Business Media, 2011.

[22] Goldenberg, L. M., B. D. Matiushkin, and M. N. Poliak. "Digital signal processing: Handbook." *Moscow Izdatel Radio Sviaz* 1 (1985).

[23] Direct-Form I. Available: https://ccrma.stanford.edu/~jos/fp/Direct_Form_I.html

[24] Levi, E. C. "Complex-Curve Fitting." *IRE Transactions on Automatic Control*. Vol. AC-4, 1959, pp. 37–44.

[25] Dennis, J. E., Jr., and R. B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Englewood Cliffs, NJ: Prentice-Hall, 1983.

[26] invfreqz - MathWorks - MATLAB and Simulink for Technical ...(n.d.). Available: http://www.mathworks.com/help/signal/ref/invfreqz.html

[27] Al-Alaoui M A. Class of digital integrators and differentiators [J]. IET Signal Processing, 2011, 5(2): 251-260.

[28] Le Bihan, J. "Novel class of digital integrators and differentiators."*Electronics Letters* 29.11 (1993): 971-973.

[29] Hoffman, Joe D., and Steven Frankel. *Numerical methods for engineers and scientists*. CRC press, 2001.

[30] Al-Alaoui, Mohamad Adnan. "A class of second-order integrators and low-pass differentiators." *Circuits and Systems I: Fundamental Theory and* Applications, *IEEE Transactions on* 42.4 (1995): 220-223.

[31] Ngo, Nam Quoc. "A new approach for the design of wideband digital integrator and differentiator." *Circuits and Systems II: Express Briefs, IEEE Transactions on* 53.9 (2006): 936-940.

[32] Thakur, Gaurav Kumar, Ankur Gupta, and D. Upadhyay. "Optimization Algorithm Approach for Error Minimization of Digital Integrators and Differentiators." *ARTCom*. 2009.

[33] Tseng, Chien-Cheng, and Su-Ling Lee. "Design of digital integrator using interlaced sampling method and Simpson integration rule." Communications *and Information Technologies (ISCIT), 2012 International Symposium on*. IEEE, 2012.

[34] Tseng, C-C. "Digital integrator design using Simpson rule and fractional delay filter." *IEE Proceedings-Vision, Image and Signal Processing* 153.1 (2006): 79-86.

[35] Al-Alaoui, Mohamad Adnan. "Class of digital integrators and differentiators."*IET Signal Processing* 5.2 (2011): 251-260.

[36] Tseng, Chien-Cheng, and Su-Ling Lee. "Digital IIR integrator design using Richardson extrapolation and fractional delay." *Circuits and* Systems *I: Regular Papers, IEEE Transactions on* 55.8 (2008): 2300-2309.

[37] Jain, Madhu, Maneesha Gupta, and Nitin Jain. "Linear phase second order recursive digital integrators and differentiators." *Radioengineering* 21.2 (2012).

[38] Goldberg, David E., and John H. Holland. "Genetic algorithms and machine learning." *Machine learning* 3.2 (1988): 95-99.

[39] Karaboga, Nurhan, and Bahadir Cetinkaya. "Design of minimum phase digital IIR filters by using genetic algorithm." *Proceedings of the 6th Nordic signal Processing Symposium-NORSIG*. Vol. 2004. 2004.

[40] Houck, Christopher R., Jeff Joines, and Michael G. Kay. "A genetic algorithm for function optimization: a Matlab implementation." *NCSU-IE TR*95.09 (1995).

[41] Tang, Kit-Sang, et al. "Genetic algorithms and their applications." *Signal Processing Magazine, IEEE* 13.6 (1996): 22-37.

[42] Holland, John H. Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. U Michigan Press, 1975.

[43] Joines, Jeffrey A., and Christopher R. Houck. "On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with GA's." *Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on*. IEEE, 1994.

[44] Gen, Mitsuo, and Runwei Cheng. *Genetic algorithms and engineering optimization*. Vol. 7. John Wiley & Sons, 2000.

[45] Galvanic Isolation, Available: http://en.wikipedia.org/wiki/Galvanic_isolation

[46] GNU MP Manual, Available: https://gmplib.org/manual/

[47] Newton–Cotes quadrature formula. Encyclopedia of Mathematics. Available: http://www.encyclopediaofmath.org/index.php?title=Newton%E2%80%93Cotes_quadrature_formula&oldid=22841

APPENDIX: MATLAB CODES

```
1       b=[1 1]
2       a=[2 -2]
3       [z,p,k]=tf2zp(b,a)
4       if abs(p)>=1
5          p=0.9995*(real(p)/abs(p)+1i*imag(p)/abs(p));
6       end
```

Program 1. Example code for Trapezoidal Digital Integrator

```
1       b=[1 4 1]
2       a=[3 0 -3]
3       [z,p,k]=tf2zp(b,a)
4       if abs(p)>=1
5          p=0.9995*(real(p)/abs(p)+1i*imag(p)/abs(p))
6       end
```

Program 2. Example code for Simpson Digital Integrator

```
1       b=[0 1]
2       a=[1 -1]
3       [z,p,k]=tf2zp(b,a)
4       if abs(p)>=1
5          p=0.9995*(real(p)/abs(p)+1i*imag(p)/abs(p));
6       end
7        [b,a]=zp2tf(z,p,k)
```

Program 3. Example code for Rectangular Digital Integrator

```
b2=[1 1]
a2=[2 -2]
[z2,p2,k2]=tf2zp(b2,a2)
 if abs(p2)>=1
   p2=0.9995*(real(p2)/abs(p2)+1i*imag(p2)/abs(p2));
end
[b2,a2]=zp2tf(z2,p2,k2)
notchHz=0.5;%Hz
width=10;%Hz
notchAngle=360*notchHz/9600;
z2=[z2 cos(notchAngle*pi/180)+1i*sin(notchAngle*pi/180)];
z2= [z2 conj(z2(2))];
r=1-(width/9600)*pi;
p2=[p2 r*cos(notchAngle*pi/180)+1i*r*sin(notchAngle*pi/180)];
p2=[p2 conj(p2(2))];
x1=29.159320336410961
[b2,a2]=zp2tf(z2',p2',k2/x1)
fvtool(b2,a2)
```

Program 4. An example of higher order trapezoidal integrator

```
Fs=9600;
w = [0 pi*25/4800 pi*50/4800 pi*100/4800 pi*1600/4800 pi*4800/4800
mag = [0 2  1 0.5 0.03125 0.01041666 ]; %magnitudes
phase = [0/180*pi - 87/180*pi - 89.75/180*pi - 89.80/180*pi –
89.9/180*pi-     89.95/180*pi ];  % phases
wt = [1    1     10    10      10     10 ];
[b,a] = invfreqz(mag.*exp(1i*phase),w,3,3,wt,1000,0.01,'trace');
[zeroes, poles, k] = tf2zpk(b, a);
```

Program 5. Integrator design with direct design method

```
notchHz=0;
width=5;
notchAngle=360*notchHz/9600;
r=1-(width/9600)*pi;
z=[z cos(notchAngle*pi/180)+1i*sin(notchAngle*pi/180)];
```

```
6       z=[z conj(z(2))];
7       p=[p r*cos(notchAngle*pi/180)+1i*r*sin(notchAngle*pi/180)];
8       p=[p conj(p(2))];
                                result
z =   [-1     1     1]
p =  [0.999500000000000   0.998363753826255   0.998363753826255]
k =  0.016361741209689
```

Program 6. Example code for generating a 3rd order filter by pole zero placement

```
1       for j=1:N
2          notchHz=rand;%Hz
3          width=10*rand;% 0-20Hz
4          notchAngle=360*notchHz/9600;
5          r=1-(width/9600)*pi;
6          if j==1
7             zn(j) =  -1 + 0.01*rand;
8             zn(j+1) = 0.99 + 0.01*rand;
9          elseif j==N
10            zn(j) = 0.99 + 0.1*rand;
11         end
12         if j==N
13            pn(j) = 0.99 + 0.01*rand;
14         else
15            pn(j)=(r*cos(notchAngle*pi/180)+1i*r*sin(notchAn
              gle*pi/180));
16            notchHz=rand;%Hz
17            width=10*rand;% 0-20Hz
18            notchAngle=360*notchHz/9600;
19            r=1-(width/9600)*pi;
20            pn(j+1)=(r*cos(notchAngle*pi/180)-1i*r*sin(notchAn
              gle*pi/180));
21         end
22            kn = 0.0162+0.0001*rand ;
23      end
```

Program 7. Example code for generating 3rd order filter of GA optimization

```
1       notchHz=rand;
2       width=20*rand;
3       notchAngle=360*notchHz/9600;
4       r=1-(width/9600)*pi;
5       zn(1) =  -1 + 0.01*rand;
6       zn(2) = 0.99 + 0.01*rand;
7       zn(3) = 0.99 + 0.1*rand;
9       pn(1) = 0.99 + 0.01*rand;
10      pn(2) = (r*cos(notchAngle*pi/180)+1i*r*sin(notchAngle*pi/180));
11      notchHz=rand; %Hz
12      width=10*rand;
13      notchAngle=360*notchHz/9600;
14      r=1-(width/9600)*pi;
15      pn(3) = (r*cos(notchAngle*pi/180)-1i*r*sin(notchAnle*pi/180));
16      kn = 0.0162+0.0001*rand ;
17      Individualn=[zn pn kn];
```

Program 8. Example code of generating a 3rd order

```
        % 3rd order filter
1       z = [-1     1     1]
2       p = [0.999500000000000   0.998363753826255   0.998363753826255]
3       k = 0.016361741209689
4       notchHz=0;
5       width=20*rand;
6       notchAngle=360*notchHz/9600;
7       r=1-(width/9600)*pi;
9       z1=[z cos(notchAngle*pi/180)+1i*sin(notchAngle*pi/180)];
10      p1=[p r*cos(notchAngle*pi/180)+1i*r*sin(notchAngle*pi/180)];
11      k1=0.0162+0.0001*rand;
                                Result
% 4th order filter
z1 = [-1    1    1    1]
p1 = [0.999500000000 0.99836375382625 0.99836375382625 0.9946676459419]
k1 = 0.016361741209689
```

Program 9. Example code for generating a 4th order integrator by adding a notch exactly at 0 Hz

```
notchHz=rand; %put zero in the range [0,5]Hz
width=5*rand; %width of notch 0-5Hz
notchAngle=360*notchHz/9600;
r=1-(width/9600)*pi;
zn(1) =  -1 + 0.01*rand;
zn(2) = 0.99 + 0.01*rand;
zn(3) = 0.99 + 0.01*rand;
zn(4)= cos(notchAngle*pi/180)+1i*sin(notchAngle*pi/180);

pn(1) = (0.99 + 0.01*rand);
pn(4) = r*cos(notchAngle*pi/180)+1i*r*sin(notchAngle*pi/180);
notchHz=10*rand; %put zero in the range [0,10]Hz
width=20*rand; %width of notch 0-20Hz
notchAngle=360*notchHz/9600;
r=1-(width/9600)*pi;
pn(2)=r*cos(notchAngle*pi/180)+1i*r*sin(notchAngle*pi/180);
notchHz=10*rand;
width=20*rand;
notchAngle=360*notchHz/9600;
r=1-(width/9600)*pi;
pn(3)=r*cos(notchAngle*pi/180)-1i*r*sin(notchAngle*pi/180);
kn=0.0162+0.0001*rand;
Individualn=[zn pn kn];
```

Program 10. Example code for creating new filters for 4th order filter optimization