UPMC
SORBONNE UNIVERSITÉS

École doctorale Informatique, Télécommunications et Électronique (Paris)

THÈSE

pour obtenir le grade de

DOCTEUR de l'UNIVERSITÉ PIERRE ET MARIE CURIE

Spécialité Informatique

Présentée par

Anastasia VOLKOVA

Towards reliable implementation of digital filters

Directeur de thèse:	Jean-Claude BAJARD
Encadrants de thèse:	Thibault HILAIRE et Christoph LAUTER
Après avis de:	Martine CEBERIO
	David DEFOUR
	Olivier SENTIEYS

Soutenue publiquement le 25 Septembre 2017 devant le jury composé de :

M.	Jean-Claude BAJARD	Université Pierre et Marie Curie
Mme.	Martine CEBERIO	Université du Texas à El Paso
M.	David DEFOUR	Université de Perpignan
M.	Thibault HILAIRE	Université Pierre et Marie Curie
M.	Lionel LACASSAGNE	Université Pierre et Marie Curie
M.	Christoph LAUTER	Université Pierre et Marie Curie
M.	Jean-Michel MULLER	CNRS, ENS de Lyon
M.	Olivier SENTIEYS	Inria, Université de Rennes I

To my brother Konstantin

ACKNOWLEDGMENTS

My first words of gratitude go to Thibault Hilaire and Christoph Lauter, for their excellent skills as thesis advisors. At all times you have given me a perfect equilibrium of guidance and freedom, as well as equilibrium of two different approaches on doing science. Thank you for providing me with interesting challenges and pushing towards better results while giving me the freedom to choose my own way. I will always be grateful to you for becoming my friends, not only professional but above all personal. Without your beleif in me, this thesis would not be possible.

I would like to thank Thibault for his welcome ever since I have arrived to France, for all those brainstorm sessions when we were on the same wavelength, for our endless conversations on a billion topics at the same time and for always finding right encouraging words. Merci!

I am grateful to Christoph for his cunning comments (in four languages and with illustrations!) on my writing, for guiding me through the French administrative hell and for always finding time for me. Thank you and your family, Olya and Sasha, for practically accepting me as your own and for your support, especially in the last moments of redaction. Danke, mein Doktor Vater!

Of course, I would like to thank Jean-Claude Bajard for being my thesis director. Even though we have never worked together on common research projects, you have always been ready to share your experience and to give me helpful advices on research and the world of science.

My sincere gratitude goes to the reviewers of this manuscript, Martine Ceberio, David Defour and Olivier Sentieys. Thank you for the thorough feedback, invaluable remarks and advices. I am grateful for the different angles of perception of my work that you have provided me with. I would also like to thank Jean-Michel Muller and Lionel Lacassagne for agreeing to be members of the jury. In addition, I thank Jean-Michel for his interest in working with me after the thesis and supporting my application all the way to a postdoc position.

It is a pleasure for me to thank my coauthors Florent de Dinechin, Matei Istoan, Fahad Qureshi and Jarmo Takala for the most interesting exchanges on the methodologies of the Fixed-Point implementation.

My warm thanks go to the members of the PEQUAN team, for providing me with a working environment full of diversity of research topics and with a friendly atmosphere. In particular, my gratitude goes to Stef for his advises, support, friendship and, of course, patience through my

“stolen passport” adventures. I also thank Irphane for his invaluable help with the administration and for sometimes making the impossible possible.

Through the three years of my thesis, I was lucky to find myself surrounded by fellow PhD students who, through our closeness of the spirit, became dear friends. I thank Benoit and Julien for being an excellent and adventurous company during the first year of my thesis. Without Vincent, Thibaut V., Matias, Charles, Clothilde, Guillaume, Alex G., Ivan, Ana-Maria and other people whose names I forgot to mention (and whom I humbly ask for forgiveness), going through this thesis would be a lot less fun. I thank Ivan, Alex G. and Thibaut V. for teaching me the “real-life” French language (though I wish I could un-learn some phrases!). And thank you, Clothilde and Charles, for always being ready to help, even without being asked to. Thanks to scientific traveling, I was also lucky to find excellent friends in colleagues from other cities: my most warm thanks go to Valentina, Silviu and Matei.

Being a foreigner is never easy but, thanks to compatriots, longing for the homeland is less strong. I thank Katia, Tania and Denis for being my small parisian Ukraine. My gratitude also goes to Lilya, Ira, Daniel and Nastia, for their continuous friendship regardless relocations to different countries.

This thesis would be impossible without my parents and my brother, without their unconditional belief, encouragements and motivation. Thank you for not letting me to choose “easy” studies and for motivating me to do challenging ones. I will be eternally grateful for the opportunities that you have given me, or have encouraged me to take. It is your good examples that have taught me to work hard for the things I aspired to achieve.

I would also like to thank Marie-Eve, Frank and Henri for being a wonderful family-in-law, for doing everything to comfort me during the redaction and for always making me feel at home.

Finally, from all my heart I thank Alex for being my support, my pillar of calmness and my source of strength in the most challenging moments. I am deeply grateful for your understanding, practical advices and for encouraging me when the tasks seemed arduous and insurmountable.

Table of Contents

NOTATION

Acronyms

SIF	Specialized Implicit Form
LTI	Linear Time-Invariant
IIR	Infinite Impulse Response
FIR	Finite Impulse Response
SISO	SIngle Input Single Output
MIMO	Multiple Input Multiple Output
LWDF	Lattice Wave Digital Filter
FP	Floating-Point
FxP	Fixed-Point
SOPC	Sum-of-Products by Constants

Conventions

Scalars / **Vectors** / **Matrices:** apart from exceptions, throughout this document scalar quantities are in lowercase (e.g. x), vectors are in lowercase boldface (e.g. $\boldsymbol{x}$), matrices are in uppercase boldface (e.g. $\boldsymbol{X}$).
Exception: we denote an impulse response matrix of a MIMO filter as $\mathbb{h}(k)$.

Absolute Values and Inequalities: apart from exceptions, all matrix absolute values and inequalities are applied element-by-element. For example, $|\boldsymbol{A}| \leq |\boldsymbol{B}|$ denotes $\left|\boldsymbol{A}_{ij}\right| \leq \left|\boldsymbol{B}_{ij}\right|$, $\forall i,j$. In addition, $\boldsymbol{A} < \varepsilon$ denotes $\boldsymbol{A}_{ij} < \varepsilon$, $\forall i,j$. Norms, such as Frobenius norm, notated $\|\boldsymbol{A}\|_F$, stay of course norms on matrices and are not to be understood element-by-element.

Zero and Identity Matrices: $\boldsymbol{0}$ denotes a matrix of zeros, the size of which is usually deduced from the context; $\boldsymbol{I}_n$ denotes an identity matrix of size $n \times n$.

Intervals: a real interval $[x]$ is defined with its lower and upper bounds $[x] := [\underline{x}, \overline{x}]$. An interval matrix is denoted by $[\boldsymbol{M}] := [\underline{\boldsymbol{M}}, \overline{\boldsymbol{M}}]$, where each element $[\boldsymbol{M}_{ij}]$ is an interval $[\boldsymbol{M}_{ij}] = [\underline{\boldsymbol{M}}_{ij}, \overline{\boldsymbol{M}}_{ij}]$.

INTRODUCTION

If you have ever watched television, taken a plane or listened to an MP3 then you have taken advantage of digital signal processing. You might have noticed that sometimes a small interference in a TV signal occurs, or that the voice of your interlocutor is distorted over the phone. This may mean that something went wrong in the way the data was processed in the device. Usually you do not even mind those occasional failures. However, when you are in an airplane you do mind if any failure happens in a critical system and the airplane crashes. When people launch a satellite into orbit, they try to ensure that all the signals are correctly processed and no failure ever happens; in the end you cannot just come and fix it!

In this thesis we consider signal processing and control algorithms for applications where guarantee and reliability are cornerstones.

The advancement of digital computers during the 1960s paved the way for many electronic devices to be emulated with digital computers where all information is usually stored in binary format, i.e., as a sequence of ones and zeros. For example, music used to be recorded and distributed in analog form such as magnetic tapes until the 1980s, when CD players have made digital recording of music common.

There are two very important advantages to digital signals. First, they can be reproduced exactly; all you have to do is to make sure to thoroughly copy the sequence of ones and zeros. Second, digital signals can be easily manipulated: once we have a discrete-time representation of a real-world signal, we are left with a sequence of numbers, e.g., temperature and audio signals both boil down to just sequences.

So what can we do with signals? We can filter out unwanted parts, combine several signals into one, enhance certain parts and discard others, etc. A transformation of a digital signal in order to amplify or attenuate some of its properties is called a Digital Filter. Any given digital filter may be computed with numerous different algorithms that compute the desired output in different ways. Some of the algorithms are faster and expensive, others are slower but cheaper, etc. The process of translating an algorithm into a computer program or a circuit is called implementation.

In this thesis we will address the question of implementation of digital filter algorithms.

Despite its advantages, digitalization comes at a price. In digital computers values are required to have a finite number of digits in their representation, i.e. finite precision. During manipulations with values in finite precision, errors often occur. Suppose all numbers that you can represent on a computer must fit into two decimal digits and you need to square 1.7. You know that $1.7^2 = 2.89$ but all that you can represent is two digits. The two-digit result that you return (2.8 or 2.9) will depend on the computer arithmetic that you use but anyway, an error between the exact and computed values inevitably occurs. This error inherently depends on the order in which computations are done. Usually, the more precision you use for your computations, the smaller the error becomes.

Thus, when we evaluate a Digital Filter with some algorithm, varying the parameters of the arithmetic and the quantity of resources (for instance precision) yields different results, i.e. different errors. The race towards smaller, faster and more energy-efficient signal processing devices dictates very strict requirements for digital filters. For implementations dedicated to embedded systems, precision is almost in direct opposition to these requirements: for example, the more digits we need to operate with, the slower the algorithm works. The goal is to find the maximum *acceptable* degradation of precision. For reliable implementations, the task is even more complicated: the error due to the finite-precision implementation must be rigorously accounted for and the implemented algorithm must always produce an output that is within this error margin from the ideal result.

For a given algorithm there exist many possible implementation settings that can be changed. The fact that each Digital Filter may be implemented with different algorithms significantly enlarges the design space. Very quickly the implementation process becomes time-consuming. And in order to produce a reliable implementation designers must analyze the effects of the arithmetic and precision choices for each algorithm. Doing this manually is not always feasible and requires a deep expertise. Thus, we come to the goal of our work.

In this thesis we work on the numerically reliable and automated implementation of digital filters.

Our focus is mainly on the automatic and rigorous analysis of finite-precision effects on an arbitrary filter algorithm from the computer arithmetic point of view. In this work you will see how to reliably determine the worst possible error induced by a finite-precision implementation. We will also show how to determine the trade-off between the precision and the output error which will help us choose the best (in certain sense) implementation. Our approach will be applicable to any possible algorithm in the broad class of Linear Time Invariant filters.

To achieve our goals, we will intertwine techniques from the areas of both digital signal processing and computer arithmetic.

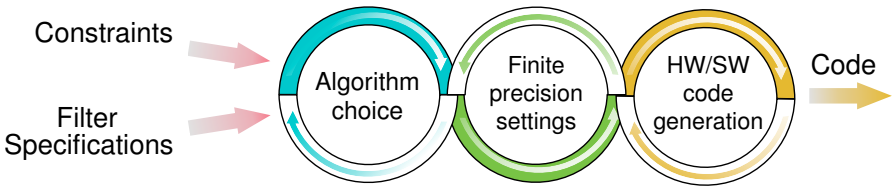


Figure Y: Iterative nature of the filter implementation flow

Implementation of digital filters

A digital filter should be seen as a set of specifications that describe the desired behavior. Usually, digital filters are designed with respect to some frequency specifications, for example audio filters that filter out high and suppress low frequencies. Mathematically, these specifications can be described with the help of a rational function, called transfer function.

A filter algorithm should be seen as a set of instructions the digital filter can be evaluated with. For every digital filter there exist numerous different evaluation algorithms; we will call them structures. Thus we come to the following: a digital filter is a set of requirements of *what* should be done, and an algorithm shows *how* it is done.

By the *implementation* of a linear digital filter we understand a transformation that goes from the transfer function (or just requirements on its behavior), via a filter algorithm to the software or hardware code.

Every implementation is based on a choice of the filter structure. All structures are based on elementary arithmetic operations, additions and multiplications by constants. These constants, called coefficients, are the parameters of the structure: changing the coefficients changes the filter that is evaluated with the structure. Thus, given a structure, the designer must determine the set of coefficients of this structure that *realize* the transfer function that is about to be implemented. This process is called the realization of a filter with a given structure.

In different structures the number and order of arithmetic operations are different, as well as the number of coefficients. The designer must choose a structure that best suits the application constraints (for example, the implementation is required to be parallelizable) and realize the filter with this structure.

Once the realization is obtained, the designer must define the parameters of its finite-precision counterpart. The quality of a finite-precision implementation is measured with by accuracy, i.e. how many digits in the output of the signal processing system are correct. For example, for hardware targets the designer needs to make several architectural choices that influence the *accuracy* of the algorithm. These choices include: (i) the choice of the arithmetic,

i.e. the rules with which all arithmetic operations are conducted; (ii) the precision with which the values will be stored. Different structures have different numerical properties. For example, to achieve the same level of accuracy, some algorithms require more resources for the intermediate computations and the others require less. Accuracy is not the only requirement, we usually seek to satisfy certain constraints on the speed, area, power consumption, etc.

If the implementation result is not satisfactory, another structure is chosen and the process repeats. The implementation of a filter becomes a long chain of trial and error when the designer moves back and forth between different implementation stages. The iterative nature of this process is illustrated in Figure Y.

The task is even more complicated when we need to deliver a *reliable implementation*. A reliable implementation is a couple of an error-bound and the software/hardware code that is guaranteed to always produce an output that differs from the ideal not more than by this error. In other words, reliable implementations are required to always deliver a given accuracy.

In signal processing, reliability and the accuracy of the filter are often determined using simulations. With this approach, the implemented system is compared to some reference implementation via extensive testing and the maximum error that occurred is taken as an error bound. While being easy to adapt to any structure, this approach is often very slow. Moreover, simulations cannot guarantee that all, even rare, cases have been checked. Another approach is to investigate the numerical properties of the filter structure and deduce the statistical distribution of errors. However, such analysis is structure-specific and, most importantly, does not provide any bounds on the error.

The reader may wonder why the finite-precision effects on filters have not been studied using the computer arithmetic approach. The reason is simple: historically the two scientific domains have only had a few interfaces and little in common. In this thesis, we will try to fill the gap between them and benefit from the best approaches of both domains.

Towards automatic filter code generation

There exist software tools that help designers with filter implementation, such as Matlab and Simulink. While significantly simplifying the implementation process, they do not make any decisions for the designers, leaving them with the burden of the proof of reliability.

To accelerate the implementation and automatize some designer choices, an automatic filter code generator may be used. There already exist some frameworks for semi- or fully automatic filter implementation [1–3]. However, the major drawback of the existing tools is that they are application- or structure-specific and do not give any guarantee that the implementation is reliable in the sense that we defined.

To remedy the disadvantages of existing solutions and to provide an *automatic* and *reliable* way to implement linear filters, an automatic code generator has been designed [4, 5]. This generator targets implementation in Fixed-Point arithmetic, which is probably the most widely used arithmetic for embedded systems. It aims at implementing linear filters, a broad class of algorithms used in both signal processing and control.

Hilaire proposed the outline of the generator in [4, 5] and Lopez continued in [6]. The code generator is based on the idea of unification: all filter structures can be described with a unified analytical representation called Specialized Implicit Form (SIF). This representation is based on writing explicitly all computations that define a filter structure in an analytical form. SIF represents at the same time a “language” for the description of any linear digital filter and a tool for their analysis. It is enough to detail the implementation process for filters in SIF and it may be applied to any structure. The workflow of the code generator follows the classic implementation flow from Figure Y with the exception that the structures are represented with SIF.

While the major outline of the code generator was developed before this thesis, the essential part is missing: there is still no approach that provides rigorous error analysis of finite-precision implementations.

The goal of this thesis is twofold: first we aim at providing a new methodology for the error analysis when linear filter algorithms are investigated from a computer arithmetic aspect. Our goal, in particular, is to analyze recursive filters. At the same time, using the SIF, we incorporate our approach into the automatic filter code generator. To do so we will also need to extend some of the functionalities of the existing code generator.

In this manuscript we are going to address the following questions:

– The SIF was conceived to be able to describe any linear digital filter structure. However, having a language for the description of digital filters and applying it are different things. Several structures have already been translated to the SIF description. However, the question is how can we automatically translate an arbitrary structure to the SIF representation? Often filter structures are described in graphical representation, such as data-flow graphs. This representation is constantly used in practical engineering implementations due to its resemblance to digital circuits. We first investigate [7] how data-flows that describe a particular structure, Lattice Wave digital filter, can be expressed in SIF. Then, we generalize [8] our approach to any data-flow describing a linear digital filter.

– When the designers manipulate a filter realization they should be able at any moment to recover the transfer function that corresponds to this realization. For example, when we slightly change the coefficients of the realization and must verify that this algorithm

still evaluates the filter we want. Thus, another question arises: how can we compute the transfer function corresponding to an arbitrary filter in SIF representation? In general, exact computation of a transfer function corresponding to a filter is not practically feasible. We will show [9] how to compute an approximation of the transfer function and give a rigorous bound on the approximation error.

– How can we build a reliable Fixed-Point algorithm with a guarantee on its accuracy? In Fixed-Point Arithmetic numbers have fixed-sized integer and fractional parts, therefore this question needs to be answered in two steps:

  – First, what are the ranges of all variables involved in the evaluation of filter algorithm? In other words, given the range of possible input signals we need to determine the range of all possible intermediate and output values that may occur in the filter algorithm. One of the approaches is based on the Worst-Case Peak Gain (WCPG) measure [10]. However, this measure cannot be computed exactly but only approximately without any guarantee on the approximation. We will reinforce this approach by providing algorithms [11, 12] for the reliable evaluation of the WCPG measure with arbitrary accuracy. They will allow us to compute a rigorous bound on the dynamic range of all variables involved in the evaluation of the exact filters, and consequently how many digits we need to dedicate to the integer parts when stocking these variables.

  – Second, given wordlength constraints for storage of all variables, what is the best position of the binary point? Here we need to achieve two goals: the binary point must respect the size of the integer parts deduced from the dynamic ranges, and at the same time we aim at having the longest fractional part possible to obtain more precision. The problem is that the ranges computed with the WCPG measure correspond to an ideal filter but not an implemented filter. In implemented filters, the fractional parts have finite lengths and arithmetic operations are not exact. These effects lead to errors that may propagate up to the integer parts and yield to a non-representable value (overflow). Again, using the WCPG we will show how to rigorously evaluate the non-linear propagation of errors due to finite-precision implementation of recursive filters. Then, we take this propagation into account in a general algorithm [13, 14] which, given wordlength constraints, tries to find the best position of binary point. At the same time, we will also provide a tight bound on the worst-case error of the implementation. Again, we will use computer arithmetic techniques to rigorously prove all error bounds.

– For reliable implementations, it is necessary to have a guarantee on the error of the implementation in the time domain. However, this guarantee is not worth anything if the eventual signal processing system does not satisfy the filter specifications in terms of its frequency response. Thus, the question is: does the implemented filter satisfy the desired frequency specifications? We will address the possibility of such a certification and show how this verification can be done for any linear filter algorithm. With our approach [9], the problem boils down to a verification of the positivity of a polynomial on some domain. Using various computer arithmetic techniques, we provide a fast implementation of this check that is rigorous in the sense that it never gives false positive answers.

– How to build a hardware circuit that guarantees an accurate output without wasting resources? We consider generation of such architectures on Field Programmable Gate Arrays (FPGAs). We aim at providing designs which have their implementation error not larger than the weight of the last bit of the output. Usually, this is achieved by using extended precision for internal computations, i.e. adding guard bits. We collaborate with the FloPoCo [15] project and provide a hardware code generator that uses just enough precision for internal computations (i.e. no resource wasted) to guarantee that level of accuracy for the output [16].

A complete methodology for the automatic and reliable Fixed-Point implementation of linear digital filters will also permit the filter designer to cover a large design space in search of the best implementation under various constraints. Using new functionalities of SIF we will be able to produce a reliable Fixed-Point implementation upon any filter algorithm. What is important, is that the kernel algorithms that we provide are rigorous and are very well-suited to be used in extensive optimization routines.

Also, in a collaboration with the Tampere University of Technology, Finland, we applied our techniques to the analysis of a particular hardware architecture implementing Fast Fourier Transform [17]. Even though this contribution concerns reliable implementation of signal processing algorithms, in reality it is situated in a context too distant to be presented in this thesis.

Overall, this thesis has led to the following publications: [7–9, 11–14, 17].

Organization of the document

This thesis follows the order of the questions mentioned above. This order is natural in terms of the workflow of the code generator and is almost chronological. The document is divided into four parts, each subdivided into chapters.

Part I: Chapters 1 and 2 give an overview of the main notions within the realm of digital filters and computer arithmetic as well as a general picture of typical problems that occur during the finite-precision implementation of digital filters. Finally, Chapter 3 describes the state of the art of the code generation tool before this thesis.

Part II: Chapter 4 is more technical than theoretical and concerns the conversion of Lattice Wave Digital filters into the SIF representation. Chapter 5 incorporates the algorithm for the conversion from data-flow graph into SIF and our approach for the approximation of the transfer function corresponding to filters in the SIF.

Part III Here we present the kernel algorithms for the Fixed-Point Implementation of linear filters. Chapter 6 concerns the computation of the dynamic range of all variables involved in the filter evaluation. We present rigorous error analysis of the computation of the Worst-Case Peak Gain measure in multiple precision arithmetic. Chapter 7 contains the general approach for the determination of reliable Fixed-Point formats. The problem of verification of an implemented filter against frequency specifications is addressed in Chapter 8.

Part IV This final part is comprised only of Chapter 9. In this last contribution we make our first steps towards reliable implementation of any linear filter on FPGAs by first proposing implementation of the Direct Form I structure.
The reader will find the conclusion of this thesis along with the perspectives in the last Chapter.

PART I

TECHNICAL PRE-REQUISITES

CHAPTER 1

DIGITAL FILTERS

In this Chapter we recall to the reader the concept of real-valued discrete-time signals and Linear Time-Invariant systems, also called filters in our context. We also give a short overview of properties of filters. We are particularly interested in recursive filters and their properties. In practice, these systems can be evaluated in many different ways that depend on the application, performance requirements, etc. These filter computational algorithms are called structures and we present a few of them.

We refer the reader to classic digital signal processing texts like [18, 19] for a systematic presentation and details.

1.1 Discrete-time signals

Discrete-time real signals are represented mathematically as sequences that can be formally written as

$$u = \{u(k)\}, \quad \forall k \in \mathbb{Z}. \tag{1.1}$$

The integer index k represents dimensionless time[1], i.e. gives a chronological order to the elements of sequence. This comes from the fact that the discrete-time signal $u(k)$ is usually obtained by constant sampling of a continuous signal $\check{u}(t)$: $u(k) = \check{u}(kT_s)$, where T_s is the sampling period.

Remark 1.1. *In the manuscript we will abuse the notation and even though $u(k)$ denotes the k^{th} element of sequence u, we will denote the whole sequence by $u(k)$ when needed.*

The following elementary operations on signals will be useful for this work:

Shift A sequence $u(k)$ shifted by an integer K is $y(k) = u(k-K)$ for all k. If K is positive, then the signal is said to be *delayed* by K samples.

Scaling A sequence $u(k)$ scaled by $\alpha \in \mathbb{R}$ is $y(k) = \alpha u(k)$.

Sum A sum of two sequences is their term-by-term sum $y(k) = u(k) + z(k)$ for all k.

[1] Throughout the text we reserve the independent variable k to denote an integer index of a discrete-time signal.

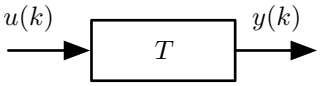


Figure 1.1: Representation of a discrete-time system.

The simplest discrete-time signal is probably the impulse signal, denoted δ and defined as

$$\forall k, \quad \delta(k) = \begin{cases} 1, & k = 0 \\ 0, & k \neq 0 \end{cases} \tag{1.2}$$

This signal is central to describing digital systems. Any signal can be expressed as a linear combination of suitably weighted and shifted impulses: $u(k) = \sum_{l=-\infty}^{\infty} \delta(k-l)u(l)$.

1.2 Discrete-time systems

A discrete-time Single-Input Single-Output (SISO) system is defined as a transformation or operator that maps the input sequence $u(k)$ to an output sequence $y(k)$:

$$y(k) = T\{u(k)\}. \tag{1.3}$$

When for all k the input signal $u(k)$ and output signal $y(k)$ are scalar (i.e. $u(k), y(k) \in \mathbb{R}$), then the system is called Single Input Single Output (SISO). When for all k the signals $\boldsymbol{u}(k) \in \mathbb{R}^q, \boldsymbol{y}(k) \in \mathbb{R}^p$ are vectors, then the system is said to be Multiple Input Multiple Output (MIMO).

In this thesis we are interested in studying the Linear Time-Invariant (LTI) discrete-time systems which we are going to define just below. A system of this type is referred to as a *filter*.

1.2.1 LTI systems

Let $u_1(k)$ and $u_2(k)$ be two signals given as inputs to a discrete-time system $\mathcal{H}$. Then the system $\mathcal{H}$ is called a Linear Time-Invariant system if it satisfies

- the Linearity property:

$$\mathcal{H}\{\alpha u_1(k) + \beta u_2(k)\} = \alpha\mathcal{H}\{u_1(k)\} + \beta\mathcal{H}\{u_2(k)\} \tag{1.4}$$

  for any $\alpha, \beta \in \mathbb{R}$, and

- the Time-Invariance property:

$$y(k) = \mathcal{H}\{u(k)\} \Leftrightarrow \mathcal{H}\{u(k-K)\} = y(k-K), \tag{1.5}$$

which basically means that if the input signal is delayed by K samples, the output is also delayed by K samples.

It turns out that an LTI system can be characterized by its *impulse response*, i.e. by the sequence $h(k) = \mathcal{H}\{\delta(k)\}$. Indeed, using the linearity and time-invariance, we obtain that

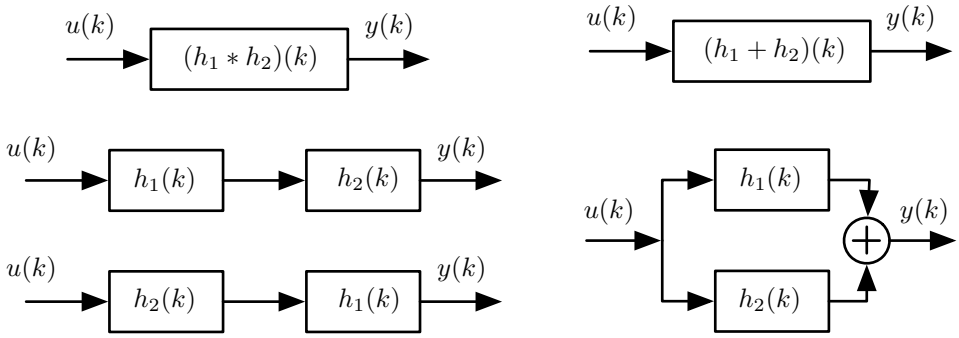
$$y(k) = \mathcal{H}\{u(k)\} = \mathcal{H}\left\{\sum_{l=-\infty}^{\infty} u(l)\delta(k-l)\right\} \tag{1.6}$$

$$= \sum_{l=-\infty}^{\infty} u(l)\mathcal{H}\{\delta(k-l)\} \tag{1.7}$$

$$= \sum_{l=-\infty}^{\infty} u(l)h(k-l) \tag{1.8}$$

$$=: (u * h)(k), \tag{1.9}$$

where " $*$ " is the convolution operator defined by (1.9). The convolution operator has a lot of useful properties, for a full list of which we refer the reader to [18]. To summarize, the associativity, distributivity over addition and commutativity of the convolution operator permit us to do various combinations of LTI systems, see Figure 1.2.



(a) Cascades

(b) Parallel combination

Figure 1.2: Equivalent operations over SISO filters.

There exist two classes of LTI systems with respect to the behavior of impulse response:

- **Finite Impulse Response** (FIR) filters that have an impulse response with finite support, i.e. there exists a finite K such that for all $k > K$ the impulse response $h(k)$ is zero;
- **Infinite Impulse Response** (IIR) filters that have an impulse response with infinite support, i.e. they do not become exactly zero past a certain point but continue indefinitely.

Stability: A system is called *Bounded-Input Bounded-Output (BIBO) stable* iff its output is bounded for all bounded input sequences. This is a very natural requirement for a filter, since it states that the output will not "blow up" when the input is bounded. LTI filters are stable iff the impulse response is absolutely summable, i.e. when $\|h\|_{\ell_1}$ exists. More practical stability[2] criteria are defined using the $\mathcal{Z}$-transform, see Section 1.3.

Causality: A system is called *causal* if its output does not depend on any "future" inputs, i.e. the output $y(k_0)$ depends only on the input samples $u(k)$ for $k \leq k_0$. That implies that for causality of LTI systems the condition is $h(k) = 0$ for all $k < 0$; see [18] for details.

In this manuscript we consider only **stable causal** LTI filters.

1.2.2 Linear Constant-Coefficient Difference Equations

A convenient way of describing LTI systems is using constant-coefficient difference equations (CCDE). CCDEs define the relationship in time domain between an input signal $u(k)$ and the output $y(k)$ as

$$\sum_{i=0}^{N_1} a_i y(k-i) = \sum_{i=0}^{N_2} b_i u(k-i), \quad N_2 \geq N_2. \tag{1.10}$$

Throughout the manuscript we restrict ourselves to the case of real coefficients a_k and b_k. We can normalize the coefficients in order to have $a_0 = 1$, so that the above equation is rearranged as

$$y(k) = \sum_{i=0}^{N_2} b_i u(k-i) - \sum_{i=1}^{N_1} a_i y(k-i). \tag{1.11}$$

CCDE (1.11) describes an IIR linear system. If $\forall i > 0, a_i = 0$, it describes a FIR system. The difference may be interpreted in the following way: in FIR filters the output is computed solely out of the input signal, while the output of the IIR filters also depends on the previous N_2 outputs.

Everywhere in the thesis we consider filters to have *zero initial conditions*, i.e. in response to the zero signal, a zero signal is generated.

1.3 $\mathcal{Z}$-transform

Discrete-time systems are often characterized by their oscillatory behavior which is analyzed in the frequency domain. Usually to obtain the frequency-domain representation of a signal, the

[2] Everywhere in the manuscript under "stable" we will understand "BIBO-stable" filters.

Fourier Transform [20] is used:

$$U(e^{j\omega}) = \sum_{k=-\infty}^{\infty} u(k)e^{-j\omega n}, \quad \omega \in [0, 2\pi). \tag{1.12}$$

However, in practice, the $\mathcal{Z}-$transform of a discrete-time signals is used:

$$U(z) = \mathcal{Z}\{u\}(z) = \sum_{k=-\infty}^{\infty} u(k)z^{-k}, \quad z \in \mathbb{C}. \tag{1.13}$$

Remark 1.2. *Just like in the case of the signals, we will abuse the notation and express the $\mathcal{Z}$-transform of a signal u not by $\mathcal{Z}\{u\}(z)$ but by $\mathcal{Z}\{u(k)\}$.*

The $\mathcal{Z}-$transform represents a counterpart of the Laplace transform [20], which is, in turn, used for continuous-time signals[3]. Indeed, $U(z)$ evaluated over the unit circle, i.e. for $z = e^{j\omega}$, is the Fourier Transform.

Two important properties of the $\mathcal{Z}-$transform are:

- Linearity: $\mathcal{Z}\{\alpha u_1(k) + \beta u_2(k)\} = \alpha U_1(z) + \beta U_2(z)$ and
- Time-Shift: $\mathcal{Z}\{u(k-K)\} = z^{-K}\mathcal{Z}\{u(k)\}$.

We can also use partial fraction expansions to compute the inverse $\mathcal{Z}$-transform [21].
Then, applying the $\mathcal{Z}-$transform to the CCDE representation (1.11) with the above two properties, we obtain

$$Y(z) = \sum_{i=0}^{N_2} b_i z^{-i} U(z) + \sum_{i=1}^{N_1} a_i z^{-i} Y(z) \tag{1.14}$$

and hence

$$Y(z) = H(z)U(z), \tag{1.15}$$

where $H(z)$ is called the *transfer function* of the filter (1.11). Obviously, in the case of FIR filters, i.e. when all $a_i = 0$, $H(z)$ is just a polynomial.
In case of IIR filters $H(z)$ is a rational function:

$$H(z) = \frac{\sum_{i=0}^{n} b_i z^{-i}}{1 + \sum_{i=1}^{n} a_i z^{-i}}, \quad z \in \mathbb{C}. \tag{1.16}$$

Here n denotes the degree of the filter and $n = \max(N_1, N_2)$. If $N_1 \neq N_2$, we complete the polynomial of smaller degree by zeros, i.e. $b_i = 0$ for $N_2 < i \leq n$ and $a_i = 0$ for $N_1 < i \leq n$.

The general relationship between the time-domain representation and frequency-domain representation is illustrated on Figure 1.3.

[3] The Laplace transform can be seen as a generalization of the Fourier Transform for continuous-time signals

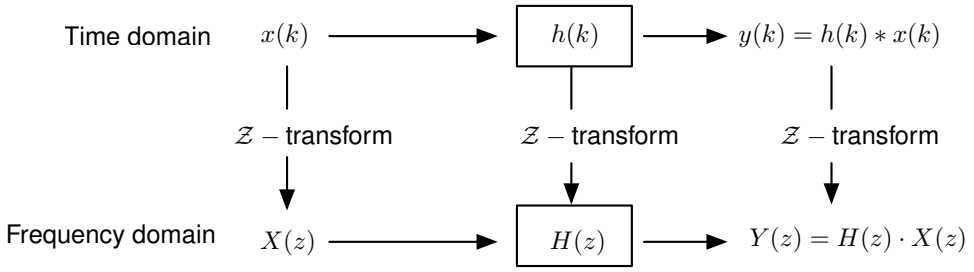


Figure 1.3: Time and frequency domains.

Zeros, poles and stability: The roots of the numerator of $H(z)$ are called the system's *zeros* and the roots of the denominator are called the system's *poles*. If a root of the denominator has multiplicity 1, it is called a simple pole, otherwise it is a multiple pole. It can be shown [18], that for an LTI system to be causal and stable, its poles must lie strictly inside the unit circle. We refer the reader to [18] for a proof.

If there are one or more simple poles exactly on the unit circle, the filter is called marginally stable. For most practical filters, all the poles are designed to lie inside the unit circle. However, for oscillators, the poles are placed on the unit circle on purpose.

1.4 Design of IIR filters

1.4.1 Frequency specifications

The dynamics of LTI systems are characterized by the system's frequency response. It is a measure of magnitude and phase of the output as a function of frequency in comparison to the input. The frequency response of a filter is the value of its transfer function evaluated on the unit circle:

$$H(e^{j\omega}) = \left|H(e^{j\omega})\right| e^{j\measuredangle H(e^{j\omega})}, \quad \omega \in [0, 2\pi), \tag{1.17}$$

where $\left|H(e^{j\omega})\right|$ is the filter's magnitude response and $\measuredangle H(e^{j\omega})$ is the phase response.

Classically, filters are designed with respect to the magnitude response specifications in the frequency domain, in order to amplify (or preserve) signals in some frequency bands, and attenuate them in other bands.

A filter specification is composed of several *passbands* (i.e. the gain of the filter for these frequencies should be between given bounds) and *stopbands* (the gain should be lower than a given bound), formally described as:

$$\underline{\beta} \leq \left|H(e^{j\omega})\right| \leq \overline{\beta}, \quad \forall \omega \in [\omega_1, \omega_2] \subseteq [0, 2\pi]. \tag{1.18}$$

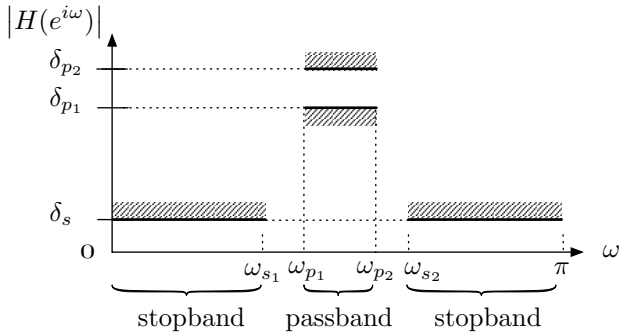


Figure 1.4: A passband filter specification.

For instance, Figure 1.4 illustrates a passband filter specification which can be described as the following system of inequalities:

$$\begin{cases} |H(e^{j\omega})| \leq \delta_s, & \forall \omega \in [0, \omega_{s_1}] \quad \text{(stopband)} \\ \delta_{p_1} \leq |H(e^{j\omega})| \leq \delta_{p_2}, & \forall \omega \in [\omega_{p_1}, \omega_{p_2}] \quad \text{(passband)} \\ |H(e^{j\omega})| \leq \delta_s, & \forall \omega \in [\omega_{s_2}, \pi] \quad \text{(stopband)} \end{cases}.$$

Remark that no conditions are applied to the magnitude response in the intermediate frequency intervals $(\omega_{s_1}, \omega_{p_1})$ and $(\omega_{p_2}, \omega_{s_2})$. These bands of frequencies are called *transition bands*. Very often filter designers aim at having small transition bands.

Here ω denotes the normalized frequency which is a unit of measurement of frequency equivalent to cycles per sample. In practical applications filter designers prefer to use frequencies $f = \frac{\omega}{2\pi} F_s$ that are measured in Hertz, where F_s is the sampling rate ($F_s = \frac{1}{T_s}$). The bounds on the magnitude response are often given in decibels (dB), x dB means $10^{\frac{x}{20}}$[4].

Due to the Nyquist-Shannon theorem [18], for causal filters with a real input/output relationship it is only necessary to specify the frequency up to $\frac{F_s}{2}$ or the normalized frequency up to π instead of 2π.

In terms of the phase response, filter designers usually seek to have a linear phase which means that the phase response of the filter is a linear function of frequency. A linear phase filter will preserve the waveshape of the signal (to the extent that it is possible given that some frequencies will be changed in amplitude by the filter). However, phase characteristics are out of the scope of this work.

[4] Deci means 10 and logically we should have $\frac{x}{10}$, however since the power measures are proportional to to squares of filed measures, there is a confusing 2 that appears and results in $\frac{x}{20}$ [22]

1.4.2 Design methods

Given frequency specifications can be satisfied by both FIR and IIR filters. IIR filters can generally approximate frequency response specification using a lower order than the FIR filters. It is in particular beneficial for filters that have short transition bands: by placing poles of the filter near the unit circle we may obtain a sharp peak (the magnitude of the transfer function becomes very large in the neighborhood of a poles).

In this thesis we only consider the implementation of the IIR filters, as a more general case.

The most common IIR design technique involves designing first an analog filter (a prototype) and then transforming it to a digital filter. This approach is popular because of the vast literature available on the subject of analog filters. Usually, the method of choice is the bilinear transformation [18] (based on the mapping of the analog plane into the digital plane) and impulse-invariant methods (based on the sampling of the continuous time impulse response) [23]. For piecewise-constant frequency band specifications, numerical approximation methods can be used. For instance, the Remez exchange algorithm can be applied [24]. However, this method can sometimes fail [25].

1.4.3 A key band specification example

To unify the demonstration of our algorithms, further in this thesis we will use the following key example of frequency specifications of a simple but realistic enough lowpass filter:

- sampling frequency $F_s = 48$kHz
- passband up to 9.6kHz with amplitude in the interval from 0 to -1dB
- stopband starting with 12kHz with amplitude bounded by -80dB

There exist infinite number of transfer functions that satisfy these specifications. For instance, designing the transfer function using Elliptic method by Matlab yields an 8^{th} order IIR filter[5] with double precision coefficients[6] that are given in Table 1.1.

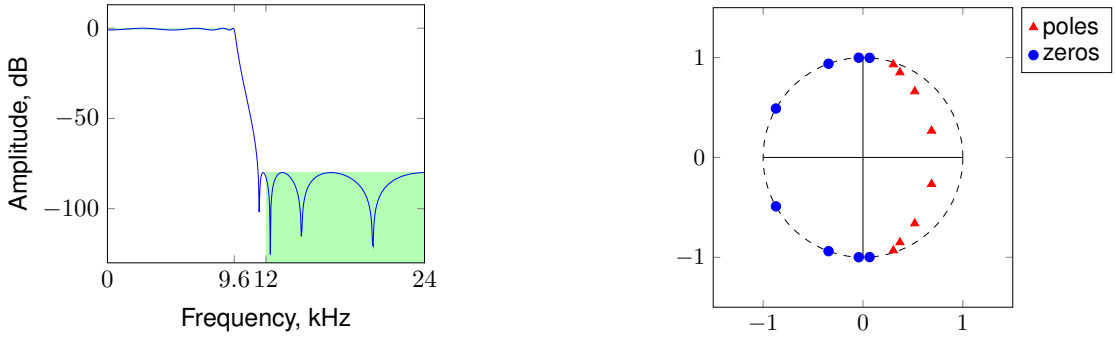
Figure 1.5a illustrates the magnitude response, and Figure 1.5b shows the position of poles and zeros of this transfer function. We see that this filter has its poles very close to the unit circle, the maximum distance from the unit circle is around 0.0199.

[5] Filter order determined with Matlab function `ellipord`

[6] Attention: Matlab does not provide any information on the *accuracy* of the obtained coefficients.

b_i , $i = 0,...8$	a_i, $i = 1,...8$
$561579990219285 \cdot 2^{-57}$	
$2680202476681091 \cdot 2^{-58}$	$-8466075925097353 \cdot 2^{-51}$
$2851083336434883 \cdot 2^{-57}$	$144214293209393 \cdot 2^{-44}$
$7945852418150609 \cdot 2^{-58}$	$-834038461885399 \cdot 2^{-46}$
$2285687050263807 \cdot 2^{-56}$	$3470975327611749 \cdot 2^{-48}$
$7945852418150609 \cdot 2^{-58}$	$-5233992444458969 \cdot 2^{-49}$
$5702166672869765 \cdot 2^{-58}$	$2801619744602567 \cdot 2^{-49}$
$2680202476681091 \cdot 2^{-58}$	$-3922384851870119 \cdot 2^{-51}$
$561579990219285 \cdot 2^{-57}$	$5713945412302955 \cdot 2^{-54}$

Table 1.1: Coefficients of the elliptic filter satisfying the key specifications.



(a) Magnitude response

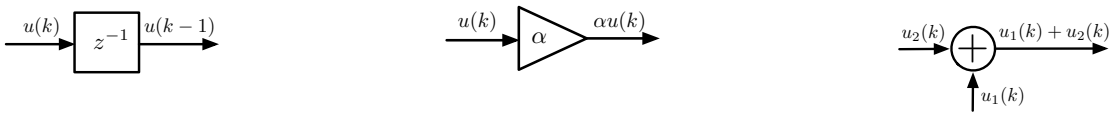
(b) Poles and zeros

Figure 1.5: The 8th order elliptic transfer function corresponding to the key specifications.

1.5 Filter Structures

In practice, an IIR filter can be evaluated with many different algorithms, we will further call them "*structures*". Each IIR structure is parametrized to evaluate any IIR filter. We will further refer to the parameters of each structure as to its *coefficients*. Different structures have different number of coefficients. The process of computation of the structure coefficients out of a filter (e.g. its transfer function) is called the "realization of a filter with the structure". Hence, we will call a particular filter evaluated with a structure a "*realization*".

A convenient way of reflecting different computational flows is using the data-flow diagrams, modeled with block diagrams. The basic building blocks for the description of LTI systems are multipliers, adders and unit delays; symbols for them are shown on Figure 1.6.



(a) delay (b) multiplication (c) addition

Figure 1.6: Basic blocks in linear block-diagrams

1.5.1 Direct Forms

A straightforward way to describe the evaluation of the transfer function (1.16) is shown in Figure 1.7a and is called Direct Form I (DFI) structure. It has the same coefficients as the filter's transfer function $H(z)$ and corresponds to the CCDE 1.11. By rearranging the structure we can obtain Direct Form II (DFII) as on Figure 1.7b. Through transposition of the data-flow diagrams [18] we can obtain Direct Form I and II transposed (DFIt and DFIIt) as shown on Figures 1.8a and 1.8b.

These structures are straightforward to design and implement since they use the coefficients of the transfer function. For example, by simply replacing coefficients in 1.7a with the coefficients of the elliptic transfer function from Table 1.1 we obtain a *realization* of the key lowpass filter with the DFI structure.

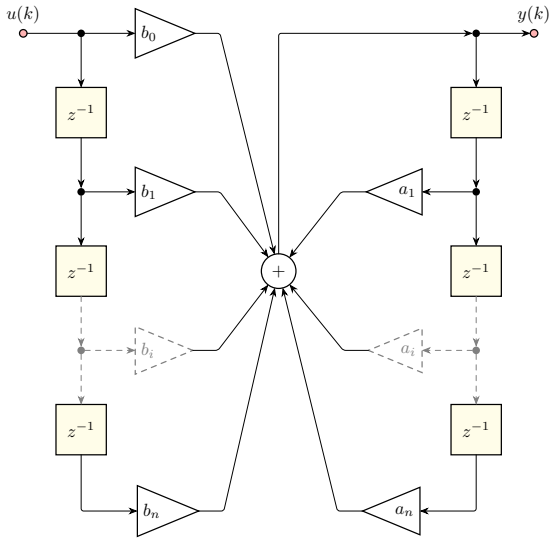
For an evaluation of an n^{th} order filter they require $2n+1$ coefficients to be stored and used. The major drawback is that they are very sensitive to finite-precision effects and low-precision implementations often have extremely high error [18].

In order to overcome this drawback, another type of structure has been derived from transfer function coefficients: decomposition into second-order sections. It is based on the partial fraction expansion of the rational function $H(z)$ into the terms of second order:

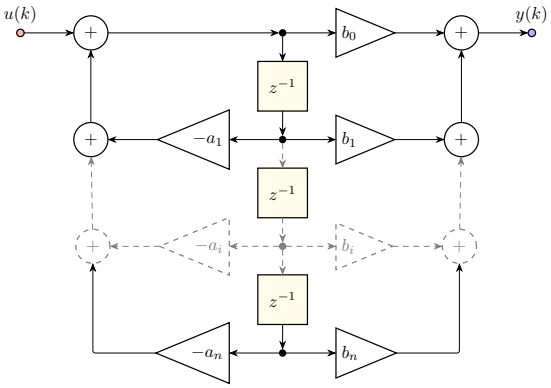
$$H(z) = \prod_{i=1}^{\lceil n/2 \rceil} \frac{b_{i0}^{'} + b_{i1}^{'} z^{-1} + b_{i2}^{'} z^{-2}}{1 + a_{i1}^{'} z^{-1} + a_{i2}^{'} z^{-2}}. \tag{1.19}$$

If we recall the properties of the convolution operator (Figure 1.2), such partial fraction expansion corresponds to a cascade of $\lceil n/2 \rceil$ second-order filters.

Various approaches have been proposed on the improvement of second-order sections behavior in finite precision [26–29]. In general, for a n^{th} order filter such structures require $2n-1$ additions, $2n$ multiplications and, storage for $2n+1$ coefficients.

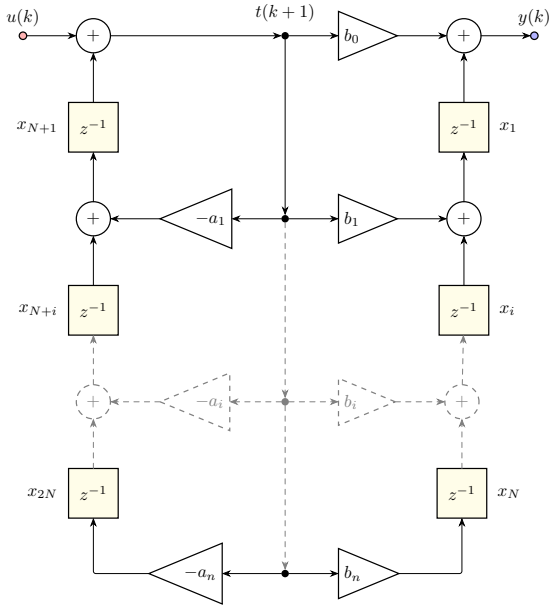


(a) Direct Form I

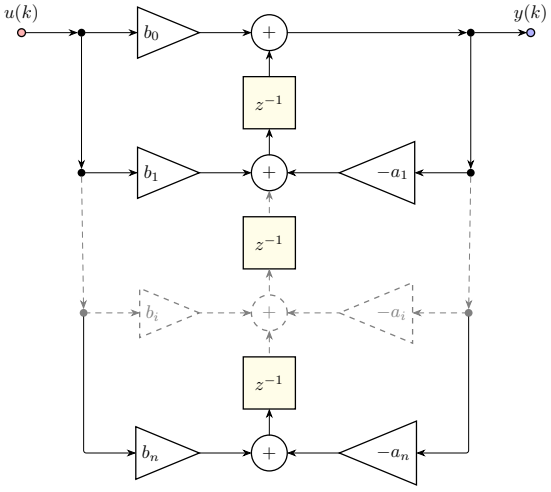


(b) Direct Form II

Figure 1.7: Direct Forms



(a) Direct Form I transposed



(b) Direct Form II transposed

Figure 1.8: Direct Forms transposed

1.5.2 State-Space

Another common representation of IIR filters is using state-space equations. State-space structure permits to easily describe MIMO systems (i.e. input and output signals are vectors) and comes from the theory of control systems. Its block-diagram is shown on Figure 1.9. Analytically a state-space system $\mathscr{H}$ describes the evolution of the state vector $\boldsymbol{x}(k)$ that depends on $\boldsymbol{x}(k-1)$ and the input vector $\boldsymbol{u}(k)$, while a new output vector $\boldsymbol{y}(k)$ is computed out of the current state and the input. An n^{th} order state-space structure with q inputs and p outputs is analytically described with

$$\mathscr{H}\begin{cases} \boldsymbol{x}(k+1) &= \boldsymbol{A}\boldsymbol{x}(k)+\boldsymbol{B}\boldsymbol{u}(k) \\ \boldsymbol{y}(k) &= \boldsymbol{C}\boldsymbol{x}(k)+\boldsymbol{D}\boldsymbol{u}(k) \end{cases}, \tag{1.20}$$

where $\boldsymbol{u}(k) \in \mathbb{R}^q$ is the input vector, $\boldsymbol{y}(k) \in \mathbb{R}^p$ the output vector, $\boldsymbol{x}(k) \in \mathbb{R}^n$ the state vector and $\boldsymbol{A} \in \mathbb{R}^{n\times n}$, $\boldsymbol{B} \in \mathbb{R}^{n\times q}$, $\boldsymbol{C} \in \mathbb{R}^{p\times n}$ and $\boldsymbol{D} \in \mathbb{R}^{p\times q}$ are the state-space matrices of the system. In the case of a SISO system, $\boldsymbol{B}$ and $\boldsymbol{C}$ are vectors, and $\boldsymbol{D}$ is a scalar, which will further be indicated by appropriate notation $\boldsymbol{b}, \boldsymbol{c}$ and d.

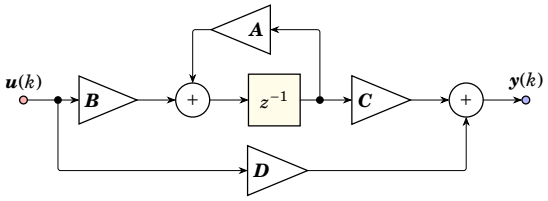


Figure 1.9: The State-Space structure

The matrix of transfer functions of the state-space system $\mathscr{H}$ is computed with

$$\mathbf{H}(z) = \boldsymbol{D} + \boldsymbol{C}(z\boldsymbol{I} - \boldsymbol{A})^{-1}\boldsymbol{B}, \quad z \in \mathbb{C}. \tag{1.21}$$

Remark 1.3. *In the MIMO case $\mathbf{H}(z)$ is called a multidimensional transfer function, i.e. can be seen as a $p \times q$ matrix containing p rational functions that share the same denumerator but have different numerators (for each output of the system) [30].*

The impulse response $\mathbb{h}(k)$[7] of the MIMO system $\mathscr{H}$ can be computed with

$$\mathbb{h}(k) = \begin{cases} \boldsymbol{D}, & k = 0 \\ \boldsymbol{C}\boldsymbol{A}^{k-1}\boldsymbol{B}, & k > 0 \end{cases}, \tag{1.22}$$

where each sample $\mathbb{h}(k)$ is a $p \times q$ matrix, and element $\mathbb{h}_{ij}(k)$ is the response of the i^{th} output to the impulse signal on the j^{th} input.

[7] Exceptionally, we denote a matrix by a small case letter, for instance $\mathbb{h}$.

The stability criterion of the state-space systems can be formulated as follows:

Property 1.1. *(Bounded Input Bounded Output Stability) Let $\mathscr{H}$ be a state-space system. Suppose an input signal $\{\boldsymbol{u}(k)\}_{k\geq 0}$ is known to be bounded by $\bar{\boldsymbol{u}}$ (i.e. $\forall k \geq 0$, $|\boldsymbol{u}_i(k)| \leq \bar{\boldsymbol{u}}_i, 1 \leq i \leq q$). So it holds that the output $\{\boldsymbol{y}(k)\}_{k\geq 0}$ is bounded iff the spectral radius $\rho(\boldsymbol{A})$ is strictly less than 1.*

See [31] for a proof.

A state-space structure can be easily obtained from the filter's transfer function [32]. For example, an n^{th} order SISO IIR filter described with a rational transfer function (1.16), the state-space matrices can be computed with

$$\boldsymbol{A} = \begin{pmatrix} -a_1 & 1 & & \\ \vdots & & \ddots & \\ \vdots & & & 1 \\ -a_n & 0 & \dots & 0 \end{pmatrix}, \quad \boldsymbol{b} = \begin{pmatrix} b_1 - a_1 b_0 \\ \vdots \\ \vdots \\ b_n - a_n b_0 \end{pmatrix}, \quad \boldsymbol{c} = \begin{pmatrix} 1 & 0 & \cdots & 0 \end{pmatrix}, \quad d = b_0. \tag{1.23}$$

The state-space systems described with the equations (1.23) are called state-space systems in the canonical observable form [18, 32].

An advantage of the state-space systems is that we can perform a linear change of the state variable coordinates, which is called a similarity transformation [33, 34]. Indeed, consider any invertible matrix $\boldsymbol{T}$ and perform a change $\boldsymbol{x}_T(k) \triangleq \boldsymbol{T}\boldsymbol{x}(k)$. Then, we can compute a transformed system $\mathscr{H}_T$ with new coefficient matrices

$$\begin{aligned} \boldsymbol{A}_T &= \boldsymbol{T}^{-1}\boldsymbol{A}\boldsymbol{T}, & \boldsymbol{C}_T &= \boldsymbol{C}\boldsymbol{T}, \\ \boldsymbol{B}_T &= \boldsymbol{T}^{-1}\boldsymbol{B}, & \boldsymbol{D}_T &= \boldsymbol{D}. \end{aligned} \tag{1.24}$$

The transformed system describes the same system in new state-variable coordinates. It can be verified that the transfer function has not changed [34].

Therefore, there exists infinitely many *mathematically* equivalent state-space realizations for any recursive linear filter. The questions of choice of the best state-space realization have been addressed in, namely, works of Gevers and Li [34].

1.5.3 Other structures

There is a huge variety of other structures and regularly more appear. Some of them are based on analytical representation, like State-Space. Others, like Wave and Lattice structures, whose coefficients are directly derived from analog filters, are analyzed by means of Signal Flow Graph theory. This approach benefits from its proximity with the analog circuits. However, the comparison issue arises from both theoretical (comparison of structures given in different

representations) and practical (the designer should have a more or less deep knowledge of numerous structures) points of view.

Regardless of the means of description, one of the most important considerations in the performance evaluation of digital signal processing structures is sensitivity under finite-precision arithmetic constraints. The topology (i.e. the placement of adders, multipliers, delays and data paths) of the realization structures significantly influences the numerical quality, investigation of which is in the heart of our work.

1.6 Conclusion

In this Chapter we have presented some basic notions of LTI IIR filters, their frequency and time domain representations. There exist numerous computational algorithms, called structures, with which we can compute the same filter. Filter structures can be described either in analytical (recurrence equations, matrices) or in graphical way (block diagrams). They possess different set of coefficients and can be even obtained without actually computing the filter's transfer function. Moreover, in further reading we should keep in mind that different structures call for different analysis methods, which significantly complicates the designer's job.

CHAPTER 2

COMPUTER ARITHMETIC

Efficient representation of real numbers on electronic computers is not a straightforward task. Since the early years of computer science, many different ways have been introduced. While radix 10 is most convenient for the understanding by humans, most of the computers are based on two-state logic. Therefore, radix-2 arithmetic is easiest to implement and we focus only on such representations. In general, a real number is represented in the machine with some finite number of digits, called precision. Finite-precision numbers form a subset of real numbers and there exist different solutions for their hardware and software representation.

In this Chapter we present basic notions of two finite-precision representations: Floating-Point and Fixed-Point. It appears that Floating-Point arithmetic (with well chosen parameters) is a very good compromise for most numerical applications. However, it comes together with relatively high cost (in terms of speed, power consumption, etc) that is often too high for embedded microprocessors. In these cases, the faster and less power-hungry Fixed-Point arithmetic is preferred.

Finally, we give a brief overview of finite-precision effects that occur during the implementation of digital filters on Fixed-Point processors.

Rounding and errors

On the one hand we have real numbers, and on the other hand we have a finite-precision arithmetic, which describes a set of discretized values with a finite number of digits. Let $\mathbb{F} \subset \mathbb{R}$ be some set of finite-precision numbers. In order to represent a real number in $\mathbb{F}$, we must apply a rule which is called *rounding*. In general, rounding is the operation of replacing a numerical value by another one which, usually, has a smaller number of digits in its representation. There are many ways of rounding numbers:

- round-to-nearest: $RN(x) \in \mathbb{F}$ is the finite-precision number that is closest to x. In the case when x is exactly between to finite-precision numbers, a tie-braking rule must be used;
- round towards $+\infty$ (up): $RU(x) \in \mathbb{F}$ is the smallest finite-precision number that is greater than or equal to x;

- round towards $-\infty$ (down): $RD(x) \in \mathbb{F}$ is the largest finite-precision number that is smaller than or equal to x.

These ways are called rounding modes.

Suppose you compute some function in finite-precision arithmetic. If the rounded result is the same as if the function were computed in infinite precision and then rounded (once), then we say that the function is *correctly rounded*. If for an exact result y one writes a finite-precision algorithm that always returns either $RU(y)$ or $RD(y)$, then the result is said to be *faithfully rounded*.

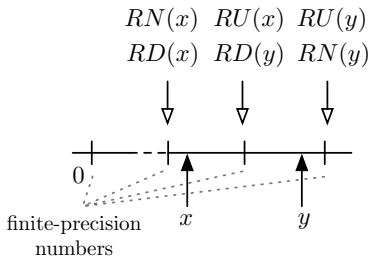


Figure 2.1: Illustration of three rounding modes of real numbers x and y.

Obviously, rounding can lead to errors that are called *rounding errors*. One of the goals of computer arithmetic is to estimate the errors in computations when using finite-precision arithmetic for algorithms. In the case when a sequence of numerical computations is subject to rounding operators, the errors may accumulate and even dominate the calculation itself.

Denote by $\circ(\cdot)$ some rounding operator. When approximating a real non-zero number x by $\circ(x)$, the *relative error* [35] is defined by

$$\delta x := \frac{\circ(x) - x}{x}. \tag{2.1}$$

Another type of measurement of error is the *absolute error [35]*:

$$\Delta x := \circ(x) - x. \tag{2.2}$$

Sources of Errors

In the following we will distinguish two sources of errors in numerical computations:

- Methodological errors: for example an infinite sum can be computed only by taking some finite number of terms, and the omitted terms constitute the truncation error or "methodological" error.

- Rounding errors that occur while working with finite precision arithmetic.

In this work, when doing the error-analysis of our algorithms, we rigorously account for both sources of errors.

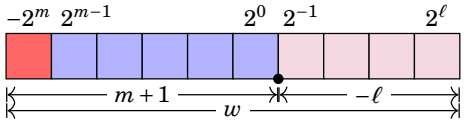


Figure 2.2: Fixed-point representation (here, $m = 5$ and $\ell = -4$).

2.1 Fixed-Point Arithmetic

A radix-2 Fixed-Point (FxP) number system [36, 37] is a subset of real numbers whose elements are integers scaled by a fixed factor and have form

$$\pm X \cdot 2^\ell, \tag{2.3}$$

where X is an integer mantissa and 2^ℓ is an implicit quantization factor. Remark that the FxP numbers are *equally spaced* by the quantization factor because it is fixed.

The power of FxP arithmetic comes from the fact that it is based on manipulation with integers and bit-wise arithmetic. It is particularly well exploited in embedded systems that permit small wordlengths and require high speed. However, the downside is that FxP representations have a relatively limited range of values that they can represent.

However, no uniformly used standard exist and, as a drawback, no recognized systematic overview nor unique notation of the FxP numbers exist either. Further in the thesis we consider two's complement Fixed-Point arithmetic [36–38], which we recall to the reader just below. We refer the reader to the PhD thesis of B. Lopez [6] (in French) for a detailed description of the format and notation that we adopt in this thesis.

2.1.1 Two's-complement numbers

Two's complement is a representation of signed numbers in integer arithmetic. Let t be a signed FxP number. It is written as

$$t = -2^m t_m + \sum_{i=\ell}^{m-1} 2^i t_i, \tag{2.4}$$

where t_i is the i^{th} bit of t, and m and ℓ are the *Most Significant Bit* (MSB) and *Least Significant Bit* (LSB) positions of t (see Figure 2.2) respectively. Denote by w the wordlength of a FxP number. It is related with the MSB and LSB positions via

$$w = m - \ell + 1. \tag{2.5}$$

A w bit FxP number t in two's-complement representation is stored as an integer mantissa $T \in [-2^{w-1}; 2^{w-1} - 1] \cap \mathbb{Z}$ scaled by the quantization factor 2^ℓ.

The range of numbers that can be represented with the wordlength w and quantization factor ℓ is the interval $[-2^m; 2^m - 2^\ell]$, called dynamic range.

Remark 2.1. *The dynamic range of two's complement is asymmetric, which in certain cases may complicate the conversion of a real number to FxP representation [39].*

2.1.2 Conversion

Consider the following problem: given a real non-zero number[1] $x \in \mathbb{R}$ and wordlength w, determine the best FxP format (m, ℓ). Obviously, we are interested in finding the least possible MSB position m. The least MSB position can be computed in the most cases with

$$m = \begin{cases} \lfloor \log_2(t) \rfloor + 1 & \text{if } t > 0 \\ \lceil \log_2(-t) \rceil & \text{if } t < 0 \end{cases} . \tag{2.6}$$

Once the MSB position is computed, we can directly determine the LSB $\ell = m - w + 1$. Finally, we can compute the value of the integer mantissa $T \in \mathbb{Z}$ with

$$T = \lfloor t \cdot 2^{-\ell} \rceil, \tag{2.7}$$

where $\lfloor \cdot \rceil$ denotes round to nearest integer operator. Then, the FxP counterpart $\hat{t}$ of the real number t is actually $\hat{t} = T \cdot 2^\ell = \lfloor t \cdot 2^{-\ell} \rceil \cdot 2^\ell$.

Remark 2.2. *When converting a real number to FxP arithmetic we seek to have the least error and usually can use round to nearest. In hardware, when converting a number from a larger FxP format to a smaller one, another rounding mode can be used. To minimize the hardware cost, truncation (understand "round-down") is often used.*

Example 2.1. *Let* $t = \sqrt[3]{2} = 1.2599210498948731647672 1...$ *and suppose we need to represent it with* $w = 8$ *bits. Using the above approach we obtain the MSB* $m = \lfloor \log_2(\sqrt[3]{2}) \rfloor + 1 = 1$ *and LSB* $\ell = 1 - 8 + 1 = -6$*. Therefore, using a round-to-nearest integer operator, the actually stored mantissa* $T = \lfloor \sqrt[3]{2} \cdot 2^6 \rceil = 81$ *and the* 8*-bit FxP representation of* t *is* $\hat{t} = 81 \cdot 2^{-6} = 1.265625$*.*

However, the MSB computation formula (2.6) is naive: the asymmetric dynamic range of the two's complement FxP arithmetic imposes limit cases that should be treated separately. We refer the reader to [6, 39] for a systematic approach on treatment of such cases.

[1] For the zero any format is suitable.

2.1.3 Overflows

When determining FxP formats for variables in a FxP algorithm we need to rigorously determine the range of all variables involved in the computations. Otherwise, an overflow may occur, i.e. at some point an integer mantissa exceeds its range $[-2^{w-1}; 2^{w-1}-1]$. Consider the example of a FxP algorithm for a vector normalization.

Example 2.2. *Let $\boldsymbol{t} = (125, 125, 125)$ be a vector in 3-dimensional space whose coordinates are represented with the FxP format $(15, -16)$. One can suppose that computing the length of $\boldsymbol{t}$ and storing it in a variable $h = \sqrt{125^2 + 125^2 + 125^2} = 216.5\ldots$ with the same format $(15, -16)$ should be enough. The problem is that the sum of those squares is equal to $46875 \notin [-32768; 32767]$, i.e. it exceeds the dynamic range of the format $(15, -16)$. Thus, a variable storing the intermediate result with the sum of squares must have larger wordlengths.*
Sometimes, determining a potential overflow is complicated. For example, the normalization will work just fine on a lot of values: vector $(100, 100, 100)$ will be normalized just fine but the vector (40,40,180) will not.

If careful investigation of the dynamic range of all variables is not possible, there exist several mechanisms on dealing with the overflow: wrap-around mode [40], saturation to min/max [40], etc. In digital signal processing, the saturation mode is widely used [40] because it prevents an overflow from being turned into an abrupt change from very big positive to very big negative values due to wrapping.

In saturation mode, overflow is dealt with by replacing the value that overflows by the largest (smallest) representable value of the target format. Such technique requires additional hardware support. Consider a number v_1 with FxP format (m_1, ℓ) that needs to be represented with a "smaller" FxP format (m_2, ℓ), $m_1 > m_2$. Then, the saturated value v_2 is

$$v_2 = \begin{cases} 2^{m_2} - 2^{\ell} & \text{if } v_1 \geq 2^{m_2} - 2^{\ell} \\ -2^{m_2} & \text{if } v_1 \leq -2^{m_2} \\ v_1 & \text{otherwise} \end{cases} . \tag{2.8}$$

Computations performed with the saturation technique may give an impression to be correct if the saturated values yielded a small overflow (in sense of the absolute value). However, in general such a suppression of the values significantly changes the accuracy of the result.

In this thesis we advocate rigorous evaluation of the range of all variables involved in a FxP implementation. Our implementations will not use saturation or other similar techniques since we will give a guarantee (based on mathematical proofs) that no overflow occurs.

2.1.4 Quantization and computational errors

We have already seen that in general rounding errors occur during the conversion of a real (exact) number to some FxP format. Further we will refer to the rounding errors of coefficients of the filter as *quantization errors*. Bounds on the quantization errors can be deduced with respect to the quantization step and rounding mode.

Consider a real number $t \in \mathbb{R}$ that is represented in FxP with format (m, ℓ). Denote by Δt the quantization error between t and its FxP approximation $\hat{t}$: $\Delta t = \hat{t} - t$. The bounds on Δt for different rounding modes are summarized [6] in Table 2.1.

Rounding mode	Error bound
round-to-nearest	$-2^{\ell-1} \leq \Delta t \leq 2^{\ell-1}$
truncation	$0 \leq \Delta t < 2^{\ell}$
faithful	$-2^{\ell} < \Delta t < 2^{\ell}$

Table 2.1: Error bounds for quantization errors according to the rounding mode.

Apart from the quantization errors, in the general case computational errors are induced after every arithmetic operation on FxP numbers. Usually, in the computational units the result of intermediate operations is stored with larger precision and then rounded to the initial precision. The errors due to this rounding can propagate and be significantly amplified.

Example 2.3. *Let $\hat{c}$ be some number rounded to FxP format (m_c, ℓ_c) and let t be a variable (exact quantity) in FxP format (m_t, ℓ_t). The product of these two numbers can be exactly representable on $w_c + w_t$ bits.*
Suppose the rounding error Δc is bounded in its absolute value $|\Delta c| < 2^{\ell}$. In other words, the last bit of $\hat{c}$ is potentially erroneous. Then, by using the classic long multiplication method we may remark that this potentially false bit will propagate through the multiplication in such way that the last $w_t - 1$ bits of the product become potentially false, too. Therefore, we obtain that the error between the exact product $c \cdot t$ and $\hat{c} \otimes t$ (here $\otimes$ means FxP product) is bounded in absolute value by $2^{w_t - 1}$.

When implementing FxP algorithms, the classic problem is to estimate an upper bound on the error of some operation, given the formats of the operands. However, during the implementation of digital filters we will look at this problem in another way: given the required upper bound on the output error, determine the least formats of the operands that guarantee this error.

The reader may have noted that filter structures (e.g. state-space) are often based on the computation of Sums-of-Products by Constants (SOPCs). To ensure some properties of the implemented filters we need to define how these Sums of Products are computed in FxP

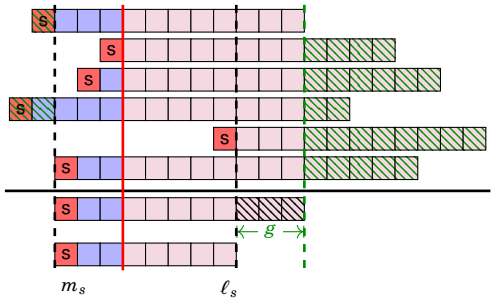


Figure 2.3: The sum is first performed with extended format and then rounded.

arithmetic. Further in this thesis we will rely on the following result presented and proved in [6, 41, 42].

Proposition 2.1 (Faithfully rounded Sums-of-Products)**.** *Let $c_1, c_2, \ldots, c_n$ be some quantized FxP numbers with respective formats (m_{c_i}, ℓ_{c_i}) and $t_1, t_2, \ldots, t_n$ be FxP variables with respective formats (m_{t_i}, ℓ_{t_i}). Suppose we need to compute the SOPC $S = \sum_{i=1}^{n} c_i t_i$ in FxP arithmetic and represent the output in the format (m_s, ℓ_s) (and we know that there will be no overflow). Then if*

- *the results of FxP multiplications are stored on $w_{c_i} + w_{t_i}$ bits, and*
- *the sum is performed on the extended by g guard bits FxP format[2] $(m_s, \ell_s - g)$, where $g = \lfloor \log_2(n) \rfloor + 1$, and then rounded to the final format (m_s, ℓ_s) (see Figure 2.3)*

we can guarantee that the FxP result is a faithful rounding of the exact sum of product, i.e. the absolute error is bounded by 2^{ℓ_s}.

2.2 Floating-Point Arithmetic

The Floating-Point (FP) number system is another way of describing numbers in computers. The concept is adapted from the scientific number notation. In 1985 the IEEE published the IEEE-754 standard for Binary Floating-Point Arithmetic which is nowadays supported in the vast majority of systems.

A key feature of the FP notation is that the represented numbers are *not uniformly spaced*: it is ensured that there are small gaps between small numbers and large gaps between large numbers. In comparison with Fixed-Point Arithmetic, FP has a larger dynamic range. According

[2] All bits to the right of $\ell_s - g$ can be truncated.

to studies [43, 44] FP implementations of digital filters yield much less error to the output[3]. However, the manufacturing cost dictates the use of cheaper, smaller and less power-hungry Fixed-Point processors.

We use Floating-Point arithmetic as our main instrument for computation of optimal implementation parameters, accurate evaluation of FxP implementation errors, etc. Therefore, in this section we present a small overview of the IEEE 754-2008 Floating Point Standard [45] and, since FP computations are not exact, the general idea of the FP error analysis techniques that we will use.

We refer the reader to the *Handbook of Floating-Point Arithmetic* by Muller et al. [35] for a detailed and formal reference on the general subject of FP arithmetic. As well as to a guide into *Accuracy and Stability of Numerical Algorithms* [46] by Higham.

2.2.1 Normalized IEEE 754 Binary Floating-Point

The 2008 version of the IEEE standard supports radix-2 and radix-10 systems. In this thesis we consider only radix-2 floating-point numbers. According to the standard, a radix-2 FP number is written as:

$$x = (-1)^s \cdot m \cdot 2^e, \tag{2.9}$$

where

- $s \in \{0, 1\}$ is the sign bit
- the exponent $e \in [e_{\min}, e_{\max}]$, with $e_{\min}$ and $e_{\max}$ specified for each standard format;
- the mantissa m is represented with p bits (p defined by the standard for each format) is normalized to the form $1.m_1m_2...m_{p-1}$. In other words, only $p-1$ trailing mantissa bits are stored.

In contrast to Fixed-Point Arithmetic, the exponent is stored explicitly along with the mantissa and is not fixed (thus the name, "floating-point"). The actual exponent e is biased in order to be stored as a positive integer: an exponent stored on w bits is biased by $b = 2^{w-1} - 1$. The smallest positive normal[4] number is $2^{e_{\min}}$. The largest finite binary floating-point number is $(2 - 2^{1-p}) \cdot 2^{e_{\max}}$. Numbers $|x| < 2^{e_{\min}}$ are called subnormal numbers and not all systems support them. In this work we do not consider sub-normal numbers.

[3] However, to our knowledge, similar to the case of FxP implementations, no systematic studies of implementation errors has been performed for FP filter implementations either.

[4] i.e. with mantissa m having the first bit set to 1

In IEEE format, both $+0$ and -0 exist, along with signed infinities and "Not a Number data", shortly NaNs, which are returned in case of invalid operations.

The IEEE 754 standard dictates several formats, i.e. standard couples of parameters p and w, and we are going to use two of them: double and single. The double precision format has a 11-bit exponent and mantissa with 53 bits of precision. The single precision format has a 8-bit exponent and mantissa with 24 bits of precision.

2.2.2 Conversion

Obviously, a real number might be not exactly representable as a FP number. This is mainly due to two reasons: (i) the number lies strictly between two consecutive floating-point numbers; (ii) number is out of range, i.e. its absolute value is larger than $2^{e_{\max}}$ or smaller than $2^{e_{\min}}$ (if subnormals are supported, the bound for complete underflow is $2^{e_{\min}-p}$).

The relative error due to rounding is given by the *unit roundoff* [46]: $u = 2^{-p}$. It can be proved that if a real number $x \in \mathbb{R}$ lies in range of the radix-2 FP format with precision p, then

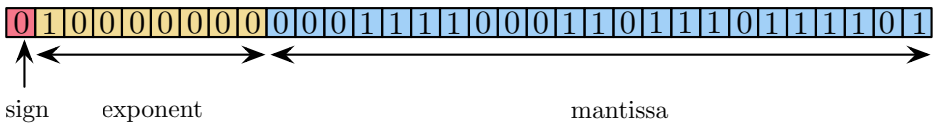
$$\circ(x) = x(1+\delta), \quad |\delta| \leq u, \quad u = 2^{-p}, \tag{2.10}$$

where $\circ(\cdot)$ is the round to nearest operator[5].

Example 2.4. *Consider the single precision format: the exponent is stored on 8 bits and the mantissa on 23 bits. Suppose we wish to represent $x = \sqrt{5}$ in this format. Binary representation of x will be*

$$10.00111100011011011110011011100101111110100101001...{}_2 \tag{2.11}$$

After normalization of the mantissa the non-biased exponent is equal to 1. We compute the biased exponent e by adding the bias equal to 127, therefore $e = 128$. Also, for a representation with 23 bits, we round it with the round-to-nearest operator. Therefore, a single precision FP approximation of x is:



The relative error is indeed bounded by the unit roundoff: we obtain that $\delta = 1.1111100001...{}_2 \cdot 2^{-27}$ while for single precision $u = 2^{-24}$.

[5]The actual bound is more complicated and depends on the rounding operator

2.2.3 Rounding errors

Any result of a finite precision computation is susceptible to rounding errors. Sometimes rounding can be beneficial: rounding errors can cancel in certain algorithms and lead to a final result that is more accurate than intermediate results [46]. However, usually we refer to rounding errors as to an undesirable effect and want to ensure relative and absolute error bounds on the result of a computation. An important remark is that rounding errors are not random. Using Kahan's examples Higham demonstrated [46] that rounding errors may have a strikingly regular pattern. Thus, modeling the rounding errors with random noise [47] can be readily argued with.

A widely used model [46] for the errors of FP operations over two FP normal numbers x and y can be expressed as (in the absence of under- and overflows and with round-to-nearest):

$$\circ(x \text{ op } y) = (x \text{ op } y) \cdot (1+\delta), \quad |\delta| \leq u, \quad \text{op} = \{+,-,\cdot,/\}. \tag{2.12}$$

For the ease of the notation we will often use circled signs of arithmetic operations $\oplus, \ominus, \otimes$ to denote FP additions, subtraction and multiplication.

However, when operands of arithmetic operations have themselves been subjects to previous rounding, catastrophic loss of significant digits can happen. This undesirable effect of rounding errors can be demonstrated with the *catastrophic cancellation*: subtraction of two nearly equal numbers can cause many of the accurate digits to disappear.

Example 2.5 (Cancellation)**.** *Consider the* $\cosh^{-1}$ *function that is computed with* $\cosh^{-1}(x) = -\log\left(x - \sqrt{x^2-1}\right)$*. Here, when* x *is large, the rounding that occurs in the square root computation leads to* $\sqrt{x^2-1} \approx x$ *and, subsequently, to invalid logarithm computation. For example,* $\cosh^{-1}(10^{10}) \approx 23.71899...$ *but in double precision it evaluates to* $-\infty$*. This occurred because* $\circ(\sqrt{10^{20}-1})$ *evaluated to* 10^{10}*.*

While for basic arithmetic operations correct rounding can be easily guaranteed [48], for more complicated operations and algorithms, correct rounding is difficult (or impossible) to obtain [35, 49]. For some mathematical functions there exist solutions such as correctly rounded library CRlibm[6] [50–53]. Another solution is Sollya which is both a tool and a library [53, 54]. Finally, code for correctly rounded elementary functions can be generated using Metalibm [55] code generator tool.

However, for an arbitrary numerical algorithm we are interested in ways to avoid effects such as cancellation, and determine a relative or absolute error bound on the output of an algorithm. In our contributions we will provide rigorous bounds on the computational errors (and their propagation) that occur in implementations of all the algorithms. We will often bound the errors with respect to some norm, such as the Frobenius norm.

[6] `https://scm.gforge.inria.fr/anonscm/git/metalibm/crlibm.git`

2.2.4 Multiple Precision Arithmetic

Sometimes bounding the error of our implementation will not suffice: the error bound may be relatively small but still too large in its absolute value. This usually means that the precision of intermediate computations must be increased.

There exist quite a few solutions for extending the precision of computations: the IEEE 754 standard proposes a few large and a few extended formats; exact arithmetic (rational numbers, continued fractions, etc.); the super-accumulator of Kulisch [56]; floating-point expansions [57]; multiple precision arithmetic [58, 59], etc.

In our algorithms we will need to change the precision dynamically and non-homogeneously (i.e. different variables may have different precision). To satisfy these requirements, we use Multiple Precision (MP) floating-point arithmetic. In particular, we are going to use GNU MPFR library [59] for our implementations.

We will often refer to some outputs of algorithms as being computed with *a priori* error bound. In fact, what we understand by this is that given a bound on the error, the algorithm dynamically adapts the precision of internal computations such that the computational errors are not larger than the given error bound and the output of the algorithm is returned as a multiple precision number. In such algorithms we will take care not to overestimate the required internal precision.

2.2.5 Interval Arithmetic

Another way to deal with the uncertainties and rounding errors in the computations is Interval Arithmetic. The term "interval arithmetic" dates from the 1950s due to works of Moore [60]. We refer the reader to [61–63] for a detailed overview of interval arithmetic.

An interval, denoted $[x] = [\underline{x}, \overline{x}]$, is a closed and bounded nonempty interval:

$$[x] = \left[\underline{x}, \overline{x}\right] = \left\{x \in \mathbb{R} | \underline{x} \leq x \leq \overline{x}\right\}, \tag{2.13}$$

where $\underline{x}$ and $\overline{x}$ are called lower and upper bounds respectively and $\underline{x} \leq \overline{x}$. In the case $\underline{x} = \overline{x}$, we will call $[x]$ a point-interval. Sometimes, we will use the mid-rad representation of an interval:

$$[x] = \langle x_m, x_r \rangle, \tag{2.14}$$

where $x_m = \frac{\overline{x}+\underline{x}}{2}$ is the midpoint and $x_r = \frac{\overline{x}-\underline{x}}{2}$ is the radius of the interval.

When we compute a function $f : \mathbb{R} \to \mathbb{R}$ on the interval argument $[x] = I$, we seek to determine the intervals around the output $[y] = J$ such that $\forall\ x \in I,\ f(x) \in J$. This property is called *inclusion property* and while it does not help finding the exact output, it gives a guarantee that the computed interval contains the exact result. All basic arithmetic operations are defined for intervals and are based on the inclusion property.

The main problem related to the naive use of interval arithmetic is the phenomenon of decorrelation, a.k.a. dependency problem. This is due to the fact that interval arithmetic cannot trace the correlation between multiple occurrences of the same variable. Consider an example of expression $y = \frac{x}{x+1}$. For $[x] = [4, 9]$, the output $[y]$ should be an interval $[0.8, 0.9]$. However, using interval evaluation, we obtain the result $[0.4, 1.8]$ which is much larger than the expected result. To avoid such situations, rewriting techniques are usually used [64]. For instance, $y = \frac{1}{1+\frac{1}{x}}$ yields a much smaller interval $[0.8, 0.9]$ for $[x] = [4, 9]$.

Intervals are often used to account for rounding errors in the computations. When a computation with intervals yields a result $[y]$, its midpoint y_m is viewed as an approximate value for the result with a worst-case error y_r. Even if a result interval is too wide to be practically useful, it can at least ensure a fail-safe mode of operation.

Practical implementations of interval arithmetic are often based on multiple precision FP arithmetic, i.e. $\left[\underline{x}, \overline{x}\right] = \left\{x \in \mathbb{R} | \underline{x} \leq x \leq \overline{x}, \underline{x}, \overline{x} \in \mathbb{F}_p\right\}$, where $\mathbb{F}_p$ denotes a set of FP numbers with precision p. All operations on the endpoints must be done with care: to ensure the inclusion property, rounding must be done to wrap up around the exact result. In this thesis we are going to use the multiple precision floating-point interval library GNU MPFI [65, 66] for all interval operations.

In Part III, sometimes, we are going to use the Theory of Verified Inclusions developed by Rump [67–69] to account for errors of some linear algebra routines. In his approach, Rump addresses error bounds on the solution of some common numerical algorithms, such as eigendecomposition problems. He shows how we can easily deduce tight intervals of the solutions of these problems. These intervals are guaranteed to contain the exact results. We are going to use this approach to find interval enclosures for some algorithms that do not provide any rigorous error bounds, such as eigendecomposition, solution of linear system of equations and matrix inverse.

2.3 Finite Precision Effects for IIR filters

Before actually implementing IIR filters, we need to ascertain the extend to which its performance will be degraded by finite-precision effects. And, if the degradation is not acceptable, find a solution. Usually, the output error of the filter can be reduced by increasing the precision of computations and of the filter's coefficients but at the expense of an increased cost.

The filter degradation in both software and hardware depends on three main factors:

- quantization of the filter's coefficients,
- specification of arithmetic and wordlengths and
- structure, i.e. algorithm for the evaluation of the filter.

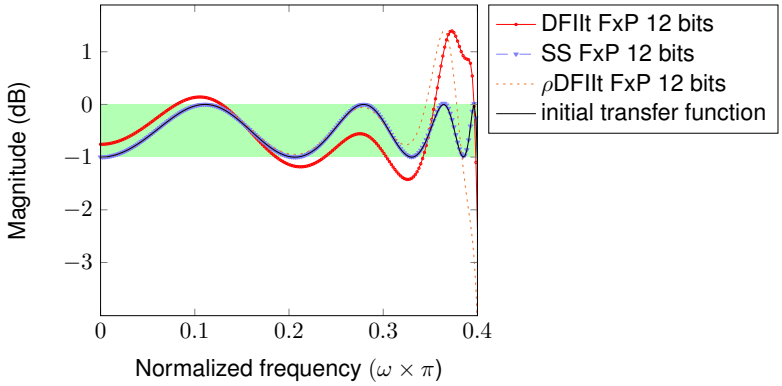


Figure 2.4: Quantization effects on the DFIIt, ρDFIIt and balanced State-Space systems

2.3.1 Coefficient quantization

Quantization of the coefficients to a finite number of bits essentially changes the filter coefficients, hence the frequency response changes as well. Quantization of the coefficients can easily yield an unstable filter. Different realizations usually demonstrate different behavior due to coefficient quantization.

Consider the key example of band specifications and corresponding transfer function presented in Chapter 1 Section 1.4.3. For instance, in the passband, i.e. for frequencies $\omega \in [0, 0.4\pi]$, the magnitude response of the filter must be between 0dB and -1dB.

Even without considerations on the FxP arithmetic (wordlengths of variables, specifications of arithmetic units, etc.), it is possible to observe how realizations obtained with different structures behave under coefficient quantizations. Consider three filter structures: Direct Form II transposed (DFIIt), balanced state space[7] and Direct Form II transposed implemented with ρ operator [70, 71].

On Figure 2.4 we can observe the magnitude response (in the passband) of the realizations obtained with the above structures and whose coefficients were quantized to 12 bits. It is clear that the majority of the realizations yields filters that do not respect the initial specifications anymore. However, the poles of each realization are still in the unit circle. Now, if we quantize the coefficients to 8 bits, poles of majority of realizations are out of the unit circle (Figure 2.5). This is all to say that quantization can yield an unstable filter and even if poles are in the unit circle, propagation of quantization errors may result in a filter that no longer respects the initial frequency specifications.

[7] As returned by the Matlab's function ss, which applies several similarity transformations upon the canonical controllable form

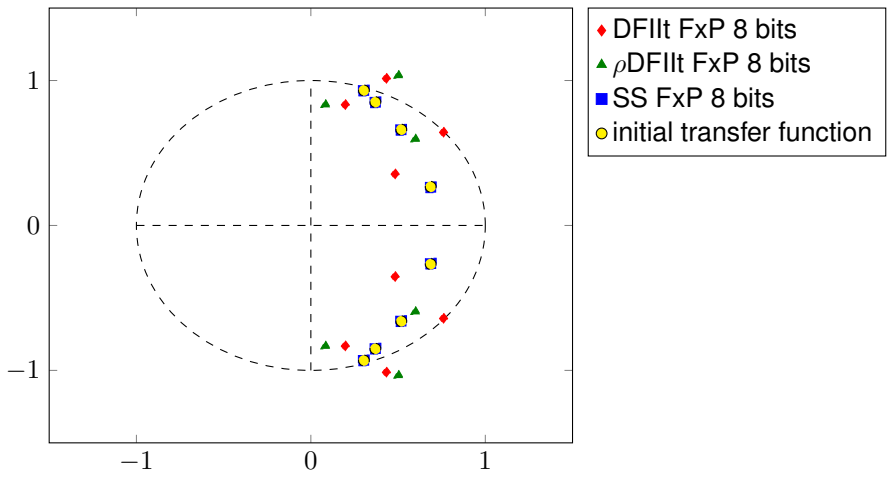


Figure 2.5: Pole locations for the initial system, DFIIt, rhoDFIIt and balanced State-Space quantized to 8 bits

2.3.2 Sensitivity analysis

Since it is desirable to predict the behavior of filter realizations prior to actually implementing them, a statistical approach is classically used. The idea is to compute *sensitivity measures* [72–74]: these measures are based on the sensitivity of the filter (its transfer function, poles, zeros, etc.) with respect to the coefficient quantization, i.e. they show how much small perturbations in the coefficients (or poles) of the filter may influence the behavior of transfer function. The first works in this direction were proposed by Tavsanoglu and Thiele [75]. An analytical form for such sensitivity measures should be derived for each structure separately. Further, if we work with state-space systems, we may seek for a similarity transformation that minimizes the overall sensitivity of the structure (with respect to ℓ_2-norm) [34, 74, 76].

However, this approach has several drawbacks: first of all, sensitivity measures can be derived only for some family of structures, and second, they do not reflect the real impact of quantization but just serve as an indication.

2.3.3 Roundoff noise and Fixed-Point Formats

To model the propagation of rounding errors the following model (see Figure 2.6) is usually used: thanks to the linearity of the systems, an implemented filter $\mathscr{H}^{\Diamond}$ is represented as a sum of the exact filter $\mathscr{H}$ and a special error-filter $\Delta\mathscr{H}$. This error-filter is not actually implemented and is only used to analyze the propagation of the computational errors $\varepsilon(k)$ that are considered as some noise signals.

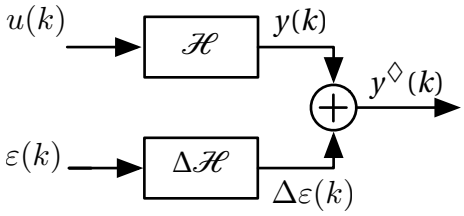
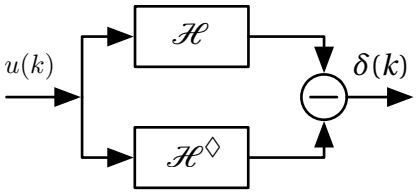
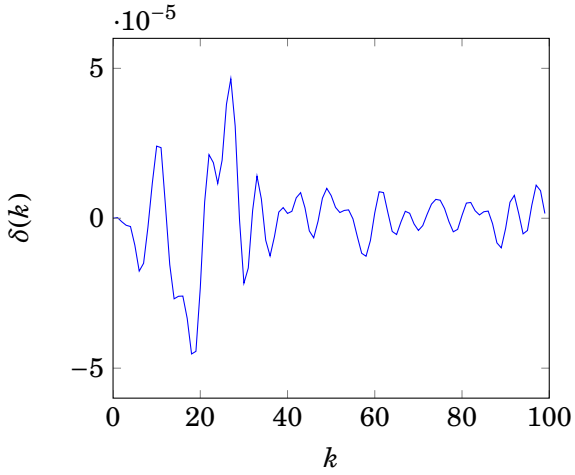


Figure 2.6: Classic decomposition of the implemented filter.

We can observe that this model indeed makes sense using our key filter example. Suppose we implement this filter with the Direct Form II structure. We represent all variables in the computational algorithm with 16 bits and on each iteration of the filter we guarantee correct rounding. Then, for some particular input signal $u(k)$ we can compute the difference $\mathscr{H} - \mathscr{H}^{\diamond}$ between the exact filter (computed with rational arithmetic) and the implemented filter as shown on the Figure 2.7a. On Figure 2.7b we can observe that the output $\delta(k)$ of the difference filter in response to the impulse signal $u(k)$ resembles some noise.



(a) Difference of two filters



(b) Output $\delta(k)$ in response to the impulse input.

Figure 2.7: Demonstration of the classic error-model.

Hence, not surprisingly, in classical signal processing, absolute errors are considered not as small intervals but as white noise signals [77]. Then, using the classic error-model, the output of the error filter is analyzed in terms of its mean and variance [78, 79]. We should remark that such an approach does not very well reflect the reality: as we said previously rounding errors are not random and the variance of the error does not provide an accurate perception of the implementation errors.

Dynamic range of variables The dynamic range of the filter's output is usually computed with simulations [80]. The idea is to: (i) perform simulations using a set of some "relevant" inputs; (ii) determine the range of each variable from these simulations; (iii) enlarge the ranges by some margin and take them as true ranges; (iv) cross fingers and hope that the inputs were "relevant" enough. Obviously, such approach is time-consuming and not reliable.

An improved approach is based on the following technique: (i) compute the probability distribution function of the outputs of the filter; (ii) determine the interval which contains all possible outputs, with some high-enough (obviously not 100%) probability; (iii) take this interval as the range of the filter's output [77]. Then, the Fixed-Point formats are determined from the dynamic range. Obviously, there is some probability that the range was underestimated and overflow might occur.

Often intermediate results in filter computation are forced to saturate rather than overflow. Implementations with saturation cannot be rigorously analyzed in terms of the rounding error: from simulations we cannot predict how far the saturated values were out of range and what is their influence on the filter's output. On top of that, we cannot use some properties of two's complement modular arithmetic that greatly simplify the additions and multiplications. These properties are also called Jackson's rule [81] which states that in sequence of additions in two's complement arithmetic certain intermediate results may overflow from the final format. However, if the result is representable in the final format, the result is valid.

Influence of rounding errors Usually, an *a posteriori* idea on the behavior of computational errors in linear filters is obtained via bit-true simulation [80, 82] of the FxP implementation and then comparison with a reference (floating-point) simulation. For instance, the difference filter shown on Figure 2.7a is used in extensive simulations to "bound" the implementation error. The advantage of this technique is that it can be applied to any realization. An obvious drawback is that simulations may not be exhaustive and comparison is not done with an exact filter but with a finite-precision evaluation. Thus, no guarantee on the result can be obtained with this approach. Moreover, simulations may take significant time [83].

Yet another solution is to apply analytical approaches once a mathematical expression of a numerical accuracy metric is determined. For example, using Interval Arithmetic [84, 85] and Affine Arithmetic [86–89]. These approaches are rigorous but they are prone to overestimations and are not well suited for recursive systems (intervals will just explode due to the decorrellation issue). There are some analytical approaches based on the noise propagation models [90, 91] or on the probability density function [92, 93]. However, these approaches do not provide a guarantee on the output error.

2.4 Conclusion

In this Chapter we have seen a very short overview of two ways to represent real numbers in finite-precision: Fixed- and Floating-Point Arithmetic. The first one is used in practical filter implementations due to its low cost while the second one is used during the design and analysis of the implementation.

The key problem with the existing approaches on filter error-analysis is that they are not suitable when a rigorous and optimal filter implementation is required. As far as we know, there exist no method that provides a *rigorous and tight bound* on the error of the implementation. Hence, for a reliable implementation we need to

- guarantee that no overflow occurs during the computations and
- provide a rigorous bound on the output error of a Fixed-Point implementation.

Moreover, the methodology must not depend on the filter evaluation algorithm and be applicable to any LTI filter realization.

CHAPTER 3

TOWARDS RELIABLE IMPLEMENTATION OF DIGITAL FILTERS

The question of automation of filter implementation is, obviously, not new. Frameworks for semi- or fully automatic filter implementation have been developed over time [1–3]. Almost every big manufacturer along with a dedicated hardware provides a tool whose goal is to *help* the designers. There is also the Matlab Simulink tool that *helps* the filter designers and proposes various toolboxes.

The drawbacks vary from one tool to other but the general idea is: there exists no tool that can implement an arbitrary realization and there is no guarantee on the implementation. Still a lot of decisions in the implementations depend on the designer. Often these decisions require high degree of expertise and a considerable amount of man-hours for the implementation.

To overcome these drawbacks a reliable filter code generation tool has been proposed [4, 6, 94, 95]. The foundation of its ideology can be expressed as four principles:

- all filter structures must be represented in a unified way
- all filter structures must be analyzed using the same criteria to enable fair comparison
- Fixed-Point implementations must be reliable, i.e. guarantee that no overflow occurs
- all implementations must come with a rigorous bound on the implementation error

Our goal is to provide the kernel methodology for this generator, improve and extend existing functionalities. In this chapter we give a brief overview of the tool ***before this thesis.***

3.1 Automatic Filter Code Generator

Automatic Filter Code Generator is a software solution the goal of which is to provide tools for reliable implementation of digital filters. It is based on the unification of existing approaches under a same notation, as well as on development of novel approaches for software and hardware implementation of filters.

One key feature that enables such a solution is the unification of representations of digital filters. In his PhD thesis [4], Hilaire developed a new formalism, called the Specialized Implicit

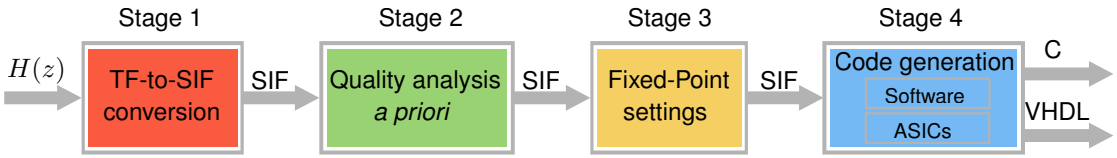


Figure 3.1: Automatic Filter Code Generator scheme before this thesis.

Form (SIF)[1] that can describe any linear filter (both FIR and IIR). It is an analytical representation, based on an extended state-space structure and will be addressed in Section 3.2. Roughly, the idea is to detail the way the computations are carried out (like in data-flows or algorithms) but to keep a matrix-based description based on a state-space structure. Its resemblance with the state-space systems rewards SIF with many useful properties. Therefore, the filter design and implementation process can be unified for this representation and then applied upon any realization of a LTI filter.

A simplified scheme of the generator work-flow is given in Figure 3.1. The process starts with a given transfer function[2] and proceeds to the hardware or software implementation. The filter-to-code generation is divided into four stages:

Stage 1: Given the coefficients of the transfer function we first represent it with SIF. Potentially, any structure describing a linear filter can be expressed with the SIF formalism. Algorithms for representations of several classical structures have been already proposed [4]. Thus, the user just needs to choose from a set of possibilities and run a conversion algorithm which returns a SIF that corresponds to a realization of the given transfer function with the desired structure. However, **support of a new structure requires a new conversion algorithm**.

Stage 2: For a chosen structure, we can compute various classical (for control systems) and new sensitivity measures [96]. These measures were directly derived for SIF matrices. Thus, using direct formulas one can compute sensitivity measures for structures that used to have only block-graph representation. These algorithms are implemented in Matlab as the Finite Wordlength Realization toolbox[3] [97].

Stage 3: For a chosen structure, we determine the parameters of Fixed-Point implementation. Given wordlength constraints (not necessarily the same for all variables), determine the position of the binary point for each constant coefficient and variable. In his thesis [6], Lopez proposed

[1] Sometimes in this thesis under “an instance of SIF” or even “a SIF’ we will actually understand “a filter in SIF representation”.

[2] In this generator, the questions of the design of transfer functions are not addressed.

[3] `svn://scm.gforge.inria.fr/svnroot/fwrtoolbox/`

conversion of structure's coefficients to FxP arithmetic by taking into account special limit cases. However, *the dynamic range* of the variables involved in the evaluation of a filter is still not determined reliably. The output interval for all variables is computed using deterministic measures, however **still no guarantee is provided on the FxP implementation or the bound on the output error**. It is easy to show that the basic bricks of linear filters are Sums-of-Products. Lopez proposed algorithms for the optimal (with area and error constraints) binary tree decomposition of operations in these Sums-of-Products.

Stage 4: Given the parameters of Fixed-Point implementation, we generate software and hardware code. Generation of C Floating- and Fixed-Point code is available. Hardware implementation on ASICs[4] using Lopez's Sums-of-Products is possible. However, **no other targets, such as Field Programmable Gate Arrays, are supported**. Moreover, Lopez's algorithms are based on non-rigorous dynamic range of Fixed-Point variables.

We see that while the main structure of the code generator is defined, it misses numerous basic bricks and could take advantage of several improvements of existing functionalities. Moreover, before this thesis some parts of the generator were implemented in Matlab while others were written in Python.

In the following we give a brief overview of the Specialized Implicit Form and its functionalities. For a full description, we refer the reader to [4, 6].

3.2 Specialized Implicit Form

The idea behind the Specialized Implicit Form is that for any structure it is not complicated to determine the state variables (those that are saved from one iteration to another) and to reflect the order of computations (additions and multiplications) in matrix form. Then, we can say that any MIMO LTI system is characterized by the input $\boldsymbol{u}$, output $\boldsymbol{y}$, state $\boldsymbol{x}$ (for IIR systems) and temporary $\boldsymbol{t}$ vectors:

$$\begin{pmatrix} \boldsymbol{J} & \boldsymbol{0} & \boldsymbol{0} \\ -\boldsymbol{K} & \boldsymbol{I}_n & \boldsymbol{0} \\ -\boldsymbol{L} & \boldsymbol{0} & \boldsymbol{I}_p \end{pmatrix} \begin{pmatrix} \boldsymbol{t}(k+1) \\ \boldsymbol{x}(k+1) \\ \boldsymbol{y}(k) \end{pmatrix} = \begin{pmatrix} \boldsymbol{0} & \boldsymbol{M} & \boldsymbol{N} \\ \boldsymbol{0} & \boldsymbol{P} & \boldsymbol{Q} \\ \boldsymbol{0} & \boldsymbol{R} & \boldsymbol{S} \end{pmatrix} \begin{pmatrix} \boldsymbol{t}(k) \\ \boldsymbol{x}(k) \\ \boldsymbol{u}(k) \end{pmatrix} \tag{3.1}$$

where

- $\boldsymbol{u}(k)$ represents the q inputs, and $\boldsymbol{y}(k)$ the p outputs;
- $\boldsymbol{x}(k+1)$ represents the n states stored at step k;

[4] Application-Specific Integrated Circuits

- $\boldsymbol{t}(k+1)$ represents the l intermediate variables in the calculations of step k (intermediate values that are used only on step k but not stored form one step to another like the states);
- $\boldsymbol{J} \in \mathbb{R}^{l\times l}$, $\boldsymbol{K} \in \mathbb{R}^{n\times l}$, $\boldsymbol{L} \in \mathbb{R}^{p\times l}$, $\boldsymbol{M} \in \mathbb{R}^{l\times n}$, $\boldsymbol{N} \in \mathbb{R}^{l\times q}$, $\boldsymbol{P} \in \mathbb{R}^{n\times n}$, $\boldsymbol{Q} \in \mathbb{R}^{n\times q}$, $\boldsymbol{R} \in \mathbb{R}^{p\times n}$, $\boldsymbol{S} \in \mathbb{R}^{p\times q}$ are coefficient matrices;
- by construction, the matrix $\boldsymbol{J}$ is lower triangular with 1 on the diagonal and denotes the order of computations for a structure.

The diagonal matrix on the left side of the implicit equation (3.1) allows us to describe the sequence of computations within a filter. For example, let $y \leftarrow \boldsymbol{m}_2(\boldsymbol{M}_1\boldsymbol{u})$ be computed as $\boldsymbol{t} \leftarrow \boldsymbol{M}_1\boldsymbol{u}$ and then $y \leftarrow \boldsymbol{m}_2\boldsymbol{t}$. So this sequence is described as

$$\begin{pmatrix} \boldsymbol{I} & \boldsymbol{0} \\ -\boldsymbol{m}_2 & 1 \end{pmatrix} \begin{pmatrix} \boldsymbol{t} \\ y \end{pmatrix} = \begin{pmatrix} \boldsymbol{M}_1 \\ 0 \end{pmatrix} \boldsymbol{u}. \tag{3.2}$$

Coefficients of a filter described with (3.1) can be regrouped into a matrix $\boldsymbol{Z}$:

$$\boldsymbol{Z} := \left(\begin{array}{c|c:c} -\boldsymbol{J} & \boldsymbol{M} & \boldsymbol{N} \\ \hdashline \boldsymbol{K} & \boldsymbol{P} & \boldsymbol{Q} \\ \hdashline \boldsymbol{L} & \boldsymbol{R} & \boldsymbol{S} \end{array}\right). \tag{3.3}$$

The minus signs "$-$" in (3.1) and (3.3) are used as a simple convention that simplifies derivation of certain sensitivity measures for SIF.

Matrix $\boldsymbol{Z}$ has a sparse nature, and possesses either trivial coefficients such as 0, 1, -1 and $\pm 2^n$ $(n > 0)$ or non-trivial coefficients that correspond to the coefficients of the structure that is described with the SIF.

The computations associated to (3.1) are ordered from top to bottom, associated in a one to one manner to the following system of equations:

$$\left\{\begin{array}{rcccccc} \boldsymbol{J}\boldsymbol{t}(k+1) & = & & & \boldsymbol{M}\boldsymbol{x}(k) & + & \boldsymbol{N}\boldsymbol{u}(k) \\ \boldsymbol{x}(k+1) & = & \boldsymbol{K}\boldsymbol{t}(k+1) & + & \boldsymbol{P}\boldsymbol{x}(k) & + & \boldsymbol{Q}\boldsymbol{u}(k) \\ \boldsymbol{y}(k) & = & \boldsymbol{L}\boldsymbol{t}(k+1) & + & \boldsymbol{R}\boldsymbol{x}(k) & + & \boldsymbol{S}\boldsymbol{u}(k) \end{array}\right. . \tag{3.4}$$

Let $\boldsymbol{J}' = \boldsymbol{J} - \boldsymbol{I}$. Then the first line in (3.4) will be

$$\left(\boldsymbol{J}' + \boldsymbol{I}\right)\boldsymbol{t}(k+1) = \boldsymbol{M}\boldsymbol{x}(k) + \boldsymbol{N}\boldsymbol{u}(k) \tag{3.5}$$

$$\boldsymbol{t}(k+1) = -\boldsymbol{J}'\boldsymbol{t}(k+1) + \boldsymbol{M}\boldsymbol{x}(k) + \boldsymbol{N}\boldsymbol{u}(k). \tag{3.6}$$

We obtain that vector $\boldsymbol{t}$ depends on itself. However, this is not a problem since $\boldsymbol{J}'$ is strictly lower triangular and the computation of i^{th} element of the temporary vector $\boldsymbol{t}_i(k+1)$ depends only on $\boldsymbol{t}_j(k+1)$ with $j < i$. This substitution will be useful further in this thesis.

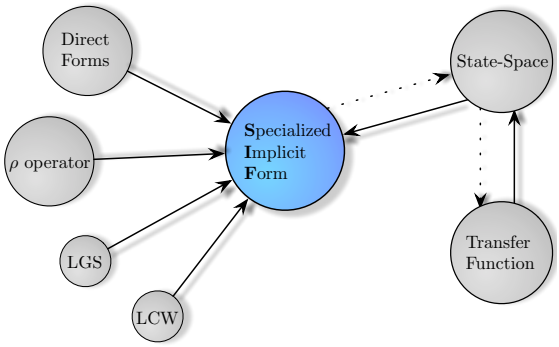


Figure 3.2: Conversion possibilities between SIF and other filter representations. Straight lines denote exact transformations, dotted lines denote error-prone computations.

3.2.1 SIF, State-Space and Transfer Function

Any LTI structures can be described with SIF. For instance, for any state-space system

$$\begin{cases} \boldsymbol{x}(k+1) &= \boldsymbol{A}\boldsymbol{x}(k)+\boldsymbol{B}\boldsymbol{u}(k) \\ \boldsymbol{y}(k) &= \boldsymbol{C}\boldsymbol{x}(k)+\boldsymbol{D}\boldsymbol{u}(k) \end{cases} \tag{3.7}$$

the equivalent SIF is given with $\boldsymbol{P}=\boldsymbol{A},\boldsymbol{Q}=\boldsymbol{B},\boldsymbol{R}=\boldsymbol{C},\boldsymbol{S}=\boldsymbol{D}$ where $\boldsymbol{J},\boldsymbol{K},\boldsymbol{L},\boldsymbol{M},\boldsymbol{N}$ are empty matrices, as a state-space structure has no temporary variables ($l=0$).
Conversely, any SIF $\{\boldsymbol{J},\boldsymbol{K},\ldots\boldsymbol{S}\}$ is equivalent in infinite precision to a state-space filter (3.7) with:

$$\begin{aligned} \boldsymbol{A} &= \boldsymbol{K}\boldsymbol{J}^{-1}\boldsymbol{M}+\boldsymbol{P}, \quad \boldsymbol{B} &= \boldsymbol{K}\boldsymbol{J}^{-1}\boldsymbol{N}+\boldsymbol{Q}, \\ \boldsymbol{C} &= \boldsymbol{L}\boldsymbol{J}^{-1}\boldsymbol{M}+\boldsymbol{R}, \quad \boldsymbol{D} &= \boldsymbol{L}\boldsymbol{J}^{-1}\boldsymbol{N}+\boldsymbol{S}. \end{aligned} \tag{3.8}$$

This transformation to the state-space structure often helps to simplify the analysis of SIF. Moreover, it serves to compute the *transfer function* that corresponds to a filter described with SIF: once a filter described with SIF is transformed to a state-space using (3.8), we can apply classical formula

$$\boldsymbol{H}(z)=\boldsymbol{D}+\boldsymbol{C}(z\boldsymbol{I}-\boldsymbol{A})^{-1}\boldsymbol{B} \tag{3.9}$$

and compute the transfer function matrix. Thus, we can determine the frequency domain representation of a filter.

However, an accuracy issue arises: the computations in both (3.8) and (3.9) are generally not performed exactly, and in some cases their naive floating-point evaluation may yield to significant roundoff errors. Thus, the computed transfer function may *not exactly correspond* to the initial structure. We will address this issue in our contributions in Chapter 5.

The possibilities of conversions between SIF and filter structures before our work are illustrated on Figure 3.2.

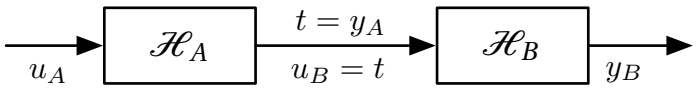


Figure 3.3: Simple cascade using SIF

3.2.2 Cascades of SIF

Consider two systems: $\mathscr{H}_A = \{\boldsymbol{J}_A, \boldsymbol{K}_A, \ldots, \boldsymbol{S}_A\}$ with inputs $\boldsymbol{u}_A$ and outputs $\boldsymbol{y}_A$; and $\mathscr{H}_B = \{\boldsymbol{J}_B, \boldsymbol{K}_B, \ldots, \boldsymbol{S}_B\}$ with inputs $\boldsymbol{u}_B$ and outputs $\boldsymbol{y}_B$. If the size of $\boldsymbol{y}_A$ is equal to the size of $\boldsymbol{u}_B$, then the systems can be *cascaded*, i.e. the outputs of the first system are reconnected with inputs of the second system.

Using SIF, this classic cascade can be done by adding a new temporary vector $\boldsymbol{t}(k+1) := \boldsymbol{y}_A(k)$, and then replacing everywhere $\boldsymbol{u}_B(k)$ by $\boldsymbol{t}(k+1)$. In other words, the outputs of the system $\mathscr{H}_A$ and inputs of $\mathscr{H}_B$ will pass to the temporary variable. This process is illustrated on Figure 3.3. By writing corresponding SIF equations (we refer the reader to [4, 98] for a step-by-step description) we can obtain that a cascaded SIF has the following coefficient matrix:

$$\boldsymbol{Z}_C = \left(\begin{array}{ccc:ccc} -\boldsymbol{J}_A & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{M}_A & \boldsymbol{0} & \boldsymbol{N}_A \\ \boldsymbol{L}_A & -\boldsymbol{I} & \boldsymbol{0} & \boldsymbol{R}_A & \boldsymbol{0} & \boldsymbol{S}_A \\ \boldsymbol{0} & \boldsymbol{N}_B & -\boldsymbol{J}_B & \boldsymbol{0} & \boldsymbol{M}_B & \boldsymbol{0} \\ \hdashline \boldsymbol{K}_A & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{P}_A & \boldsymbol{0} & \boldsymbol{Q}_A \\ \boldsymbol{0} & \boldsymbol{Q}_B & \boldsymbol{K}_B & \boldsymbol{0} & \boldsymbol{P}_B & \boldsymbol{0} \\ \hdashline \boldsymbol{0} & \boldsymbol{S}_B & \boldsymbol{L}_B & \boldsymbol{0} & \boldsymbol{R}_B & \boldsymbol{0} \end{array}\right). \tag{3.10}$$

We refer the reader to [4, 98] for more detailed description of SIF properties.

3.3 Conclusion

Considerable amount of work was done in the foundation and development of a reliable automatic filter code generator. However, the current generator still lacks its kernel functionality for the reliable implementation in FxP Arithmetic. Some work should be done towards a better way to represent any structure with the SIF. In the following, we are going to seek to improve the generator, extend its functionalities and provide kernel algorithms for the reliable implementation of linear filters in the FxP Arithmetic. On top of that, we aim at providing a single tool that incorporates all stages of the generator.

PART II

IMPROVEMENTS TO THE SPECIALIZED IMPLICIT FORM

CHAPTER 4

SPECIALIZED IMPLICIT FORM FOR LATTICE WAVE DIGITAL FILTERS

To improve and further develop the functionalities of the unifying framework, we started by describing a new structure with the Specialized Implicit Form (SIF) representation. We chose Lattice Wave Digital filters [99] (LWDF) as the target structure. As we shall see, these Single Input Single Output filters have numerous advantages and are widely used in practical applications. Lattice Wave algorithms are usually described with block-diagrams, and their coefficients are computed not with the transfer function but with a bilinear transformation of the analog Lattice Wave filters [18], i.e. via mapping the analog plane into the digital plane.

In this rather technical Chapter we show how to convert LWDF to SIF, i.e. how to express in the analytical SIF representation the order of computations in LWDF block diagrams. First, out of the coefficients of LWDFs we need to design the realization, i.e. determine which basic building blocks of LWDF to use (this will depend on the coefficient values), their order, etc. Then, we will need to express the computations of the overall LWDF in SIF. We propose a generic algorithm that, given the coefficients of LWDFs, performs those two tasks “on the fly” and returns the corresponding SIF realization.

We must admit that this Chapter is extremely technical in the sense that it is full of subtle manipulations with graph theoretical and analytical representations of filters. We first show how data-flow graphs are usually interpreted in SIF formalism. However, to facilitate several sub-steps of the conversion algorithm, we will have to come up with a modification of the usual interpretation algorithms.

This work led to a publication at the European Signal Processing Conference (EUSIPCO) in 2015 [7].

4.1 Lattice Wave Digital Filters

LWDF is a class of IIR digital filters that have several good properties, such as stability of implementation, possibility of suppression of parasitic oscillations [99] and possibility of construction

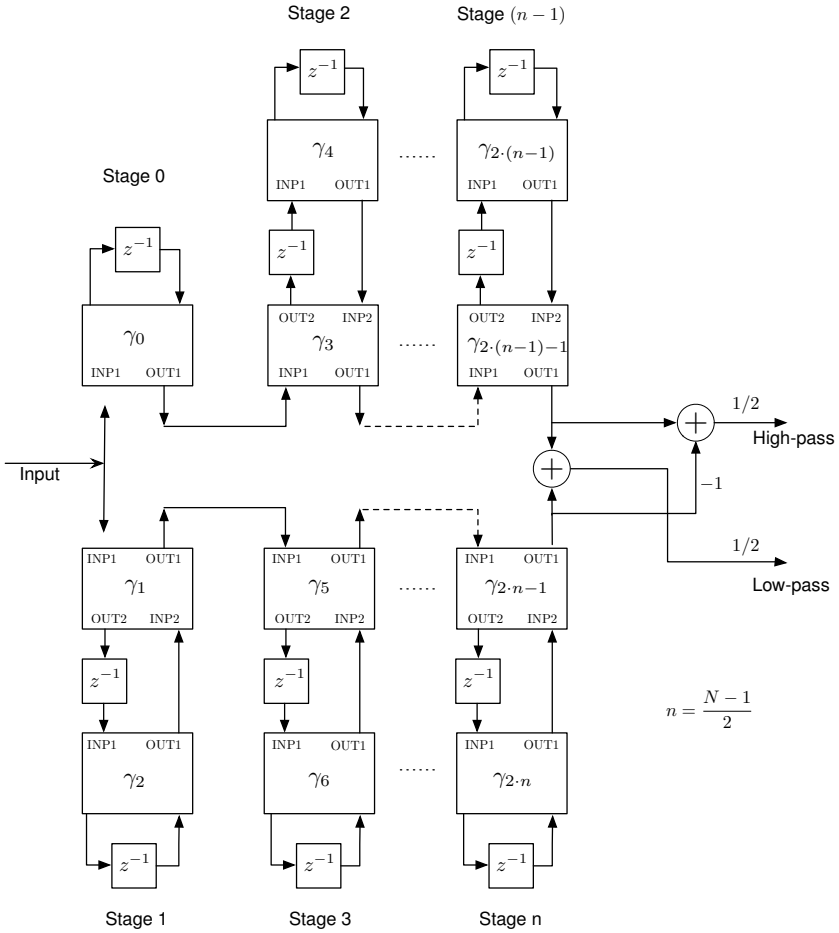


Figure 4.1: Block diagram of a N^{th} order LWDF filter.

of linear-phase designs [100]. LWDF can be either derived from analog reference filters [99] or using explicit formulas [101].

The LWDF structure is highly modular and can be easily parallelized, which makes it suitable for Very Large Scale Implementations (VLSI). Its stability qualities [99] make it a good candidate for adaptive filtering [102] and Hilbert transformations design [103].

The general block diagram of the LWDF is illustrated on Figure 4.1. In its block-diagram, an LWDF is represented by two parallel branches which realize all-pass filters [22, 104]. An all-pass filter has its magnitude response equal to one for all frequencies, so it treats all frequencies identically with respect to gain. In terms of the transfer function form, all-pass means that all poles and zeros come in conjugate reciprocal pairs [22].

The basic bricks of each branch are called two-port adaptors [22]. Each adaptor contains

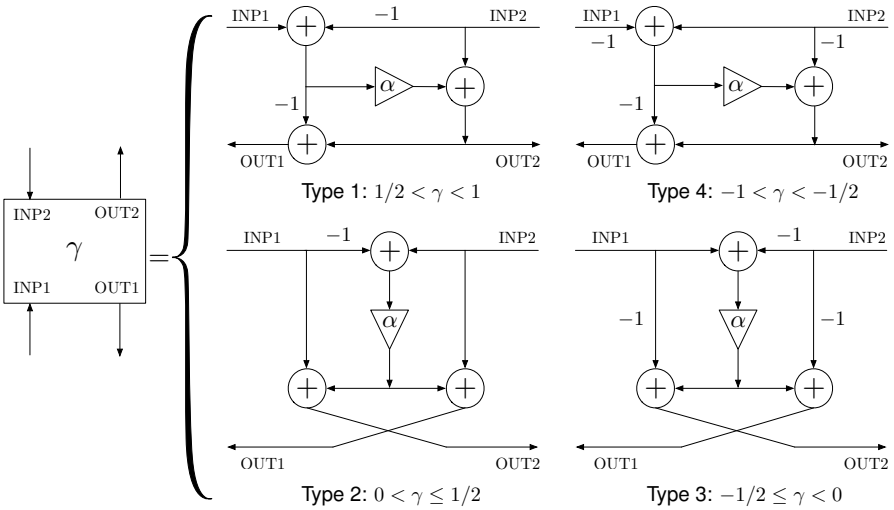


Figure 4.2: Two-port adaptor structures, for which an actual multiplier factor α is computed out of γ using Table 4.1.

three adders and one multiplier. According to [101], the multiplier coefficients γ must fall into the interval $-1 < \gamma < 1$ to guarantee the stability of the LWDF filter. To simplify the multipliers, it was proposed to use Richard's structures for the adaptors [99]. The dynamic range of γ is divided into four parts, and four different adaptor structures are used depending on the value of γ. This way multiplication by $0 < \alpha \leq 1/2$ (instead of γ) can be optimized in hardware implementations [99]. The block diagrams[1] of the two-ports are illustrated on Figure 4.2 and the conventional correspondence between the γ and α coefficients is summarized in Table 4.1.

Type	γ **range**	**Value of** α
1	$1/2 < \gamma < 1$	$\alpha = 1 - \gamma$
2	$0 < \gamma \leq 1/2$	$\alpha = \gamma$
3	$-1/2 \leq \gamma < 0$	$\alpha = \lvert\gamma\rvert$
4	$-1 < \gamma < -1/2$	$\alpha = 1 + \gamma$

Table 4.1: γ to α conversion for different γ ranges.

The transfer function of the low-pass LWDF can be expressed through the transfer functions of two stable all-pass filters corresponding to the upper and lower branch:

$$H(z) = \frac{1}{2}(H_1(z) + H_2(z)), \tag{4.1}$$

[1] Here we used the "-1" on the data-flow arcs to indicate a change of sign before the addition.

where $H_1(z)$ and $H_2(z)$ are stable all-pass filters of upper and lower branches. The frequency response can be written as

$$H\left(e^{j\omega T}\right) = \frac{1}{2}\left(e^{j\angle H_1(\omega T)} + e^{j\angle H_2(\omega T)}\right) \tag{4.2}$$

where $\angle H_1(\omega T)$ and $\angle H_2(\omega T)$ are the phase responses of $H_1(z)$ and $H_2(z)$ respectively. Therefore the magnitude of the overall filter is limited by 1.

It was shown in [99] that in order to make sure that only one passband and only one stop-band occur, the orders of the upper and lower branches must differ by one. Therefore the overall order n of the filter is odd. The high-pass filter may be simultaneously obtained by changing the sign of the all-pass lower branch. Band-pass and band-stop filters are obtained by cascading low- and high-pass filters.

Usually, wave structures are derived from analog filters: first, a reference analog filter is designed out of frequency specifications and then it is transformed to a digital filter. For several common filter design methods, such as Butterworth, Cauer (Elliptic) and Chebyshev, explicit formulas for LWDF coefficients exist [101].

Due to their good qualities, LWD filters are considered in numerous different applications, including studies on linear-phase structures [105], design of multiplierless LWDFs [106] and energy-efficient structures [107]. However, all studies on lattice wave structures implementation in finite word-length arithmetic are performed *a posteriori*, i.e. when the implementation parameters are known [108]. The implementation of LWDF is often based on two- or three-step algorithms: first, a coefficient quantization scheme based on solving optimization problems for infinite-precision filter models is developed, and then it is adjusted for a finite-precision filter. These models are specific to the LWDF and are not really suitable for a fair comparison with other structures. Hence, the problem in front of us can be formulated as:

Problem

Given a set of coefficients γ that correspond to a Lattice Wave Digital filter, determine its analytical SIF representation as the set of coefficient matrices $\boldsymbol{J},\ldots,\boldsymbol{S}$.

4.2 A LWDF-to-SIF conversion algorithm

The main idea is to exploit the modularity of the LWDFs. We propose to first divide the structure into small “building blocks”. Then, manually translate data-flow graphs of those “building blocks” to SIF representation (as we shall see, there will by only a few types of blocks, so the work is not time-consuming). Finally, we can build the SIF representation for the overall filter by simply cascading the SIFs of corresponding “building blocks”. In the following we give details of this rather technical process.

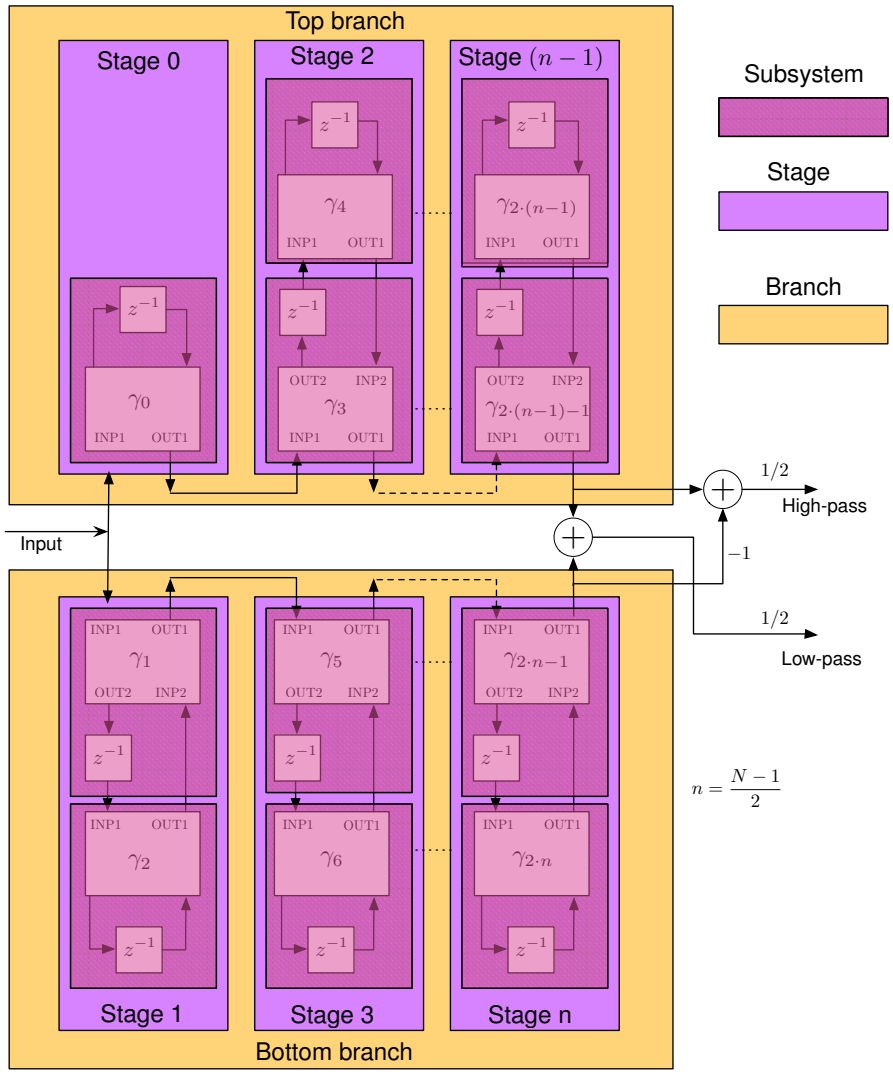
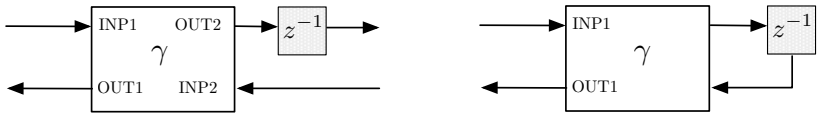


Figure 4.3: Division of a LWDF structure into subsystems, stages and branches.

As seen on Figure 4.1, a LWDF consists of two branches, and each branch is a cascade of stages. Each stage may be considered as a cascade of *subsystems* of two types, which are shown on Figure 4.4. In a stage there may be one or two subsystems, no more and no less. We denote a subsystem with 1 input and 1 output as Type A. The subsystem with 2 inputs and 2 outputs is denoted Type B. The overall decomposition of a LWDF into branches, stages and subsystems is illustrated on Figure 4.3.

Depending on the value of the coefficient γ, and therefore the type of 2-port adaptor, we obtain 8 possible subsystems (they are our "building blocks"). For example, the Type A subsystem with adaptor of Type 1 will be called Type 1-A, etc.



(a) Type A (b) Type B

Figure 4.4: Subsystems of Type A and B built out of two-port adaptors.

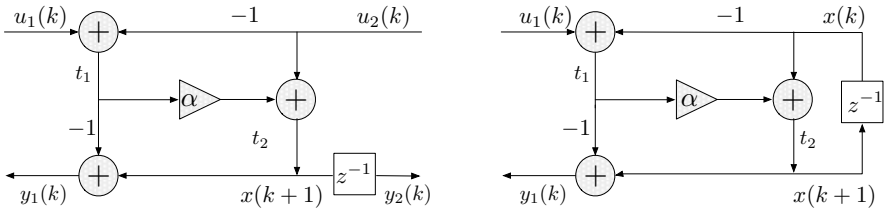
Given the filter's coefficients γ_i, the conversion algorithm can be divided into four steps:

1. According to the value of the coefficients γ_i, deduce the SIFs for basic brick subsystems;
2. Cascade the subsystems into stages;
3. Cascade the stages into the lower and upper branches;
4. Combine the two branches into the final system.

In the following we describe the algorithms for each stage.

Subsystem conversion: this can be done by applying SIF notation to the block diagrams of the subsystems. Data-flow graphs for the Type 1-A and 1-B subsystems are given on Figure 4.5. For a complete reference on all subsystems, see Appendix 1. We have annotated the graphs in the following way:

- results of additions and multiplications are temporary variables;
- delayed variables are states;



(a) Type 1-A (b) Type 1-B

Figure 4.5: Subsystems for adaptors of Type 1.

By representing these annotated block diagrams for Type 1-A and 1-B subsystems as equations, we obtain:

$$\mathscr{H}_{1A}\left\{\begin{array}{rcl} \boldsymbol{t}_1^A(k+1) &=& \boldsymbol{u}_1^A(k)-\boldsymbol{u}_2^A(k)\\ \boldsymbol{t}_2^A(k+1) &=& \alpha\boldsymbol{t}_1^A(k)+\boldsymbol{u}_2^A(k)\\ \boldsymbol{x}^A(k+1) &=& \boldsymbol{t}_2^A(k+1)\\ \boldsymbol{y}_1^A(k) &=& -\boldsymbol{t}_1^A(k+1)+\boldsymbol{t}_2^A(k+1)\\ \boldsymbol{y}_2^A(k) &=& \boldsymbol{x}^A(k)\end{array}\right. \qquad \mathscr{H}_{1B}\left\{\begin{array}{rcl} \boldsymbol{t}_1^B(k+1) &=& \boldsymbol{u}_1^B(k)-\boldsymbol{x}^B(k)\\ \boldsymbol{t}_2^B(k+1) &=& \alpha\boldsymbol{t}_1^B(k)+\boldsymbol{x}^B(k)\\ \boldsymbol{x}^B(k+1) &=& \boldsymbol{t}_2^B(k+1)\\ \boldsymbol{y}^B(k) &=& -\boldsymbol{t}_1^B(k+1)+\boldsymbol{t}_2^B(k+1)\end{array}\right. , \tag{4.3}$$

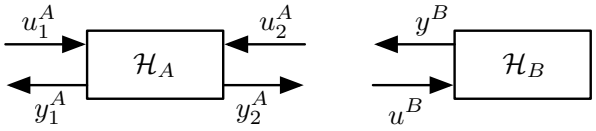


Figure 4.6: Connection between subsystems to form a Stage of a LWDF.

which can be represented as:

$$\mathscr{H}_{1A}: \quad \left(\begin{array}{ccc:cc} 1 & 0 & 0 & 0 & 0 \\ -\alpha & 1 & 0 & 0 & 0 \\ \hdashline 0 & -1 & 1 & 0 & 0 \\ \hdashline 1 & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{array}\right) \begin{pmatrix} \boldsymbol{t}_1^A(k+1) \\ \boldsymbol{t}_2^A(k+1) \\ \boldsymbol{x}^A(k+1) \\ \boldsymbol{y}_1^A(k) \\ \boldsymbol{y}_2^A(k) \end{pmatrix} = \left(\begin{array}{ccc:cc} 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 & 1 \\ \hdashline 0 & 0 & 0 & 0 & 0 \\ \hdashline 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{array}\right) \begin{pmatrix} \boldsymbol{t}_1^A(k+1) \\ \boldsymbol{t}_2^A(k+1) \\ \boldsymbol{x}^A(k) \\ \boldsymbol{u}_1(k) \\ \boldsymbol{u}_2(k) \end{pmatrix} \tag{4.4}$$

$$\mathscr{H}_{1B}: \quad \left(\begin{array}{ccc:c} 1 & 0 & 0 & 0 \\ -\alpha & 1 & 0 & 0 \\ \hdashline 0 & -1 & 1 & 0 \\ \hdashline 1 & -1 & 0 & 1 \end{array}\right) \begin{pmatrix} \boldsymbol{t}_1^B(k+1) \\ \boldsymbol{t}_2^B(k+1) \\ \boldsymbol{x}^B(k+1) \\ \boldsymbol{y}^B(k) \end{pmatrix} = \left(\begin{array}{cc:cc} 0 & 0 & -1 & 1 \\ 0 & 0 & 1 & 0 \\ \hdashline 0 & 0 & 0 & 0 \\ \hdashline 0 & 0 & 0 & 0 \end{array}\right) \begin{pmatrix} \boldsymbol{t}_1^B(k+1) \\ \boldsymbol{t}_2^B(k+1) \\ \boldsymbol{x}^B(k) \\ \boldsymbol{u}_1(k) \end{pmatrix} .. \tag{4.5}$$

Thus, we can deduce the SIF matrices $\boldsymbol{Z}_{1A}$ and $\boldsymbol{Z}_{1B}$ that correspond to the systems $\mathscr{H}_{1A}$ and $\mathscr{H}_{1B}$:

$$\boldsymbol{Z}_{1A} = \left(\begin{array}{cc:cc:c} -1 & 0 & 0 & 1 & -1 \\ \alpha & -1 & 0 & 0 & 1 \\ \hdashline 0 & 1 & 0 & 0 & 0 \\ \hdashline -1 & 1 & 0 & 0 & 0 \\ \hdashline 0 & 0 & 1 & 0 & 0 \end{array}\right), \quad \boldsymbol{Z}_{1B} = \left(\begin{array}{cc:cc} -1 & 0 & -1 & +1 \\ \alpha & -1 & 1 & 0 \\ \hdashline 0 & 1 & 0 & 0 \\ \hdashline -1 & -1 & 0 & 0 \end{array}\right). \tag{4.6}$$

Cascade subsystems into stages: we need to perform a so-called "circular" cascade of a subsystem of Type A (MIMO system with 2 inputs and 2 outputs) with a subsystem of Type B (SISO system) in the following way: output $\boldsymbol{y}_2^A(k)$ is connected with input $u^B(k)$ and output $y^B(k)$ is connected with input $u_2^A(k)$. Figure 4.6 graphically illustrates the operations that need to be performed. This operation cannot be done using the classic cascade formula (3.10).

Hence, we propose a new algorithm for such a circular cascade. According to the cascading principle from Chapter 3 Section 3.2.2, each input and output of the subsystems will eventually pass into the temporary vector of the final SIF. To facilitate the manipulation of subsystems and minimize the number of trivial temporary variables ("trivial" in the sense of simple reassignment, e.g. $\boldsymbol{t}(k+1) = \boldsymbol{u}(k+1)$) in the SIF matrix after cascades, we came up with a technical "trick". We modify the subsystem conversion such that intermediate subsystems no longer represent meaningful filters in SIF representation but some systems with the input and output variables passed into the temporary vector. Then, cascading two subsystems boils down to simply writing the equations corresponding to both subsystems together and assigning a few temporary variables. In the end of the general conversion algorithm we "reconnect" the output of the filter

with the corresponding temporary variable and "assign" the input variable to the corresponding temporary variable.

This convention is just a technical "trick" that facilitates the automation of the conversion algorithm and does not change in any manner the order of computations of the initial system.

Remark 4.1. *The reader might rightfully find it frustrating to revisit and change the subsystem conversion algorithm. However, the revisited conversion will facilitate the cascading process.*

Subsystem conversion revisited: our approach is better explained with an example. Consider again a Type 1 adaptor, for which we build intermediate subsystems $\mathscr{H}_{1A}$ and $\mathscr{H}_{1B}$. Instead of (4.3) we write:

$$\mathscr{H}_{1A}\begin{cases} \boldsymbol{t}^A_{u_1}(k+1) &= 0 \\ \boldsymbol{t}^A_{u_2}(k+1) &= 0 \\ \boldsymbol{t}^A_{1}(k+1) &= \boldsymbol{t}^A_{u_1}(k) - \boldsymbol{t}^A_{u_2}(k) \\ \boldsymbol{t}^A_{2}(k+1) &= \alpha^A \boldsymbol{t}_1(k) + \boldsymbol{t}^A_{u_2}(k) \\ \boldsymbol{t}^A_{y_1}(k+1) &= -\boldsymbol{t}^A_{1}(k+1) + \boldsymbol{t}^A_{2}(k+1) \\ \boldsymbol{t}^A_{y_2}(k+1) &= \boldsymbol{x}^A(k) \\ \boldsymbol{x}^A(k+1) &= \boldsymbol{t}^A_{2}(k+1) \end{cases} \qquad \mathscr{H}_{1B}\begin{cases} \boldsymbol{t}^B_{u}(k+1) &= 0 \\ \boldsymbol{t}^B_{1}(k+1) &= \boldsymbol{t}^B_{u_1}(k+1) - \boldsymbol{x}^B(k) \\ \boldsymbol{t}^B_{2}(k+1) &= \alpha^B \boldsymbol{t}^B_{1}(k) + \boldsymbol{x}^B(k) \\ \boldsymbol{t}^B_{y}(k+1) &= -\boldsymbol{t}^B_{1}(k+1) + \boldsymbol{t}^B_{2}(k+1) \\ \boldsymbol{x}^B(k+1) &= \boldsymbol{t}^B_{2}(k+1) \end{cases} . \tag{4.7}$$

Here we replaced all occurrences of the input $\boldsymbol{u}$ by temporary variables that will be later connected with outputs if other subsystems during the cascade. The computations that correspond to the outputs are also moved into a temporary vector.

New circular cascade algorithm: now the circular cascade boils down to the manipulation with the temporary variables as shown on Figure 4.7.

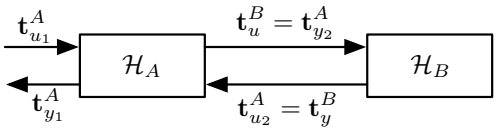


Figure 4.7: Cascade of revisited subsystems is just manipulation over temporary variables.

Formally, to cascade systems $\mathscr{H}_{1A}$ and $\mathscr{H}_{1B}$ into the system $\mathscr{H}_C$ we do the following:

- regroup all the vectors into the cascaded system $\mathscr{H}_C$ with coefficients Z_C as follows:

$$Z_C = \left(\begin{array}{cc:cc:c} -\boldsymbol{J}_A & \boldsymbol{0} & \boldsymbol{M}_A & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & -\boldsymbol{J}_B & \boldsymbol{0} & \boldsymbol{M}_B & \boldsymbol{0} \\ \hdashline -\boldsymbol{K}_A & \boldsymbol{0} & \boldsymbol{P}_A & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & -\boldsymbol{K}_B & \boldsymbol{0} & \boldsymbol{P}_B & \boldsymbol{0} \\ \hdashline \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \end{array}\right) \tag{4.8}$$

- do the circular connection by assigning $\boldsymbol{t}^B_u = \boldsymbol{t}^A_{y_2}$ and $\boldsymbol{t}^A_{u_2} = \boldsymbol{t}^B_y$ in the resulting system $\mathscr{H}_C$.

The last step is possible thanks to the “trick” with the temporary variables in revised basic brick conversion. We can compare this new cascading to the classic formula from Chapter 3 Section 3.2.2 and see that in the new cascading algorithm there are no additional temporary variables added to do the connection of inputs/outputs.

However, the drawback of this approach, is that after the circular cascade, matrix $\boldsymbol{J}_C$ loses its lower triangular form: the assignment $\boldsymbol{t}^A_{u_2} = \boldsymbol{t}^B_y$ violates the lower-triangularity of $\boldsymbol{J}_C$.

The lower-triangular form of the matrix $\boldsymbol{J}$ in any SIF denotes the order of operations in the filter: first, $\boldsymbol{t}_1$ is computed, then $\boldsymbol{t}_2$ may depend on $\boldsymbol{t}_1$ and so on. In other words, $\boldsymbol{t}_i$ depends only on $\boldsymbol{t}_k$ with $k < i$. We can interpret $\boldsymbol{J} - \boldsymbol{I}$ (we subtract the identity matrix to avoid cycles) as a Directed Acyclic Graph. In this graph $\boldsymbol{t}_i$ are the vertices and dependencies between temporary variables are edges. Therefore, a topological sort [109] of the graph can be performed: the vertices of the graph are ordered such that for every directed edge $\boldsymbol{t}_k\boldsymbol{t}_l$ for vertex $\boldsymbol{t}_k$ to vertex $\boldsymbol{t}_l$, $\boldsymbol{t}_k$ comes before $\boldsymbol{t}_l$ in the ordering. We perform Depth-First Search [109] sort to order the graph, i.e. return matrix $\boldsymbol{J}$ to its lower-triangular form. The sorting algorithm guarantees by construction that the cascaded system correctly describes the order of computations in the LWDF structure.

Algorithm II.1 illustrates our approach for the cascade of subsystems.

Algorithm II.1: `cascadeSubsystems`: cascade of LWDF subsystems into a stage

Input: $\mathscr{H}_A$ (with 2 inputs, 2 outputs and l_A intermediate variables)
$\mathscr{H}_B$ (with 1 input, 1 output and l_B intermediate variables)
Output: $\mathscr{H}_C$ (with 1 input, 1 output and $l_A + l_B$ intermediate variables)

1 $\{\boldsymbol{J}_C, \boldsymbol{K}_C, \ldots, \boldsymbol{S}_C\} \leftarrow$ `apply` (4.8) `upon` $\mathscr{H}_A$ `and` $\mathscr{H}_B$
2 $i_{u^B} \leftarrow$ `index of` $\boldsymbol{t}^B_u$ `in` $\boldsymbol{J}_C$
3 $i_{y_2{}^A} \leftarrow$ `index of` $\boldsymbol{t}^A_{y_2}$ `in` $\boldsymbol{J}_C$
4 $i_{u_2{}^A} \leftarrow$ `index of` $\boldsymbol{t}^A_{u_2}$ `in` $\boldsymbol{J}_C$
5 $i_{y^B} \leftarrow$ `index of` $\boldsymbol{t}^B_y$ `in` $\boldsymbol{J}_C$
6 $\boldsymbol{J}_C(i_{u^B}, i_{y_2{}^A}) \leftarrow 1$
7 $\boldsymbol{J}_C(i_{u_2{}^A}, i_{y^B}) \leftarrow 1$
8 $\boldsymbol{J}_C \leftarrow$ `TopologicalSort`$(\boldsymbol{J}_C)$
9 **return** $SIF(\{\boldsymbol{J}_C, \boldsymbol{K}_C, \ldots, \boldsymbol{S}_C\})$

Cascading stages into branches: analogously, it is straightforward to derive an algorithm for the classic sequential cascade of our revisited subsystems, see Algorithm II.2. Based on this algorithm, the cascade of stages into branches can be derived. At this point we can "reconnect" the input variable $\boldsymbol{u}(k)$ of the branch with the first element of the vector $\boldsymbol{t}(k+1)$ (which implicitly represents the input). For this, we just need to set $\boldsymbol{t}_1(k+1) = \boldsymbol{u}(k)$. See Algorithm II.3 for details.

Algorithm II.2: `sequentialCascade`: sequential cascade of two SIFs

Input: $\mathscr{H}_A$ (with 1 input, 1 output and l_A intermediate variables)
$\mathscr{H}_B$ (with 1 input, 1 output and l_B intermediate variables)
Output: $\mathscr{H}$ (with 1 input, 1 output and $l_A + l_B$ intermediate variables)

$\{\boldsymbol{J},\boldsymbol{K},\ldots,\boldsymbol{S}\} \leftarrow$ `apply` (4.8) `upon` $\mathscr{H}_A$ `and` $\mathscr{H}_B$
$i_{y^A} \leftarrow$ `index of` $\boldsymbol{t}_y^A$ `in` $\boldsymbol{J}$
$i_{u^B} \leftarrow$ `index of` $\boldsymbol{t}_u^B$ `in` $\boldsymbol{J}$
$\boldsymbol{J}(i_{u^B}, i_{y^A}) \leftarrow 1$
$\boldsymbol{J}_C \leftarrow$ `TopologicalSort`$(\boldsymbol{J}_C)$
return $\{\boldsymbol{J},\boldsymbol{K},\ldots,\boldsymbol{S}\}$

Algorithm II.3: `cascadeStages`: cascade of LWDF stages into a branch

Input: stages $\mathscr{H}_i$, $i = 1,\ldots,s$ (with 1 input, 1 output and l_i intermediate variables)
Output: branch $\mathscr{H}$ (with 1 input, 1 output and $\sum_{i=1}^{s} l_i$ intermediate variables)

$\mathscr{H} \leftarrow \mathscr{H}_1$
for $i \leftarrow 2$ **to** s **do**
  $\{\boldsymbol{J},\boldsymbol{K},\ldots,\boldsymbol{S}\} \leftarrow$ `sequentialCascade`$(\mathscr{H},\mathscr{H}_i)$ `// here we use Algorithm II.2`
  $\mathscr{H} \leftarrow$ `SIF` $(\{\boldsymbol{J},\boldsymbol{K},\ldots,\boldsymbol{S}\})$
end
$\boldsymbol{N}(1,1) \leftarrow 1$ `// assign the input to the temporary variable`
$\mathscr{H} \leftarrow$ `SIF` $(\{\boldsymbol{J},\boldsymbol{K},\ldots,\boldsymbol{S}\})$
return $\mathscr{H}$

Cascading the branches into the LWDF filter: finally, we combine the outputs of the top and bottom branches to obtain a lowpass or highpass filter (see Figure 4.1). Denote by $\mathscr{H}_1$ the top branch (with 1 input, 1 output, l_1 intermediate variables) and $\mathscr{H}_2$ the bottom branch (with 1 input, 1 output, l_2 intermediate variables). By construction, the outputs of the branches are zero vectors and the actual outputs are stored in the last element of the temporary vector. Then, the idea is to add a temporary variable, which reflects the addition (subtraction) of two branches and then to link the output of the combined system with this new temporary variable.
Then, the SIF for a LWDF filter is:

$$\boldsymbol{Z} = \left(\begin{array}{ccc:ccc} -\boldsymbol{J}_1 & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{M}_1 & \boldsymbol{0} & \boldsymbol{N}_1 \\ \boldsymbol{0} & -\boldsymbol{J}_2 & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{M}_2 & \boldsymbol{N}_2 \\ \boldsymbol{t}_1 & \boldsymbol{t}_2 & -1 & \boldsymbol{0} & \boldsymbol{0} & 0 \\ \hdashline \boldsymbol{K}_1 & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{P}_1 & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & -\boldsymbol{K}_2 & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{P}_2 & \boldsymbol{0} \\ \hdashline \boldsymbol{0} & \boldsymbol{0} & -\frac{1}{2} & \boldsymbol{0} & \boldsymbol{0} & 0 \end{array}\right) \tag{4.9}$$

where

$$\boldsymbol{t}_1 = (\underbrace{0,\ldots,0}_{l_1},1) \qquad \boldsymbol{t}_2 = \begin{cases} (\underbrace{0,\ldots,0}_{l_2},1) & \text{if highpass} \\ (\underbrace{0,\ldots,0}_{l_2},-1) & \text{if lowpass} \end{cases} . \tag{4.10}$$

Algorithm II.4 illustrates this final stage of the conversion.

Algorithm II.4: `cascadeBranches`: cascade of LWDF branches into a lowpass/highpass filter

Input: $\mathscr{H}_1$ top-branch (with 1 input, 1 output and l_1 intermediate variables)
$\mathscr{H}_2$ bottom-branch (with 1 input, 1 output and l_2 intermediate variables)
Output: SIF $\mathscr{H}$

$\boldsymbol{t}_1 \leftarrow (0,\ldots,0,1)$ `// with` l_1 `zeros`
$\boldsymbol{t}_2 \leftarrow (0,\ldots,0,1)$ `// with` l_2 `zeros`
if *lowpass* **then**
  $\boldsymbol{t}_2 \leftarrow -\boldsymbol{t}_2$
end
$\{\boldsymbol{J},\boldsymbol{K},\ldots,\boldsymbol{S}\} \leftarrow$ `apply` (4.9) `upon` $\mathscr{H}_1$, $\mathscr{H}_2$ `and` $\boldsymbol{t}_1,\boldsymbol{t}_2$
$\mathscr{H} \leftarrow$ `SIF`$(\{\boldsymbol{J},\boldsymbol{K},\ldots,\boldsymbol{S}\})$
return $\mathscr{H}$

Final algorithm Given a filter's coefficients γ_i, $i = 0,\ldots,n$ the conversion algorithm can be divided into four steps:

1. Deduce the SIF representation for each subsystem according to its γ_i value using the revised conversion algorithm;
2. Cascade the subsystems into stages using Algorithm II.1;
3. Cascade the stages into top and bottom branches using Algorithm II.3 which is based on Algorithm II.2;
4. Combine the two branches into the final system using Algorithm II.4.

Remark 4.2. *The reader might have noticed that there may occur some amount of trivial temporary variables that are just assignments. In order to remove them, we apply a simplification algorithm which nevertheless preserves the initial order of computations in the LWDF structure.*

γ_i	0.72689	-0.79125	0.51319	-0.97439	0.30983	-0.62505	0.77261	-0.90732	0.36682
α_i	0.27311	0.20875	0.48681	0.02561	0.30983	0.37495	0.22739	0.09268	0.36682
Type	1	4	1	4	2	4	1	4	2

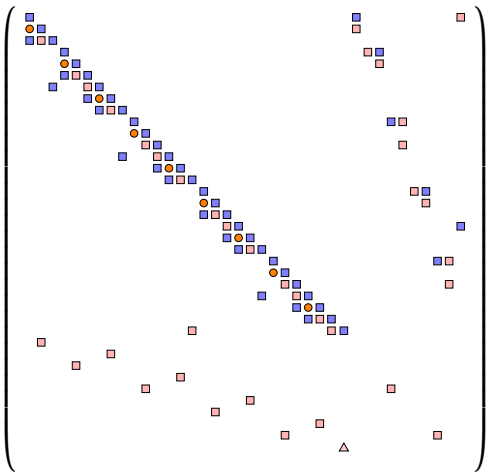
Table 4.2: Initial coefficients of LWDF and adaptor settings.

4.3 Conversion example

To provide a conversion of LWDF to SIF we must first obtain the coefficients of the Lattice Wave structure. We do this using the *Lattice Wave Digital Filter Toolbox* for Matlab [110] that was developed at TU Delft. Naturally, we implemented our conversion algorithm using Matlab as well[2]. Using Matlab here is no danger to reliability because the conversion algorithm just copies data and does not perform Floating-Point computations.

Consider the specifications of the SISO low-pass filter from Section 1.4.3 in Chapter 1. A 9^{th} order LWDF filter satisfying these specifications was obtained using the LWDF Toolbox. The first digits of its coefficients γ_i and the actually implemented adaptor coefficients α_i are given in Table 4.2. Applying the conversion algorithm described above, we obtain the SIF with coefficient matrix $\boldsymbol{Z}$ given by

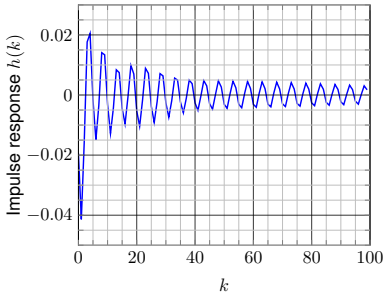
$$\boldsymbol{Z} = \left(\right. \qquad \left. \right) \tag{4.11}$$



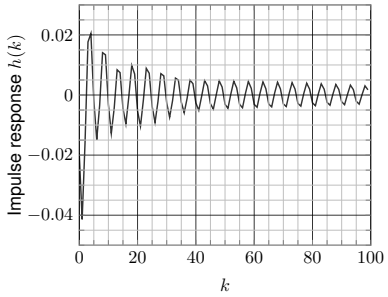
where blue and pink rectangles represent '-1' and '1' respectively and circles are the non-trivial coefficients α_i. The only triangle represents the final division by two (i.e. just a shift operation) of the sum of outputs of lower and upper branches (see Figure 4.1).

Remark 4.3. *Given the structure coefficients, no computational errors are induced during the conversion algorithm, since our algorithm simply copies them into right places in SIF matrices.*

[2]However, we also embedded the conversion algorithm into the filter code generator tool written in Python. It is based on Matlab API for Python, i.e. an installation of Matlab is required.



(a) Impulse response of the LWDF provided by the LWDF Toolbox



(b) Impulse response of the LWDF realization in SIF.

Figure 4.8: Comparison of the impulse responses of the LWDF and the corresponding SIF.

We can compare the SIF realization with the reference Lattice Wave filter by several means. For instance, we may trace the impulse response of both systems and compare them. Figure 4.8 demonstrates that impulse responses (at least the first 100 terms) are identical.

4.4 Conclusion

The Lattice Wave Digital filter structure has been well-studied over the years and is widely used in real-life applications. However, a fair comparison with other structures was not trivial due to the block diagram description of the filter.

We provided an algorithm that without any computational errors represents any LWDF realization in the SIF. Using LWDF coefficients, our task was to first design a LWDF realization and then represent it as an algorithm in the SIF. For this technical task we provided new system cascade algorithms for SIF. Our algorithms ensure that the initial numerical properties are preserved by construction in SIF representation.

Now, using SIF capabilities we can compare this structure fairly with other filter realizations. We refer the reader to [7] for an example of a comparison of LWDF with Direct Forms and state-space structures according to several classic signal processing quality measures. This work permitted us to more deeply understand the relationship between data-flows and SIF representation.

CHAPTER 5

GENERAL ALGORITHMS FOR CONVERSION

After taking a closer look at the Specialized Implicit Form and our "hands-on" experience with the creation of SIF corresponding to Lattice Wave filters, we are ready to propose several improvements to this unifying framework.

Before this thesis the usual user scenario of our code generator started with the choice of possible structures from a dictionary of filter realizations. Obviously, it would be too time-consuming to write algorithms for conversion of all existing structures to SIF. Another typical scenario is when we need to analyze an already existing structure described with a data-flow. Hence, we developed and implemented an algorithm for the conversion of an arbitrary linear filter described as a data-flow in Matlab Simulink format to SIF representation.

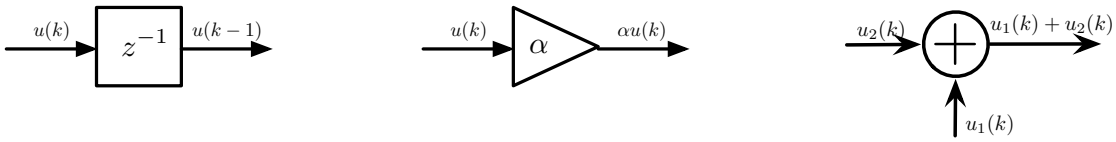
Now we can take any data-flow describing a linear filter and obtain a SIF representation. For example, we can convert an existing Simulink design of a LWDF to SIF. The main difference in this case is that we will need to design the actual data-flow graph, while the algorithm from the previous Chapter automatically created the LWDF design out of the coefficients.

The Simulink-to-SIF conversion is a significant improvement to the framework functionality, since it makes the comparison of different linear filters practical. We shall detail this point in Section 5.1. This work led to a publication [12] at the IEEE SiPS conference in 2016.

Further, we proposed an algorithm for the Multiple Precision (MP) computation of the transfer function of any linear filter given in SIF representation. First we compute a Floating-Point approximation of the transfer function and then rigorously bound the error. To bound the error, we use a well-known object, the Worst-Case Peak Gain, which is just a ℓ_1-norm of filter's impulse response. We shall detail this point in Sections 5.2 and 5.3. This work is a part of contribution that was published at the 24th IEEE ARITH Symposium in 2017 [9].

5.1 Conversion of data-flow graphs to SIF

Matlab Simulink is a widely used software tool for the model-based design of digital systems, their simulation and implementation. Engineers worldwide use Simulink for academic and industrial applications, including some for reducing fuel emissions, developing safety-critical autopilot



(a) delay (b) multiplication (gain) (c) addition

Figure 5.1: Basic blocks in Simulink block-diagrams.

software, and designing wireless LTE systems. While not giving any proof of the accuracy or guarantee on the finite-precision implementations, Simulink is no doubt a state-of-the-art tool for modeling linear filters. This conversion will enable the comparison and implementation of any data-flow describing linear filters.

5.1.1 Simulink

Simulink mainly uses a graphical block diagram to describe the model. The building blocks of such diagrams are adders, multipliers and delays (see Figure 5.1).

Internally, the block diagram is stored using extensible markup language (xml) (more specifically in `.slx` format). In this format the `<System>` tag contains the model description, and the `<Block>` and `<Line>` tags inside hold blocks of elements and their interconnections.

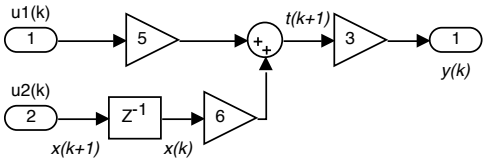


Figure 5.2: A very simple Simulink block diagram.

Consider the very simple data-flow diagram given on Figure 5.2. It consists of gain, delay and sum blocks. All the information on the block parameters can be easily retrieved by parsing the xml file. For instance, its sum and gain operators are described in xml with the following code:

```
<Block BlockType="Sum" Name="Sum" SID="3">
<P Name="Ports"> [2, 1] </P>
<P Name="Inputs"> |++ </P>
...
</Block>
```

```
<Block BlockType="Gain" Name="5" SID="4">
<P Name="Gain">5</P>
...
</Block>
```

In this example, the sum block has the identification SID=3 and the gain with constant 5 has the identification SID=4. The output of the gain is connected into the sum operator with following code:

```
<Line>
<P Name="ZOrder">7</P>
<P Name="Src">4#out:1</P>
<P Name="Dst">3#in:1</P>
</Line>
```

We see that all the information concerning the model can be easily extracted from the xml file. Thus, the problem in front of us can be formulated as follows:

Problem

Given a file that contains a Simulink model of a data-flow describing a linear digital filter, return a SIF corresponding to this data-flow.

5.1.2 Simulink-to-SIF conversion algorithm

Our goal is, given a block-diagram described with a Simulink model and stored in a file, to identify all temporary and state variables, as well as to determine the coefficients of the Sums-of-Products by Constants (SOPCs) that constitute the system. In order to do that, we assume the following set of rules:

- since each delay element represents a computation that is saved from one time instance to another, in SIF we note it as a state variable;
- the gain elements represent the coefficients of variables in the corresponding SOPC;
- from each sum element, we can deduce one equation (as a sum of its inputs);
- all equations produced by a sum block that feed a delay are state equations;
- all equations by a sum block connected to the model output are output equations;
- the others (that feed another sum blocks) are intermediate equations.

As each Simulink block has its identification number (SID), we can easily label each block and deduce the equation for each sum block by listing all connected blocks at their inputs. All chained sum blocks can be gathered into one sum with several operands. Then we have a system of equations that defines our model. In case of a subsystem present in the diagram, the design is flattened before obtaining the block equations.

The most important consideration in the Simulink to SIF conversion is the order of the computations. Different order of computations leads to different numerical properties after we pass to finite-precision arithmetic. Therefore, when we represent a data-flow in SIF, we must ensure that the order of computations stays the same.

For instance, the simple example of a block-diagram given in Figure 5.2 corresponds to the following system of equations:

$$\begin{cases} t(k+1) &= 6 \cdot x(k) + 5 \cdot u_1(k) \\ x(k+1) &= 1 \cdot u_2(k) \\ y &= 3 \cdot t(k+1) \end{cases} \tag{5.1}$$

From this system of equations, it is then straightforward to identify the matrices $\boldsymbol{J}$, $\boldsymbol{K}$, $\boldsymbol{L}$, $\boldsymbol{M}$, $\boldsymbol{N}$, $\boldsymbol{P}$, $\boldsymbol{Q}$, $\boldsymbol{R}$, $\boldsymbol{S}$ of the SIF. In our toy example $\boldsymbol{J} = \begin{pmatrix} 1 \end{pmatrix}$, $\boldsymbol{L} = \begin{pmatrix} 3 \end{pmatrix}$, $\boldsymbol{M} = \begin{pmatrix} 6 \end{pmatrix}$, $\boldsymbol{N} = \begin{pmatrix} 0 & 5 \end{pmatrix}$, $\boldsymbol{Q} = \begin{pmatrix} 0 & 1 \end{pmatrix}$ and other matrices are null.

However, the order of labeling the temporary variables during the conversion can compromise the lower triangularity of the matrix $\boldsymbol{J}$. As in the case of Lattice Wave digital filters, we return $\boldsymbol{J}$ to its lower-triangular form by interpreting $\boldsymbol{J} - \boldsymbol{I}$ as the adjacency matrix of a Directed Acyclic Graph (DAG). We perform a topological sort [109] of this DAG: if for some i the element $\boldsymbol{t}_i$ depends on $\boldsymbol{t}_j$ with $j > i$, then element $\boldsymbol{t}_j$ is placed before $\boldsymbol{t}_i$ and corresponding changes in $\boldsymbol{J}$ are done (along with changes in $\boldsymbol{N}$, $\boldsymbol{K}$ and $\boldsymbol{L}$). In other words, we restore the order of operations up to some possible permutations of operations that do not depend on one another, for example the ones that on the data-flow graph are parallel.

5.1.3 Conversion example

We implemented our algorithm in Python. It requires as only input a Simulink model description file. Given some reference test results, we can automatically test the correctness of the SIF representation via comparison (e.g. compare simulation output).

Consider the 3^{rd} order Lattice filter described with Simulink block-diagram given on Figure 5.3. Applying the general conversion algorithm, we obtain the SIF described with the following

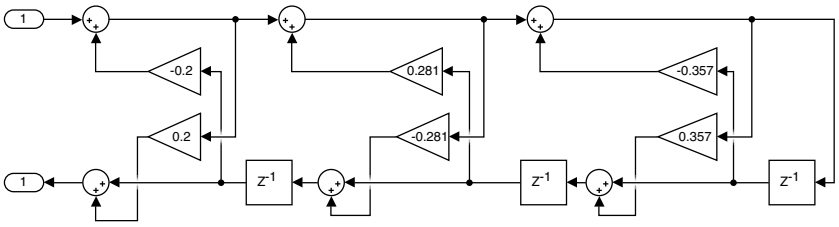


Figure 5.3: Simulink diagram of a 3^{rd} order Lattice filter.

coefficient matrix:

$$\boldsymbol{Z} = \left(\begin{array}{ccccc:c:cc:c} -1 & 0 & 0 & 0 & 0 & 0 & -0.2 & 0 & 1 \\ 1 & -1 & 0 & 0 & 0 & 0.281 & 0 & 0 & 0 \\ 0 & -0.281 & -1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 & 0 & -0.357 & 0 \\ 0 & 0 & 0 & 0.357 & -1 & 0 & 0 & 1 & 0 \\ \hdashline 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ \hdashline 0.2 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{array}\right). \tag{5.2}$$

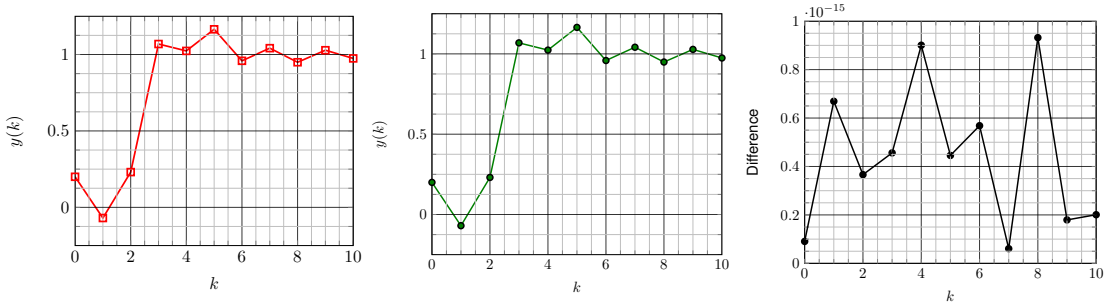
We check that both representations are equivalent by comparing the outputs of both systems. We perform some simulations of Simulink and SIF models in double precision floating-point arithmetic. For example, Figure 5.4 shows the response of the Simulink model and of the computed SIF realizations to a constant unit signal. The output seems to be the same. However, since we have no control over double precision simulation in Simulink (what rounding is used, whether Fused Multiply and Add is available, etc.), the outputs of Simulink and SIF simulations may not be exactly the same. Figure 5.4c illustrates the difference between outputs of data-flow and SIF simulations.

If the transfer function corresponding to the initial Simulink model is available, we can compare it with the transfer function of the SIF realization. However, as described in Section 3.2.1 Chapter 3, rounding errors may occur during the computation of a transfer function for the SIF realization. Thus, we need to improve the algorithm for conversion to the transfer function.

5.2 Conversion of arbitrary structures to transfer functions

Accurate determination of the transfer functions of arbitrary filters is important for two major reasons:

- to analyze an already existing filter realization (e.g. a data-flow graph with its coefficients is given but no information on the transfer function is available);



(a) Simulink output (b) SIF output (c) Difference between outputs

Figure 5.4: Comparison of responses of Simulink and SIF models to a constant unit signal.

- during implementation, coefficients in a realization may be modified (scaled, quantized, etc.). We must always test the properties of the corresponding transfer function: to check stability of the modified filter, its spectral behavior, etc.

If the structure type is determined and well-studied, there may be known closed formulas for the transfer function computation (e.g. Lattice Wave filters, state-space). For some structures, such as Direct Forms, determining corresponding transfer function is straightforward: the structure coefficients are the same as those of the transfer function. Then, for example, after quantization of the coefficients in the data-flow graph, the transfer function corresponding to quantized filter can be easily computed.

However, if the structure type is not given, other approaches must be used. In [111, 112] authors propose various algorithms for the determination of a transfer function by analyzing the data-flow graph. However, no error bound with respect to a certain transfer function norm is given. In Matlab Simulink, extraction of a model's transfer function is based on sampling the frequency response of the system and then interpolation. Obviously, such approach may be prone to computational and approximation errors.

Therefore, our goal is to propose an approach for reliable computation of a transfer function corresponding to an arbitrary linear data-flow graph; reliable in the sense that the computed transfer function is proven to satisfy some error bound (with respect to a certain given norm).

Since any linear data-flow graph can be represented as a SIF structure, it is enough to propose a conversion algorithm for a SIF.

5.2.1 Conversion of a SIF to Transfer Function

For simplicity of demonstration, in this Chapter we consider only SISO systems, i.e. with 1 input and 1 output. However, our reasoning is easily generalized for the case of MIMO filters.

Let $\mathscr{R} = (\boldsymbol{J}, \boldsymbol{K}, \ldots, s)$ be a realization of a filter described with a SIF. Its coefficient matrix is

$$\boldsymbol{Z} = \begin{pmatrix} -\boldsymbol{J} & \boldsymbol{M} & \boldsymbol{n} \\ \boldsymbol{K} & \boldsymbol{P} & \boldsymbol{q} \\ \boldsymbol{l} & \boldsymbol{r} & s \end{pmatrix} \tag{5.3}$$

where, since the system has 1 output, $\boldsymbol{n}$ and $\boldsymbol{q}$ are column vectors, $\boldsymbol{l}$ and $\boldsymbol{r}$ are row vectors and s is a scalar. We seek to compute the transfer function H that corresponds to $\mathscr{R}$.

As it was mentioned in Chapter 3, Section 3.2.1, a direct way to obtain the transfer function of a given SIF is to first convert it to a discrete-time state-space representation $\mathscr{S} := \{maA, \boldsymbol{b}, \boldsymbol{c}, d\}$

$$\mathscr{S} \begin{cases} \boldsymbol{x}(k+1) &= \boldsymbol{A}\boldsymbol{x}(k) + \boldsymbol{b}u(k) \\ y(k) &= \boldsymbol{c}\boldsymbol{x}(k) + du(k) \end{cases} \tag{5.4}$$

with

$$\boldsymbol{A} = \boldsymbol{K}\boldsymbol{J}^{-1}\boldsymbol{M} + \boldsymbol{P}, \qquad \boldsymbol{b} = \boldsymbol{K}\boldsymbol{J}^{-1}\boldsymbol{n} + \boldsymbol{q}, \tag{5.5a}$$

$$\boldsymbol{c} = \boldsymbol{l}\boldsymbol{J}^{-1}\boldsymbol{M} + \boldsymbol{r}, \qquad d = \boldsymbol{l}\boldsymbol{J}^{-1}\boldsymbol{n} + s. \tag{5.5b}$$

Before this thesis, the conversion of a filter from SIF to state-space representation was done naively in Floating-Point arithmetic and applying rigorous analysis and implementation methods upon an error-prone state-space is meaningless. In this thesis, we propose to avoid the rounding errors of this conversion by simply computing the coefficients $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}, d)$ exactly: matrix multiplications and additions can be done using rational[1] arithmetic, and the inverse of matrix $\boldsymbol{J}$ can be computed exactly as well. Since $\boldsymbol{J}$ is lower-triangular with 1 on the diagonal, its inverse $\boldsymbol{J}^{-1}$ can be found using forward descend. Therefore, we can compute an exact inverse using rational numbers.

By applying the $\mathscr{Z}$-transform on (5.4) [18], we obtain

$$\begin{cases} zX(z) &= \boldsymbol{A}X(z) + \boldsymbol{b}U(z) \\ Y(z) &= \boldsymbol{c}X(z) + dU(z) \end{cases}. \tag{5.6}$$

The transfer function $H(z)$ is the ratio $\frac{Y(z)}{U(z)}$. First, we need to express $X(z)$:

$$(z\boldsymbol{I} - \boldsymbol{A})X(z) = \boldsymbol{b}U(z) \tag{5.7}$$

$$X(z) = (z\boldsymbol{I} - \boldsymbol{A})^{-1}\boldsymbol{b}U(z). \tag{5.8}$$

Then, for $Y(z)$ we have

$$Y(z) = \boldsymbol{c}(z\boldsymbol{I} - \boldsymbol{A})^{-1}\boldsymbol{b}U(z) + dU(z) \tag{5.9}$$

$$= \left(\boldsymbol{c}(z\boldsymbol{I} - \boldsymbol{A})^{-1}\boldsymbol{b} + d\right)U(z). \tag{5.10}$$

[1] In practice, we use dyadic rational arithmetic, i.e. the denominators are powers of two.

We obtain that the transfer function of the state-space $\mathscr{S}$ is

$$H(z) = \boldsymbol{c}(z\boldsymbol{I} - \boldsymbol{A})^{-1}\boldsymbol{b} + d. \tag{5.11}$$

Thus, we may determine the SIF's transfer function through the corresponding state-space realization. This is not the only way but rather our technological choice. In the following we propose an approach on the accurate computation of the transfer function corresponding to a state-space realization.

5.2.2 Accurate computation of the transfer function of a state-space

The main bottleneck in the computation of (5.11) is the inverse $(z\boldsymbol{I} - \boldsymbol{A})^{-1}$. A commonly used approach, e.g. in Matlab, is to find the inverse symbolically[2] (with z as unknown) and compute the transfer function coefficients using rational numbers. However, such approach is computationally inefficient: the state-space system may be large and the transfer function may be required to be computed for numerous times during the implementation. In practical implementations, e.g. Matlab, we first get a symbolical expression for the transfer function and then the computations are done in double precision. Obviously, numerical errors may occur but no error bound is given.

Another approach is to completely avoid the closed formula (5.11) for the transfer function but try to bring the state-space realization to a canonical form using similarity transformations. We remind the reader that canonical form of a state-space is directly based on the coefficients of the transfer function. For example, this approach is implemented in Python library SciPy [113]. However, the canonical form is obtained by numerous similarity transformations, i.e. matrix multiplications which are performed without any concern on rounding errors. As usual, no error bound on a certain norm of the transfer function is given.

In most applications[3], exact computation of the Transfer function is not required but an approximation with a rigorous error bound (w.r.t. a certain given norm) on the transfer function is enough. Since double precision might be not enough to ensure such an error bound, we propose to perform the computations in Multiple Precision (MP) arithmetic. Therefore, the problem is

Problem

Given a small $\varepsilon > 0$, compute an approximation $\widehat{H}$ on the transfer function H such that

$$\left|H(z) - \widehat{H}(z)\right| < \varepsilon, \quad \forall z \in \left\{e^{j\omega} | \omega \in [0; 2\pi]\right\}. \tag{5.12}$$

We propose yet another closed formula for computation of the transfer function.

[2] By first setting all matrices to be symbolical expressions and then deducing a symbolical expression of the inverse using classical algorithm (using minors and cofactors).

[3] In practical filter implementations some design margin is always present.

Lemma 5.1. *Let $\mathscr{S} = (\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}, d)$ be a state-space realization of a stable causal linear filter. Suppose $\boldsymbol{A} \in \mathbb{R}^{n_x \times n_x}$ be diagonalizable with no multiple eigenvalues*[4]*. Let $\boldsymbol{V} \in \mathbb{C}^{n_x \times n_x}$ be its eigenvalue matrix and let $\boldsymbol{E} \in \mathbb{C}^{n_x \times n_x}$ contain its associated eigenvalues $\{\lambda_i\}_{1 \leq i \leq n_x}$ on the diagonal. Then the transfer function associated with the filter $\mathscr{S}$ can be written as a rational function $H(z) = \frac{b(z)}{a(z)}$ with*

$$a(z) = \prod_{j=1}^{n_x} (z - \lambda_j), \tag{5.13}$$

$$b(z) = \sum_{i=1}^{n_x} (\boldsymbol{c}\boldsymbol{V})_i (\boldsymbol{V}^{-1}\boldsymbol{b})_i \prod_{j \neq i} (z - \lambda_j) + d \cdot a(z). \tag{5.14}$$

Proof. Consider the eigendecomposition $\boldsymbol{V}\boldsymbol{E}\boldsymbol{V}^{-1}$ of the matrix $\boldsymbol{A}$:

$$H(z) = \boldsymbol{c}(z\boldsymbol{I} - \boldsymbol{V}\boldsymbol{E}\boldsymbol{V}^{-1})^{-1}\boldsymbol{b} + d \tag{5.15}$$

$$= \boldsymbol{c}\boldsymbol{V}(z\boldsymbol{I} - \boldsymbol{E})^{-1}\boldsymbol{V}^{-1}\boldsymbol{b} + d \tag{5.16}$$

$$= \boldsymbol{c}\boldsymbol{V} \begin{pmatrix} \frac{1}{z-\lambda_1} & & \\ & \ddots & \\ & & \frac{1}{z-\lambda_{n_x}} \end{pmatrix} \boldsymbol{V}^{-1}\boldsymbol{b} + d. \tag{5.17}$$

By explicitly expressing the scalar products, we obtain:

$$H(z) = \left((\boldsymbol{c}\boldsymbol{V})_1 \frac{1}{z - \lambda_1}, \ldots, (\boldsymbol{c}\boldsymbol{V})_{n_x} \frac{1}{z - \lambda_{n_x}} \right) \begin{pmatrix} (\boldsymbol{V}^{-1}\boldsymbol{b})_1 \\ \vdots \\ (\boldsymbol{V}^{-1}\boldsymbol{b})_{n_x} \end{pmatrix} + d \tag{5.18}$$

$$= \sum_{i=1}^{n_x} (\boldsymbol{c}\boldsymbol{V})_i (\boldsymbol{V}^{-1}\boldsymbol{b})_i \frac{1}{z - \lambda_i} + d. \tag{5.19}$$

By gathering all terms on the same denominator, we can easily obtain the closed formulas (5.13)-(5.14). ■

We shall remark that the accuracy of the computation of the transfer function relies on the accuracy of the computation of its eigendecomposition and on the computation of the scalar products. For the latter ones, it is not complicated to propose a MP algorithm that will satisfy a certain *a priori* given error bound.

However, exhibiting a bound on the accuracy of the eigendecomposition is a non-trivial task. There exist algorithms that perform all the computations in the eigendecomposition algorithm with MP. Nevertheless, no sharp bound on the accuracy of computed MP eigenvalues has yet

[4] Analogously, the case of multiple eigenvalues can be considered.

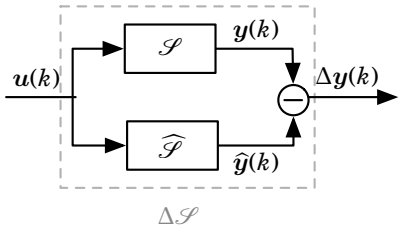


Figure 5.5: $\Delta\mathscr{S}$ is a difference between two filters.

been proposed. Therefore, for the moment, it is not possible to obtain the least required precision of the internal computations in (5.13) and (5.14) that will yield (5.12).

We propose to iteratively increase the accuracy of the computed transfer function. Suppose we have available MP basic bricks for matrix arithmetic and a MP eigensolver such that its absolute error decreases when its working precision is increased. Then, increasing the precision of the computation of $\widehat{H}$ decreases its error $\left|(H-\widehat{H})(e^{j\omega})\right|$ for all $\omega \in [0,2\pi]$.

In order to bound this error we propose the following iterative approach that goes through four steps:

Step 1: we first compute the approximation $\widehat{H}$ on the transfer function of $\mathscr{S} = (\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}, d)$ using a MP eigendecomposition and MP computations for equations (5.13) and (5.14);

Step 2: we deduce the system $\widehat{\mathscr{S}}$ which exactly corresponds to the approximation $\widehat{H}(z)$ using the controllable canonical state-space [18], see Chapter 1 Section 1.5.2.

Step 3: we compute the state-space difference $\Delta\mathscr{S} = \mathscr{S} - \widehat{\mathscr{S}}$ which is defined as the difference between outputs of the two state-spaces (see Figure 5.5). No computational errors are induced on this step, coefficients of $\Delta\mathscr{S}$ are copied from $\mathscr{S}$ and $\widehat{\mathscr{S}}$ in the following manner:

$$\boldsymbol{\Delta A} = \begin{pmatrix} \boldsymbol{A} & \boldsymbol{0} \\ \boldsymbol{0} & \widehat{\boldsymbol{A}} \end{pmatrix}, \qquad \boldsymbol{\Delta b} = \begin{pmatrix} \boldsymbol{b} \\ \widehat{\boldsymbol{b}} \end{pmatrix}, \tag{5.20a}$$

$$\boldsymbol{\Delta c} = \begin{pmatrix} \boldsymbol{c} & -\widehat{\boldsymbol{c}} \end{pmatrix}, \qquad \boldsymbol{\Delta} d = d - \widehat{d}. \tag{5.20b}$$

Here the subtraction $d - \hat{d}$ can be performed exactly using rational arithmetic.

Step 4: we compute a bound on the approximation error $\left|(H-\widehat{H})(e^{j\omega})\right|$ for all $\omega \in [0,2\pi]$: analogously to the case of the state-space, we define the difference ΔH of the transfer functions H and $\widehat{H}$. In other words, ΔH is just the transfer function of $\Delta\mathscr{S}$. The relationships between H, $\widehat{H}$, ΔH and their corresponding state-space systems are illustrated on Figure 5.6.

In [114], it was shown that

$$\left|\Delta H(e^{j\omega})\right| \leq \sup_{\omega} \left|\Delta H(e^{j\omega})\right| \leq \|\Delta h\|_1 \quad \forall \omega \in [0,2\pi] \tag{5.21}$$

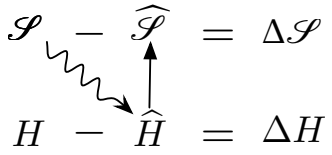


Figure 5.6: Straight lines: exact transformations. Curved lines: MP transformation.

where Δh is the impulse response of the system $\Delta\mathscr{S}$, and $\|\Delta h\|_1$ is its ℓ_1-norm. It is defined as

$$\|\Delta h\|_1 \triangleq \sum_{k=0}^{\infty} |\Delta h(k)|. \tag{5.22}$$

Remark 5.1. *We bring the reader's attention to the elegance with which* (5.21) *binds the errors from the frequency domain with the errors in the time domain. Indeed, we can easily consider the difference between H and $\widehat{H}$ to be due to the quantization of filter coefficients. Then, to evaluate the impact of the quantization upon filter's spectrum, we just compute the ℓ_1-norm of the difference filter ΔH. We are going to explain this approach in details in Chapter 8.*

Obviously, (5.22) is in infinite sum and cannot be exactly computed in finite time. However, in Chapter 6 we show how to compute it with arbitrary precision. Thus, we obtain an arbitrarily precise bound on the error of the approximation $\widehat{H}$:

$$\left|H(e^{j\omega}) - \widehat{H}(e^{j\omega})\right| \leq \Theta, \quad \forall \omega \in [0, 2\pi], \tag{5.23}$$

where $\Theta = \|\Delta h\|_1 + \varepsilon_\Theta$ and $\varepsilon_\Theta > 0$ denotes the upper bound on the error of approximation of $\|\Delta h\|_1$, this bound can be arbitrarily small.

Hence, to ensure that for a small given ε the bound (5.12) is satisfied, we must ensure that $\Theta < \varepsilon$. This can be achieved by increasing the precision of the approximation of $\widehat{H}$.

We choose the precision of the computations based on an heuristic which increases the compute precision in a loop and is, hence, reasonably fast and accurate but does not provide any guarantee that the precision will eventually be enough. The result will always be reliable, though.

Algorithm II.5 illustrates our approach: the initial precision INIT_PREC of the eigensolver is increased by a factor PREC_FACTOR as long as the bound (5.23) is not satisfied. The process will necessarily end if our assumptions on the eigensolver are satisfied.

Algorithm II.5 uses following routines that are not detailed in the manuscript but can be found directly in the implementation:

- `multiprec_eigendecomp` for the computation of an eigendecomposition of a real matrix, internal operations are performed with the precision given in argument;

- `multiprec_tf` for the computation of a transfer function as given in Lemma 5.1;
- `tf2canonical` for the exact construction of a canonical state-space system out of a transfer function;
- `l1norm` algorithm for the computation of the ℓ_1-norm of a state-space system in arbitrary precision (this algorithm will be detailed in Chapter 6);
- `difference` for the exact construction of the difference of two filters (see Figure 5.5).

Algorithm II.5: `computeTF` - accurate Transfer Function computation

Input: state-space coefficients $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}, d)$
$\varepsilon > 0$ bound on approximation error
Output: $\widehat{H}$
`// set initial precision for eigensolver computations`
$p \leftarrow$ INIT_PREC ;
do
  `// eigendecomposition with working precision` p
  $\boldsymbol{V}, \boldsymbol{E} \leftarrow$ `multiprec_eigendecomp`$(\boldsymbol{A}, p)$;
  `// compute transfer funciton with accuracy` p
  $\widehat{H} \leftarrow$ `multiprec_tf`$(\boldsymbol{c}, \boldsymbol{b}, \boldsymbol{d}, \boldsymbol{V}, \boldsymbol{E}, p)$;
  `// construct corresponding canonical state-space`
  $\widehat{\mathscr{S}} \leftarrow$ `tf2canonical`$(\widehat{H})$;
  `// construct state-space of difference`
  $\Delta\mathscr{S} \leftarrow$ `difference`$(\mathscr{S}, \widehat{\mathscr{S}})$;
  `// compute its` ℓ_1`-norm with absolute error bounded by` $\varepsilon/2$
  $\Theta \leftarrow$ `l1norm`$(\Delta\mathscr{S}, \varepsilon/2) + \varepsilon/2$;
  `// increase the precision by some factor PREC_FACTOR`
  $p \leftarrow$ PREC_FACTOR $\cdot\, p$;
while $\Theta > \varepsilon$;
return $\widehat{H}$

5.3 Numerical examples

We implemented Algorithm II.5 in Python using the `mpmath` library [115] and the implementation of the ℓ_1-norm of a filter that will be presented in Chapter 6.

Consider our key example of a 8^{th}-order transfer function from Chapter 1 Section 1.4.3.

Example 1: Suppose we implement the filter with Direct Form II transposed structure and truncate the coefficients to 32 bits. Denote by H_{DFII} its transfer function. We wish to compute

an approximation $\widehat{H}_{\text{DFII}}$ of the transfer function that corresponds to the quantized realization with the error bound $\varepsilon = 2^{-32}$. We just apply our algorithm and obtain a multiple precision approximation $\widehat{H}_{\text{DFII}}$. The algorithm returns an answer after the first run, i.e. after computing the eigendecomposition with the initial precision INIT_PREC (which we set to 53 bits in our implementation).

In case of the Direct Forms, quantization of the realization coefficients is equivalent to the quantization of the transfer function. We can verify that the SIF respects this property. Denote by $H_q(z) = \frac{a_q(z)}{b_q(z)}$ a truncated to 32 bits version of the initial example transfer function. We obtain that its coefficients are the same as those returned by our algorithm:

$$\left\| b_q(z) - \widehat{b}(z) \right\|_\infty = 0, \quad \left\| a_q(z) - \widehat{a}(z) \right\|_\infty = 0, \tag{5.24}$$

where the ∞-norm gives the magnitude of the largest value. However, with other structures quantization of transfer function is not equivalent to quantization of the coefficients of the realization.

Example 2: Suppose we realize the example transfer function with a balanced[5] state-space structure and truncate the coefficients of the state matrices to 32 bits. Denote by H_{SS} its exact transfer function. We apply our algorithm with an a priori error bound $\varepsilon = 2^{-32}$ and obtain an approximation $\widehat{H}_{\text{SS}}$ after computing the eigendecomposition with 128 bits of precision. In this case we obtain that

$$\left\| b_q(z) - \widehat{b}(z) \right\|_\infty = 1.3 \cdot 10^{-9}, \quad \left\| a_q(z) - \widehat{a}(z) \right\|_\infty = 5.5 \cdot 10^{-9}, \tag{5.25}$$

This difference is due to the fact that during balancing of the state-space realization some rounding errors occurred.

The main goal of these examples is to demonstrate that in general truncating the coefficients of the filter realization is not the same as truncation of the coefficients of the corresponding transfer function, and vice versa.

5.4 Conclusion

As first contribution of this Chapter, we developed an algorithm for the conversion of any data-flow describing a linear filter in Simulink format to SIF. This contribution has a major impact on the functionality of the automatic filter code generator: it permits us to implement and compare any linear filter using our flow. Now, analysis and implementation techniques that are developed

[5]The balanced form was obtained using function `ss` in Python SciPy.

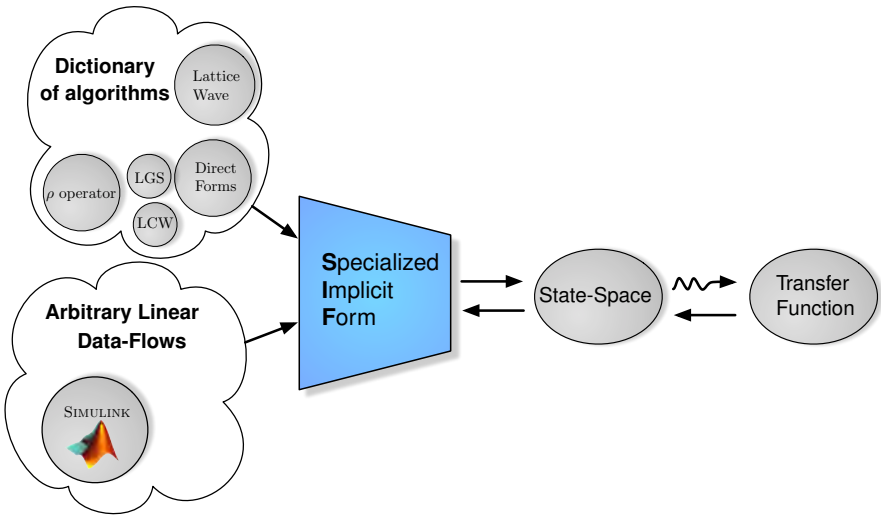


Figure 5.7: Interactions between SIF and other representations. Straight arrows denote the exact and curved arrows denote multiple precision transformations.

for SIF can be generalized to any filter algorithm. In perspective, we should develop an algorithm for the conversion from SIF representation to a Simulink data-flow graph. This can be done using considerations analogous to those from Section 5.1.2.

In the second contribution, we provided an algorithm for the accurate computation of a transfer function of a SIF and thus, for any linear filter algorithm. It is especially useful during the analysis of existing implementations or for structures whose coefficients are not directly coefficients of the transfer function. Our algorithm is based on the reliable computation of the ℓ_1 – norm of a digital filter[6] and on multiple precision eigendecomposition[7] of a real matrix. In practice, we do not compute the eigendecomposition with the *least* precision[8] but start with a certain initial precision and increase it by some factor until the error-bound condition is met. As perspective, we would like to improve this point and, ideally, provide an algorithm for eigendecomposition of a real non-symmetric matrix that adapts the precision of internal computations just enough to meet an a priori given error bound.

To conclude both Chapters 4 and 5, we proposed algorithms that permit us to finally use SIF as a unifying framework in practice. We proposed a new conversion algorithm for a widely-used structure, generalized the approach for any linear data-flow and ensured that computation of a transfer function for SIFs is accurate. An overview of the possibilities of conversions between SIF and other representations is illustrated on Figure 5.7 (compare to Figure 3.2 on p. 47).

[6] We address this question in Chapter 6.

[7] In the sense that all internal operations are performed with some a priori set precision.

[8] We do not think it is possible to determine the least precision without error bounds on the computed eigenvalues.

PART III

RELIABLE FIXED-POINT IMPLEMENTATION OF DIGITAL FILTERS

CHAPTER 6

RELIABLE EVALUATION OF THE DYNAMIC RANGE OF AN EXACT FILTER

Once a structure for filter implementation is chosen, we must convert the obtained realization into a Fixed-Point algorithm. As we have seen in Part I Chapter 2, for a Fixed-Point implementation, we must know beforehand the dynamic range of all the variables involved in the computations. If the dynamic range is underestimated, there is a risk of an overflow at some point in algorithm execution. On the other hand, overestimating induces a higher cost of the implementation. Our goal is to *accurately* determine the dynamic range of all variables involved in computations of *any* LTI algorithm. An important remark is that we do not use any assumptions on the behavior of the input signal apart from it being bounded. In other words, we focus on the worst-case dynamic range.

We have seen in the previous Chapters that any LTI filter can be represented with the Specialized Implicit Form (SIF), which can be exactly converted to a state-space. Without loss of generality and in order to make the notation and equation simpler, further in this Chapter we demonstrate our algorithms for the case of state-space systems.

In this Chapter we propose a novel approach for the reliable determination of the dynamic range of a digital filter's output: we compute with arbitrary accuracy a bound on the output interval of the filter. Our algorithm is based on the computation of the so-called Worst-Case Peak Gain (WCPG) of a digital filter which is just ℓ_1-norm of the filter's impulse response.

For state-space systems, this measure is classically given as an infinite sum and has matrix powers in each summand. These problems are both known to be non-trivial. In this Chapter we propose a detailed algorithm that ensures that the WCPG is computed with an absolute error rigorously bounded by an a priori given value ε. For these purposes several multiprecision algorithms with rigorous bounds were developed. This is achieved by adapting the precision of intermediate computations. Therefore, we present not only the error analysis of the approximations made on each step of the WCPG computation but we also deduce the required accuracy for our kernel multiprecision algorithms such that the overall error bound is satisfied.

We analyze the error induced by truncating the infinite sum and give a direct formula for the computation of a lower bound on truncation order required for a desired absolute error. The truncation order algorithm involves Interval Arithmetic computations and uses the Theory of Verified Inclusions.

We also investigate the case of uncertain systems when the state-space coefficients are represented as small intervals. We give a brief description on how to adapt our initial algorithm for interval computations.

This work is mostly based on the article [11] published at IEEE Symposium on Computer Arithmetic (ARITH) in 2015 and [8] presented at the 17th International Symposium on Scientific Computing, Computer Arithmetics and Verified Numerics (SCAN) in 2016.

Notation: we remind the reader that all matrix and vector absolute values and inequalities are applied element-by-element.

6.1 State of the Art

As we stated in Chapter 2 Section 2.3.3, several approaches on the estimation of a filter's dynamic range exist.

In particular, the most commonly used approach is based on simulations [80, 82]. Another way to estimate the dynamic range is the determination with a certain probability of an overflow [90–93]. Indeed, some applications, like telecommunications, can tolerate a slight degradation of the accuracy due to overflow; more precisely, due to techniques that deal with the overflows. For example, when the values that overflow are saturated towards maximum/minimum, the accuracy of the output cannot be determined.

All these approaches do not take into account the worst cases that can be extremely rare.

We, on the other hand, target the applications that require a *guarantee* on the quality of the filter's output and that no overflow occurs. For example, in emerging drone and autonomous vehicle industries the safety standards are extremely high and require such guarantees. In [39] Hilaire proposed an approach on the determination of the filter's dynamic range which is based on the well-known following result: the Worst-Case Peak Gain theorem [10, 116]. However, this approach cannot be used in practice because the WCPG measure cannot be computed exactly but only approximately. In the following we detail the WCPG theorem and formally state the problem that we will solve in this Chapter.

6.1.1 Worst-Case Peak Gain theorem

Without loss of generality, consider a state-space system $\mathscr{H}$:

$$\mathscr{H}\begin{cases} \boldsymbol{x}(k+1) &= \boldsymbol{A}\boldsymbol{x}(k)+\boldsymbol{B}\boldsymbol{u}(k) \\ \boldsymbol{y}(k) &= \boldsymbol{C}\boldsymbol{x}(k)+\boldsymbol{D}\boldsymbol{u}(k) \end{cases} \tag{6.1}$$

where $\boldsymbol{u}(k) \in \mathbb{R}^q$ is the input vector, $\boldsymbol{y}(k) \in \mathbb{R}^p$ the output vector, $\boldsymbol{x}(k) \in \mathbb{R}^n$ the state vector and $\boldsymbol{A} \in \mathbb{R}^{n\times n}$, $\boldsymbol{B} \in \mathbb{R}^{n\times q}$, $\boldsymbol{C} \in \mathbb{R}^{p\times n}$ and $\boldsymbol{D} \in \mathbb{R}^{p\times q}$ are the state-space matrices of the system.

Theorem 6.1 (Worst-Case Peak Gain)**.** *Let $\mathscr{H}$ be a BIBO-stable*[1] *n^{th} order state-space system with q inputs, p outputs. If an input signal $\{\boldsymbol{u}(k)\}_{k\geq 0}$ is bounded in magnitude by $\bar{\boldsymbol{u}}$ (i.e. $\forall k \geq 0, \quad |\boldsymbol{u}_i(k)| \leq \bar{\boldsymbol{u}}_i, \quad 1 \leq i \leq q$), then the output $\boldsymbol{y}(k)$ is (element-by-element) bounded by*

$$\forall k, \quad |\boldsymbol{y}| \leq \langle\langle \mathscr{H} \rangle\rangle \, \bar{\boldsymbol{u}} \tag{6.2}$$

where $\langle\langle \mathscr{H} \rangle\rangle \in \mathbb{R}^{p\times q}$ is the Worst-Case Peak Gain matrix [10] of the system. It can be computed as the ℓ_1-norm of the system's impulse response. In case of a state space, this norm can be computed with:

$$\langle\langle \mathscr{H} \rangle\rangle := |\boldsymbol{D}| + \sum_{k=0}^{\infty} \left| \boldsymbol{C}\boldsymbol{A}^k\boldsymbol{B} \right|. \tag{6.3}$$

Proof. Let $\mathbb{h}(k)$ be the impulse response matrix[2] of the system at instance k, i.e. $\mathbb{h}_{ij}(k)$ is the response on the i^{th} output to the impulse at time $k = 0$ on the j^{th} input. With (6.1), we have

$$\mathbb{h}(k) = \begin{cases} \boldsymbol{D} & \text{if } k = 0 \\ \boldsymbol{C}\boldsymbol{A}^{k-1}\boldsymbol{B} & \text{if } k > 0. \end{cases} \tag{6.4}$$

Since the input $\{\boldsymbol{u}(k)\}_{k\geq 0}$ can be seen as a weighted sum of impulses (see Chapter 1), and thanks to the linearity and time invariance property of LTI systems [31], we get

$$\boldsymbol{y}(k) = \sum_{l=0}^{k} \mathbb{h}(l)\boldsymbol{u}(k-l). \tag{6.5}$$

($\{\boldsymbol{y}\}_{k\geq 0}$ is the result of the convolution of $\{\mathbb{h}\}_{k\geq 0}$ by $\{\boldsymbol{u}\}_{k\geq 0}$).Then the output is (element-by-element) bounded by

$$|\boldsymbol{y}(k)| \leq \bar{\boldsymbol{u}} \sum_{l=0}^{k} |\mathbb{h}(l)|, \quad \forall k \geq 0. \tag{6.6}$$

Finally

$$\forall k \geq 0, \quad |\boldsymbol{y}(k)| < \left(\sum_{l=0}^{\infty} |\mathbb{h}(k)| \right) \bar{\boldsymbol{u}}. \tag{6.7}$$

By writing explicitly the impulse response (6.4) in (6.7) we can directly obtain (6.3). ■

[1] i.e. $\rho(\boldsymbol{A}) < 1$, see Property 1.1 on p.23

[2] Here, we exceptionally use a lowcase bold notation for a matrix.

Remark 6.1. *An important remark is that it is possible to find a finite input signal* $\{\boldsymbol{u}(k)\}_{0\le k\le K}$ *that yields an output that approaches the* $\langle\langle\mathscr{H}\rangle\rangle\,\bar{\boldsymbol{u}}$ *up to any arbitrarily small distance.*

Indeed, in (6.6)*, we obtain the equality for the* i^{th} *output if the input is such that*

$$\boldsymbol{u}_j(l) = \bar{\boldsymbol{u}}_j \cdot sign\big(\mathbb{h}_{ij}(k-l)\big), \quad \forall 0 \le l \le k, \quad \forall 0 \le j \le q \tag{6.8}$$

where $sign(x)$ *returns* ± 1 *or* 0 *depending on the sign of* x*.*

Remark 6.2. *This proposition can be completed when considering intervals for the input, instead of bounds (corresponding to symmetric intervals). In that case, the Worst-Case Peak Gain matrix indicates by how much the radius of the input interval is amplified on the output [39].*

Remark 6.3. *From* (6.7) *we see that the WCPG is actually the* ℓ_1*-norm of a filter's impulse response. Therefore, in what follows we will use the term WCPG instead of* ℓ_1*-norm.*

Obviously, the infinite sum in (6.3) cannot be computed exactly. However, we may need to compute an arbitrarily precise approximation:

- when we determine the dynamic range of the filter's states and outputs. If the computed bound is very close to the power of 2, we might require to increase the accuracy of the bound (in order to be sure not to overestimate);
- we compute an approximation on a transfer function of a filter using Algorithm II.5 from Chapter 5 we must compute the ℓ_1-norm, which is just the WCPG;
- we must provide rigorous error-analysis of a finite-precision implementation of a filter, as we shall see in the next Chapter.

Hence, the problem can be formulated as follows:

Problem

Given a small $\varepsilon > 0$ compute an approximation $\boldsymbol{S}$ on the WCPG matrix $\langle\langle\mathscr{H}\rangle\rangle$ such that

$$\big|\langle\langle\mathscr{H}\rangle\rangle_{i,j} - \boldsymbol{S}_{i,j}\big| < \varepsilon \quad \text{for } i = 0,\dots,p \text{ and } j = 0,\dots,q \tag{6.9}$$

Of course, we need to first truncate the sum to some finite number of terms N (further called truncation order). In practice, some "sufficiently large" truncation order is often chosen, e.g. 500 or 1000 terms. The following example demonstrates that it may be very dangerous.

Example 6.1. *Consider a stable* 5^{th} *order random SISO filter*[3]*. A naive computation of the WCPG in double precision with 1000 terms in the sum* (6.3) *yields* $\langle\langle\mathscr{H}\rangle\rangle_{naive} = 105.66$*. Suppose all*

[3] Motivated reader can find exact coefficients in the Appendix 3.1

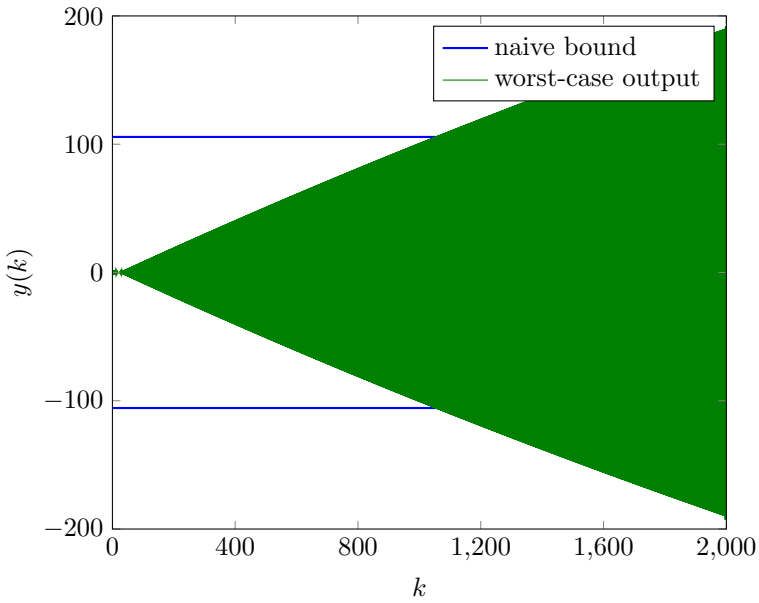


Figure 6.1: Worst-case output is out of the naively determined bounds.

the inputs are in the interval $[-1, 1]$*. Then, according to the WCPG theorem, outputs must be in the interval* $[-105.66, 105.66]$*.*
Now, consider the input signal from Remark 6.1, i.e. the one that yields the worst-case output. Figure 6.1 demonstrates that it easily goes out of the bounds determined by the naive WCPG. It reaches the value 192.2 *in just two thousand iterations.*

In [10] Balakrishnan and Boyd propose "simple" lower and upper bounds on the truncation order. However, they describe their algorithm in terms of exact arithmetic, *i.e.* do not propose any error analysis. This iterative algorithm has several difficulties: first of all, there is a matrix $\boldsymbol{A}$ exponentiation, which would require a non-trivial error analysis. Second, on each iteration (the quantity of which may reach a high order) a solution of Lyapunov equations [117–119] is required for which there exists no ready-to-use solution with rigorous error bounds on the result. Therefore, *numerically computing* a reliable lower bound on the truncation order N is not possible with this approach as it is.

A competing approach would be not to start with truncation order determination but to immediately go for summation and to stop when adding more terms does not improve accuracy. For example, if we try increasing the truncation order in the Example 6.1, we obtain the dynamic of the WCPG approximations shown on Figure 6.2.

However, naive computation of terms in (6.3) with double-precision arithmetic may yield significant rounding errors and would not allow the final approximation error to be bounded in an

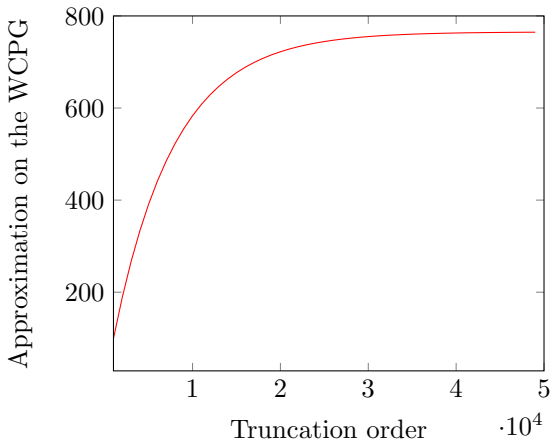


Figure 6.2: The approximations of the WCPG with the increase of truncation order.

a priori way by an arbitrary ε. Even if the computations were performed using multiple precision arithmetic, we would need to determine the required precision (and prove it) without yet knowing how many operations would be performed in the end.

Therefore, in the following we propose a new approach on the evaluation of the WCPG in multiple precision. Our goal is to not only perform rigorous error analysis of approximations but also to deduce the required accuracy for each computation in the evaluation of the WCPG. By adapting the precision of intermediate computations we achieve an *a priori* bound on the overall approximation error.

Notation: In further discussions we bound the error matrices with respect to their Frobenius norm. The Frobenius norm is sub-multiplicative and has several good properties used in numerical analysis. Let $\boldsymbol{K}$ be some matrix, $\boldsymbol{K} \in \mathbb{C}^{n\times m}$, then

$$\left|\boldsymbol{K}_{ij}\right| \leq \|\boldsymbol{K}\|_F \quad \forall i,j \tag{6.10}$$

$$\|\boldsymbol{K}\|_2 \leq \|\boldsymbol{K}\|_F \leq \sqrt{\min(m,n)}\,\|\boldsymbol{K}\|_2\,, \tag{6.11}$$

where $\|\boldsymbol{K}\|_2$ is the spectral-norm, i.e. equal to the largest singular value of $\boldsymbol{K}$.

Let $\boldsymbol{K}$ be a square $n \times n$ matrix with $\|\boldsymbol{K}\|_2 \leq 1$, then for all k, $\left\|\boldsymbol{K}^k\right\|_2 \leq 1$ and $\left\|\boldsymbol{K}^k\right\|_F \leq \sqrt{n}$.

6.2 Algorithm for Worst-Case Peak Gain evaluation

Given a BIBO stable LTI filter in state-space realization (6.1) and ε, a desired absolute approximation error, we want to determine the Worst-Case Peak Gain matrix $\langle\langle \mathscr{H} \rangle\rangle$ of this filter, defined in (6.3). While computing such an approximation, various errors, such as truncation and summation errors, are made.

Algorithm III.1: Floating-point evaluation of the Worst-Case Peak Gain

Input: $\boldsymbol{A} \in \mathbb{F}^{n\times n}, \boldsymbol{B} \in \mathbb{F}^{n\times q}, \boldsymbol{C} \in \mathbb{F}^{p\times n}, \boldsymbol{D} \in \mathbb{F}^{p\times q}, \varepsilon > 0$

Output: $\boldsymbol{S}_N \in \mathbb{F}^{p\times q}$

Step 1: Compute N

Step 2: Compute $\boldsymbol{V}$ from an eigendecomposition of $\boldsymbol{A}$

$\boldsymbol{T} \leftarrow \text{inv}(\boldsymbol{V}) \otimes \boldsymbol{A} \otimes \boldsymbol{V}$

if $\|\boldsymbol{T}\|_2 > 1$ **then return** $\perp$

Step 3: $\boldsymbol{B}' \leftarrow \text{inv}(\boldsymbol{V}) \otimes \boldsymbol{B}$

$\boldsymbol{C}' \leftarrow \boldsymbol{C} \otimes \boldsymbol{V}$

$\boldsymbol{S}_{-1} \leftarrow |\boldsymbol{D}|, \quad \boldsymbol{P}_{-1} \leftarrow \boldsymbol{I}_n$

for k **from** 0 **to** N **do**

Step 4: $\quad \boldsymbol{P}_k \leftarrow \boldsymbol{T} \otimes \boldsymbol{P}_{k-1}$

Step 5: $\quad \boldsymbol{L}_k \leftarrow \boldsymbol{C}' \otimes \boldsymbol{P}_k \otimes \boldsymbol{B}'$

Step 6: $\quad \boldsymbol{S}_k \leftarrow \boldsymbol{S}_{k-1} \oplus \text{abs}(\boldsymbol{L}_k)$

end

return $\boldsymbol{S}_N$

Instead of directly computing the infinite sum $\left|\boldsymbol{C}\boldsymbol{A}^k\boldsymbol{B}\right|$ for any $k \geq 0$, we will use an approximate eigenvalue decomposition of $\boldsymbol{A}$ (i.e. $\boldsymbol{A} \approx \boldsymbol{V}\boldsymbol{T}\boldsymbol{V}^{-1}$) and compute the floating-point sum $\left|\boldsymbol{C}\boldsymbol{V}\boldsymbol{T}^k\boldsymbol{V}^{-1}\boldsymbol{B}\right|$ for $0 \leq k \leq N$.

Our approach to compute the approximation $\boldsymbol{S}_N$ of $\langle\langle\mathscr{H}\rangle\rangle$ is summarized in algorithm III.1 where all the operations ($\otimes$, $\oplus$, inv, abs, etc.) are floating-point multiple precision operations done at various precisions to be determined such that the overall error is less than ε:

$$|\langle\langle\mathscr{H}\rangle\rangle - \boldsymbol{S}_N| \leq \varepsilon. \tag{6.12}$$

The overall error analysis is decomposed into 6 steps, where each one expresses the impact of a particular approximation (or truncation), and provides the accuracy requirements for the associated operations such that the result is rigorously bounded by ε. These steps are discussed in detail in Sections 6.3 and 6.4:

Step 1: Let $\langle\langle\mathscr{H}\rangle\rangle_N$ be the truncated sum

$$\langle\langle\mathscr{H}\rangle\rangle_N := \sum_{k=0}^{N} \left|\boldsymbol{C}\boldsymbol{A}^k\boldsymbol{B}\right| + |\boldsymbol{D}|. \tag{6.13}$$

We compute a truncation order N of the infinite sum $\langle\langle\mathscr{H}\rangle\rangle$ such that the truncation error is less than $\varepsilon_1 > 0$:

$$|\langle\langle\mathscr{H}\rangle\rangle - \langle\langle\mathscr{H}\rangle\rangle_N| \leq \varepsilon_1. \tag{6.14}$$

See Section 6.3 for more details.

Step 2: Error analysis for computing the powers $\boldsymbol{A}^k$ of a full matrix $\boldsymbol{A}$, when the k reaches several hundreds, is a significant problem, especially when the norm of $\boldsymbol{A}$ is larger than 1 and its eigenvalues are close to 1. However, if $\boldsymbol{A}$ may be represented as $\boldsymbol{A} = \boldsymbol{X}\boldsymbol{E}\boldsymbol{X}^{-1}$ with $\boldsymbol{E} \in \mathbb{C}^{n\times n}$ strictly diagonal and $\boldsymbol{X} \in \mathbb{C}^{n\times n}$, then powering of $\boldsymbol{A}$ reduces to powering the diagonal matrix $\boldsymbol{E}$, which is more convenient.

Suppose we have a matrix $\boldsymbol{V}$ approximating $\boldsymbol{X}$. We require this approximation to be just quite accurate so that we are able to discern the different associated eigenvalues and be sure their absolute values are less than 1.

We may then consider the matrix $\boldsymbol{V}$ to be exact and compute an approximation $\boldsymbol{T}$ to $\boldsymbol{V}^{-1}\boldsymbol{A}\boldsymbol{V}$ with sufficient accuracy such that the error of computing $\boldsymbol{V}\boldsymbol{T}^k\boldsymbol{V}^{-1}$ instead of matrix $\boldsymbol{A}^k$ is less than $\varepsilon_2 > 0$:

$$\left| \langle\langle \mathscr{H} \rangle\rangle_N - \sum_{k=0}^{N} \left| \boldsymbol{C}\boldsymbol{V}\boldsymbol{T}^k\boldsymbol{V}^{-1}\boldsymbol{B} \right| \right| \leq \varepsilon_2. \tag{6.15}$$

See Section 6.4.1 for more details.

Step 3: We compute approximations $\boldsymbol{B}'$ and $\boldsymbol{C}'$ of $\boldsymbol{V}^{-1}\boldsymbol{B}$ and $\boldsymbol{C}\boldsymbol{V}$, respectively. We require that the propagated error committed in using $\boldsymbol{B}'$ instead of $\boldsymbol{V}^{-1}\boldsymbol{B}$ and $\boldsymbol{C}'$ instead of $\boldsymbol{C}\boldsymbol{V}$ be less than $\varepsilon_3 > 0$:

$$\left| \sum_{k=0}^{N} \left| \boldsymbol{C}\boldsymbol{V}\boldsymbol{T}^k\boldsymbol{V}^{-1}\boldsymbol{B} \right| - \sum_{k=0}^{N} \left| \boldsymbol{C}'\boldsymbol{T}^k\boldsymbol{B}' \right| \right| \leq \varepsilon_3. \tag{6.16}$$

See Section 6.4.2.

Step 4: We compute in $\boldsymbol{P}_k$ the powers $\boldsymbol{T}^k$ of $\boldsymbol{T}$ with a certain accuracy. It is required that the propagated error be less than $\varepsilon_4 > 0$:

$$\left| \sum_{k=0}^{N} \left| \boldsymbol{C}'\boldsymbol{T}^k\boldsymbol{B}' \right| - \sum_{k=0}^{N} \left| \boldsymbol{C}'\boldsymbol{P}_k\boldsymbol{B}' \right| \right| \leq \varepsilon_4. \tag{6.17}$$

See Section 6.4.3.

Step 5: We compute in $\boldsymbol{L}_k$ each summand $\boldsymbol{C}'\boldsymbol{P}_k\boldsymbol{B}'$ with a error small enough such that the overall approximation error induced by this step is less than $\varepsilon_5 > 0$:

$$\left| \sum_{k=0}^{N} \left| \boldsymbol{C}'\boldsymbol{P}_k\boldsymbol{B}' \right| - \sum_{k=0}^{N} \left| \boldsymbol{L}_k \right| \right| \leq \varepsilon_5. \tag{6.18}$$

See Section 6.4.4.

Step 6: Finally, we sum $\boldsymbol{L}_k$ in $\boldsymbol{S}_N$ with enough precision so that the absolute error bound for summation is bounded by $\varepsilon_6 > 0$:

$$\left| \sum_{k=0}^{N} |\boldsymbol{L}_k| - \boldsymbol{S}_N \right| \leq \varepsilon_6. \tag{6.19}$$

See Section 6.4.5.

By ensuring that each step verifies its bound ε_i, and taking $\varepsilon_i = \frac{1}{6}\varepsilon$, we get $\varepsilon_1 + \varepsilon_2 + \varepsilon_3 + \varepsilon_4 + \varepsilon_5 + \varepsilon_6 \leq \varepsilon$, hence (6.12) will be satisfied if inequalities (6.14) to (6.19) are.

Our approach hence determines first a truncation order N and then performs summation up to that truncation error, whilst adjusting precision in the different summation steps.

6.3 Truncation order and truncation error

In this Section we propose a direct formula for the lower bound on N along with a reliable evaluation algorithm.

The goal is to determine a lower bound on the truncation order N of the infinite sum (6.3) such that its tail is smaller than the given ε_1. Obviously, $\langle\langle \mathscr{H} \rangle\rangle_N$ is a lower bound on $\langle\langle \mathscr{H} \rangle\rangle$ and increases monotonically to $\langle\langle \mathscr{H} \rangle\rangle$ with increasing N. Hence the truncation error is

$$|\langle\langle \mathscr{H} \rangle\rangle - \langle\langle \mathscr{H} \rangle\rangle_N| = \sum_{k>N} \left| \boldsymbol{C}\boldsymbol{A}^k\boldsymbol{B} \right|. \tag{6.20}$$

6.3.1 A bound on the truncation error

Many simple bounds on (6.20) are possible. For instance, if the eigendecomposition of $\boldsymbol{A}$ is computed

$$\boldsymbol{A} = \boldsymbol{X}\boldsymbol{E}\boldsymbol{X}^{-1} \tag{6.21}$$

where $\boldsymbol{X} \in \mathbb{C}^{n\times n}$ is the right hand eigenvector matrix, and $\boldsymbol{E} \in \mathbb{C}^{n\times n}$ is a diagonal matrix holding the eigenvalues $\boldsymbol{\lambda}_l$, the terms $\boldsymbol{C}\boldsymbol{A}^k\boldsymbol{B}$ can be written

$$\boldsymbol{C}\boldsymbol{A}^k\boldsymbol{B} = \boldsymbol{\Phi}\boldsymbol{E}^k\boldsymbol{\Psi} = \sum_{l=1}^{n} \boldsymbol{R}_l \boldsymbol{\lambda}_l^k \tag{6.22}$$

where $\boldsymbol{\Phi} \in \mathbb{C}^{p\times n}$, $\boldsymbol{\Psi} \in \mathbb{C}^{n\times q}$ and $\boldsymbol{R}_l \in \mathbb{C}^{p\times q}$ are defined by

$$\boldsymbol{\Phi} := \boldsymbol{C}\boldsymbol{X}, \qquad \boldsymbol{\Psi} := \boldsymbol{X}^{-1}\boldsymbol{B}, \qquad (\boldsymbol{R}_l)_{ij} := \boldsymbol{\Phi}_{il}\boldsymbol{\Psi}_{lj}. \tag{6.23}$$

In this setting, we obtain

$$|\langle\langle \mathscr{H} \rangle\rangle - \langle\langle \mathscr{H} \rangle\rangle_N| = \sum_{k>N} \sum_{l=1}^{n} \left| \boldsymbol{R}_l \boldsymbol{\lambda}_l^k \right|. \tag{6.24}$$

As required by Proposition 6.1, all eigenvalues $\boldsymbol{\lambda}_l$ of matrix $\boldsymbol{A}$ must be strictly smaller than one in magnitude. We may therefore notice that the outer sum is in geometric progression with a common ratio $|\boldsymbol{\lambda}_l| < 1$. So the following bound is possible (we remind the reader that inequalities and absolute values are considered to be element by element):

$$\begin{aligned} |\langle\langle \mathscr{H} \rangle\rangle - \langle\langle \mathscr{H} \rangle\rangle_N| &\leq \sum_{k=N+1}^{\infty} \sum_{l=1}^{n} |\boldsymbol{R}_l| \left| \boldsymbol{\lambda}_l^k \right| && (6.25) \\ &\leq \sum_{l=1}^{n} |\boldsymbol{R}_l| \frac{\left| \boldsymbol{\lambda}_l^{N+1} \right|}{1 - |\boldsymbol{\lambda}_l|} \\ &= \rho(\boldsymbol{A})^{N+1} \sum_{l=1}^{n} \frac{|\boldsymbol{R}_l|}{1 - |\boldsymbol{\lambda}_l|} \left(\frac{|\boldsymbol{\lambda}_l|}{\rho(\boldsymbol{A})} \right)^{N+1}. && (6.26) \end{aligned}$$

Since $\frac{|\boldsymbol{\lambda}_l|}{\rho(\boldsymbol{A})} \leq 1$ holds for all terms, we may leave out the powers. Notate

$$\boldsymbol{M} := \sum_{l=1}^{n} \frac{|\boldsymbol{R}_l|}{1 - |\boldsymbol{\lambda}_l|} \frac{|\boldsymbol{\lambda}_l|}{\rho(\boldsymbol{A})} \in \mathbb{R}^{p \times q}. \tag{6.27}$$

The tail of the infinite sum is hence bounded by

$$|\langle\langle \mathscr{H} \rangle\rangle - \langle\langle \mathscr{H} \rangle\rangle_N| \leq \rho(\boldsymbol{A})^{N+1} \boldsymbol{M}. \tag{6.28}$$

Remark 6.4. *Other bounds are possible. For instance,*

$$|\langle\langle \mathscr{H} \rangle\rangle - \langle\langle \mathscr{H} \rangle\rangle_N| \leq \rho(\boldsymbol{A})^{N+1-K} \sum_{l=1}^{n} \frac{|\boldsymbol{R}_l|}{1 - |\boldsymbol{\lambda}_l|} \left(\frac{|\boldsymbol{\lambda}_l|}{\rho(\boldsymbol{A})} \right)^{K}, \quad \forall N > K. \tag{6.29}$$

This bound takes into account the weight of each eigenvalue.

Remark 6.5. *A similar bound for the truncation error may be obtained when the eigenvalues of the system are not distinct, i.e. when matrix $\boldsymbol{A}$ has multiple eigenvalues. The impulse response will be [120]*

$$\boldsymbol{J}(k) = \begin{cases} \boldsymbol{D} & \text{if } k = 0 \\ \sum_{i=1}^{n} \sum_{j=1}^{m_i} \boldsymbol{R}_i \frac{(k-1)(k-2)\ldots(k-j+1)}{(j-1)!} \boldsymbol{\lambda}_i^{k-j} & \text{if } k > 0 \end{cases} \tag{6.30}$$

where pole $\boldsymbol{\lambda}_i$ has multiplicity m_i, $i = 1 \ldots n$. An appropriate matrix $\boldsymbol{M}$ can be then deduced.

6.3.2 Deducing a lower bound on the truncation order

In order to get (6.28) bounded by ε_1, it is required that element-by-element

$$\rho(\boldsymbol{A})^{N+1}\boldsymbol{M} \leq \varepsilon_1.$$

Solving this inequality for N leads us to the following bound:

$$N \geq \left\lceil \frac{\log \frac{\varepsilon_1}{m}}{\log \rho(\boldsymbol{A})} \right\rceil \tag{6.31}$$

where m is defined as $m := \min_{i,j} \boldsymbol{M}_{i,j}$.

However we cannot compute exact values for all quantities occuring in (6.31) when using finite-precision arithmetic. We only have approximations for them. Thus, in order to reliably determine a lower bound on N, we must compute lower bounds on m and $\rho(\boldsymbol{A})$, from which we can deduce an upper bound on $\log \frac{\varepsilon_1}{m}$ and a lower bound on $\log \rho(\boldsymbol{A})$ to eventually obtain a lower bound on N.

Due to the implementation of (6.21) and (6.23) with finite-precision arithmetic, only approximations on $\boldsymbol{\lambda}, \boldsymbol{X}, \boldsymbol{\Phi}, \boldsymbol{\Psi}, \boldsymbol{R}_l$ can be obtained. There exist many floating-point libraries, such as LAPACK[4], providing functions for an eigendecomposition as needed for (6.21) and to solve linear systems of equations in (6.23). They usually deliver good and fast approximations to the solution of a given numerical problem but there is neither verification nor guarantee about the accuracy of that approximation. LAPACK only gives an estimation of the absolute error which we will nevertheless exploit in our approach.

We propose to combine LAPACK floating-point arithmetic with Interval Arithmetic [121] enhanced with Rump's Theory of Verified Inclusions [67–69, 122] in order to obtain trusted intervals on the eigensystem and, eventually, a rigorous bound on N.

The Theory of Verified Inclusions is a set of algorithms that compute guaranteed bounds on solutions of various numerical problems. The verification process is performed by means of checking an interval fixed point and yields to a trusted interval for the solution, i.e. it is verified that the result interval contains an exact solution of given numerical problem.

It permits us to quickly obtain trusted error bounds on the truncation order without significant impact on algorithm performance, since this computation is done only once. In addition, if the spectral radius of $\boldsymbol{A}$ cannot be shown less than 1, we stop the algorithm.

Using the ideas proposed by Rump in [67], we obtain trusted intervals for the eigensystem with the following steps:

[4] `http://www.netlib.org/lapack/`

1. Using the LAPACK eigensolver, we compute floating-point approximations $\boldsymbol{V}$ for the eigenvectors $\boldsymbol{X}$ and $\boldsymbol{\alpha}$ for the eigenvalues $\boldsymbol{\lambda}$, along with error estimates $\boldsymbol{\varepsilon}_X$ and $\boldsymbol{\varepsilon}_\lambda$. These error estimates are such that $|\boldsymbol{\lambda} - \boldsymbol{\alpha}| \leq \boldsymbol{\varepsilon}_\lambda$ and $|\boldsymbol{X} - \boldsymbol{V}| \leq \boldsymbol{\varepsilon}_X$ should be not far from the truth.

2. We construct, verify and possibly adjust intervals for $[\boldsymbol{\lambda}] = [\boldsymbol{\alpha} - \boldsymbol{\varepsilon}_\lambda, \boldsymbol{\alpha} + \boldsymbol{\varepsilon}_\lambda]$ and $[\boldsymbol{X}] = [\boldsymbol{V} - \boldsymbol{\varepsilon}_X, \boldsymbol{V} + \boldsymbol{\varepsilon}_X]$ such that for all vectors $\boldsymbol{\lambda}' \in [\boldsymbol{\lambda}]$ there exists a matrix $\boldsymbol{X}' \in [\boldsymbol{X}]$ satisfying $\boldsymbol{A}\boldsymbol{X}' = \boldsymbol{X}' \cdot \mathrm{diag}(\boldsymbol{\lambda}')$ and such that for all matrices $\boldsymbol{X}' \in [\boldsymbol{X}]$ there exists a vector $\boldsymbol{\lambda}' \in [\boldsymbol{\lambda}]$ satisfying $\boldsymbol{A}\boldsymbol{X}' = \boldsymbol{X}' \cdot \mathrm{diag}(\boldsymbol{\lambda}')$. In this process, first intervals for the eigensystem are constructed from the error estimates $\boldsymbol{\varepsilon}_\alpha$ and $\boldsymbol{\varepsilon}_V$ as radii and the approximate solutions $\boldsymbol{V}$ and $\boldsymbol{\alpha}$ as mid-points. Further, these intervals are verified with inclusion algorithms [67]. If the verification does not succeed, the intervals are extended by some small factor and process is repeated until it succeeds or until there exists an eigenvalue interval which contains 1.

For the solution of the linear system of equations (LSE) appearing in (6.23), the algorithm for interval verification is based on [69] and consists of two steps:

1. Using LAPACK, compute a floating-point approximation $\boldsymbol{\Omega}$ on the solution of $\boldsymbol{V}\boldsymbol{\Psi} = \boldsymbol{B}$ along with an error estimate $\boldsymbol{\varepsilon}_\Psi$ such that $|\boldsymbol{\Psi} - \boldsymbol{\Omega}| \leq \boldsymbol{\varepsilon}_\Psi$ should be not far from the truth.

2. Construct, verify and adjust intervals $[\boldsymbol{\Psi}] = [\boldsymbol{\Omega} - \varepsilon_\Psi, \boldsymbol{\Omega} + \varepsilon_\Psi]$ such that for all matrices $\boldsymbol{X}' \in [\boldsymbol{X}]$ there exists $\boldsymbol{\Psi}' \in [\boldsymbol{\Psi}]$ such that $\boldsymbol{X}'\boldsymbol{\Psi}' = \boldsymbol{B}$ holds.

The intervals for verification are constructed in the same way as for the eigensystem solution. We require the existence of the exact solution of the linear system not for $\boldsymbol{V}\boldsymbol{\Psi} = \boldsymbol{B}$ but for $[\boldsymbol{X}]\boldsymbol{\Psi} = \boldsymbol{B}$, i.e. $[\boldsymbol{\Psi}]$ must contain the exact solution for each element of the already verified interval $[\boldsymbol{X}]$.

Finally, the intervals for (6.23), (6.27) and (6.31) are computed with Interval Arithmetic. Our complete algorithm to determine a reliable lower bound on N is given with algorithm III.2.

6.4 Summation

Once the truncation order determined, we need to provide a summation scheme which is reliable in floating-point arithmetic, i.e. such that the error of computations is bounded by an a priori given value. To do so we propose to perform all operations in multiple precision arithmetic whilst adapting precision dynamically where needed. Several multiple precision algorithms were therefore developed:

• `multiplyAndAdd`$(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \delta)$ that computes $\boldsymbol{A} \cdot \boldsymbol{B} + \boldsymbol{C} + \boldsymbol{\Delta}$, where the error matrix $\boldsymbol{\Delta}$ is bounded by $|\boldsymbol{\Delta}| < \delta$, for the given a priori bound δ. We shall notate $\boldsymbol{A} \otimes \boldsymbol{B}$ for the output of `multiplyAndAdd` when $\boldsymbol{C}$ is the zero matrix.

Algorithm III.2: Lower bound of truncation order

Input: $\boldsymbol{A} \in \mathbb{F}^{n\times n}, \boldsymbol{B} \in \mathbb{F}^{n\times q}, \boldsymbol{C} \in \mathbb{F}^{p\times n}, \varepsilon_1 > 0$
Output: $N \in \mathbb{N}$

$\boldsymbol{\alpha}, \boldsymbol{V}, \boldsymbol{\varepsilon}_\alpha, \boldsymbol{\varepsilon}_V \leftarrow$ LAPACK eigendecomposition for $\boldsymbol{A}$;
$\boldsymbol{\Omega}, \boldsymbol{\varepsilon}_\Psi \leftarrow$ LAPACK solver for $\boldsymbol{V\Psi} = \boldsymbol{B}$;
$[\boldsymbol{\lambda}], [\boldsymbol{X}] \leftarrow$ Eigensystem verification algorithm;
$[\boldsymbol{\Psi}] \leftarrow$ LSE solution verification algorithm;
$[\boldsymbol{\Phi}] \leftarrow \boldsymbol{C}[\boldsymbol{X}]$;
$[\boldsymbol{R}_l]_{i,j} \leftarrow [\boldsymbol{\Phi}_{i,l}][\boldsymbol{\Psi}_{l,j}]$;
$\lceil\rho\rfloor \leftarrow \max_i \left|[\boldsymbol{\lambda}_i]\right|$;
$[\boldsymbol{M}] \leftarrow \sum_{i=1}^{n} \frac{\left|[\boldsymbol{R}_i]\right|}{1-\left|[\boldsymbol{\lambda}_i]\right|} \frac{\left|[\boldsymbol{\lambda}_i]\right|}{\lceil\rho\rfloor}$;
$[m] \leftarrow \min_{i,j} \left|[\boldsymbol{M}]_{i,j}\right|$;
$N \leftarrow \sup\left(\left\lceil \frac{\log \frac{\varepsilon_1}{[m]}}{\log\lceil\rho\rfloor} \right\rceil\right)$;
return N

• $\texttt{sumAbs}(\boldsymbol{A}, \boldsymbol{B}, \delta)$ that computes $\boldsymbol{A} + |\boldsymbol{B}| + \boldsymbol{\Delta}$, where the error matrix $\boldsymbol{\Delta}$ is bounded by $|\boldsymbol{\Delta}| < \delta$, for the given δ. With a slight notational abuse, we shall also notate $\boldsymbol{A} \oplus \texttt{abs}(\boldsymbol{B})$ for `sumAbs`.

• $\texttt{inv}(\boldsymbol{V}, \delta)$ that computes the inverse $\boldsymbol{V}^{-1} + \boldsymbol{\Delta}$, where the error matrix $\boldsymbol{\Delta}$ is bounded by $|\boldsymbol{\Delta}| < \delta$, for the given δ. See Section 6.5.

These computation kernels adapt the precision of their intermediate computations where needed. The algorithms we use for these basic bricks will be discussed in Section 6.5.

6.4.1 Step 2: using the Eigendecomposition

6.4.1.1 Error propagation

As seen, in each step of the summation, a matrix power, $\boldsymbol{A}^k$, must be computed. In [46] Higham devotes an entire chapter to error analysis of matrix powers but this theory is in most cases inapplicable for state matrices $\boldsymbol{A}$ of linear filters, as the requirement $\rho(|\boldsymbol{A}|) < 1$ does not necessarily hold here. Therefore, despite taking $\boldsymbol{A}$ to just a finite power k, the sequence of computed matrices may explode in norm since k may take an order of several hundreds or thousands. Thus, even extending the precision is not a solution, as an enormous number of bits would be required.

In real life the state matrices are usually diagonalizable, i.e. there exists a matrix $\boldsymbol{X} \in \mathbb{C}^{n\times n}$ and diagonal $\boldsymbol{E} \in \mathbb{C}^{n\times n}$ such that $\boldsymbol{A} = \boldsymbol{XEX}^{-1}$. Then $\boldsymbol{A}^k = \boldsymbol{XE}^k\boldsymbol{X}^{-1}$. A good choice of $\boldsymbol{X}$

and $\boldsymbol{E}$ are the eigenvector and eigenvalue matrices obtained using eigendecomposition (6.21). However, with LAPACK we can compute only approximations of them and we cannot control their accuracy. Therefore, we propose the following method to *almost* diagonalize matrix $\boldsymbol{A}$. The method does not make any assumptions on matrix $\boldsymbol{V}$ except for it being *some* approximation on $\boldsymbol{X}$. Therefore, for simplicity of further reasoning we treat $\boldsymbol{V}$ as an exact matrix.

Using our multiprecision algorithms for matrix inverse and multiplication we may compute a complex $n \times n$ matrix $\boldsymbol{T}$:

$$\boldsymbol{T} := \boldsymbol{V}^{-1}\boldsymbol{A}\boldsymbol{V} - \boldsymbol{\Delta}_2, \tag{6.32}$$

where $\boldsymbol{V} \in \mathbb{C}^{n\times n}$ is an approximation on $\boldsymbol{X}$, $\boldsymbol{\Delta}_2 \in \mathbb{C}^{n\times n}$ is a matrix representing the element-by-element errors due to the two matrix multiplications and the inversion of matrix $\boldsymbol{V}$.

Although the matrix $\boldsymbol{E}$ is strictly diagonal, $\boldsymbol{V}$ is not exactly the eigenvector matrix and consequently $\boldsymbol{T}$ is a full matrix. However it has its prevailing elements on the main diagonal. Thus $\boldsymbol{T}$ is an approximation on $\boldsymbol{E}$.

We require for matrix $\boldsymbol{T}$ to satisfy $\|\boldsymbol{T}\|_2 \le 1$. This condition is stronger than $\rho(\boldsymbol{A}) < 1$, and Section 6.4.1.2 provides a way to test it. In other words, this condition means that there exist some margin for computational errors between the spectral radius and 1.

Notate $\boldsymbol{\Xi}_k := (\boldsymbol{T} + \boldsymbol{\Delta}_2)^k - \boldsymbol{T}^k$. Hence $\boldsymbol{\Xi}_k \in \mathbb{C}^{n\times n}$ represents the error matrix which captures the propagation of error $\boldsymbol{\Delta}_2$ when powering $\boldsymbol{T}$. Since

$$\boldsymbol{A}^k = \boldsymbol{V}(\boldsymbol{T} + \boldsymbol{\Delta}_2)^k\boldsymbol{V}^{-1}, \tag{6.33}$$

therefore

$$\boldsymbol{C}\boldsymbol{A}^k\boldsymbol{B} = \boldsymbol{C}\boldsymbol{V}\boldsymbol{T}^k\boldsymbol{V}^{-1}\boldsymbol{B} + \boldsymbol{C}\boldsymbol{V}\boldsymbol{\Xi}_k\boldsymbol{V}^{-1}\boldsymbol{B}. \tag{6.34}$$

Thus the error of computing $\boldsymbol{V}\boldsymbol{T}^k\boldsymbol{V}^{-1}$ instead of $\boldsymbol{A}^k$ in (6.13) is bounded by

$$\left|\sum_{k=0}^{N}\left|\boldsymbol{C}\boldsymbol{A}^k\boldsymbol{B}\right| - \sum_{k=0}^{N}\left|\boldsymbol{C}\boldsymbol{V}\boldsymbol{T}^k\boldsymbol{V}^{-1}\boldsymbol{B}\right|\right| \le \tag{6.35}$$

$$\sum_{k=0}^{N}\left|\boldsymbol{C}\boldsymbol{A}^k\boldsymbol{B} - \boldsymbol{C}\boldsymbol{V}\boldsymbol{T}^k\boldsymbol{V}^{-1}\boldsymbol{B}\right| \le \sum_{k=0}^{N}\left|\boldsymbol{C}\boldsymbol{V}\boldsymbol{\Xi}_k\boldsymbol{V}^{-1}\boldsymbol{B}\right|. \tag{6.36}$$

Here and further on each step of the algorithm we use inequalities with left side in form (6.36) rather than (6.35), i.e. we will instantly use the triangular inequality $\big||a| - |b|\big| \le |a - b|$ $\forall a, b$ applied element-by-element to matrices.

In order to determine the accuracy of the computations on this step such that (6.36) is bounded by ε_2, we need to perform detailed analysis of $\boldsymbol{\Xi}_k$, with spectral-norm. Using the definition of $\boldsymbol{\Xi}_k$ the following recurrence can be easily obtained:

$$\|\boldsymbol{\Xi}_k\|_2 \le \|\boldsymbol{\Xi}_{k-1}\|_2 + \|\boldsymbol{\Delta}_2\|_2\left(\|\boldsymbol{\Xi}_{k-1}\|_2 + 1\right) \tag{6.37}$$

If $\|\Xi_{k-1}\|_2 \leq 1$, which must hold in our case since Ξ_k represent an error-matrix, then

$$\|\Xi_k\|_2 \leq \|\Xi_{k-1}\|_2 + 2\|\Delta_2\|_2 \tag{6.38}$$

As $\|\Xi_1\|_2 = \|\Delta_2\|_2$ we can get the desired bound capturing the propagation of Δ_2 with Frobenius norm:

$$\|\Xi_k\|_F \leq 2\sqrt{n}(k+1)\|\Delta_2\|_F . \tag{6.39}$$

Substituting this bound to (6.36) and folding the sum, we obtain

$$\sum_{i=0}^{N} \left|\boldsymbol{C}\boldsymbol{V}\Xi_k\boldsymbol{V}^{-1}\boldsymbol{B}\right| \leq \beta\|\Delta_2\|_F \|\boldsymbol{C}\boldsymbol{V}\|_F \left\|\boldsymbol{V}^{-1}\boldsymbol{B}\right\|_F , \tag{6.40}$$

with $\beta = \sqrt{n}(N+1)(N+2)$. Thus, we get a bound on the error of approximation of $\boldsymbol{A}$ by $\boldsymbol{V}\boldsymbol{T}\boldsymbol{V}^{-1}$. Since we require it to be less than ε_2 we obtain a condition for the error on the inversion and two matrix multiplications:

$$\|\Delta_2\|_F \leq \frac{1}{\beta}\frac{\varepsilon_2}{\|\boldsymbol{C}\boldsymbol{V}\|_F \left\|\boldsymbol{V}^{-1}\boldsymbol{B}\right\|_F}. \tag{6.41}$$

Using this bound we can deduce the desired accuracy of our multiprecision algorithms for complex matrix multiplication and inverse as a function of ε_2.

6.4.1.2 Checking $\|\boldsymbol{T}\|_2 \leq 1$

Since $\|\boldsymbol{T}\|_2^2 = \rho(\boldsymbol{T}^*\boldsymbol{T})$, we study the eigenvalues of $\boldsymbol{T}^*\boldsymbol{T}$, where "$*$" denotes conjugate transpose. According to Gershgorin's circle theorem [123], each eigenvalue μ_i of $\boldsymbol{T}^*\boldsymbol{T}$ is in the disk centered in $(\boldsymbol{T}^*\boldsymbol{T})_{ii}$ with radius $\sum_{j\neq i}\left|(\boldsymbol{T}^*\boldsymbol{T})_{ij}\right|$.

Let us decompose $\boldsymbol{T}$ into $\boldsymbol{T} = \boldsymbol{F} + \boldsymbol{G}$, where $\boldsymbol{F}$ is diagonal and $\boldsymbol{G}$ contains all the other terms ($\boldsymbol{F}$ contains the approximate eigenvalues, $\boldsymbol{G}$ contains small terms and is zero on its diagonal). Denote $\boldsymbol{Y} := \boldsymbol{T}^*\boldsymbol{T} - \boldsymbol{F}^*\boldsymbol{F} = \boldsymbol{F}^*\boldsymbol{G} + \boldsymbol{G}^*\boldsymbol{F} + \boldsymbol{G}^*\boldsymbol{G}$. Then

$$\begin{aligned}
\sum_{j\neq i}\left|\left(\boldsymbol{T}^*\boldsymbol{T}\right)_{ij}\right| &= \sum_{j\neq i}\left|\boldsymbol{Y}_{ij}\right| \\
&\leq (n-1)\|\boldsymbol{Y}\|_F \\
&\leq (n-1)\left(2\|\boldsymbol{F}\|_F\|\boldsymbol{G}\|_F + \|\boldsymbol{G}\|_F^2\right) \\
&\leq (n-1)\left(2\sqrt{n} + \|\boldsymbol{G}\|_F\right)\|\boldsymbol{G}\|_F .
\end{aligned} \tag{6.42}$$

Each eigenvalue of $\boldsymbol{T}^*\boldsymbol{T}$ is in the disk centered in $(\boldsymbol{F}^*\boldsymbol{F})_{ii} + (\boldsymbol{Y})_{ii}$ with radius γ, where γ is equal to $(n-1)\left(2\sqrt{n} + \|\boldsymbol{G}\|_F\right)\|\boldsymbol{G}\|_F$, computed in a rounding mode that makes the result become an upper bound (round-up).

As $\boldsymbol{G}$ is zero on its diagonal, the diagonal elements $(\boldsymbol{Y})_{ii}$ of $\boldsymbol{Y}$ are equal to the diagonal elements $(\boldsymbol{G}^*\boldsymbol{G})_{ii}$ of $\boldsymbol{G}^*\boldsymbol{G}$. They can hence be bounded as follows:

$$|(\boldsymbol{Y})_{ii}| = \left|\left(\boldsymbol{G}^*\boldsymbol{G}\right)_{ii}\right| \leq \|\boldsymbol{G}\|_F^2 . \tag{6.43}$$

Then, it is easy to see that Gershgorin circles enclosing the eigenvalues of $\boldsymbol{F}^*\boldsymbol{F}$ can be increased, meaning that if $(\boldsymbol{F}^*\boldsymbol{F})_{ii}$ is such that

$$\forall i, \quad \left|\left(\boldsymbol{F}^*\boldsymbol{F}\right)_{ii}\right| \leq 1 - \|\boldsymbol{G}\|_F^2 - \gamma, \tag{6.44}$$

it holds that $\rho(\boldsymbol{T}^*\boldsymbol{T}) \leq 1$ and $\|\boldsymbol{T}\|_2 \leq 1$.

This condition can be tested by using floating-point arithmetic with directed rounding modes (round-up for instance).

After computing $\boldsymbol{T}$ out of $\boldsymbol{V}$ and $\boldsymbol{A}$ according to (6.32), the condition on $\boldsymbol{T}$ should be tested in order to determine if $\|\boldsymbol{T}\|_2 \leq 1$. This test failing means that $\boldsymbol{V}$ is not a sufficient approximate of $\boldsymbol{X}$ or that the error $\boldsymbol{\Delta}_2$ done computing (6.32) is too large, i.e. the accuracy of our multiprecision algorithm for complex matrix multiplication and inverse should be increased. The test is required for rigor only. We do perform the test in the implementation of our WCPG method, and, on the real-world examples we tested, never saw it give a negative answer. In case when matrix $\boldsymbol{T}$ does not pass check our algorithm is programmed to return an error.

6.4.2 Step 3: computing $\boldsymbol{CV}$ and $\boldsymbol{V}^{-1}\boldsymbol{B}$

We compute approximations on matrices $\boldsymbol{CV}$ and $\boldsymbol{V}^{-1}\boldsymbol{B}$ with a certain precision and need to determine the required accuracy of these multiplications such that the impact of these approximations is less than ε_3.

Notate $\boldsymbol{C}' := \boldsymbol{CV} + \boldsymbol{\Delta}_{3_C}$ and $\boldsymbol{B}' := \boldsymbol{V}^{-1}\boldsymbol{B} + \boldsymbol{\Delta}_{3_B}$, where $\boldsymbol{\Delta}_{3_C} \in \mathbb{C}^{p\times n}$ and $\boldsymbol{\Delta}_{3_B} \in \mathbb{C}^{n\times q}$ are error-matrices containing the errors of the two matrix multiplications and the inversion.

Using Frobenius norm, we can bound the error in the approximation of $\boldsymbol{CV}$ and $\boldsymbol{V}^{-1}\boldsymbol{B}$ by $\boldsymbol{C}'$ and $\boldsymbol{B}'$ as follows:

$$\begin{aligned}&\sum_{k=0}^{N}\left|\boldsymbol{CV}\boldsymbol{T}^k\boldsymbol{V}^{-1}\boldsymbol{B} - \boldsymbol{C}'\boldsymbol{T}^k\boldsymbol{B}'\right| \leq \\ &\sum_{k=0}^{N}\left\|\boldsymbol{\Delta}_{3_C}\boldsymbol{T}^k\boldsymbol{B}' + \boldsymbol{C}'\boldsymbol{T}^k\boldsymbol{\Delta}_{3_B} + \boldsymbol{\Delta}_{3_C}\boldsymbol{T}^k\boldsymbol{\Delta}_{3_B}\right\|_F .\end{aligned} \tag{6.45}$$

Since $\|\boldsymbol{T}\|_2 < 1$ holds we have (using Frobenius norm properties)

$$\begin{aligned}&\left\|\boldsymbol{\Delta}_{3_C}\boldsymbol{T}^k\boldsymbol{B}' + \boldsymbol{C}'\boldsymbol{T}^k\boldsymbol{\Delta}_{3_B} + \boldsymbol{\Delta}_{3_C}\boldsymbol{T}^k\boldsymbol{\Delta}_{3_B}\right\|_F \leq \\ &\sqrt{n}\left(\left\|\boldsymbol{\Delta}_{3_C}\right\|_F\left(\left\|\boldsymbol{B}'\right\|_F + \left\|\boldsymbol{\Delta}_{3_B}\right\|_F\right) + \left\|\boldsymbol{C}'\right\|_F\left\|\boldsymbol{\Delta}_{3_B}\right\|_F\right).\end{aligned} \tag{6.46}$$

This bound represents the impact of our approximations for each $k = 0 \dots N$. If (6.46) is bounded by $\frac{1}{N+1} \cdot \varepsilon_3$, then the overall error is less than ε_3. Hence, bounds on the two error-matrices are:

$$\left\| \boldsymbol{\Delta}_{3_C} \right\|_F \leq \frac{1}{3\sqrt{n}} \cdot \frac{1}{N+1} \frac{\varepsilon_3}{\left\| \boldsymbol{C}' \right\|_F} \tag{6.47}$$

$$\left\| \boldsymbol{\Delta}_{3_B} \right\|_F \leq \frac{1}{3\sqrt{n}} \cdot \frac{1}{N+1} \frac{\varepsilon_3}{\left\| \boldsymbol{B}' \right\|_F}. \tag{6.48}$$

Therefore, using bounds on $\left\| \boldsymbol{\Delta}_{3_C} \right\|_F$ and $\left\| \boldsymbol{\Delta}_{3_B} \right\|_F$, we can deduce the required accuracy of our multiprecision matrix multiplication and inversion according to ε_3.

6.4.3 Step 4: powering $\boldsymbol{T}$

Given a square complex matrix $\boldsymbol{T}$ with prevailing main diagonal we need to compute its k^{th} power. Notate

$$\boldsymbol{P}_k := \boldsymbol{T}^k - \boldsymbol{\Pi}_k, \tag{6.49}$$

where $\boldsymbol{\Pi}_k \in \mathbb{C}^{n \times n}$ represents element-by-element the error on the matrix powers, including error propagation from the first to the last power. Using the same simplification as in (6.35) and (6.36) we get the error of computing the approximations $\boldsymbol{P}_k$ rather than the exact powers bounded by

$$\sum_{k=0}^{N} \left| \boldsymbol{C}' \boldsymbol{T}^k \boldsymbol{B}' - \boldsymbol{C}' \boldsymbol{P}_k \boldsymbol{B}' \right| \leq \sum_{k=0}^{N} \left| \boldsymbol{C}' \boldsymbol{\Pi}_k \boldsymbol{B}' \right|. \tag{6.50}$$

Thus a bound on a norm of $\boldsymbol{\Pi}_k$, say $\|\boldsymbol{\Pi}_k\|_F$, is required.

Since we need all the powers of $\boldsymbol{T}$ from 1 to N, we use an iterative scheme to compute them. It is then evident that we may write the recurrence

$$\boldsymbol{P}_k = \boldsymbol{T} \boldsymbol{P}_{k-1} + \boldsymbol{\Gamma}_k, \tag{6.51}$$

where $\boldsymbol{\Gamma}_k \in \mathbb{C}^{n \times n}$ is the error matrix representing the error of the matrix multiplication at step k.

With $\boldsymbol{P}_0 = \boldsymbol{I}$, $\boldsymbol{P}_1 = \boldsymbol{T}$ and using (6.51) we obtain

$$\boldsymbol{P}_k = \boldsymbol{T}^k + \sum_{l=2}^{k} \boldsymbol{T}^{k-l} \boldsymbol{\Gamma}_l. \tag{6.52}$$

Using the condition $\|\boldsymbol{T}\|_2 \leq 1$ and properties of the Frobenius norm we get

$$\|\boldsymbol{\Pi}_k\|_F \leq \left\| \sum_{l=2}^{k} \boldsymbol{T}^{k-l} \boldsymbol{\Gamma}_l \right\|_F \leq \sqrt{n} \sum_{l=2}^{k} \|\boldsymbol{\Gamma}_l\|_F. \tag{6.53}$$

Therefore the impact of approximation of the matrix powers is bounded by

$$\sum_{k=0}^{N} \left| \boldsymbol{C}' \boldsymbol{\Pi}_k \boldsymbol{B}' \right| \le \sqrt{n}(N+1) \sum_{l=2}^{N} \left\| \boldsymbol{C}' \right\|_F \| \boldsymbol{\Gamma}_l \|_F \left\| \boldsymbol{B}' \right\|_F . \tag{6.54}$$

Obviously, if the error of matrix multiplication $\boldsymbol{\Gamma}_l$ satisfies

$$\| \boldsymbol{\Gamma}_l \|_F \le \frac{1}{\sqrt{n}} \cdot \frac{1}{N-1} \cdot \frac{1}{N+1} \cdot \frac{\varepsilon_4}{\left\| \boldsymbol{C}' \right\|_F \left\| \boldsymbol{B}' \right\|_F} \tag{6.55}$$

for $l = 2 \ldots N$, then we have (6.54) to be less than ε_4.

Using (6.55) we may now deduce the required accuracy of matrix multiplications on each step in dependency of ε_4. First, denote the right side of (6.55) as γ. We must guarantee that the error-matrix $\boldsymbol{\Gamma}$ satisfies $\| \boldsymbol{\Gamma} \|_F \le \gamma$, or in other words, that $\sum_{i=1}^{n} \sum_{j=1}^{n} \left| \boldsymbol{\Gamma}_{i,j} \right|^2 \le \gamma^2$. We can strengthen the requirement on $\boldsymbol{\Gamma}$ by requiring for all i, j that $n^2 \max \left| \boldsymbol{\Gamma}_{i,j} \right|^2 \le \gamma^2$. This inequality holds if, for instance, $\left| \Gamma_{i,j} \right| \le \frac{\gamma^2}{n^2}$. Thus, to guarantee the bound (6.55) it is sufficient to call our basic brick algorithm `multiplyAndAdd` with the element-by-element absolute error bound $\delta = \frac{\gamma^2}{n^2}$.

6.4.4 Step 5: computing $\boldsymbol{L}_k$

Once the matrices $\boldsymbol{C}', \boldsymbol{B}'$ and $\boldsymbol{P}_k$ are pre-computed and the error of their computation is bounded, we must evaluate their product. Let $\boldsymbol{L}_k$ be the approximate product of these three matrices at step k:

$$\boldsymbol{L}_k := \boldsymbol{C}' \boldsymbol{P}_k \boldsymbol{B}' + \boldsymbol{\Upsilon}_k, \tag{6.56}$$

where $\boldsymbol{\Upsilon}_k \in \mathbb{C}^{p \times q}$ is the matrix of element-by-element errors for the two matrix multiplications. Then it may be shown that the error of computations induced by this step is bounded by $\sum_{k=0}^{N} |\boldsymbol{\Upsilon}_k|$.

If we want the overall error of approximation on this step to be less than ε_5 then we can easily deduce the required accuracy of each of those multiplications on every iteration of summation algorithm.

6.4.5 Step 6: final summation

Finally the absolute value of the $\boldsymbol{L}_k$ must be taken and the result accumulated in the sum. We remind the reader that if all previous computations were exact, the matrix $\boldsymbol{L}_k$ would be a real matrix and the absolute-value-operation would have been an exact sign manipulation. However, as the computations were in finite-precision arithmetic, $\boldsymbol{L}_k$ is complex with a small imaginary part, which is naturally caused by the errors of computations and must not be neglected. Therefore the element-by-element absolute value of the matrix must be computed.

Since we perform $N+1$ accumulations of absolute values in the result sum $\boldsymbol{S}_N$, it is evident that bounding the error of each such computation by $\frac{1}{N+1}\varepsilon_6$ is sufficient.

Therefore, using this bound for each invokation of our basic brick algorithm `sumAbs` we guarantee bound (6.19).

6.5 Basic bricks

In Section 6.4, we postulated the existence of the following three basic floating-point algorithms: `multiplyAndAdd`, `sumAbs` and `inv`, computing, respectively, the product-sum, the sum in absolute value and the inverse of matrices. Each of these operators was required to satisfy an absolute error bound $|\boldsymbol{\Delta}| < \delta$ to be ensured by the matrix of errors $\boldsymbol{\Delta}$ with respect to scalar δ, given in argument to the algorithm.

Ensuring such an *absolute* error bound is not possible in general when fixed-precision floating-point arithmetic is used. Any such algorithm, when returning its result, must round into that fixed-precision floating-point format. Hence, when the output grows sufficiently large, the unit-in-the-last-place of that format and hence that final rounding error in fixed-precision floating-point arithmetic will grow larger than a set absolute error bound.

In multiple precision floating-point arithmetic, such as offered by software packages like MPFR[5] [124], it is sometimes possible to develop algorithms that will generically determine the output precision of the floating-point variables they return their results in. Hence an absolute error bound as the one we require can be guaranteed. In contrast to classical floating-point arithmetic, such as Higham analyzes, there is no longer any clear, overall *computing precision*, though. Variables just bear the precision that had been determined for them by the previous computation step.

This preliminary clarification made, general description of our three basic bricks `sumAbs`, `inv` and `multiplyAndAdd` is easy. See Appendix 2 for detailed description of the error analysis behind our approach.

For $\texttt{sumAbs}(\boldsymbol{A},\boldsymbol{B},\delta) = \boldsymbol{A} + |\boldsymbol{B}| + \boldsymbol{\Delta}$, we can reason element by element. We need to approximate $\boldsymbol{A}_{ij} + \sqrt{\Re\boldsymbol{B}_{ij}{}^2 + \Im\boldsymbol{B}_{ij}{}^2}$ with absolute error no larger than δ, where $\Re z$ and $\Im z$ are the real and imaginary parts of the complex z. This can be ensured by considering the floating-point exponents of each of $\boldsymbol{A}_{ij}$, $\Re\boldsymbol{B}_{ij}$ and $\Im\boldsymbol{B}_{ij}$ with respect to the floating-point exponent of δ.

For $\texttt{multiplyAndAdd}(\boldsymbol{A},\boldsymbol{B},\boldsymbol{C},\delta) = \boldsymbol{A}\cdot\boldsymbol{B} + \boldsymbol{C} + \boldsymbol{\Delta}$, we can reason in terms of scalar products between $\boldsymbol{A}$ and $\boldsymbol{B}$. The scalar products boil down to summation of products which, in turn, can be done exactly, as we can determine the precision of the $\boldsymbol{A}_{ik}$ and $\boldsymbol{B}_{kj}$. As a matter of course the very same summation can capture the matrix elements $\boldsymbol{C}_{ij}$. Finally, multiple precision

[5] `http://www.mpfr.org/`

floating-point summation with an absolute error bound can be performed with a modified, software-simulated Kulisch accumulator [56], which does not need to be exact but bear just enough precision to satisfy the absolute accuracy bound δ.

Finally, once the `multiplyAndAdd` operator is available, it is possible to implement the matrix inversion algorithm `inv` using a Newton-Raphson-like iteration [125]:

$$\begin{aligned} \boldsymbol{U}_0 &\leftarrow \text{some seed inverse matrix for } \boldsymbol{V}^{-1} \\ \boldsymbol{R}_k &\leftarrow \boldsymbol{V}\boldsymbol{U}_k - \boldsymbol{I}_n \\ \boldsymbol{U}_{k+1} &\leftarrow \boldsymbol{U}_k - \boldsymbol{U}_k\boldsymbol{R} \end{aligned} \tag{6.57}$$

where the iterated matrices $\boldsymbol{U}_k$ converge to $\boldsymbol{V}^{-1}$ provided the `multiplyAndAdd` operations computing $\boldsymbol{R}_k$ and $\boldsymbol{U}_{k+1}$ are performed with enough accuracy, i.e. small enough δ and the seed matrix satisfies some additional properties. In order to ensure these properties with an explicit check, an operator to compute the Frobenius norm of a matrix with a given a priori absolute error bound δ is required. Implementing such a Frobenius norm operator again boils down to summation, as above.

6.6 Numerical examples

The algorithms discussed above were implemented in `C`, using GNU MPFR version 3.1.12, GNU MPFI[6] version 1.5.1 and CLAPACK[7] version 3.2.1. The source code is available online[8]. Our implementation was tested on several real-life and random examples:

- The first example comes from Control Theory: the LTI system is extracted from an active controller of vehicle longitudinal oscillation [126], and WCPG matrix is used to determine the fixed-point arithmetic scaling of state and output.

- The second is a 12^{th}-order Butterworth filter, described in ρ-Direct Form II transposed [127], where WCPG is used during the computation of the transfer function.

- The third one is a large random BIBO stable filter (obtained from the `drss` command of Matlab), with 60 states, 14 inputs and 28 outputs.

- The last one is our random filter from Example 6.1 with 1 input and 1 output.

[6] `https://gforge.inria.fr/projects/mpfi/`
[7] `http://www.netlib.org/clapack/`
[8] `https://scm.gforge.inria.fr/anonscm/git/metalibm/wcpg.git`

Experiments were done on a laptop computer with an Intel Core i5 processor running at 2.8 GHz and 16 GB of RAM.

The numerical results detailed in Table 6.1 show that our algorithm for Worst-Case Peak Gain matrix evaluation with a priori error bound exhibits reasonable performance on typical examples.

Even when the a priori error bound is pushed to compute WCPG results with an accuracy way beyond double precision, the algorithm succeeds in computing a result, even though execution time grows pretty high. We see that for our random filter from Example 6.1 the actual truncation order for a WCPG accurate to double precision is much larger than the initial guess.

Our algorithm includes checks testing that certain properties of matrices are verified, in particular that $\rho(\boldsymbol{A}) < 1$ and $\|\boldsymbol{T}\|_2 \leq 1$. For example, we tested our algorithm on an artificial system with distance between poles and unit circle less than 2^{-60}. Our algorithm correctly detected that the conditions on the system's poles are not fulfilled (i.e. LAPACK and inclusion principles cannot guarantee that $\rho(\boldsymbol{A}) < 1$) and refused to compute the result.

6.7 Extending the WCPG theorem to the range of the state variables

It is easy to apply the WCPG to determine the output interval of state vector as well. For this, we "concatenate" the state vector with the system's output. Denote vector $\boldsymbol{\zeta}(k) := \begin{pmatrix} \boldsymbol{x}(k) \\ \boldsymbol{y}(k) \end{pmatrix} \in \mathbb{R}^{n+p}$ to be the new output vector. Then the state-space relationship takes the form:

$$\mathscr{H}_\zeta \begin{cases} \boldsymbol{x}(k+1) = & \boldsymbol{A}\boldsymbol{x}(k) & + & \boldsymbol{B}\boldsymbol{u}(k) \\ \boldsymbol{\zeta}(k) = & \begin{pmatrix} \boldsymbol{I} \\ \boldsymbol{C} \end{pmatrix} \boldsymbol{x}(k) & + & \begin{pmatrix} \boldsymbol{0} \\ \boldsymbol{D} \end{pmatrix} \boldsymbol{u}(k) \end{cases} . \tag{6.58}$$

Hence, the first n elements of $\boldsymbol{\zeta}(k)$ are just copies of the state vector and the following p elements are computed just like the initial output $\boldsymbol{y}(k)$.

Applying the WCPG upon the system $\mathscr{H}_\zeta$ we obtain a bound on $\boldsymbol{\zeta}(k)$, i.e. both state and output vectors of the initial system.

	Example 1			Example 2			Example 3			Example 4
sizes n, p and q	$n=10,\quad p=11,\quad q=1$			$n=12,\quad p=1,\quad q=25$			$n=60,\quad p=28,\quad q=14$			$n=5,\quad p=1,\quad q=1$
$1-\rho(\boldsymbol{A})$	1.39×10^{-2}			8.65×10^{-3}			1.46×10^{-2}			1.44×10^{-4}
$\max(\boldsymbol{S}_N)$	3.88×10^{1}			5.50×10^{9}			2.64×10^{2}			7.72×10^{2}
$\min(\boldsymbol{S}_N)$	1.29×10^{0}			1.0×10^{0}			1.82×10^{1}			7.72×10^{2}
ε	2^{-5}	2^{-53}	2^{-600}	2^{-5}	2^{-53}	2^{-600}	2^{-5}	2^{-53}	2^{-600}	2^{-53}
N	220	2153	29182	308	4141	47811	510	1749	27485	251510
Inversion iterations	0	2	4	2	3	5	1	2	4	2
overall max precision (bits)	212	293	1401	254	355	1459	232	306	1416	179
Overall execution time (sec)	0.11	1.53	60.06	0.85	11.54	473.20	45.62	177.90	9376.86	5.99

Table 6.1: Numerical results for 2 real-world and 2 constructed example

6.8 WCPG for interval systems

In our problem statement we assume that the coefficients of filter structures are exactly representable in some floating-point format. However, in practical applications coefficients of the filter structures may be represented as intervals with small radii. This representation may be due to measurement errors: for instance in closed-loop control systems [128] state-matrices are often auto-corrected after measurements. To take into account the measurement uncertainty, matrices are represented as intervals. In this Section we propose several ideas on the computation of the WCPG of a system whose coefficients are small[9] intervals.

Absolute value of an interval: there exist several ways to define an absolute value of an interval number. We are going to use the definition by Neumaier [129], which states that for an interval number $[x] = [\underline{x}, \overline{x}]$, its absolute value is a real number $|[x]| \in \mathbb{R}$ such that $|[x]| = \max\left\{\left|\underline{x}\right|, |\overline{x}|\right\}$. In his work [130], Neumaier shows how to achieve a distributivity for interval arithmetic.

Notation: to improve the clarity of formulas, in this section we assume that an interval matrix $[\boldsymbol{M}]$ is centered at $\boldsymbol{M}_c$ and has radius $\Delta\boldsymbol{M}$. We suppose all interval arithmetic to be in multiple-precision. All matrix inequalities and absolute values are applied element by element.

Problem

Given a state-space system $\mathscr{H} = ([\boldsymbol{A}], [\boldsymbol{B}], [\boldsymbol{C}], [\boldsymbol{D}])$, compute an approximation $\boldsymbol{S}$ on the WCPG

$$\langle\langle \mathscr{H} \rangle\rangle = |[\boldsymbol{D}]| + \sum_{k=0}^{\infty} \left| [\boldsymbol{C}][\boldsymbol{A}]^k [\boldsymbol{B}] \right| \tag{6.59}$$

such that two properties are ensured:

1. bound property: $\langle\langle \mathscr{H} \rangle\rangle \leq \boldsymbol{S}$ element-by-element;
2. if coefficients' radii $\to 0$ and precision $\to \infty$ then the exact $\langle\langle \mathscr{H} \rangle\rangle$ is contained in an ε neighborhood of the approximation $\boldsymbol{S}$ for an a priori given small $\varepsilon > 0$.

Naive computation of the Worst-Case Peak Gain with interval matrices may yield to interval explosion due to strong decorrelation. We, on the other hand, propose to adopt the same approach of six-step truncation and summation.

Truncation: in case of interval matrices we can apply the same approach for the truncation order computation as in Section 6.3. However, the problem of computing eigenvalues of an interval matrix arises.

[9] The width of the intervals is of the order of rounding errors due to double precision.

Eigendecomposition of interval matrix: given an interval matrix $[\boldsymbol{A}]$ centered at $\boldsymbol{A}_c$ with radius $\Delta\boldsymbol{A}$ we need to compute the set $[\boldsymbol{\Lambda}]$ which contains the eigenvalue matrices of all $\boldsymbol{A} \in [\boldsymbol{A}]$. To compute the enclosure on $\boldsymbol{\Lambda}$, we use the result of Xu *et al.* [131] which is based on the Generalized Gershgorin discs. It can be summarized in the following way.

Denote $[\boldsymbol{E}_c] = \text{diag}([\lambda_{c_1}], \ldots, [\lambda_{c_n}])$ to be an interval matrix enclosing the eigenvalues of matrix $\boldsymbol{A}_c$. Let $[\boldsymbol{V}_c]$ be an interval matrix of corresponding eigenvectors, i.e. $\forall \boldsymbol{E} \in [\boldsymbol{E}]\ \exists \boldsymbol{V}_c \in [\boldsymbol{V}]$ such that $\boldsymbol{A}_c \subseteq \boldsymbol{V}_c \boldsymbol{E} \boldsymbol{V}_c^{-1}$. Then, the enclosure on the eigenvalues of $[\boldsymbol{A}]$ is a diagonal interval matrix $[\boldsymbol{E}] = \text{diag}([\lambda_1], \ldots, [\lambda_n])$ such that for $i = 1, \ldots, n$

$$[\lambda_i] = [\lambda_{c_i}] + \sum_{j \neq i}^{n} \boldsymbol{H}_{ij}, \tag{6.60}$$

where matrix $\boldsymbol{H} = |[\boldsymbol{V}_c]|\, \Delta\boldsymbol{A}\, \left|[\boldsymbol{V}_c]^{-1}\right|$. Xu *et al.* prove that $[\boldsymbol{\Lambda}] \subseteq [\boldsymbol{E}]$.

Then, we can compute the enclosure $[\boldsymbol{V}]$ on the eigenvectors corresponding to $[\boldsymbol{E}]$ by solving the linear systems of interval equations. In [132] Corsaro *et al.* give a good overview of the State of the Art on the solution of interval linear systems. For example, Neumaier proposes in [133] to combine the Interval Gauss Elimination with preconditioning and proves that such combination gives good results when $\Delta\boldsymbol{A}$ is small.

Finally, using [134–136], we can compute an enclosure on the inverse of an interval matrix. This operation is required during computation of both truncation order and final summation.

Summation For the summation stage we apply the technique similar to the one discussed in Section 6.4. The first intuition would be just to simply use interval arithmetic for the computations. However, to satisfy the second property in the problem statement we must control the precision of computations of centers and radii of all interval matrices, just like in the floating-point case.

6.9 Conclusion

With this work, a reliable, rigorous multiple precision algorithm to compute the Worst-Case Peak Gain matrix has been developed. It relies on Theory of Verified Inclusion, eigenvalue decomposition to perform matrix powering, some multiple-precision arithmetic basic bricks developed to satisfy absolute error bounds and a detailed step-by-step error analysis. A `C` library[10] has been developed and now can be used in our automatic code generator.

To conclude, for a given MIMO filter in state-space representation we can now reliably determine the dynamic range of the output signal, including the worst-case values. Moreover, our algorithm easily extends to the determination of dynamic range of the state variables as well,

[10] `https://scm.gforge.inria.fr/anonscm/git/metalibm/wcpg.git`

which we demonstrate in the next Chapter together with the determination of FxP formats for all variables. Through the conversion between the SIF and state-space representations we may easily deduce the dynamic range for the temporary variables as well.

Since WCPG is just the ℓ_1-norm of the filter's impulse response, our algorithm ensures reliable computation of a digital filter's transfer function (see Chapter 5). We shall also see in the next Chapter that rigorous error analysis of a finite-precision implementation is based on the accurate Worst-Case Peak Gain measure.

However, some efforts are still required to overcome double precision eigenvalue decomposition in LAPACK (specially for close-to-instability LTI systems) by using a multiple precision eigensolver. Additionally, as the proofs on the error bounds are pretty complicated, they should be formalized in a Formal Proof Checker, such as Coq or HolLight.

Another point to be investigated is the distribution of the error budget among computation stages. At the moment we are using an equal distribution between all six stages. It would be interesting to see whether increasing truncation error (hence, decreasing number of terms to be summed) can be beneficial for the speed of the computations. Our intuition is that in terms of time, the dependency between number of terms and accuracy is somewhat linear and, therefore, changing error budget repartition does not bring a real improvement.

We have also proposed a modification of our algorithm for the case of interval matrices. The interval WCPG can be beneficial in implementation of uncertain systems, when uncertainties are due to limited precision of measurements. However, the bottleneck of the proposed approach is the width of the interval matrices: increasing intervals' radii may yield to singularities in the intermediate computations.

CHAPTER 7

DETERMINING RELIABLE FIXED-POINT FORMATS

Once the dynamic range of all variables is determined using the WCPG, we must determine the Fixed-Point formats of all variables for an implementation. At this step we determine the accuracy of the output and the cost of implementation. Usually, we seek to minimize the cost (e.g. in terms of memory or power consumption) while satisfying certain output error bound. To solve this problem, various optimization processes may be applied. To ensure satisfactory optimization time, the evaluation of the FxP formats and output error for each given set of wordlength constraints must be fast.

The goal is, given realization coefficients and wordlength constraints, to determine the Most and the Least Significant Bit (MSB and LSB respectively) positions that will ensure that no overflow occurs. We must also determine the worst-case error bound on the *implemented* system. We remind the reader that we consider the case of recursive LTI filters. The difficulty in implementation of those filters comes from the non-linear propagation of computational errors from one filter iteration to another. Even small rounding errors may be significantly amplified and accumulated through the feedback loop. On top of that, the computation of the FxP formats is based on the estimation of the dynamic range which is an approximated measure. We will show how to rigorously take the eventual approximation error into account and ensure that we compute MSB positions either exactly or overestimate them by at most 1 bit.

In the following we propose a novel analytical approach for determination of the FxP formats for an *arbitrary* filter realization. We use SIF to encompass all LTI filter realizations but again, we demonstrate our approach on state-space structures.

This work is based on an article published at the 49^{th} Asilomar Conference on Signals and Systems in 2015 [13].

7.1 Determining the Fixed-Point Formats

Existing approaches Usually, an idea on the behavior of computational errors in linear filters can be obtained via bit-true simulation [80, 82] of the FxP implementation and then comparison with a reference (floating-point) simulation. The advantage of such technique is that it can be

applied to any realization. An obvious drawback is that simulations may not be exhaustive and comparison is not done with an exact filter but with a floating point, i.e. finite-precision, evaluation. Thus, no guarantee on the result can be obtained with this approach. Moreover, simulations may take significant time [83].

Another way is to apply analytical approaches once a mathematical expression of a numerical accuracy metric is determined. For example, using Interval Arithmetic [84, 85] or Affine Arithmetic [87–89]. These approaches may be more or less efficient, i.e. the dynamic range estimation is relatively fast, but they do not support all kinds of systems. If a new structure is developed, a corresponding analysis approach must be adopted too.

Thus, a new rigorous methodology must be developed. The idea to use the WCPG theorem has already been there for a while [10, 137] but without guarantee on the evaluation of the WCPG it was not rigorous. Moreover, numerous details such as the errors of computation of MSBs and taking into account the propagation of computational errors when determining MSBs were not worked through. In the following we propose a rigorous approach that accounts for all details and proposes a complete methodology for the reliable FxP implementation.

Problem statement The problem of determining the FxP formats for a filter realization $\mathscr{H}$ can be formulated as follows. Let $\mathscr{H}$ be an n^{th} order stable filter in state-space representation:

$$\mathscr{H}\begin{cases} \boldsymbol{x}(k+1) &= \boldsymbol{A}\boldsymbol{x}(k)+\boldsymbol{B}\boldsymbol{u}(k) \\ \boldsymbol{y}(k) &= \boldsymbol{C}\boldsymbol{x}(k)+\boldsymbol{D}\boldsymbol{u}(k) \end{cases} \tag{7.1}$$

with q inputs and p outputs.

Suppose all the inputs to be in an interval bounded by $\bar{\boldsymbol{u}}$: $\forall k\ |\boldsymbol{u}_i(k)| \leq \bar{\boldsymbol{u}}_i$ for $i = 0,\ldots,q$ and that no other information on the spectrum of the input signal is available.

Let $\boldsymbol{w}_x \in \mathbb{Z}^n$ and $\boldsymbol{w}_y \in \mathbb{Z}^p$ be vectors with wordlength constraints on the state and output variables respectively. We wish to determine the least MSB positions $\boldsymbol{m}_x$ and $\boldsymbol{m}_y$ that will ensure that no overflow occurs (see Chapter 2 Section 2.1), i.e. we seek $\boldsymbol{m}_y$ and $\boldsymbol{m}_x$ such that

$$\forall k, \quad \boldsymbol{y}(k) \in [-2^{\boldsymbol{m}_y}; 2^{\boldsymbol{m}_y} - 2^{\boldsymbol{m}_y - \boldsymbol{w}_y + 1}], \tag{7.2}$$

$$\forall k, \quad \boldsymbol{x}(k) \in [-2^{\boldsymbol{m}_x}; 2^{\boldsymbol{m}_x} - 2^{\boldsymbol{m}_x - \boldsymbol{w}_x + 1}]. \tag{7.3}$$

Since the filter $\mathscr{H}$ is linear and input interval is centered at zero, the output interval is also centered at zero. This leads to the following formulation of the problem:

Problem

Given a filter realization $\mathscr{H}$ in the state-space form (7.1), determine the least MSB positions $\boldsymbol{m}_y$ and $\boldsymbol{m}_x$ for the output and state vectors respectively such that

$$\forall k, \quad |\boldsymbol{y}(k)| \leq 2^{\boldsymbol{m}_y} - 2^{\boldsymbol{m}_y - \boldsymbol{w}_y + 1}, \tag{7.4}$$

$$\forall k, \quad |\boldsymbol{x}(k)| \leq 2^{\boldsymbol{m}_x} - 2^{\boldsymbol{m}_x - \boldsymbol{w}_x + 1}. \tag{7.5}$$

7.1.1 Applying the WCPG to compute MSB positions

Applying the WCPG theorem on the filter $\mathscr{H}$ yields a bound on the output interval:

$$\forall k \quad |\boldsymbol{y}_i(k)| \leq (\langle\langle \mathscr{H} \rangle\rangle \, \bar{\boldsymbol{u}})_i, \quad i = 1, \ldots, p. \tag{7.6}$$

Let $\bar{\boldsymbol{y}} := \langle\langle \mathscr{H} \rangle\rangle \, \bar{\boldsymbol{u}}$ be the bound vector. Then, we can determine the FxP formats for the output of a LTI filter $\mathscr{H}$ with the following lemma.

Lemma 7.1. *Let $\mathscr{H} = (\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D})$ be a BIBO-stable MIMO LTI filter and $\bar{\boldsymbol{u}}$ be a bound on the input interval. Suppose the wordlengths $\boldsymbol{w}_y$ are known and $\boldsymbol{w}_{y_i} > 1$, $i = 1, \ldots, p$.*

If for $i = 1, \ldots, p$ the MSBs are computed with

$$\boldsymbol{m}_{y_i} = \left\lceil \log_2(\bar{\boldsymbol{y}}_i) - \log_2\left(1 - 2^{1 - \boldsymbol{w}_{y_i}}\right) \right\rceil \tag{7.7}$$

and the LSBs are computed with $\boldsymbol{\ell}_{y_i} = \boldsymbol{m}_{y_i} + 1 - \boldsymbol{w}_{y_i}$, then for all k $|\boldsymbol{y}_i(k)| \leq 2^{\boldsymbol{m}_{y_i}} - 2^{\boldsymbol{m}_{y_i} - \boldsymbol{w}_{y_i} + 1}$ and $\boldsymbol{m}_{y_i}$ is the least.

Proof. We look for the least $\boldsymbol{m}_y$ such that (7.4) holds. Using the definition of two's complement FxP format from Chapter 2 Section 2.1 and the fact that the bound $\bar{\boldsymbol{y}}$ can be reached, it is sufficient to require that:

$$\bar{\boldsymbol{y}}_i \leq 2^{\boldsymbol{m}_{y_i}} - 2^{\boldsymbol{m}_{y_i} - \boldsymbol{w}_{y_i} + 1}. \tag{7.8}$$

Solving this inequality for $\boldsymbol{m}_{y_i}$ we obtain that the smallest integer, which satisfies the above inequality is given by (7.7). ■

7.1.2 Modification of filter $\mathscr{H}$ to determine bounds on the state variables

Using Lemma 7.1 we can determine the FxP formats for the output of a filter. In order to determine the FxP formats for the state variables, we modify the filter $\mathscr{H}$ like in Section 6.7.

Denote vector $\boldsymbol{\zeta}(k) := \begin{pmatrix} \boldsymbol{x}(k) \\ \boldsymbol{y}(k) \end{pmatrix}$ to be the new output vector. Then the state-space relationship takes the form:

$$\mathscr{H}_\zeta \begin{cases} \boldsymbol{x}(k+1) = \boldsymbol{A}\boldsymbol{x}(k) + \boldsymbol{B}\boldsymbol{u}(k) \\ \boldsymbol{\zeta}(k) = \begin{pmatrix} \boldsymbol{I} \\ \boldsymbol{C} \end{pmatrix} \boldsymbol{x}(k) + \begin{pmatrix} \boldsymbol{0} \\ \boldsymbol{D} \end{pmatrix} \boldsymbol{u}(k) \end{cases} . \tag{7.9}$$

Performing the concatenation of wordlength constraints vectors $\boldsymbol{w}_x$ and $\boldsymbol{w}_x$ gives $\boldsymbol{w}_\zeta \in \mathbb{Z}^{n+p}$. Hence the problem is to find the least MSB vector $\boldsymbol{m}_\zeta$ such that (element-by-element)

$$\forall k, \quad |\boldsymbol{\zeta}(k)| \leq 2^{\boldsymbol{m}_\zeta} - 2^{\boldsymbol{m}_\zeta - \boldsymbol{w}_\zeta + 1}. \tag{7.10}$$

Now, applying the WCPG theorem on the filter $\mathscr{H}_\zeta$ and using Lemma 7.1, we can deduce the MSB positions of the state and output vectors for an implementation of the filter $\mathscr{H}$:

$$\boldsymbol{m}_{\zeta_i} = \left\lceil \log_2(\bar{\boldsymbol{\zeta}}_i) - \log_2\left(1 - 2^{1-\boldsymbol{w}_{\zeta_i}}\right) \right\rceil \quad \text{for } i = 1, \ldots, n+p. \tag{7.11}$$

7.2 Taking rounding errors into account

However, due to the finite-precision degradation what we actually compute is not the exact filter $\mathscr{H}_\zeta$ but an implemented filter $\mathscr{H}_\zeta^\Diamond$:

$$\mathscr{H}_\zeta^\Diamond \begin{cases} \boldsymbol{x}^\Diamond(k+1) = \Diamond_{\boldsymbol{\ell}_x}\left(\boldsymbol{A}\boldsymbol{x}^\Diamond(k) + \boldsymbol{B}\boldsymbol{u}(k)\right) \\ \boldsymbol{\zeta}^\Diamond(k) = \Diamond_{\boldsymbol{\ell}_\zeta}\left(\begin{pmatrix} \boldsymbol{I} \\ \boldsymbol{C} \end{pmatrix} \boldsymbol{x}^\Diamond(k) + \begin{pmatrix} \boldsymbol{0} \\ \boldsymbol{D} \end{pmatrix} \boldsymbol{u}(k)\right) \end{cases} \tag{7.12}$$

where the Sums-of-Products (accumulation of scalar products on the right side) are computed with some rounding operator $\Diamond_\ell$. Suppose, this operator ensures faithful rounding [35], i.e.:

$$|\Diamond_\ell(x) - x| < 2^\ell, \tag{7.13}$$

where ℓ is the Least Significant Bit position of the operator's output.

In [42, 138] it was shown that such an operator can be implemented using some extra guard bits for the accumulation.

Denote the errors due to operator $\Diamond_\ell$ as $\boldsymbol{\varepsilon}_x(k)$ and $\boldsymbol{\varepsilon}_y(k)$ for the state and output vectors, respectively. Essentially, the vectors $\boldsymbol{\varepsilon}_x(k)$ and $\boldsymbol{\varepsilon}_y(k)$ may be associated with the noise which is induced by the filter implementation. Then the implemented filter can be rewritten as

$$\mathscr{H}_\zeta^\Diamond \begin{cases} \boldsymbol{x}^\Diamond(k+1) = \boldsymbol{A}\boldsymbol{x}^\Diamond(k) + \boldsymbol{B}\boldsymbol{u}(k) + \boldsymbol{\varepsilon}_x(k) \\ \boldsymbol{\zeta}^\Diamond(k) = \begin{pmatrix} \boldsymbol{I} \\ \boldsymbol{C} \end{pmatrix} \boldsymbol{x}^\Diamond(k) + \begin{pmatrix} \boldsymbol{0} \\ \boldsymbol{D} \end{pmatrix} \boldsymbol{u}(k) + \begin{pmatrix} \boldsymbol{0} \\ \boldsymbol{I} \end{pmatrix} \boldsymbol{\varepsilon}_y(k) \end{cases} , \tag{7.14}$$

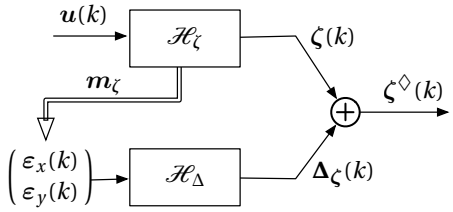


Figure 7.1: Implemented filter decomposition.

where

$$|\boldsymbol{\varepsilon}_x(k)| < 2^{\ell_x}, \quad \left|\boldsymbol{\varepsilon}_y(k)\right| < 2^{\ell_y}.$$

It should be remarked that since the operator $\diamondsuit_l$ is applied $\boldsymbol{\varepsilon}_x(k) \neq \boldsymbol{x}(k) - \boldsymbol{x}^\diamondsuit(k)$ and $\boldsymbol{\varepsilon}_y(k) \neq \boldsymbol{y}(k) - \boldsymbol{y}^\diamondsuit(k)$. As the rounding also affects the filter state, the $\boldsymbol{x}^\diamondsuit(k)$ drifts away from $\boldsymbol{x}(k)$ over time, whereas with $\boldsymbol{\varepsilon}_x(k)$ we consider the error due to one step only.

It can be observed that at each instance of time the state and output vectors are computed out of $\boldsymbol{u}(k)$ and error-vectors, which can be considered as inputs as well. Thanks to the linearity of the filters, we can decompose the actually implemented filter into a sum of the exact filter and an "error-filter" $\mathscr{H}_\Delta$ as shown in Figure 7.1. Note that this "error-filter" is an artificial one; it is not required to be implemented by itself and serves exclusively for error-analysis purposes.

The filter $\mathscr{H}_\Delta$ is obtained by computing the difference between $\mathscr{H}_\zeta^\diamondsuit$ and $\mathscr{H}_\zeta$. This filter takes the rounding errors $\boldsymbol{\varepsilon}(k) := \begin{pmatrix}\boldsymbol{\varepsilon}_x(k)\\ \boldsymbol{\varepsilon}_y(k)\end{pmatrix}$ as input and returns the result of their propagation through the filter:

$$\mathscr{H}_\Delta \begin{cases} \boldsymbol{\Delta}_x(k+1) &= \boldsymbol{A}\boldsymbol{\Delta}_x(k) &+ \begin{pmatrix}\boldsymbol{I} & \boldsymbol{0}\end{pmatrix}\boldsymbol{\varepsilon}(k) \\ \boldsymbol{\Delta}_\zeta(k) &= \begin{pmatrix}\boldsymbol{I}\\ \boldsymbol{C}\end{pmatrix}\boldsymbol{\Delta}_x(k) + & \begin{pmatrix}\boldsymbol{0} & \boldsymbol{0}\\ \boldsymbol{0} & \boldsymbol{I}\end{pmatrix}\boldsymbol{\varepsilon}(k) \end{cases}, \tag{7.15}$$

where, the vector $\boldsymbol{\varepsilon}(k)$ is guaranteed to be in the interval bounded by $\bar{\boldsymbol{\varepsilon}} := 2^{\ell_\zeta}$.

Once the decomposition is done, we can apply the WCPG theorem on the "error-filter" $\mathscr{H}_\Delta$ and deduce the output interval of the computational errors propagated through filter:

$$\forall k, \quad \left|\boldsymbol{\Delta}_\zeta(k)\right| \leq \langle\langle \mathscr{H}_\Delta \rangle\rangle \cdot \bar{\boldsymbol{\varepsilon}}. \tag{7.16}$$

Hence, the output of the implemented filter is bounded with

$$\left|\boldsymbol{\zeta}^\diamondsuit(k)\right| = \left|\boldsymbol{\zeta}(k) + \boldsymbol{\Delta}_\zeta(k)\right| \leq |\boldsymbol{\zeta}(k)| + \left|\boldsymbol{\Delta}_\zeta(k)\right|. \tag{7.17}$$

Remark 7.1. *Obviously, when applying the triangular inequality in* (7.17) *we actually overestimate the bound. From a practical point of view, it can be interpreted as an assumption that*

the input signal that leads to the worst-case output also leads to the worst-case rounding errors. Obviously, this is not generally true. Thus, the triangular inequality bound is not generally attained. Consequently, the "least" MSB positions that we compute further are not the least possible but the least for our way to model the errors and their propagation. In Section 7.8 we propose an approach on dealing with this potential overestimation.

Applying Lemma 7.1 on the implemented filter and using (7.17) we obtain that the MSB vector $\boldsymbol{m}_\zeta^\Diamond$ can be upper bounded by

$$\boldsymbol{m}_{\zeta_i}^\Diamond = \left\lceil \log_2 \left(\left(\langle\langle \mathscr{H}_\zeta \rangle\rangle \cdot \bar{\boldsymbol{u}} \right)_i + \left(\langle\langle \mathscr{H}_\Delta \rangle\rangle \cdot \bar{\boldsymbol{\varepsilon}} \right)_i \right) - \log_2 \left(1 - 2^{1-\boldsymbol{w}_{\zeta_i}} \right) \right\rceil . \tag{7.18}$$

Therefore, the FxP formats $(\boldsymbol{m}_\zeta^\Diamond, \boldsymbol{\ell}_\zeta^\Diamond)$ guarantee that *no overflows occur for the implemented filter*.

Since the input of the error filter $\mathscr{H}_\Delta$ depends on the FxP formats chosen for implementation, we cannot directly use (7.18). The idea is to first compute the FxP formats of the variables in the exact filter $\mathscr{H}$, where computational errors are not taken into account, and use it as an initial guess for implemented filter $\mathscr{H}_\zeta^\Diamond$. Hence, we obtain the following two-step algorithm:

Step 1: Determine the FxP formats $(\boldsymbol{m}_\zeta, \boldsymbol{\ell}_\zeta)$ for the exact filter $\mathscr{H}_\zeta$

Step 2: Construct the "error-filter" $\mathscr{H}_\Delta$, which gives the propagation of the computational errors induced by format $(\boldsymbol{m}_\zeta, \boldsymbol{\ell}_\zeta)$; then, compute the FxP formats $(\boldsymbol{m}_\zeta^\Diamond, \boldsymbol{\ell}_\zeta^\Diamond)$ of the actually implemented filter $\mathscr{H}_\zeta^\Diamond$ using (7.18).

The above algorithm takes into account the filter implementation errors. However, the algorithm itself is implemented in finite-precision and can suffer from rounding errors, which influence the output result. All operations in the MSB computation will induce errors, so what we actually compute are only floating-point approximations $\widehat{\boldsymbol{m}_\zeta}$ and $\widehat{\boldsymbol{m}_\zeta}^\Diamond$. In what follows, we propose an error-analysis of the floating-point evaluation of the MSB positions via (7.7) and (7.18).

7.3 Error analysis of the MSB computation formula

Let us consider the case of $\widehat{\boldsymbol{m}_{\zeta_i}}^\Diamond$ and show afterwards that $\widehat{\boldsymbol{m}_{\zeta_i}}$ is a special case. To reduce the size of expressions, denote

$$\mathfrak{m} := \log_2 \left(\left(\langle\langle \mathscr{H}_\zeta \rangle\rangle \cdot \bar{\boldsymbol{u}} \right)_i + \left(\langle\langle \mathscr{H}_\Delta \rangle\rangle \cdot \bar{\boldsymbol{\varepsilon}} \right)_i \right) - \log_2 \left(1 - 2^{1-\boldsymbol{w}_{\zeta_i}} \right). \tag{7.19}$$

Handling floating-point analysis of multiplications and additions in (7.18) is straightforward using approach by Higham [46]. The difficulty comes from the WCPG matrices which cannot be

computed exactly. Both approximations $\widehat{\langle\langle \mathscr{H}_\zeta \rangle\rangle}$ and $\widehat{\langle\langle \mathscr{H}_\Delta \rangle\rangle}$, even if computed with arbitrary precision, bear some errors ε_{WCPG_ζ} and $\varepsilon_{WCPG_\Delta}$ that satisfy

$$0 \leq \widehat{\langle\langle \mathscr{H}_\Delta \rangle\rangle} - \langle\langle \mathscr{H}_\Delta \rangle\rangle \leq \varepsilon_{WCPG_\zeta} \cdot \mathbb{1}, \tag{7.20}$$

$$0 \leq \widehat{\langle\langle \mathscr{H}_\zeta \rangle\rangle} - \langle\langle \mathscr{H}_\zeta \rangle\rangle \leq \varepsilon_{WCPG_\Delta} \cdot \mathbb{1}. \tag{7.21}$$

Introducing the errors on the WCPG computations into the formula (7.18) we obtain that what we actually compute is

$$\widehat{\boldsymbol{m}_{\zeta_i}}^{\Diamond} \leq \left\lceil \mathfrak{m} + \log_2 \left(1 + \frac{\varepsilon_{WCPG_\zeta} \sum_{j=1}^{q} \bar{\boldsymbol{u}}_j + \varepsilon_{WCPG_\Delta} \sum_{j=1}^{n+p} \bar{\boldsymbol{\varepsilon}}_j}{\left(\langle\langle \mathscr{H}_\zeta \rangle\rangle \bar{\boldsymbol{u}}\right)_i + \left(\langle\langle \mathscr{H}_\Delta \rangle\rangle \bar{\boldsymbol{\varepsilon}}\right)_i} \right) \right\rceil. \tag{7.22}$$

The error term in (7.22) cannot be zero (apart from trivial case with zero $\bar{\boldsymbol{u}}$). However, since we can control the accuracy of the WCPG matrices, we can deduce conditions for the approximation $\widehat{\boldsymbol{m}_{\zeta_i}}^{\Diamond}$ to be off by at most one. Moreover, with the following Lemma we prove that we never underestimate the MSB positions.

Lemma 7.2. *If the WCPG matrices* $\langle\langle \mathscr{H}_\zeta \rangle\rangle$ *and* $\langle\langle \mathscr{H}_\Delta \rangle\rangle$ *are computed such that* (7.20) *and* (7.21) *hold with*

$$\varepsilon_{WCPG_\Delta} < \frac{1}{2} \frac{\left(\underline{\langle\langle \mathscr{H}_\Delta \rangle\rangle} \cdot \bar{\boldsymbol{\varepsilon}}\right)_i}{\sum_{j=1}^{p+n} \bar{\boldsymbol{\varepsilon}}_i} \tag{7.23}$$

$$\varepsilon_{WCPG_\zeta} < \frac{1}{2} \frac{\left(\underline{\langle\langle \mathscr{H}_\zeta \rangle\rangle} \cdot \bar{\boldsymbol{u}}\right)_i}{\sum_{j=1}^{q} \bar{\boldsymbol{u}}_i}, \tag{7.24}$$

where $\underline{\langle\langle \mathscr{H} \rangle\rangle} := |\boldsymbol{D}| + |\boldsymbol{CB}| + |\boldsymbol{CAB}|$*, then*

$$0 \leq \widehat{\boldsymbol{m}_{\zeta_i}}^{\Diamond} - \boldsymbol{m}_{\zeta_i}{}^{\Diamond} \leq 1. \tag{7.25}$$

Proof. Proof by construction, we reason as follows: since the error-term caused by the WCPG floating-point evaluation is positive and the ceil function is increasing, then

$$\widehat{\boldsymbol{m}_{\zeta_i}}^{\Diamond} - \boldsymbol{m}_{\zeta_i}{}^{\Diamond} \geq 0, \tag{7.26}$$

i.e. the floating-point approximation $\widehat{\boldsymbol{m}_{\zeta_i}}^{\Diamond}$ is guaranteed to never be underestimated. However, it can overestimate the MSB position by

$$\widehat{\boldsymbol{m}_{\zeta_i}}^{\Diamond} - \boldsymbol{m}_{\zeta_i}{}^{\Diamond} \leq \left\lceil \underbrace{\mathfrak{m} - \lceil \mathfrak{m} \rceil}_{-1 < \cdot \leq 0} + \log_2 \left(1 + \frac{\varepsilon_{WCPG_\zeta} \sum_{j=1}^{q} \bar{\boldsymbol{u}}_j + \varepsilon_{WCPG_\Delta} \sum_{j=1}^{n+p} \bar{\boldsymbol{\varepsilon}}_j}{\left(\langle\langle \mathscr{H}_\zeta \rangle\rangle \bar{\boldsymbol{u}}\right)_i + \left(\langle\langle \mathscr{H}_\Delta \rangle\rangle \bar{\boldsymbol{\varepsilon}}\right)_i} \right) \right\rceil. \tag{7.27}$$

The approximation $\widehat{\boldsymbol{m}_{\zeta_i}}^{\Diamond}$ overestimates at most by one bit if and only if the error term is contained in the interval $[0,1)$, i.e. if

$$0 \leq \log_2 \left(1 + \frac{\varepsilon_{WCPG_\zeta} \sum_{j=1}^{q} \bar{\boldsymbol{u}}_j + \varepsilon_{WCPG_\Delta} \sum_{j=1}^{n+p} \bar{\boldsymbol{\varepsilon}}_j}{\left(\langle\langle \mathscr{H}_\zeta \rangle\rangle \bar{\boldsymbol{u}} \right)_i + (\langle\langle \mathscr{H}_\Delta \rangle\rangle \bar{\boldsymbol{\varepsilon}})_i} \right) < 1. \tag{7.28}$$

Hence, using the above condition we can deduce the required upper bounds on the ε_{WCPG_ζ} and $\varepsilon_{WCPG_\Delta}$:

$$0 \leq \frac{\varepsilon_{WCPG_\zeta} \sum_{j=1}^{q} \bar{\boldsymbol{u}}_j + \varepsilon_{WCPG_\Delta} \sum_{j=1}^{n+p} \bar{\boldsymbol{\varepsilon}}_j}{\left(\langle\langle \mathscr{H}_\zeta \rangle\rangle \bar{\boldsymbol{u}} \right)_i + (\langle\langle \mathscr{H}_\Delta \rangle\rangle \bar{\boldsymbol{\varepsilon}})_i} < 1. \tag{7.29}$$

Since all the terms are positive, the left inequality is always true. The right inequality in (7.29) is satisfied for instance if

$$\frac{\varepsilon_{WCPG_\zeta} \sum_{j=1}^{q} \bar{\boldsymbol{u}}_j}{\left(\langle\langle \mathscr{H}_\zeta \rangle\rangle \cdot \bar{\boldsymbol{u}} \right)_i} < \frac{1}{2} \qquad \frac{\varepsilon_{WCPG_\Delta} \sum_{j=1}^{n+p} \bar{\boldsymbol{\varepsilon}}_j}{(\langle\langle \mathscr{H}_\Delta \rangle\rangle \cdot \bar{\boldsymbol{\varepsilon}})_i} < \frac{1}{2} \tag{7.30}$$

Rearranging terms we obtain following inequalities on the WCPG computation with error:

$$\varepsilon_{WCPG_\zeta} < \frac{1}{2} \cdot \frac{\left(\langle\langle \mathscr{H}_\zeta \rangle\rangle \bar{\boldsymbol{u}} \right)_i}{\sum_{j=1}^{q} \bar{\boldsymbol{u}}_j} \qquad \varepsilon_{WCPG_\Delta} < \frac{1}{2} \cdot \frac{(\langle\langle \mathscr{H}_\Delta \rangle\rangle \bar{\boldsymbol{\varepsilon}})_i}{\sum_{j=1}^{n+p} \bar{\boldsymbol{\varepsilon}}_j} \tag{7.31}$$

Unfortunately, the above results cannot be used in practice, since they depend themselves on the exact WCPG matrices. Instead, we may use a lower bound of the WCPG matrix, which can be shown to be $\underline{\langle\langle \mathscr{H} \rangle\rangle}$. We can compute this matrix exactly. Obviously,

$$\frac{\left(\underline{\langle\langle \mathscr{H}_\Delta \rangle\rangle} \cdot \bar{\boldsymbol{\varepsilon}} \right)_i}{\sum_{j=1}^{p+n} \bar{\boldsymbol{\varepsilon}}_i} \leq \frac{(\langle\langle \mathscr{H}_\Delta \rangle\rangle \cdot \bar{\boldsymbol{\varepsilon}})_i}{\sum_{j=1}^{p+n} \bar{\boldsymbol{\varepsilon}}_i} \tag{7.32}$$

and

$$\frac{\left(\underline{\langle\langle \mathscr{H}_\zeta \rangle\rangle} \cdot \bar{\boldsymbol{u}} \right)_i}{\sum_{j=1}^{q} \bar{\boldsymbol{u}}_i} \leq \frac{\left(\langle\langle \mathscr{H}_\zeta \rangle\rangle \cdot \bar{\boldsymbol{u}} \right)_i}{\sum_{j=1}^{q} \bar{\boldsymbol{u}}_i}. \tag{7.33}$$

Hence, if the WCPG matrices in the right sides of (7.31) are substituted with their lower bounds, the condition (7.29) stays satisfied and we obtain bounds (7.23) and (7.24).

■

Analogously, Lemma 7.2 can be applied to the computation of $\widehat{\boldsymbol{m}}_{\zeta_i}$ with the terms concerning filter $\mathscr{H}_\Delta$ set to zero.

Algorithm III.3: Reliable determination of the Fixed-Point formats

Input: system $\mathscr{H} = (\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D})$;
input interval bound $\bar{\boldsymbol{u}}$;
wordlength constraints $\boldsymbol{w}_x, \boldsymbol{w}_y$

Output: Formats $(\boldsymbol{m}_x, \boldsymbol{m}_y)$ or an error

$\boldsymbol{w}_\zeta \longleftarrow \begin{pmatrix} \boldsymbol{w}_x \\ \boldsymbol{w}_y \end{pmatrix}$

$\mathscr{H}_\zeta \longleftarrow \left(\boldsymbol{A}, \boldsymbol{B}, \begin{pmatrix} \boldsymbol{I} \\ \boldsymbol{C} \end{pmatrix}, \begin{pmatrix} \boldsymbol{0} \\ \boldsymbol{D} \end{pmatrix}\right)$

$\mathscr{H}_\Delta \longleftarrow \left(\boldsymbol{A}, \begin{pmatrix} \boldsymbol{I} \\ \boldsymbol{0} \end{pmatrix}, \begin{pmatrix} \boldsymbol{I} \\ \boldsymbol{C} \end{pmatrix}, \begin{pmatrix} \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} \end{pmatrix}\right)$

for $i = 0, \ldots, n+p$ **do**
  $[\boldsymbol{m}_{\zeta_i}] \longleftarrow \texttt{interval}\left(\left[\log_2\left(\left\langle\left\langle \mathscr{H}_\zeta \right\rangle\right\rangle \cdot \bar{\boldsymbol{u}}\right)_i - \log_2\left(1 - 2^{1-\boldsymbol{w}_{\zeta_i}}\right)\right]\right)$
  $\boldsymbol{m}_{\max_i} \longleftarrow \overline{\boldsymbol{m}}_{\zeta_i} + \boldsymbol{w}_{\zeta_i} + 1$
end
do
  for $i = 0, \ldots, n+p$ **do**
    $\bar{\boldsymbol{\varepsilon}}_{\zeta_i} \longleftarrow 2^{\overline{\boldsymbol{m}}_{\zeta_i} - \boldsymbol{w}_{\zeta_i} + 1}$
    $[\boldsymbol{m}^{\Diamond}_{\zeta_i}] \longleftarrow \texttt{interval}\left(\left[\log_2\left(\left(\left\langle\left\langle \mathscr{H}_\zeta \right\rangle\right\rangle \cdot \bar{\boldsymbol{u}}\right)_i + \left(\left\langle\left\langle \mathscr{H}_\Delta \right\rangle\right\rangle \cdot \bar{\boldsymbol{\varepsilon}}\right)_i\right) - \log_2\left(1 - 2^{1-\boldsymbol{w}_{\zeta_i}}\right)\right]\right)$
  end
  if $[\boldsymbol{m}^{\Diamond}_{\zeta_i}] == [\boldsymbol{m}_{\zeta_i}]$ *for* $i = 0, \ldots, n+p$ **then**
    return $\overline{\boldsymbol{m}_\zeta}$
  end
  else
    $[\boldsymbol{m}_{\zeta_i}] \longleftarrow [\boldsymbol{m}_{\zeta_i}] + 1$ for $i = 0, \ldots, n+p$
  end
while $\overline{\boldsymbol{m}}^{\Diamond}_\zeta < \boldsymbol{m}_{\max}$;
return *Error*

7.4 Complete algorithm

The two-step algorithm, presented in subsection 7.2 takes into account accumulation of computational errors in a filter over time and Lemma 7.2 presents error-analysis of the MSB position computation procedure. However, one additional fact has not been taken into account.

In most cases the MSB vectors $\widehat{\boldsymbol{m}}_\zeta$ (computed on Step 1) and $\widehat{\boldsymbol{m}}^{\Diamond}_\zeta$ (computed on Step 2) are the same. However, in some cases they are not, which can happen due to one of the following reasons:

- the accumulated rounding error due to the FxP formats $(\widehat{\boldsymbol{m}}_\zeta, \widehat{\boldsymbol{\ell}}_\zeta)$ makes the output of the

actually implemented filter pass over to the next binade; or

- the floating-point approximation $\widehat{\boldsymbol{m}}_{\zeta}^{\Diamond}$ is off by one.

Moreover, we remind the reader that we consider that wordlength constraints are hard, i.e. they cannot be changed. Then, if the MSB position after Step 2 of the algorithm is increased, the LSB position moves along and increases the error. Therefore, the modified format must be re-checked to verify whether the increased error had not propagated in such way that the MSB positions must be increased even more. Hence, the FxP formats determination algorithm gets transformed into the following iterative procedure that goes through three steps:

Step 1: Determine the FxP formats $(\widehat{\boldsymbol{m}}_{\zeta}, \widehat{\boldsymbol{\ell}}_{\zeta})$ for the exact filter $\mathscr{H}_{\zeta}$;

Step 2: Construct the "error-filter" $\mathscr{H}_{\Delta}$ which describes the propagation of the computational errors induced by format $(\widehat{\boldsymbol{m}}_{\zeta}, \widehat{\boldsymbol{\ell}}_{\zeta})$; then, compute the FxP formats $(\widehat{\boldsymbol{m}}_{\zeta}^{\Diamond}, \widehat{\boldsymbol{\ell}}_{\zeta}^{\Diamond})$ of the actually implemented filter $\mathscr{H}_{\zeta}^{\Diamond}$;

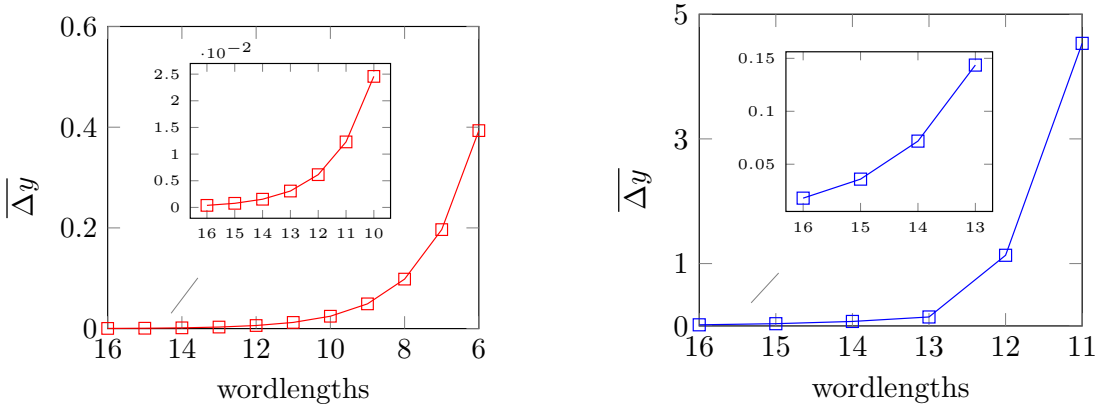
Step 3: If the formats $\widehat{\boldsymbol{m}}_{\zeta_i}^{\Diamond}$ computed on Step 2 are the same as formats computed on Step 1, i.e. $\widehat{\boldsymbol{m}}_{\zeta_i}^{\Diamond} == \widehat{\boldsymbol{m}}_{\zeta_i}$, then we return the formats $(\widehat{\boldsymbol{m}}_{\zeta}^{\Diamond}, \widehat{\boldsymbol{\ell}}_{\zeta}^{\Diamond})$. Otherwise, we increase $\widehat{\boldsymbol{m}}_{\zeta_i}$ by one and repeat the process from Step 2.

Obviously, if the given wordlengths are too small and the filter simply cannot be reliably implemented with those constraints, our procedure enters an infinite loop. A practical stop condition is to let the LSB position to move up until the initial guess of the MSB position. Moving the LSB position further is meaningless since it means that the quantization error is larger in magnitude than the output of the exact filter.

The final procedure is described with the Algorithm III.3, where operator `interval` implies that all internal computations are done with interval arithmetic and vector $\boldsymbol{m}_{\max}$ denotes the maximum bound on the MSBs of the implemented filter. In this algorithm we compute the WCPG matrices with the error bounds deduced with Lemma 7.2.

7.5 Numerical results

The above described algorithm was implemented as a C library, using GNU MPFR version 3.1.12, GNU MPFI version 1.5.1 and the WCPG library [7]. Experiments were done on a laptop computer with an Intel Core i5 processor running at 2.8 GHz and 16 GB of RAM. Consider two examples, one being our key lowpass filter and second being a real-life filter from Software Defined Radio applications.



(a) Example 1: simple lowpass filter

(b) Example 2: real-life filter

Figure 7.2: Evolution of the worst-case error in dependency with the wordlength constraints.

Example 1: Consider our key filter example from Chapter 1 Section 1.4.3 and suppose that all inputs are situated in an interval bounded by $\bar{u} = 1$. We realized this transfer function with a balanced state-space structure[1]. We applied our algorithm to determine the FxP formats for an implementation with wordlengths (for state and output variables) decreasing from 16 to 4 bits. We obtain that implementation is possible only for wordlengths larger than 5 bits. For the worldengths set to 5 our algorithm determined that the computational errors with such wordlengths will yield an overflow. In other words, in Algorithm III.3, we tried to move the MSB positions in additional steps up until the condition on line 19 was satisfied. Interestingly, for implementation with wordlengths from 16 to 6 bits no additional steps were required, i.e. the initial guess formats were always enough.

Via the error filter $\mathscr{H}_\Delta$ we determine the bound on the implementation error for state and output variables of our filter. In Figure 7.2a we illustrate the evolution of the worst-case error of the output $y(k)$ in dependency with the wordlength constraints.

Example 2: This example comes from a recent article [139] on the implementation of Software Defined Radio on reconfigurable architectures. The filter in question is a SISO bandpass filter with a very narrow passband. We realize this filter again with a balanced state-space structure[2] An interested reader may find the coefficients of the structure in Appendix 3.2.

We applied our algorithm to determine the FxP formats for an implementation with wordlengths (for state and output variables) decreasing from 16 to 4 bits. Our algorithm indicates that a reliable implementation is possible only up to 11 bits of wordlength for each state and output.

[1] Coefficients of the state-space were obtained using the standard Python SciPy function `ss`.

[2] The structure choice is based only on its possibility to highlight certain features of the algorithm.

	16 bits		12 bits			
	Step 1	Step 2	Step 1	Step 2	Additional Step 1	Additional Step 2
$\boldsymbol{x}_1$	5	5	5	5	5	5
$\boldsymbol{x}_2$	5	5	5	5	5	5
$\boldsymbol{x}_3$	4	4	4	5	5	5
$\boldsymbol{x}_4$	4	4	4	5	5	5
$\boldsymbol{x}_5$	3	3	3	3	4	4
$\boldsymbol{x}_6$	3	3	3	3	4	4
y	1	1	1	1	2	2
time	1.35s		3.06s			

Table 7.1: Evolution of MSB positions through the algorithm

Decreasing wordlengths to 10 bits and less yields computational errors that may lead to an overflow and there exist no FxP formats that are guaranteed to avoid that.

In contrast to Example 1, our algorithm required several additional steps, i.e. came back to Step 2 several times, for some wordlength constraints. Table 7.1 illustrates the evolution of MSB positions through our algorithm for the wordlength constraints set to 16 and 12 bits. We see that for the wordlength constraints set to 16 bits the MSB positions computed on Step 2 are the same is the initially guessed MSBs, and hence our algorithm stops. For the wordlengths set to 12 bits, the MSBs of $\boldsymbol{x}_3$ and $\boldsymbol{x}_4$ are increased on Step 2. We check whether increasing the quantization error may yield to an overflow and obtain that MSBs for $\boldsymbol{x}_5, \boldsymbol{x}_6$ and y must be increased. After performing one more check we obtain that these formats are indeed reliable.

Analogously to Example 1, we may determine the bound on the implementation error. On Figure 7.2b we illustrate the evolution of a bound on the worst-case error $\overline{\Delta y}$ of the output in dependency with the wordlengths constraints.

7.6 Application to the Specialized Implicit Form

The above approach can easily be extended to the case of filter realizations described with the Specialized Implicit Form (SIF).

Let $\mathscr{R} = \{\boldsymbol{J}, \boldsymbol{K}, \boldsymbol{L}, \boldsymbol{M}, \boldsymbol{N}, \boldsymbol{P}, \boldsymbol{Q}, \boldsymbol{R}, \boldsymbol{S}\}$ be some realization of a MIMO filter described with SIF. Analogously to (7.9), we denote by $\boldsymbol{\zeta}(k) := \begin{pmatrix} \boldsymbol{t}(k+1) \\ \boldsymbol{x}(k) \\ \boldsymbol{y}(k) \end{pmatrix}$ a vector holding the temporary, state and output variables. Then, the corresponding SIF $\mathscr{R}_\zeta$ is described with the following set of

equations:

$$\mathscr{R}_\zeta \begin{cases} \boldsymbol{t}(k+1) &= -\boldsymbol{J}'\boldsymbol{t}(k+1) &+ \boldsymbol{M}\boldsymbol{x}(k) &+ \boldsymbol{N}\boldsymbol{u}(k) \\ \boldsymbol{x}(k+1) &= \boldsymbol{K}\boldsymbol{t}(k+1) &+ \boldsymbol{P}\boldsymbol{x}(k) &+ \boldsymbol{Q}\boldsymbol{u}(k) \\ \boldsymbol{\zeta}(k) &= \begin{pmatrix} \boldsymbol{I} \\ \boldsymbol{0} \\ \boldsymbol{L} \end{pmatrix} \boldsymbol{t}(k+1) &+ \begin{pmatrix} \boldsymbol{0} \\ \boldsymbol{I} \\ \boldsymbol{R} \end{pmatrix} \boldsymbol{x}(k) &+ \begin{pmatrix} \boldsymbol{0} \\ \boldsymbol{0} \\ \boldsymbol{S} \end{pmatrix} \boldsymbol{u}(k) \end{cases}. \tag{7.34}$$

where $\boldsymbol{J}' = \boldsymbol{J} - \boldsymbol{I}$. Then, using the same considerations over the rounding operator as in Section 7.2, the actually implemented filter can be modeled as $\mathscr{R}_\zeta^\Diamond$ described with the following set of equations:

$$\mathscr{R}_\zeta^\Diamond \begin{cases} \boldsymbol{t}^\Diamond(k+1) &= -\boldsymbol{J}'\boldsymbol{t}^\Diamond(k+1) &+ \boldsymbol{M}\boldsymbol{x}^\Diamond(k) &+ \boldsymbol{N}\boldsymbol{u}(k) &+ \begin{pmatrix} \boldsymbol{I} & \boldsymbol{0} & \boldsymbol{0} \end{pmatrix} \boldsymbol{\varepsilon}(k) \\ \boldsymbol{x}^\Diamond(k+1) &= \boldsymbol{K}\boldsymbol{t}^\Diamond(k+1) &+ \boldsymbol{P}\boldsymbol{x}^\Diamond(k) &+ \boldsymbol{Q}\boldsymbol{u}(k) &+ \begin{pmatrix} \boldsymbol{0} & \boldsymbol{I} & \boldsymbol{0} \end{pmatrix} \boldsymbol{\varepsilon}(k) \\ \boldsymbol{\zeta}^\Diamond(k) &= \begin{pmatrix} \boldsymbol{I} \\ \boldsymbol{0} \\ \boldsymbol{L} \end{pmatrix} \boldsymbol{t}^\Diamond(k+1) &+ \begin{pmatrix} \boldsymbol{0} \\ \boldsymbol{I} \\ \boldsymbol{R} \end{pmatrix} \boldsymbol{x}^\Diamond(k) &+ \begin{pmatrix} \boldsymbol{0} \\ \boldsymbol{0} \\ \boldsymbol{S} \end{pmatrix} \boldsymbol{u}(k) &+ \begin{pmatrix} \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{I} \end{pmatrix} \boldsymbol{\varepsilon}(k) \end{cases} \tag{7.35}$$

where $\boldsymbol{\varepsilon}(k) := \begin{pmatrix} \boldsymbol{\varepsilon}_t(k+1) \\ \boldsymbol{\varepsilon}_x(k) \\ \boldsymbol{\varepsilon}_y(k) \end{pmatrix}$ holds the errors due to rounding while computing $\boldsymbol{t}^\Diamond, \boldsymbol{x}^\Diamond$ and $\boldsymbol{y}^\Diamond$.

Analogously to (7.15), the error-filter $\Delta\mathscr{R}_\zeta$ is obtained by computing the difference between $\mathscr{R}_\zeta^\Diamond$ and $\mathscr{R}_\zeta$, and has coefficient matrices

$$\Delta\mathscr{R}_\zeta := \left\{ \boldsymbol{J}, \boldsymbol{K}, \begin{pmatrix} \boldsymbol{I} \\ \boldsymbol{0} \\ \boldsymbol{L} \end{pmatrix}, \boldsymbol{M}, \begin{pmatrix} \boldsymbol{I} & \boldsymbol{0} & \boldsymbol{0} \end{pmatrix}, \boldsymbol{P}, \begin{pmatrix} \boldsymbol{0} & \boldsymbol{I} & \boldsymbol{0} \end{pmatrix}, \begin{pmatrix} \boldsymbol{0} \\ \boldsymbol{I} \\ \boldsymbol{R} \end{pmatrix}, \begin{pmatrix} \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{I} \end{pmatrix} \right\}. \tag{7.36}$$

As it was mentioned in the state of the art, Chapter 3 Section 3.2.1, we can convert any SIF to a state-space model without any computational errors. Thus, we can compute the WCPG $\langle\langle\mathscr{R}\rangle\rangle$ of a system $\mathscr{R}$ in SIF representation by first converting $\mathscr{R}$ into a corresponding state-space system and then applying the WCPG algorithm. Hence, we need to make the following changes in the Algorithm III.3 to adapt to the case of systems in SIF representation:

- instead of $\mathscr{H}_\zeta$ we use $\mathscr{R}_\zeta$ as in (7.34);
- instead of $\mathscr{H}_\Delta$ we use $\mathscr{R}_\Delta$ as in (7.36);
- WCPGs of systems in the SIF representation are computed through conversion to the exact conversion from SIF to state-space;
- vectors $\boldsymbol{m}_\zeta$ and $\boldsymbol{m}_\zeta^\Diamond$ bear the MSB positions of the temporary, state and output variables for the initial and implemented filter respectively.

7.7 Conclusion

In this Chapter we proposed an algorithm for the reliable determination of the FxP formats for all variables involved in a recursive filter. We assume that the wordlength constraints and a bound on the input interval[3] are given. We take computational errors as well as their propagation over time fully into account. We achieve this by decomposing the actually implemented filter into a sum of the exact filter and a special error-filter. By applying the WCPG theorem upon the error filter we get a bound on the worst-case error. We take this bound into account while computing the MSB positions for the variables.

We provide error analysis of the MSB computation formula and show that by adjusting the accuracy of the WCPGs, the computed MSB positions are either exact or overestimated by one. Our approach is fully reliable and we do not use any simulations anywhere in our algorithms. Even despite the off-by-one problem, to our knowledge, our algorithm is the first existing approach that given wordlength constraints provides reliable MSB positions along with a rigorous bound on the computational errors. Moreover, it is easy to turn the problem the "other way around" and, given some output error bound, determine the least MSB positions that ensure this bound. We also support multiple wordlength paradigm, i.e. wordlengths are not necessarily the same for all variables.

By extending this approach to the case of SIF, we enable reliable FxP implementation of any LTI digital filter structure. This contribution represents the kernel functionality of our automatic filter code generator.

Remark that in this work we derived our algorithms only for the case of errors only due to rounding in intermediate computations. As it was mentioned before, another source of error is the quantization of the coefficients. Obviously, both computational and quantization errors must be treated together. In his thesis [6], Lopez modified the decomposition of an implemented filter such that the error filter $\mathscr{H}_\Delta$ also takes into account the propagation of the quantization errors. Using this updated error-filter, we simply apply the Algorthm III.3 upon it and obtain the general approach for reliable FxP implementation of digital filters.

The execution time of our algorithm is dominated by the computation of the WCPG. In most cases, we do not require large accuracy for the WCPG. On the contrary, we often need the WCPG to be accurate to even less than double precision which speeds up the computations. Overall, the execution time of our algorithm permits us to use it repetitively, for instance as part of optimization routines. For example, for hardware targets that support multiple wordlength paradigm, we usually seek to minimize the wordlengths while maintaining certain quality of the output. Or, we may seek to minimize the power consumption of the eventual architecture. To

[3] The interval is supposed to be centered at zero.

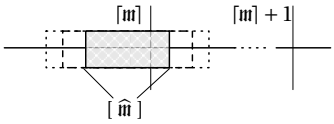


Figure 7.3: Performing Ziv's iteration.

prove the optimality of the computed solutions, we must ensure that we always compute the least MSB positions.

In other words, we need to solve the off-by-one problem of our algorithm. In Section 7.8 we present a possible approach on tackling this problem. We showed that by solving a particular Integer Linear Programming (ILP) problem we may prove that it is safe to take the smaller MSB positions. The condition for that is the absence of solutions of the ILP problem. However, if there exists a solution, we still do not have a guarantee that there indeed exists an input signal that forces our FxP algorithm to fail (e.g. overflow occurs). There is still some work required to investigate this question.

Finally, it is interesting to investigate the case when some information is available concerning the spectrum of the input signal. This information may considerably reduce the dynamic range of the filter and, consequently, the implementation cost. Our goal is to propose an approach on exploiting this information on the behavior of the input signal while still guaranteeing the reliability of the computed FxP Formats. In the Section 7.9 we give the problem statement and our approach on tackling this problem.

7.8 Ongoing work: Off-by-One Problem

In Section 7.3 we showed that since the computation of MSB positions is based on the WCPG measure, which itself cannot be computed exactly, we compute the MSB positions with some error. In Lemma 7.2 we give the minimum accuracy of the WCPG matrices such that the MSB positions are overestimated at most by one. However, the computation of the MSB is based on the quantities (the WCPG and logarithm) that cannot be computed exactly in finite number of steps but only approximately. We deal with the error of computation of logarithm by using multiple precision interval arithmetic but then the problem is to determine what accuracy of the WCPG to choose.

Let $[\widehat{\mathfrak{m}}]$ be an interval estimation of a MSB position via (7.19), where the WCPG matrices were computed with the error bounds deduced in Lemma 7.2. The integer MSB positions are computed as $\left\lceil [\widehat{\mathfrak{m}}] \right\rceil$. However, we may be in trouble if the exact value $\mathfrak{m}$ is very close to the integer $\left\lceil \mathfrak{m} \right\rceil$, since in this case the interval with an approximated MSB position will contain both $\left\lceil \mathfrak{m} \right\rceil$ and

$\left\lceil \mathfrak{m} \right\rceil + 1$. Then, we need to determine the smallest accuracy of the WCPG such that we do not overestimate the MSB position, i.e. the upper bound of interval $\left\lceil [\widehat{\mathfrak{m}}] \right\rceil$ with an approximation of MSB is the same as $\left\lceil \mathfrak{m} \right\rceil$. This problem is an instance of the Table Maker's Dilemma (TMD) [35], which occurs during the implementation of correctly rounded transcendental functions.

One of the strategies of solving the TMD is performing Ziv's iteration [140]. In this approach we reduce the width of the interval $[\widehat{\mathfrak{m}}]$ by iteratively increasing the accuracy of the WCPG computation. However, even after numerous iterations the interval may still contain the integer $\left\lceil \mathfrak{m} \right\rceil$ (see Figure 7.3). This may be due to the following reasons:

(i) the interval is still too large due to the rounding errors;

(ii) the interval is too large due to the triangular inequality in (7.17);

(iii) the propagation of the rounding errors indeed yields the larger MSB position, i.e. $\mathfrak{m} > z$.

Thus, we cannot simply continue increasing the precision of the computations. We propose the following strategy:

- increase the accuracy of the WCPG several times;
- if the interval $[\widehat{\mathfrak{m}}]$ still contains the integer z, try to find whether there exist a state and input vector that yield an overflow if the eventual MSB position is set to z. Roughly said, we try to use the smaller format and prove that an overflow is not possible.

To prove that an overflow (underflow) is not possible, we propose to solve an instance of the following Integer Linear Programming [141–143] problem.

7.8.1 Optimization problem

Let the input signal $\boldsymbol{u}$ be represented in some FxP Format. Suppose that we determine the FxP Formats for the state and output variables and, in case of the Off-by-one problem, we choose the smaller MSB positions. Let $\underline{\boldsymbol{x}}$, $\underline{\boldsymbol{y}}$, $\underline{\boldsymbol{u}}$ be the minimal and $\overline{\boldsymbol{x}}$, $\overline{\boldsymbol{y}}$, $\overline{\boldsymbol{u}}$ the maximum authorized values for the state, output and input vectors respectively.

Then, our goal is to find $\begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{u} \end{pmatrix}$ that are in the deduced FxP formats but for which

$$\begin{pmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{D} \end{pmatrix} \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{u} \end{pmatrix} = \begin{pmatrix} \overline{\boldsymbol{x}} \\ \overline{\boldsymbol{y}} \end{pmatrix} + \begin{pmatrix} \boldsymbol{\delta_x} \\ \boldsymbol{\delta_y} \end{pmatrix} \tag{7.37}$$

with $\begin{pmatrix}\boldsymbol{\delta_x}\\ \boldsymbol{\delta_y}\end{pmatrix} \geq 0$. In other words, we are looking for $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\delta_x}, \boldsymbol{\delta_y}$ such that

$$\begin{pmatrix}\boldsymbol{A} & \boldsymbol{B}\\ \boldsymbol{C} & \boldsymbol{D}\end{pmatrix}\begin{pmatrix}\boldsymbol{x}\\ \boldsymbol{u}\end{pmatrix} \leq \begin{pmatrix}\overline{\boldsymbol{x}}\\ \overline{\boldsymbol{y}}\end{pmatrix} + \begin{pmatrix}\boldsymbol{\delta_x}\\ \boldsymbol{\delta_y}\end{pmatrix} \tag{7.38}$$

$$\begin{pmatrix}\boldsymbol{A} & \boldsymbol{B}\\ \boldsymbol{C} & \boldsymbol{D}\end{pmatrix}\begin{pmatrix}\boldsymbol{x}\\ \boldsymbol{u}\end{pmatrix} \geq \begin{pmatrix}\overline{\boldsymbol{x}}\\ \overline{\boldsymbol{y}}\end{pmatrix} + \begin{pmatrix}\boldsymbol{\delta_x}\\ \boldsymbol{\delta_y}\end{pmatrix} \tag{7.39}$$

$$\begin{pmatrix}\boldsymbol{I} & \boldsymbol{0}\\ \boldsymbol{0} & \boldsymbol{I}\end{pmatrix}\begin{pmatrix}\boldsymbol{x}\\ \boldsymbol{u}\end{pmatrix} \leq \begin{pmatrix}\overline{\boldsymbol{x}}\\ \overline{\boldsymbol{u}}\end{pmatrix} \tag{7.40}$$

$$\begin{pmatrix}\boldsymbol{I} & \boldsymbol{0}\\ \boldsymbol{0} & \boldsymbol{I}\end{pmatrix}\begin{pmatrix}\boldsymbol{x}\\ \boldsymbol{u}\end{pmatrix} \geq \begin{pmatrix}\underline{\boldsymbol{x}}\\ \underline{\boldsymbol{u}}\end{pmatrix} \tag{7.41}$$

Denote $\boldsymbol{x} := \underline{\boldsymbol{x}} + \boldsymbol{x}'$ and $\boldsymbol{u} := \underline{\boldsymbol{u}} + \boldsymbol{u}'$. To formalize the optimization problem, we need to bring the above inequalities to the canonical form, i.e. bring all inequalities to the direction "≤".

Then, the optimization problem is the following:

$$\text{maximize } \boldsymbol{t}^\top \boldsymbol{\xi} \tag{7.42}$$

subject to the following constraints:

$$\boldsymbol{\mathfrak{F}}\boldsymbol{\xi} \leq \boldsymbol{r} \tag{7.43}$$

where

$$\boldsymbol{\xi} = \begin{pmatrix}\boldsymbol{x}'\\ \boldsymbol{u}'\\ \boldsymbol{\delta_x}\\ \boldsymbol{\delta_y}\end{pmatrix} \geq 0, \quad \boldsymbol{t} = \begin{pmatrix}\boldsymbol{0}\\ \boldsymbol{0}\\ \boldsymbol{1}\\ \boldsymbol{1}\end{pmatrix} \tag{7.44}$$

and

$$\boldsymbol{\mathfrak{F}} = \begin{pmatrix}\boldsymbol{A} & \boldsymbol{B} & -\boldsymbol{I} & \boldsymbol{0}\\ \boldsymbol{C} & \boldsymbol{D} & \boldsymbol{0} & -\boldsymbol{I}\\ -\boldsymbol{A} & -\boldsymbol{B} & \boldsymbol{I} & \boldsymbol{0}\\ -\boldsymbol{C} & -\boldsymbol{D} & \boldsymbol{0} & \boldsymbol{I}\\ \boldsymbol{I} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0}\\ \boldsymbol{0} & \boldsymbol{I} & \boldsymbol{0} & \boldsymbol{0}\end{pmatrix}, \quad \boldsymbol{r} = \begin{pmatrix}\left(\begin{pmatrix}\overline{\boldsymbol{x}}\\ \overline{\boldsymbol{y}}\end{pmatrix} - \begin{pmatrix}\boldsymbol{A} & \boldsymbol{B}\\ \boldsymbol{C} & \boldsymbol{D}\end{pmatrix}\begin{pmatrix}\underline{\boldsymbol{x}}\\ \underline{\boldsymbol{y}}\end{pmatrix}\right)\\ \left(\begin{pmatrix}\boldsymbol{A} & \boldsymbol{B}\\ \boldsymbol{C} & \boldsymbol{D}\end{pmatrix}\begin{pmatrix}\underline{\boldsymbol{x}}\\ \underline{\boldsymbol{y}}\end{pmatrix} - \begin{pmatrix}\overline{\boldsymbol{x}}\\ \overline{\boldsymbol{y}}\end{pmatrix}\right)\\ \begin{pmatrix}\overline{\boldsymbol{x}} - \underline{\boldsymbol{x}}\\ \overline{\boldsymbol{u}} - \underline{\boldsymbol{u}}\end{pmatrix}\end{pmatrix}. \tag{7.45}$$

For a proof of equivalence, see Appendix 4.

Suppose the coefficient matrices $\boldsymbol{A},\boldsymbol{B},\boldsymbol{C},\boldsymbol{D}$ are representable in some FxP format. Then, if we scale the constraints $\mathfrak{f}$ and $\boldsymbol{r}$, then it may be shown that the above optimization problem becomes an Integer Linear Programming problem.

Remark 7.2. *Here we considered only the case of overflow. For the case of underflow we need look for $\boldsymbol{x},\boldsymbol{y},\boldsymbol{\Delta}_{\boldsymbol{x}},\boldsymbol{\Delta}_{\boldsymbol{y}}$ such that*

$$\begin{pmatrix}\boldsymbol{A} & \boldsymbol{B}\\ \boldsymbol{C} & \boldsymbol{D}\end{pmatrix}\begin{pmatrix}\boldsymbol{x}\\ \boldsymbol{u}\end{pmatrix} = \begin{pmatrix}\underline{\boldsymbol{x}}\\ \underline{\boldsymbol{y}}\end{pmatrix} - \begin{pmatrix}\boldsymbol{\Delta}_{\boldsymbol{x}}\\ \boldsymbol{\Delta}_{\boldsymbol{y}}\end{pmatrix} \tag{7.46}$$

with $\begin{pmatrix}\boldsymbol{\Delta}_{\boldsymbol{x}}\\ \boldsymbol{\Delta}_{\boldsymbol{y}}\end{pmatrix} \geq 0$*. Obviously, we can proceed analogously.*

To ensure that we find the exact solution, we propose to use a solver over rational numbers. Such solver is available in GNU Linear Programming Kit[4,5] or in SCIP Optimization Suite[6] [144–146].

If there does not exist any solution of the above problem, then the overestimation of the MSB position was due to the application of triangular inequality in (7.17) (see Remark 7.1) and it is safe to take the smaller MSB positions.

However, even if a solution exists, it does not necessarily mean that there is actual overflow: the state vector $\boldsymbol{x}$ in the solution $\begin{pmatrix}\boldsymbol{x}\\ \boldsymbol{u}\end{pmatrix}$ may be not reachable for the dynamic system implemented with the given FxP formats. For continuous-time dynamic problems, the reachability of a given state $\boldsymbol{x}$ is easily verified and there exist some results for the discrete-time models [147–149]. However, these results are not applicable in our case since all the computations are performed with some FxP formats and the set of reachable states is actually an integer "grid". Possible approach consists in representation of inputs, states and outputs as vectors in euclidean integer lattices [150] defined by the FxP formats and then "unrolling" the solution of the optimization problem to the initial state of the system (i.e. zero state). This might be feasible using some SMT solver [151].

Interestingly, after conducting tests over numerous artificial and real-life filters, we have not found an example for which the optimization problem would find the solution. Obviously, our tests are far from being exhaustive, hence we cannot make any conclusion over the results.
To conclude, we have proposed an algorithm of verification whether using a smaller MSB position in the case of Off-by-one problem may yield to an overflow (underflow). Still, in case of

[4] `https://www.gnu.org/software/glpk/`
[5] The exact solver is based on the GMP rational numbers. It is available as an unofficial patch.
[6] `http://scip.zib.de/`

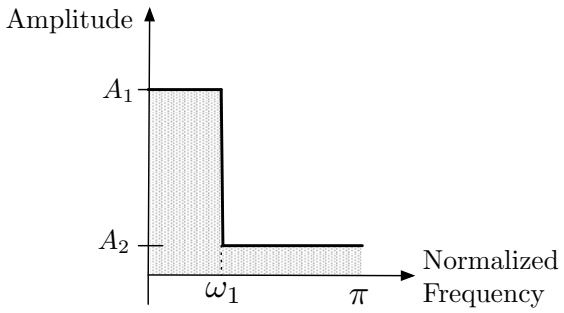


Figure 7.4: Example of input signal specifications

positive answer some work must be done to prove that the computed solution is reachable for the dynamic system or not. While being interesting from the theoretical point of view, in practical applications this problem would be often neglected and 1 bit larger MSB formats would be taken.

7.9 Ongoing work: Taking into account the spectrum of the input signal

When we determine the FxP formats for the reliable implementation of digital filters, we rely on the dynamic range determined via the WCPG measure. This approach is completely rigorous and one can always construct a finite input signal that yields an output arbitrarily close to the bound determined with the WCPG. This input sequence is composed of the supremum of the possible inputs multiplied by the sign of the impulse response shifted in time. In most of the applications this input sequence is highly unlikely to be met though still must be considered to guarantee the reliability of the implementation. However, it may be possible to take into account some information on the behavior of the input signal while still guaranteeing the rigorous evaluation of the dynamic range. In this section we present some ideas on that account.

Often a digital filter's input signal is the output of an existing signal processing system, or describes particular physical process dynamics of which can be expressed as frequency (spectrum) specifications. For example, an input signal that describes temperature usually lies in low frequencies (assuming high enough sampling rate), with higher frequencies dedicated to possible measurement noise.

Generally, specifications of the input signal consist of multiple bands. We will model the frequency specification by a function G of the normalized frequency ω bounding the Discrete-Time Fourier Transform $U(e^{j\omega})$ of the input signal:

$$|U(e^{j\omega})| \leq G(\omega), \quad \forall \omega \in [0, \pi]. \tag{7.47}$$

Let $G(\omega)$ be the frequency specification for an input signal. Then the considered filter $\mathscr{H}$ to implement is given on Figure 7.5.

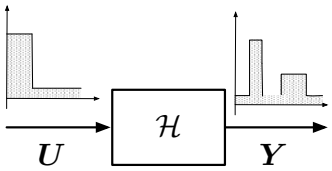


Figure 7.5: Filter that we need to implement.

The idea is to model the initial filter as a cascade of two filters: (1) a system $\mathscr{G}$ that produces an output with frequency response $G(\omega)$; (2) the initial filter. While the first filter is not going to be actually implemented, it permits to take into account the dynamics of the initial input signal when the WCPG theorem is applied upon the cascaded system.

We propose to proceed in following steps:

Step 1: Only an ideal filter can be modeled from the input signal specification. To remove that constraint, we use classical filter design tools (based on Parks-McClellan approximation [152] for example) to find a filter $\mathscr{G}^*$ that has a frequency response $G^*(\omega)$ greater than $G(\omega)$, for example between $G(\omega)$ and $G(\omega)$ enlarged by some margin Δ, as shown on Figure 7.6, i.e:

$$G(\omega) \leq |G^*(e^{j\omega})| \leq G(\omega) + \Delta, \quad \forall \omega \in [0, \pi]. \tag{7.48}$$

Step 2: However, the available tools (Matlab, Scipy, etc.) propose some filter $\mathscr{G}^*$ with no reliable guarantee that $\mathscr{G}^*$ is between $\mathscr{G}$ and $\mathscr{G} + \Delta$ for all ω, due to their internal finite-precision errors. Even though the error is relatively small, this is not sufficient for a reliable magnitude upper-bound. For that purpose, we are going to propose in the next Chapter a reliable method to verify that $\mathscr{G}^*$ satisfies (7.48). If verification does not pass, our algorithm gives an indication by how much $\mathscr{G}^*$ violates its constraints and we can increase Δ and perform again the verification of Step 2.

Step 3: Then, cascade the filters $\mathscr{G}^*$ and $\mathscr{H}$ into a filter $\mathscr{F}$. We obtain the system that is illustrated on Figure 7.7. The WCPG applied on $\mathscr{G}^*$ gives a reliable upper-bound for the magnitude of the output.

Step 4: Finally, apply the rigorous fixed-point format determination algorithm from this Chapter upon $\mathscr{F}$, slightly modified to account for the fact that $\mathscr{G}^*$ will not be part of the implemented filter, and thus no errors will propagate through it.

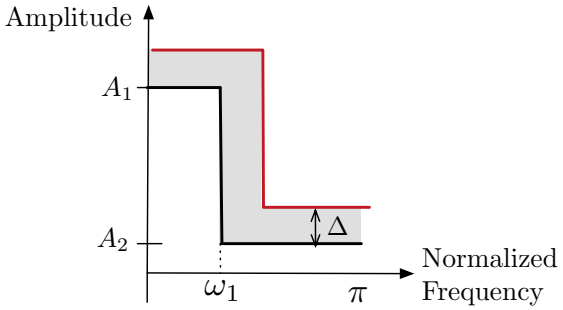


Figure 7.6: Filter $\mathscr{G}^*$ respects $G(\omega)$ enlarged by Δ.

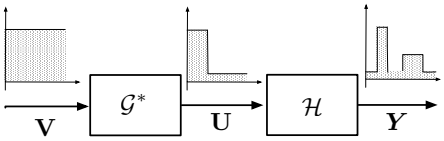


Figure 7.7: Cascaded system $\mathscr{F}$.

With our algorithm from this Chapter we guarantee that for the filter $\mathscr{F}$ the determined MSB positions may be overestimated at most by one. However, for the initial filter $\mathscr{H}$ formats may potentially be overestimated by a larger amount. This is due to the overestimation of the input signal frequency response by Δ (see Figure 7.6), which is required to guarantee an upper bound on the MSBs. However, we can reduce this by making Δ arbitrarily small using high-degree rational Remez approximation, or a Finite Impulse Response Filter for $\mathscr{G}^*$ [153].

The implementation of the algorithm being in the development stage, no numerical examples are available. However, we believe that the approach proposed above may significantly decrease the memory requirements for the implemented system while guaranteeing its reliability An abstract describing this approach has been accepted at the 51st Asilomar Conference on Signals, Systems and Computers.

CHAPTER 8

RIGOROUS VERIFICATION OF IMPLEMENTED FILTER AGAINST ITS FREQUENCY SPECIFICATION

In the previous Chapters we considered only errors in the time domain. We showed how to bound the errors due to the finite-precision computations and quantization. Obviously, we want the implemented filter's output not to be far from the ideal output. However, at the end of the day, we are rather interested in the frequency-domain behavior of the implemented filter, i.e. behavior of its frequency response. People should not think that having a small output error in the time domain necessarily implies a small error in the frequency domain. Thus, for a rigorous implementation of linear filters we must verify whether the implemented filter respects the desired frequency constraints.

In this Chapter we give an approach on the verification of an arbitrary Single Input Single Output linear filter algorithm against given frequency specifications. We first derive an algorithm for the verification of an exact transfer function. This verification boils down to the check of a positivity of a real polynomial on some domain. We use a combination of interval and rational arithmetic in the Sollya [54] tool to provide this rigorous verification. Then, using our transfer function computation algorithm from Chapter 5 and the WCPG, we show how to extend this verification to the transfer function of any filter structure. We provide several use-cases of our algorithm: as a verification tool of an existing design, as a criterion during the choice of the realization and as a tool for the verification of transfer function design methods.

With this approach we provide an easy method to tie the errors due to the coefficient quantization in the time domain with the errors in the frequency domain. Obviously, the finite-precision implementation also influences the frequency response of the filter and the impact of finite-precision computations should be taken into account during filter verification. We give an idea how it can be done with respect ot the spectral behavior of the filter's output.

This work is based on the article [9] published at the IEEE Symposium on Computer Arithmetic (ARITH) in 2017.

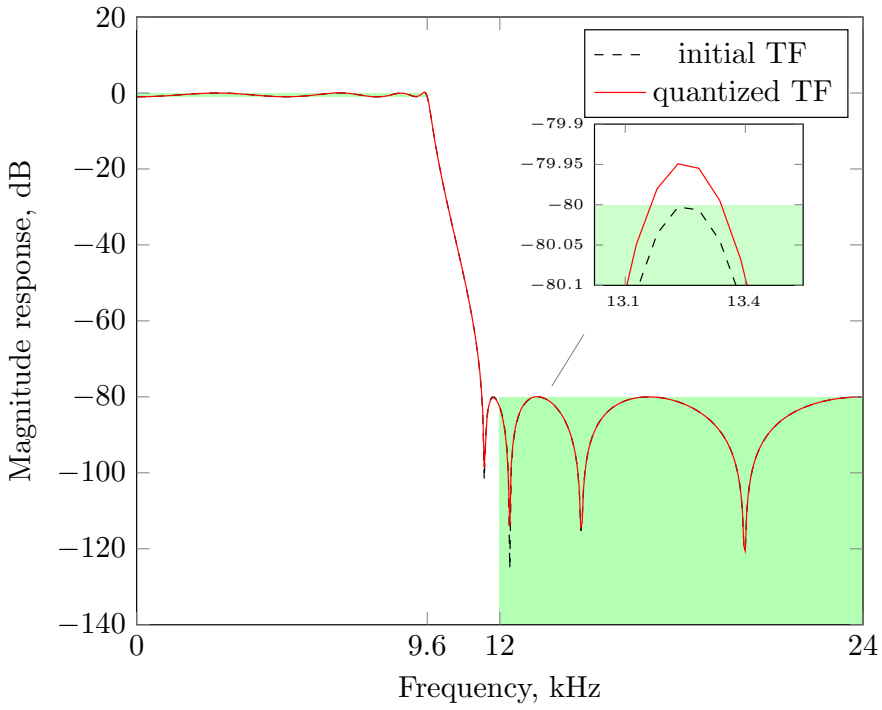


Figure 8.1: Difference between the magnitude responses of the initial and quantized transfer functions.

8.1 Problem statement

Let us start with an example that illustrates that a small error in the time domain does not necessarily implies a small error in the frequency domain.

Example 8.1. *Consider again our key filter example along with the corresponding transfer function from Chapter 1 Section 1.4.3. Suppose that the coefficients of this transfer functions are quantized to 16 bits and we let the difference between the initial and quantized functions be* $\Delta\mathscr{H}$*. Computing the WCPG corresponding to the* $\Delta\mathscr{H}$*, we obtain that the quantization errors are bounded by* $1.92 \cdot 10^{-6}$*. However, we see from Figure 8.1 that the difference between the magnitude responses of the initial and quantized transfer functions is much larger than the time-domain error.*

Thus, we always must ensure that a filter implementation really does satisfy the desired frequency specifications. Formally, band specifications can be described as

$$\underline{\beta} \leq \left|H(e^{j\omega})\right| \leq \overline{\beta}, \quad \forall \omega \in [\omega_1, \omega_2] \subseteq [0, 2\pi]. \tag{8.1}$$

The lower bound $\underline{\beta}$ is equal to 0 for stopbands. Due to Nyquist-Shannon theorem [18], it is only necessary to consider the frequencies in the interval $[0, \pi]$.

For instance, our key filter specifications can be written as

$$\begin{cases} 0 \leq \left|H(e^{j\omega})\right| \leq & 1, \quad \forall \omega \in [0, 0.4\pi] \quad \text{(passband)} \\ \left|H(e^{j\omega})\right| \leq & -80, \quad \forall \omega \in [0.5\pi, \pi] \quad \text{(stopband)}. \end{cases} \tag{8.2}$$

For the sake of generality, we will further describe specifications with a set of inequalities as in (8.1). This way of specifying filters is not the only one: instead of constant bounds $\underline{\beta}$ and $\overline{\beta}$, one might consider upper and lower bounds be given as polynomials varying in the normalized frequency[1] or even measures allowing spectral densities and partial violations of bounds to be taken into account. Considering these alternative ways of filter specifications shall be left to future work.

Then, the verification problem can be stated as follows:

Problem

Given an LTI filter implementation (e.g. data-flow graph) and frequency specifications where constraints on each band are given in the form (8.1), verify whether the implemented filter's transfer function respects the bounds for any frequency. We shall consider all frequency specification bounds as hard constraints. Our algorithm must return – in the first place – a boolean answer whether the specification is satisfied or not. The algorithm must ensure reliable verification, i.e. that no false positive answers are returned.

We tackle the problem by first verifying bounds on a transfer function. Then we use SIF as a unifying framework to encapsulate any digital filter and show how to apply the verification of a transfer function upon any SIF.

8.2 Verifying bounds on a transfer function

The purpose of this Section is to detail our method that verifies that the modulus of a transfer function H stays between two bounds $\underline{\beta}$ and $\overline{\beta}$ for all z taken on a segment of the unit circle, corresponding to a certain frequency band, i.e. $z = e^{j\omega}$ for all $\omega \in \Omega \subseteq [0, 2\pi]$. In the case when the given bounds cannot be verified, our intention is to compute approximations to problematic frequencies for which the bounds are violated.

We proceed in three steps. In Section 8.2.1, we show how we can reduce the given problem to showing that a rational function with real coefficients stays between two bounds for real arguments taken in a subset of $[0, 1]$. In Section 8.2.2, we then further reduce the problem to

[1] Normalized frequency is a unit of measurement of frequency equivalent to cycles/sample. See Chapter 1 Section 1.4.1

showing that a polynomial stays non-negative over a subset of $[0,1]$. In Section 8.2.3, we briefly describe our approach to computing problematic frequencies in the case when the verification does not succeed.

8.2.1 Reducing the problem to a real rational function

We wish to verify that $\underline{\beta} \le |H(z)| \le \overline{\beta}$ for all $z = e^{j\omega}$ with $\omega \in \Omega \subseteq [0, 2\pi]$. We suppose that H is given as a rational function $H(z) = \frac{b(z)}{a(z)}$ with real coefficients. Since we can suppose without lack of generality that $\underline{\beta} \ge 0$, this is equivalent to showing that

$$\underline{\beta}^2 \le |H(z)|^2 \le \overline{\beta}^2, \quad \forall z = e^{j\omega}, \omega \in \Omega. \tag{8.3}$$

Since $z = e^{j\omega}$ and the numerator and denominator polynomials a and b have real coefficients, conjugation of z yields $z^\star = 1/z$ and conjugation of the polynomials has no effect. So we have

$$\begin{aligned} |H(z)|^2 &= \frac{b(z)\,b^\star(z^\star)}{a(z)\,a^\star(z^\star)} \\ &= \frac{b(z)\,b(1/z)}{a(z)\,a(1/z)} \\ &= \frac{v(z)}{w(z)}, \end{aligned} \tag{8.4}$$

where v and w also are polynomials with real coefficients, obtained by simplifying the fraction $\frac{b(z)\,b(1/z)}{a(z)\,a(1/z)}$.

We have hence reduced the problem to verifying that

$$\underline{\beta}^2 \le \frac{v(z)}{w(z)} \le \overline{\beta}^2, \quad \forall z = e^{j\omega}, \omega \in \Omega. \tag{8.5}$$

Taking now $t = \tan\frac{\omega}{2}$, we can write $z = e^{j\omega}$ as

$$z = e^{j\omega} = \cos\omega + j\sin\omega = \frac{1-t^2}{1+t^2} + j\,\frac{2t}{1+t^2}. \tag{8.6}$$

By formally composing v and w with the expression $z = \frac{1-t^2}{1+t^2} + j\,\frac{2t}{1+t^2}$, for example by formal evaluation with Horner's scheme, and clearing numerators and denominators, we can hence obtain four polynomials r, s, $\mathfrak{a}$, $\mathfrak{b}$, all with real coefficients, such that

$$|H(z)|^2 = \frac{v(z)}{w(z)} = \frac{v\left(\frac{1-t^2}{1+t^2} + j\,\frac{2t}{1+t^2}\right)}{w\left(\frac{1-t^2}{1+t^2} + j\,\frac{2t}{1+t^2}\right)} = \frac{r(t) + j\,\mathfrak{a}(t)}{s(t) + j\,\mathfrak{b}(t)}. \tag{8.7}$$

We can now observe that $|H(z)|^2$ is a real number and that the ratio $\frac{r(t)}{s(t)}$ is hence equal to the complex ratio $\frac{r(t)+j\,\mathfrak{a}(t)}{s(t)+j\,\mathfrak{b}(t)}$. We may therefore drop $\mathfrak{a}$ and $\mathfrak{b}$. We have now reduced the problem to verifying that

$$\underline{\beta}^2 \le \frac{r(t)}{s(t)} \le \overline{\beta}^2, \quad \forall t = \tan\frac{\omega}{2}, \omega \in \Omega \subseteq [0, 2\pi], \tag{8.8}$$

where the both polynomials r and s have real coefficients and all other quantities, $\underline{\beta}^2, \overline{\beta}^2, \omega$ and t are all real numbers. We must hence no longer deal with complex ratios and complex numbers and have reduced the problem to verifying the bounds of a real rational function over an interval, subset of the reals.

Unfortunately, the mapping $t = \tan\frac{\omega}{2}$ maps the possible frequencies $\omega \in \Omega \subseteq [0, 2\pi]$ onto to the whole real axis. In our experiments, we found this difficult to handle, partly because the tool we used, Sollya, has very little support for unbounded intervals and partly because having unbounded intervals meant searching for the zeros of certain functions over such unbounded intervals, which we found numerically unstable (see Section 8.2.3 for more details). We hence apply a second mapping: $t = \frac{1-2\xi}{\xi(1-\xi)}$. Still by formally composing the polynomials r and s with the expression $t = \frac{1-2\xi}{\xi(1-\xi)}$, we obtain two polynomials p and q with real coefficients such that

$$|H(z)|^2 = \frac{r(t)}{s(t)} = \frac{p(\xi)}{q(\xi)}. \tag{8.9}$$

In the same step, we reduce the resulting rational function to its least terms to obtain $\frac{p(\xi)}{q(\xi)}$. In order to do so, we extended the tool we used, Sollya, with an algorithm to compute the gcd of two polynomials [154].

As the inverse mapping $\xi = \frac{t+2-\sqrt{t^2+4}}{2t}$ maps the reals onto the interval $[0, 1]$, we have hence reduced our problem to verifying that

$$\underline{\beta}^2 \leq \frac{p(\xi)}{q(\xi)} \leq \overline{\beta}^2, \quad \forall \xi \in \Xi \subseteq [0, 1]. \tag{8.10}$$

8.2.2 Verifying the bounds of a rational function by showing the non-negativity of a polynomial

In order to verify an instance of (8.10), we can suppose that the interval Ξ the arguments ξ vary in is not reduced to a point and that $\underline{\beta}^2 \neq \overline{\beta}^2$ (otherwise a simple evaluation or a check whether p and q are constant polynomials suffice). We can hence reduce the problem further to obtain:

$$-1 \leq \frac{2p(\xi) - \left(\overline{\beta}^2 + \underline{\beta}^2\right) q(\xi)}{\left(\overline{\beta}^2 - \underline{\beta}^2\right) q(\xi)} \leq 1, \quad \forall \xi \in \Xi. \tag{8.11}$$

Let

$$g(\xi) = 2p(\xi) - \left(\overline{\beta}^2 + \underline{\beta}^2\right) q(\xi)$$

and

$$h(\xi) = \left(\overline{\beta}^2 - \underline{\beta}^2\right) q(\xi).$$

It hence suffices to verify that

$$\frac{g(\xi)^2}{h(\xi)^2} \leq 1, \quad \forall \xi \in \Xi \tag{8.12}$$

which is equivalent to showing that

$$h(\xi)^2 - g(\xi)^2 \geq 0, \quad \forall \xi \in \Xi. \tag{8.13}$$

Let $f(\xi) = h(\xi)^2 - g(\xi)^2$. Again f is a polynomial with real coefficients. We have reduced our problem to showing that the value of this polynomial $f(\xi)$ stays non-negative over all $\xi \in \Xi \subseteq [0,1]$, where the interval Ξ is easily obtained from the original frequency domain $\Omega = [\omega_1, \omega_2]$.

Our approach to showing that f stays non-negative over Ξ is similar to the one set out in [49]. We typically perform the following checks:

(i) Check whether f is positive at some (arbitrarily chosen) point $\xi_1 \in \Xi$ by (interval arithmetic) evaluation of f at ξ_1 and that f has no zero over the whole interval Ξ. If so, f is non-negative over the whole interval Ξ.

(ii) Check whether f is positive at both endpoints of the interval Ξ by (interval) evaluation at these endpoints and that it has exactly one zero over whole interval Ξ, not counting multiplicities. The zero it has in the interval hence is of even multiplicity and the polynomial stays non-negative over the whole interval.

(iii) Check whether the interval Ξ can be split into subintervals such that one of the two aforementioned checks become satisfied.

We test whether a polynomial (with real coefficients) has no, one or more zeros over an interval, bounded subset of the reals, utilizing Sturm's theorems on the Sturm sequence of the polynomial, similarly as done in [49]. Sturm's theorem yields the number of real zeros of a real polynomial over a bounded interval, not counting multiplicities [155]. The tool we used, Sollya, includes a fast but rigorous implementation of Sturm's technique [54].

8.2.3 Numerically computing problematic frequencies

In the case when our checks verifying if a given transfer function $H(z)$ stays bounded in modulus by the two bounds $\underline{\beta}$ and $\overline{\beta}$ does not succeed, we numerically compute a list of problematic frequencies $\tilde{\omega}_i$ at which one of the bounds is violated. In contrast to the verification step which is completely rigorous in the sense that will never return a positive answer (i.e. the transfer function satisfies the given bounds while the function actually does not), this numerical step is not fully rigorous. It may miss certain frequencies at which the bounds are violated. It is nevertheless pretty efficient with respect to speeding up the complete LTI filter verification algorithm we set

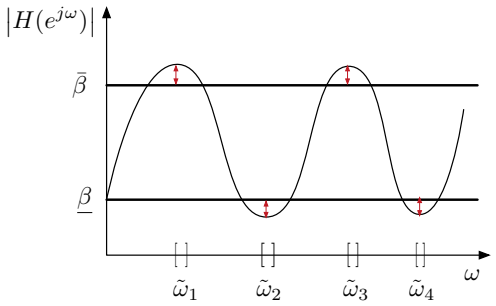


Figure 8.2: If needed our algorithm returns the problematic frequencies as small intervals.

out in Section 8.3, in particular concerning determining a reasonable verification margin (see below).

In our approach, we couple the verification process, described in Sections 8.2.1 and 8.2.2, with the possibly needed step of computing problematic frequencies. These frequencies actually correspond, through the different mappings $\omega \mapsto t \mapsto \xi$, to points ξ at which the polynomial eventually obtained, f, takes negative values. We determine these points as the (negative) extremum points of f. We therefore differentiate f and compute approximations to the zeros of f' by root isolation (still using Sturm's technique) and refinement with Newton-Raphson iterations. The tool we used, Sollya, offers all necessary basic bricks for these computations [54]. Once we obtain a list of points ξ_i at which f becomes negative, we remap these values ξ_i to a list of problematic frequencies, by following the inverse mappings $\xi \mapsto t \mapsto \omega$.

8.3 Verifying bounds for any LTI realization

Once we provided the algorithm for the verification of transfer functions, we may encompass verification of any implemented filter using SIF. We have proposed in Chapter 5 Section 5.2.1 an algorithm for the computation of transfer function corresponding to an arbitrary SIF. Given a SIF realization and an error bound δ, we can compute an approximation $\widehat{H}$ such that $\left|H(e^{j\omega}) - \widehat{H}(e^{j\omega})\right| \leq \delta$ for all frequencies ω. This algorithm is based on the following idea: we compute the difference between the initial filter realization and a realization that exactly corresponds to the approximated transfer function. Then, we can compute a bound on any output of this "error-filter" using the WCPG which also provides a bound on the filter's magnitude response.

Hence, to verify whether the magnitude response of the implemented filter is in the interval $[\underline{\beta}; \bar{\beta}]$, it is sufficient that the approximation $\left|\widehat{H}(e^{j\omega})\right|$ for $\omega \in \Omega$ is in the interval $[\underline{\beta} + \delta; \bar{\beta} - \delta]$. See Figure 8.3 for an illustration.

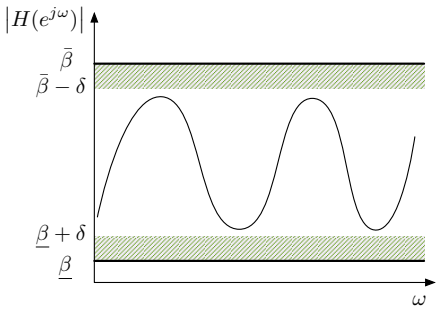
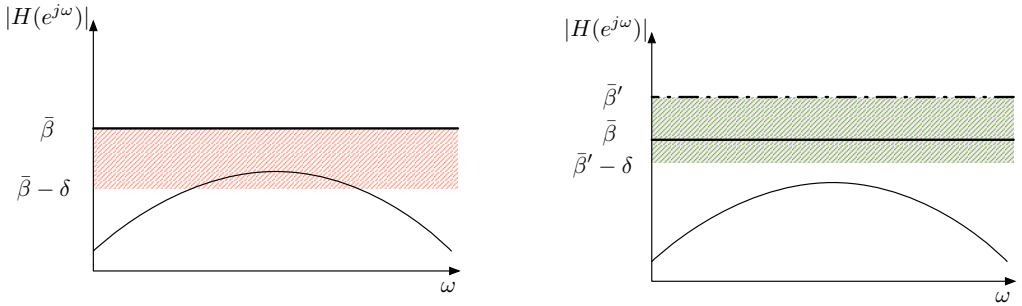


Figure 8.3: Verify whether $\left|\widehat{H}(e^{j\omega})\right|$ is in $[\underline{\beta}+\delta;\bar{\beta}-\delta]$



(a) (b)

Figure 8.4: If verification fails (i), we enlarge the bound (ii).

If the verification is not successful, we use a heuristic to compute a verification margin, such that after adding it to the bounds our algorithm gives a positive answer. First, we obtain a list of problematic frequencies with the algorithm from Section 8.2. Thus, we can compute the maximum excess of the bounds, enlarge the band by this amount and repeat the process. However, we do not know for sure whether this excess is because δ is too large or the magnitude response indeed does not pass the verification.
For example, on Figure 8.4a we do not know whether the approximation $\left|\widehat{H}(e^{j\omega})\right|$ is too close to the bound or δ is too large. In this case, we enlarge the band margin up to some new bound $\overline{\beta}'$ and at the same time decrease δ, and perform the verification again with the updated band. This algorithm may yield a false negative answer if the initial accuracy of the transfer function computation was not large enough, which we never observed on the numerical examples we ran the algorithm on. Algorithm III.4 details our heuristic for the verification. It makes use of several basic brick algorithms:

- `computeTF`$(\mathscr{R},\varepsilon)$ is the Algorithm II.5 which computes the transfer function corresponding to the SIF $\mathscr{R}$; the result approximation is guaranteed to have an approximation error

bounded by ε.

- `checkTF`(g,H,m) is an algorithm for the verification of a transfer function against frequency specifications g enlarged (or reduced, if $m<0$) by the margin $m \in \mathbb{R}$, see Section 8.2 for more detailed description. This algorithm returns a boolean answer.
- `findMinimumMargin`(g,H,m) is an algorithm that, given a transfer function H that does not satisfy frequency specifications g, iteratively deduces the maximum excess of H out of the bands in g,then enlarges bands by this amount and repeats until the verification of H against specifications g is not successful.

The constants INIT_ERROR, LEAST_ERROR and ERROR_FACTOR denote the initial error bound on the transfer function, the smallest error with which we are ready to compute the approximation on the transfer function and the factor by which we decrease the error bound each iteration respectively.

Algorithm III.4: Verification of a realization against frequency specifications

Input: $\mathscr{R}$ - SIF describing filter realization
g - set of frequency specificaions
Output: (boolean, margin) - result of verification and verification margin

```
1  δ ⟵ INIT_ERROR
2  H ⟵ computeTF(ℛ, δ)
3  margin ⟵ findMinimumMargin(g, H, 0)
4  while δ > INIT_ERROR do
5      vefificationMargin ⟵ margin − δ
6      result ⟵ checkTF(g, H, verificationMargin)
7      if result is True then
8          return (True, verificationMargin)
9      end
10     margin ⟵ max{margin + δ, findMinimumMargin(g, H, verificationMargin)}
11     δ ⟵ ERROR_FACTOR ·δ
12     H ⟵ computeTF(ℛ, δ)
13 end
14 return (False, _)
```

8.3.1 Taking into account computational errors

If for the filter which needs to be verified we have information concerning its Fixed-Point implementation, we should take into account the influence of the computational errors (from the time domain) on the frequency behavior of the filter.

Like in the previous Chapter, we consider the filter $\mathscr{H}^{\diamondsuit}$ implemented in Fixed-Point as a sum of the exact filter $\mathscr{H}$ and a special error-filter $\Delta\mathscr{H}$, see Figure 8.5. The error-filter takes as input the signal $\varepsilon(k)$ bearing the bounds on the computational errors that occur on each step of filter computation.

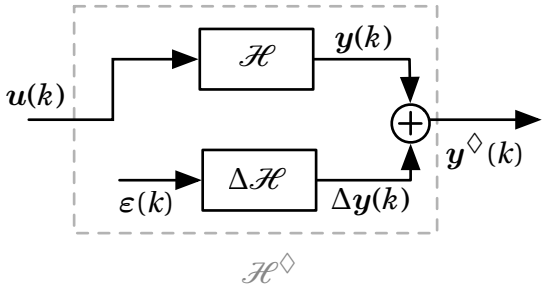


Figure 8.5: Implemented filter

We cannot naively apply the technique similar to the case of transfer function computation, i.e. simply subtract the bound $\|\boldsymbol{\varepsilon}(k)\|\langle\langle\Delta\mathscr{H}\rangle\rangle$ from the bands on filter's frequency response. The reason is that the magnitude response is a sort of a "relative" measure: rescaling the input signal in the time domain will not change the magnitude response. However, even though the signal $\boldsymbol{\varepsilon}(k)$ depends on the inputs signal $\boldsymbol{u}(k)$, this dependence is not linear (and is actually dependent on our rounding operator $\diamondsuit(\cdot)$).

All we can say at this point is that the spectrum of the output of an implemented filter is

$$Y^{\diamondsuit}(z) = H(z)U(z) + \Delta H(z)E(z), \quad |z| < 1, \tag{8.14}$$

where $E(z)$ denotes the $\mathscr{Z}-$transform of the signal $\boldsymbol{\varepsilon}(k)$. If we re-write this equality as

$$Y^{\diamondsuit}(z) = \left(H(z) + \Delta H(z)\frac{E(z)}{U(z)}\right)U(z), \quad |z| < 1, \tag{8.15}$$

we see that the factor $\frac{E(z)}{U(z)}$ represents a sort of "rescaling" the errors relatively to the inputs. Because of the $U(z)$ in the denominator, bounding $\Delta H(z)\frac{E(z)}{U(z)}$ is difficult because we would need to determine the infimum of $U(z)$ for all z on the unit circle.

Thus, at the time being, we can deduce the bound $\|\boldsymbol{\varepsilon}\|_{\infty}\langle\langle\Delta\mathscr{H}\rangle\rangle$ from the $|Y(z)|$, i.e. from the "absolute" measure of the output magnitude and not from the "relative" measure $|H(z)|$.

We believe nevertheless that information on the bound on the output of the error filter might help us to establish clearly the relation between the errors in time and frequency domains.

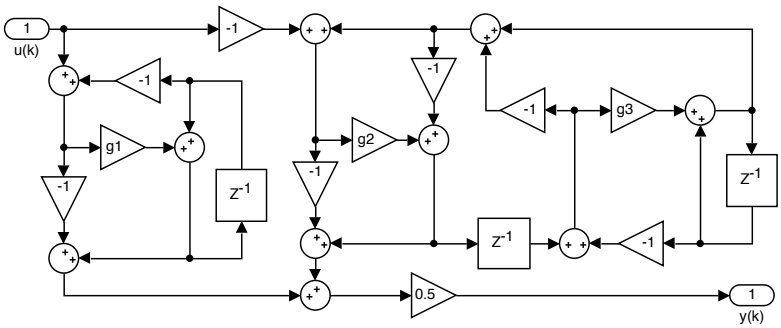


Figure 8.6: Implemented digital filter.

8.4 Numerical examples

The algorithm presented in Section 8.2 was implemented using Python version 2.7.10, the Sollya tool[2] and pythonSollya[3], which is just a wrapper of Sollya tool for usage in Python. We use our implementation of the algorithm for the WCPG computation in arbitrary precision from Chapter 6, which is written in C, using GNU MPFR version 3.1.12, GNU MPFI version 1.5.1 and CLAPACK version 3.2.1. Experiments were done on a laptop computer with an Intel Core i5 processor running at 2.8 GHz and 16 GB of RAM.

We present three different use cases of our algorithm illustrated with examples. The first example is based on a filter given with its Simulink, the second example is based on four different implementations of a filter with simple frequency constraints and the third example is based on our key filter specifications from Chapter 1 Section 1.4.3.

Example 1: Our tool can be used to certify an *already existing filter implementation*. Consider the implementation represented as the Simulink data-flow graph of Figure 8.6, with gains given as 8-bit constants $g_1 = 0.34765625$, $g_2 = 0.3359375$ and $g_3 = 0.08496094$.

The task is to verify whether the given implementation is a lowpass filter which satisfies following normalized frequency constraints: passband $\omega_p = 0.1$ with passband amplitude between 1 dB and 3 dB, stopband starting $\omega_s = 0.3$ with minimum attenuation 20 dB.
First, we exactly convert the Simulink graph to SIF (for which we can compute the transfer function with arbitrary precision). Then, we apply our algorithm from Section 8.3 and obtain a positive result in 1.9s. Thus, we obtain a guarantee that the given filter implementation satisfies the desired frequency response requirements. Remark that Figure 8.6 represents a Lattice Wave Digital Filter, the coefficients of which are usually derived from the specifications using direct formulas without actually computing the filter's transfer function [101].

[2]Commit 2828 at master, `https://scm.gforge.inria.fr/anonscm/git/sollya/sollya.git`
[3]Commit 146 at master `https://scm.gforge.inria.fr/anonscm/git/metalibm/pythonsollya.git`

	wordlength	32	16	8
DFIIt	margin	✓	unstable	unstable
	time	12s	-	-
ρ DFIIt	margin	✓	✓	$4.68 \cdot 10^{-3}$ dB
	time	13s	4s	104s
state-space Balanced	margin	$6.16 \cdot 10^{-10}$ dB	✓	$6.71 \cdot 10^{-1}$ dB
	time	12s	18s	92s
Lattice Wave	margin	$3.80 \cdot 10^{-10}$ dB	✓	$1.73 \cdot 10^{-2}$ dB
	time	920s	4s	200s

Table 8.1: Checking quantized realizations.

Example 2: Suppose we design a filter satisfying following frequency constraints: sampling frequency $F_s = 48$ kHz, passband up to 2.4 kHz with amplitude in $[-0.5\text{dB}; 0.5\text{dB}]$; stopband starting 7.2 kHz with minimum attenuation 80 dB.

Suppose also that we have four different filter realizations: a Direct Form II transposed, a Direct Form II transposed with optimized ρ operator [156], a balanced state-space [157] and a Lattice Wave Digital Filter [101]. These realizations have different number of coefficients and were designed using different approaches that are beyond the scope of this demonstration. Their floating-point coefficients are results of various approximations and optimizations specific for each structure (the reader can find them in the Appendix 3.3).

Our goal is to verify whether after quantization of the coefficients to different formats these realizations will satisfy the given frequency response constraints.

The results of verification of the above realizations are listed in Table 8.1 (here we were interested in quantization to 32, 16 and 8 bits). We can observe that for certain filters a specification is fulfilled only if some margin is added; for certain cases that margin is as large as 0.67 dB. Some interesting effects may occur: the balanced state-space and Lattice Wave structures with 32 bits coefficients do not pass the verification but the coefficient quantization effects cancel out for 16 bits and verification “luckily” passes.

If 16-bit quantization is the target format, we see that except DFIIt, all structures verify the specifications and now the designer may concentrate on choosing the best realization according to other criteria (for example, number of coefficients or datapath delay). If 8-bit quantization is the target format, the verification margins computed with our tool can be used to redesign the realizations and repeat the verification. On the other hand, if the realizations are impossible to redesign, we can determine what is the maximum quantization of the coefficients for which our tool gives a positive answer.

Finally, for the cases when the transfer function respects the band specifications our algorithm

		Butterworth	Chebyshev	Elliptic
Matlab	OK/margin	$5.44 \cdot 10^{-12}$ dB	✓	$8.94 \cdot 10^{-2}$ dB
	time	167s	17s	16s
Python SciPy	OK/margin	$8.99 \cdot 10^{-5}$ dB	$7.34 \cdot 10^{-2}$	$8.94 \cdot 10^{-2}$
	time	70s	97s	22s

Table 8.2: Verification of transfer function design methods for our key specification examples.

gives an answer quickly. Most time is spent on computing the verification margin, especially when an overflow is on the edge of the band.

Example 3: Finally, our algorithm can be used to certify the result of different design methods even before structure choice or quantization.

Recall our key example of frequency specifications for a lowpass filter. We saw in the beginning of this chapter that the quantization of the corresponding transfer function (designed with Elliptic method in Matlab) to 16 bits resulted in obvious violation of frequency specifications. Now, we propose to verify whether the Elliptic method from Matlab generated a valid transfer function or, perhaps, some other design method is better.

We propose to generate transfer functions (of degrees determined as minimal by Matlab) for our key filter example using Butterworth, Chebyshev II and Elliptic methods. We use standard implementations of these methods in Matlab and SciPy.

From Table 8.2 we see that only Chebyshev design method from Matlab instantly satisfies the specifications. However, Butterworth method from Matlab has a small design margin that is probably due to the double precision computations. On the other hand, we see that there are clearly some issues in the design of the Elliptic filters in both SciPy and Matlab. This may be due to the fact that both tools propose a way too small degree for the transfer function of this type. On the other hand, even these margins may be acceptable depending on filter designer's needs.

Thus, our algorithm can be used to certify that the filter design method is rigorous, or give the designer a perception of the sufficient correction of the design margin.

8.5 Conclusion

In this Chapter, a rigorous method to verify a transfer function of LTI digital filters against band specifications in the frequency domain has been developed. It relies on translating the problem of verifying bounds on magnitude response evaluated on a unit circle to the verification of positivity of a real polynomial. Our algorithm guarantees that no false positives occur. In the

case of unsuccessful verification a list of problematic frequencies is provided, for which we compute the maximum violation of the band specification. We propose an implementation using a combination of interval and rational arithmetic in Sollya tool.

We applied this method to develop an approach on the verification of *any* implemented filter. It relies on the multiple precision computation of a transfer function corresponding to the filter in SIF representation that we have developed in Chapter 5. The core idea behind the method is to bound the error of approximation of the transfer function using the WCPG. Thus, through the WCPG measure we obtain a relationship between the frequency domain errors (due to the approximation or quantization) with the time domain errors.

This approach opens various possibilities on the verification of digital filters, as well as for transfer function design tools. Even a naive comparison of the state-of-the-art tools and methods revealed weaknesses in the Python SciPy library, as well as frequent issues with the Elliptic method for IIR filter design in both Scipy and Matlab.

Moreover, our verification algorithm can be applied to compare different filter structures with various Fixed-Point settings for the coefficients. Such a comparison offers a filter designer an overview of the implementation possibilities. On top of that, the information on the verification margin can be used to correct a design that fails to verify the initial band specifications. For instance, we can narrow down the initial band by the verification margin and re-design the filter with the new band specifications. Then, the re-computed filter with quantized coefficients should respect the initial conditions. Finally, using the list of problematic frequencies we can probably improve the rational Remez [24] algorithm that is often behind the transfer function design.

To conclude, this algorithm perfectly integrates in our automatic code generator as a tool for the *a posteriori* verification of an implemented filter, i.e. a certification that after all manipulations with filter's coefficients the frequency response indeed has desired behavior. Also, we can use our algorithm as a tool assisting the filter designer even on early stages of the implementation.

However, overall time-efficiency of our implementation can still be improved. Passing too much time on the computation of the verification margin can be a significant drawback were the algorithm to be used during the exploration of a large design space. Nonetheless, this limitation can be overcome by setting an initial acceptable design margin, which directly depends on the application of the filter.

Naturally, the next step would be determining the relationship between the *computational* errors in time domain with the behavior of filter's magnitude response. However, we show that this task is not as straightforward as in case of quantization errors. For the moment, we only described the spectrum of the output of the filter. We have a strong belief that the information on the upper bound of the output error (error due to the rounding errors) will help us in solving this problem.

PART IV

HARDWARE CODE GENERATION

CHAPTER 9

LTI FILTERS COMPUTED JUST RIGHT ON FPGA. IMPLEMENTATION OF DIRECT FORM I

In this Chapter we present our first steps towards reliable implementation of recursive filters on hardware targets, in particular on Field Programmable Gate Arrays (FPGAs) [158]. This work was done in collaboration with the FloPoCo [15] project that provides tools for the generation of VHDL, a hardware description language. In particular, FloPoCo provides code generation for Fixed-Point (FxP) cores under the motto *"computed just right"*. More precisely, the generated architectures are guaranteed to have an implementation error no larger than the weight of the least significant bit of the output.

One of the cores provided by FloPoCo is a Sum of Products by Constants (SOPC), i.e. an architecture computing

$$r = \sum_{i=1}^{N} c_i v_i \tag{9.1}$$

for a set of real constants c_i and a set of FxP inputs v_i such that the computed result is a faithful rounding of the exact result r.

On the other hand, one of the simplest structures for the SISO recursive filters, Direct Form I (DFI) is one big SOPC plus some delays. While not being used for an implementation of high-order filters due to high sensitivity to rounding errors, this structure can be used for second- or third-order filters. Thus, we decided to do first experiments in the reliable implementation of LTI filters in hardware by proposing the implementation of DFI using FloPoCo cores as basic bricks. Looking ahead, we can say that this work will directly lead to the general approach applicable to any filter structure, described with SIF.

In Chapter 7 we showed how to determine reliable FxP formats with hard wordlength constraints and how to estimate the eventual output error using the Worst-Case Peak Gain (WCPG) of the system. Now we are turning the problem around: the error constraint is considered to be hard (in the sense that it is *guaranteed* be met) while the wordlengths may be changed, i.e. increased if needed.

To specify an implementation, a designer needs not only to determine the formats of the

variables involved in computations but also to quantize the coefficients of the structure. Obviously, we are interested in the coarsest quantization while respecting the output error bound. The main contribution of this Chapter is to show that these design decisions can be automated such that designer may focus on other design parameters.

We use the techniques proposed in the previous Chapters to provide an error analysis that captures not only the rounding errors but also their infinite accumulation in IIR filters. This error analysis then guides the design of hardware satisfying the accuracy specification at the minimal hardware cost.

This work is based on a paper "Hardware IIR Filters: Direct Form I Computing Just Right" which has not yet been submitted but available online as a technical report [16].

9.1 Introduction

We remind the reader that the transfer function $H(z)$ of a LTI filter is given with

$$H(z) = \frac{\sum_{i=0}^{n_b} b_i z^{-i}}{1 + \sum_{i=1}^{n_a} a_i z^{-i}}, \quad \forall z \in \mathbb{C}. \tag{9.2}$$

In time domain it corresponds to the following Constant-Coefficient Difference Equation:

$$y(k) = \sum_{i=0}^{n_b} b_i u(k-i) - \sum_{i=1}^{n_a} a_i y(k-i). \tag{9.3}$$

Equation (9.2) or (9.3), along with a mathematical definition of each coefficient a_i and b_i, constitute the *mathematical specification* of the filter algorithm that we will implement.

In the following we deal with the *implementation* of such a specification as FxP hardware operating on low-precision data. Figure 9.1 illustrates the simple interface of the tool that we propose. The coefficients a_i and b_i are considered as real numbers: they may be provided as high-precision numbers from e.g. Matlab, or even as mathematical formulae such as `sin(3*pi/8)`. The integers ℓ_{in} and ℓ_{out} respectively denote the bit position of the least significant bits of the input and of the result. In the proposed approach, ℓ_{out} specifies output precision and output accuracy. Without loss of generality, the MSB of the input is set to 1.

Our tool provides the construction of a minimal-cost architecture of proven last-bit accuracy. We provide implementation on FPGAs based on Look-Up Tables (LUTs). We put the FloPoCo tool in charge of the automatic generation of VHDL for our architecture.

Our tool also incorporates several architectural novelties. The SOPCs are built using a modification of the KCM[1] algorithm [159, 160] that manages multiplications by a real constant

[1] Ken Chapman's multiplier for constant coefficient multiplication.

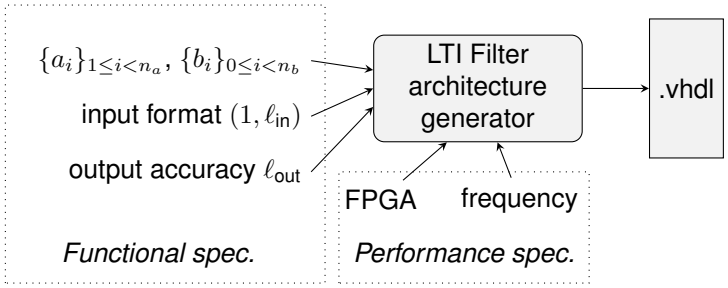


Figure 9.1: Interface to the proposed tool.

without needing to truncate it first [161]. The summation is efficiently performed thanks to the BitHeap framework recently introduced in FloPoCo [162]. These technical choices lead to logic-only architectures suited even to low-end FPGAs, a choice motivated by work on implementing the ZigBee protocol standard [163]. However, the same philosophy could be used to build other architecture generators, for instance exploiting embedded multipliers and DSP blocks.

First, we justify our choice of the faithful rounding as the main goal.

9.1.1 Perfect and faithful rounding

The rounding of a real such as our ideal output y to the nearest fixed-point number of precision ℓ bits is denoted $\circ_\ell(y)$. In the case of round to nearest, it entails an error $|\circ_\ell(y(k)) - y(k)| \leq 2^{\ell-1}$. So, the best we can do, when implementing (9.3) with a precision-ℓ output, is a *perfectly rounded* computation with an error bound $\overline{\varepsilon}_{\text{out}} = 2^{\ell-1}$.

Unfortunately, reaching perfect rounding accuracy may require arbitrarily large intermediate precision. This is not acceptable in an architecture that has very limited resources. We therefore impose a slightly relaxed constraint: $\overline{\varepsilon}_{\text{out}} < 2^{\ell}$. We call this *last-bit accuracy*, because the error must be smaller than the value of the last (LSB) bit of the result. It is sometimes also *faithful rounding* in the literature.

The main reason for choosing last-bit accuracy over perfect rounding is that, as will be shown in the sequel, it can be reached with very limited hardware overhead. Therefore, in terms of cost and efficiency, an architecture that is last-bit-accurate to ℓ bits makes more sense than a perfectly rounded architecture to $\ell - 1$ bits, for the same accuracy bound 2^{ℓ}.

The main conclusion of this discussion is the following: specifying the output precision (ℓ_{out} on Figure 9.1) is enough to also specify the accuracy of the implementation.

This is a huge improvement over classical approaches, such as the various Matlab toolboxes that generate hardware filters. In such approaches [164], one must provide ℓ_{out} and various other parameters that impact the accuracy, then measure via simulations the resulting accuracy,

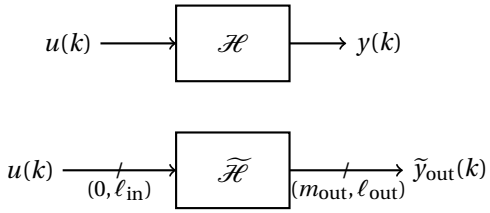


Figure 9.2: The ideal filter (top) and its implementation (bottom)

and iterate until a satisfactory implementation has been reached. Not only is the proposed interface simpler, it also enables architecture optimization under a strict accuracy constraint. An optimal architecture will be an architecture that is accurate enough but no more.

9.2 Error analysis of direct-form LTI filter implementations.

On the one hand we have the exact mathematical definition (9.3) of a filter $\mathscr{H}$. On the other hand, we aim at providing a fixed-point implementation $\widetilde{\mathscr{H}}$ with a last-bit accuracy. The two filters $\mathscr{H}$ and $\widetilde{\mathscr{H}}$ are exhibited on Figure 9.2.

The first thing to do is to determine the MSB position of the output (m_{out}). We use the Worst-Case Peak Gain Algorithm presented in the Chapter 6 and our classical decomposition of an implemented filter into a sum of an error-filter and the exact filter. As we have seen in Chapter 7, the rounding errors can propagate all the way to the MSB. Since we are going to ensure that the rounding errors will be bounded by $2^{\ell_{\text{out}}}$, we just use this information and apply Lemma 7.2 from Chapter 7 to compute the MSB position.

Instead of computing $y(k)$ with equation (9.3), we will compute an approximation $\widetilde{y}(k)$ of the involved Sum of Product by Constants (SOPC) using some internal format $(m_{\text{out}}, \ell_{\text{ext}})$. Our goal is to determine this internal LSB position that satisfies the last-bit accuracy for the format $(m_{\text{out}}, \ell_{\text{out}})$.

The value $\widetilde{y}(k)$ is an approximation of the ideal value $y(k)$ computed with some rounding errors, and the final output $\widetilde{y}_{\text{out}}(k)$ will be the rounding of this intermediate value $\widetilde{y}(k)$ to the output format. This scheme is summed up by the abstract architecture of Figure 9.3. Then, our goal is to solve the following problem:

Problem

For a LTI filter evaluated with the abstract architecture from Figure 9.3, determine the least internal format $(m_{\text{out}}, \ell_{\text{ext}})$ that guarantees the output faithfully rounded to the format $(m_{\text{out}}, \ell_{\text{out}})$.

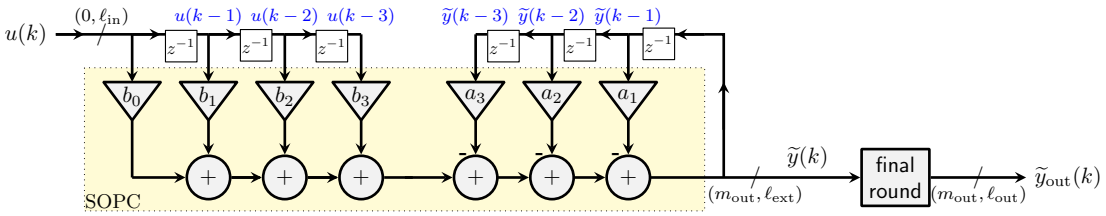


Figure 9.3: Abstract architecture for the direct form realization of an LTI filter

Formally, the overall evaluation error may be defined as

$$\varepsilon_{\text{out}}(k) = \widetilde{y}_{\text{out}}(k) - y(k). \tag{9.4}$$

Let us now decompose this error into its sources and adapt our approach from Chapter 7 for this new task.

Final rounding of the internal format The first source of error that is easy to isolate is the final rounding error. The architecture needs to internally use a fixed-point format that offers extended precision with respect to the input/output format. This extended format $(m_{\text{out}}, \ell_{\text{ext}})$ offers additional LSB bits sometimes called *guard bits*. Eventually we need to round the intermediate result in this extended format to the output format (in the "final round" box on Figure 9.3). This entails an additional error ε_{f}, formally defined as

$$\varepsilon_{\text{f}}(k) = \widetilde{y}_{\text{out}}(k) - \widetilde{y}(k). \tag{9.5}$$

This error may be bounded by $\overline{\varepsilon}_{\text{f}} = 2^{\ell_{\text{out}}-1}$, as round to nearest is easy to achieve here.

Remark that we feed back the intermediate result $\widetilde{y}(k)$ (on the extended format), not the output result $\widetilde{y}_{\text{out}}(k)$. This prevents an amplification of $\varepsilon_{\text{f}}(k)$ by the feedback loop that could compromise the goal of faithful rounding.

Rounding and quantization errors in the sum of products Our approach proposed in Chapter 7, is based on the estimation of the error amplification through the feedback loop. For this, we have fed the bounds on the errors due to the computations on each filtering iteration into a special error filter. These error bounds depend on the particular way we do the computations.

In our case, denote by $\varepsilon_{\text{r}}(k)$ the errors due to computation of SOPCs as

$$\varepsilon_{\text{r}}(k) = \widetilde{y}(k) - \left(\sum_{i=0}^{n_b} b_i u(k-i) - \sum_{i=1}^{n_a} a_i \widetilde{y}(k-i) \right). \tag{9.6}$$

This equation should be read as follows: $\varepsilon_{\text{r}}(k)$ measures how much a result $\widetilde{y}(k)$ computed by the SOPC architecture diverges from that computed by an ideal SOPC (that would use the

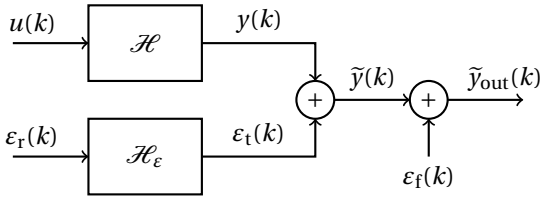


Figure 9.4: A signal view of the error propagation with respect to the ideal filter.

infinitely accurate coefficients a_i and b_i, and be free of rounding errors), this ideal SOPC being applied on the same inputs $u(k-i)$ and $\widetilde{y}(k-i)$ as the architecture.

These rounding errors $\varepsilon_{\text{r}}(k)$ can be made as small as required if we increase the internal precision of the SOPCs denoted ℓ_{ext}. In Section 9.3 we give an idea on how to build an SOPC architecture that achieves a given accuracy goal at minimum cost.

9.2.1 Complete error model

So, let us now give more details on the complete error model. As in the Chapter 7, we model the actually implemented filter as a sum of the exact filter and special error-filter (see Figure 9.4).

Applying the WCPG theorem upon the error-filter $\mathscr{H}_\varepsilon$ we obtain a bound $\overline{\varepsilon}_{\text{t}}$ on its output

$$\overline{\varepsilon}_{\text{t}} = \langle\!\langle \mathscr{H}_\varepsilon \rangle\!\rangle \, \overline{\varepsilon}_{\text{r}}, \tag{9.7}$$

where the WCPG $\langle\!\langle \mathscr{H}_\varepsilon \rangle\!\rangle$ is a scalar since we consider the error $\varepsilon_t(k)$ is scalar and the error-filter is SISO. Remark that we can reduce the $\overline{\varepsilon}_{\text{t}}$ by increasing the internal precision ℓ_{ext}.
Then, the overall error of the implementation is

$$\overline{\varepsilon}_{\text{out}} \ = \overline{\varepsilon}_{\text{f}} + \langle\!\langle \mathscr{H}_\varepsilon \rangle\!\rangle \, \overline{\varepsilon}_{\text{r}} \tag{9.8}$$

The objective of faithful rounding translates to the accuracy constraint $\overline{\varepsilon}_{\text{out}} < 2^{\ell_{\text{out}}}$. Taking into account the final rounding (which implies the error $\overline{\varepsilon}_{\text{f}} = 2^{\ell_{\text{out}}-1}$), we obtain the constraint on the internal precision of SOPCs that is required to satisfy the faithful rounding of the result:

$$2^{\ell_{\text{ext}}} < \frac{2^{\ell_{\text{out}}-1}}{\langle\!\langle \mathscr{H}_\varepsilon \rangle\!\rangle} \quad . \tag{9.9}$$

The least integer value of ℓ_{ext} is

$$\ell_{\text{ext}} = \ell_{\text{out}} - \lceil \log_2 \langle\!\langle \mathscr{H}_\varepsilon \rangle\!\rangle \rceil - 1. \tag{9.10}$$

The reader might have remarked that the MSB of the internal format is the same as that of the result (m_{out}). Some overflows may occur in the internal computation but since the computation is performed modulo $2^{m_{\text{out}}}$, the final result will be correct.

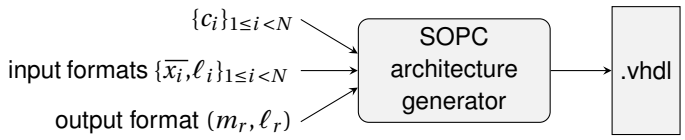


Figure 9.5: Interface to a sum-of-product-by-constant generator.

9.3 Sum of products computing just right

Our implementation is based on an arithmetic unit for the computation of SOPCs provided by FloPoCo project. Our error model is based on the assumption that we can obtain an architecture computing the SOPC (9.1) for a set of real constants c_i, and a set of fixed-point inputs v_i such that the computed result respects a priori given error bound.

Previously, our co-authors [165] proposed a tool for code generation of the faithfully rounded SOPC architectures. However, they had an assumption that all the v_i shared the same format. In the context of an LTI filter, this is no longer true: on Figure 9.3, we have a single SOPC where the c_i may be a_i or b_i, and the v_i may be either some delayed u_i, or some delayed y_i. The format of the y_i, as determined by previous section, is in general different from that of the u_i.

Therefore, we propose a more generic interface to the SOPC generator, where the format of each input may be specified independently. This interface is shown on Figure 9.5. Specifically, the input LSBs are provided as ℓ_i. Instead requiring the input MSBs, the interface uses the maximum absolute values $\overline{x_i}$ of each v_i. This is an implementation choice adopted from the initial SOPC interface proposed in [165]. Here again, the weight ℓ_r of the least significant bit of the SOPC output also specifies the accuracy of this SOPC: the result is guaranteed to be faithfully rounded to 2^{ℓ_r} bits.

Thus, to use the above interface in the context of filter implementation, we must require the SOPCs to give the result in the format $m_r = m_{\text{out}}$ and $\ell_r = \ell_{\text{ext}}$.

The aspects of hardware implementation of the SOPC unit are out of scope of this thesis. We refer the reader to [16] for the detailed description of the approach. However, we will try to give a general idea on the approach below.

9.3.1 General idea

The task is to compute the SOPC $r = \sum c_i v_i$ for a set of real constants c_i and a set of FxP inputs v_i such that such that the output in format (m_r, ℓ_r) is faithfully rounded. Performing all the internal computations to the output precision ℓ_r would in general not be accurate enough to achieve faithful rounding, due to the accumulation of rounding errors. The solution is, as previously, to use a slightly extended precision $\ell_r - g$ for the internal computation: g is a number

of “guard” bits. As this extended precision will require more hardware, we need to compute the extended precision that will minimize this hardware overhead.

The overall error of the architecture depends on the errors of each multiplication and on the errors of accumulation. We repartition the error budget 2^{ℓ_r} and deduce the maximum error of these multiplications such that the result is faithfully rounded to the LSB ℓ_r. This part is quite independent on the target technology: it could apply to ASIC synthesis as well as FPGA.

It can be shown (see the original report [16] for details) that a significant cost gain can be achieved by using look-up tables (LUTs) [158] for multiplication on FPGAs. The idea is to tabulate possible values for v_i and thus build a faithfully rounded multiplier. The pre-computation of table values must be performed with large enough accuracy (using multiple-precision software) to ensure the correct rounding of each entry.

Our implementation first invokes, for each constant, an algorithm that returns the maximum error that will be entailed by a multiplier by this constant. This error is expressed in units in the last place (ulp), whatever the value of g will be. The implementation sums these errors, then uses this sum to compute the number of guard bits g that will enable faithful rounding. Once this g has been determined it may proceed with the actual construction of the multipliers.

It may appear to be rather costly to use a large mount of LUTs to compute the SOPCs. However, using the associativity of fixed-point addition, the final summation can be implemented very efficiently using compression techniques developed for multipliers [166] and recently applied to sums of products [167, 168]. In this work, we use the bit-heap framework introduced in [162].

9.4 Implementation results

The approach described in this Chapter was implemented as the `FixIIR` operator of FloPoCo. `FixIIR` offers the interface shown on Figure 9.1, and inputs the c_i as arbitrary-precision numbers. We performed synthesis for the Xilinx Virtex-6 (6vhx380tff1923-3) using ISE 14.7.

Consider again our key filter example of a 8^{th} order lowpass filter with the corresponding transfer function from Chapter 1 Section 1.4.3. We propose to compare implementations that guarantee different accuracy, for example for 8, 12 and 16 bits. We suppose that inputs are in every implementation are as accurate as the required output.

The results of synthesis are displayed in the Table 9.1, where the smaller area or the number of LUTs the better and the higher speed the better. In this example we had that the number of guard bits for the internal computations is $g = 14$, therefore we ask for SOPC faithful to 2^{-22}, 2^{-26} and 2^{-30} for the case of 8, 12 and 16 bit implementations respectively.

This example shows that our tool is ready-to-use and that no special knowledge of filter error analysis is required from the user.

	Area		Speed
	Registers	LUTs	MHz
8 bits	272	2584	168.7
12 bits	336	3352	169.6
16 bits	400	4432	151.9

Table 9.1: Synthesis results for the key filter example.

9.5 Conclusion

In this Chapter we claim that sum-of-product architectures for LTI filters should be faithfully rounded and no more. We demonstrate that it gives a much clearer view on the trade-off between accuracy and performance, freeing the designer from several difficult choices. We provided an actual open-source "push-button" tool that offers the highest-level interface.

Designers of non-recursive LTI filters may compare their implementations using repositories of benchmarks[2]. Unfortunately, no benchmarks for IIR filters are available. Moreover, very few of the publications mention reports on the accuracy results. Thus, it is extremely difficult to compare our designs with the existing ones. We plan to provide a repository of benchmarks for IIR filters to enable such comparison.

We have only considered here the implementation of a filter once its coefficients are given. Approximation algorithms, such as Parks-McClellan [169] that compute these coefficients essentially answer the question "what is the best filter with *real* coefficients that matches this specification". It is legitimate to wonder if asking the question: "what is the best filter with low-precision coefficients" could not lead to a better result. We believe that in this case our algorithm for the verification of digital filters against frequency specifications can come at hand not only as an a posteriori verification tool but as an indicator of the direction for the rounding.

Finally, the reader may rightfully remark that in all previous chapters we relied on the SIF and extended our algorithms to any filter while in this Chapter our approach works only for the Direct Form I. In fact, SIF can be seen just as a sum of products: each element of the temporary, state and output vector is computed with a SOPC. Thus, to adapt the above approach for SIF, we just need to provide an error analysis determining the least accuracy of each sum of product in the SIF computation that guarantees that the filter's output is faithfully rounded to the output format. This work is already in progress, again in collaboration with the authors of the FloPoCo.

[2] `http://www.firsuite.net/`

CONCLUSION AND PERSPECTIVES

In this thesis we aimed at improving and extending the basic bricks for an automatic code generator for digital Linear Time-Invariant filters. We were particularly interested in bringing rigor and reliability into the practices of Fixed-Point implementation of recursive filters. While this methodology for rigorous finite-precision implementation was central to our work, we also improved other functionalities of the code generator: we extended the possibilities of the unifying framework, added new applications and provided reliable hardware code generation. Figure Z illustrates the new scheme of the tool. Sometimes a small detail would become an indispensable part and often one question would hide ten others giving us smooth transitions between subjects but at the same time leaving numerous “open doors”. In the following we present the contributions and perspectives of this thesis.

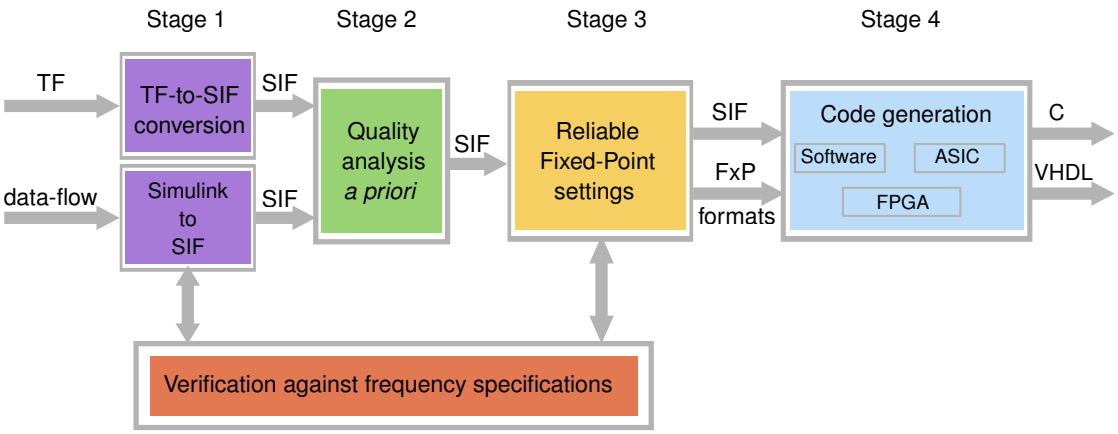


Figure Z: The workflow of the automatic code generator.

Doing more – extending and improving SIF

The automatic code generator we worked with uses its own internal format, called Specialized Implicit Form (SIF), to represent any linear filter. SIF encapsulates in an analytical form the description of different computational algorithms for filters. A typical use-case of the code generator starts with conversion of a filter's transfer function into a SIF model describing some particular structure. We started by improving and extending the possibilities of the SIF model.

First we extended the dictionary of available SIF models by adding a new filter structure, Lattice Wave digital filters. Given a filter's transfer function we can now compute the coefficients of the Lattice Wave structure and model the order of computations with SIF. This rather tedious and technical task led to the deeper understanding of the SIF [7].

Then we proposed a completely new possibility for the framework: a conversion of data-flows which describe LTI filters into the SIF. Based on the filter's graph description we determine the feedback loop and order of computations. We support data-flows in Matlab Simulink format which is one the most widely used formats in both academia and industry. With this new conversion algorithm we can analyze and implement an already existing filter design with our generator [12].

Finally, we proposed an algorithm for reliable computation of a transfer function for any filter [9]. We provided an algorithm that computes a multiple precision approximation of a transfer function along with a rigorous bound on the approximation error in terms of a certain norm (we can easily go beyond double precision). To achieve this, we use reliable evaluation of the ℓ_1-norm of a filter's impulse response, and a multiple precision eigendecomposition[3]. This contribution is important not only for our automatic code generator but also as a standalone tool: it provides an easy way for anyone working with filter algorithms to reliably determine the corresponding transfer functions.

Doing things rigorously – reliable determination of the filter's dynamic range

To provide a reliable implementation of any algorithm in the Fixed-Point arithmetic the first step is to determine the dynamic range of all variables involved in the exact filter algorithm.

With our work we provided an algorithm [11] to compute with arbitrary accuracy the upper bound on the output of a filter. It is based on a well-known object,the Worst-Case Peak Gain measure which, however, could not be *reliably* computed before but only approximated without any bound on the approximation error. We provided a new technique that permits a reliable

[3] In the sense that all operations are be performed with multiple precision. However, no information on the accuracy of the computed eigenvalues is available.

evaluation of the WCPG. This part of the thesis required a rigorous error analysis of the *floating-point* evaluation of the WCPG.

The WCPG measure is based on the evaluation of an infinite series. We proposed a formula for the determination of truncation order with an a priori bound on the truncation error. For this, we use a combination of multiple precision interval arithmetic and of the Theory of Verified Inclusions. To evaluate the truncated series we use multiple precision computations and adapt the precision just enough to meet an a priori given requirement on the accuracy. A non-trivial part was getting around the absence of a *multiple precision eigensolver with reliable error bounds* which we needed at some point in our algorithm. All error bounds are based on proofs and the algorithm was extensively tested.

Using our result, filter designers now have a proven guarantee on the dynamic range of all variables involved in exact filter evaluation. We also provide an adaptation of our algorithm for the case when the filter coefficients are represented as intervals. This adaptation is useful when a filter's parameters are results of some measurements or computations.

Considering everything – taking into account the accumulation of computational errors

During the filter implementation, the designer must make several choices that influence the cost and the accuracy of the implemented system. Usually, you get the one at the expense of the other and, therefore, a Pareto-optimal solution that respects a trade-off must be found. The basic problem in the determination of the trade-off is: given wordlength constraints for each variable to determine the best Fixed-Point formats that ensure that system is reliable (i.e. no overflow occurs) and to obtain a tight bound on the error. In other words, we look for the least Most Significant Bit (MSB) position that ensures that no overflow occurs. We provided a rigorous methodology [13] for the solution of this problem. The difficulty consisted in the fact that the dynamic ranges determined with the WCPG measure correspond to the exact filter and not the implemented one. In some cases rounding errors due to the finite-precision computations may propagate up to the MSB. To fully take this fact into account we had to rigorously capture the non-linear propagation of the computational and quantization errors through recursive filters. We achieved this by exploiting certain properties of the linear filters and using our reliable WCPG measure. We prove that we either determine the MSB positions exactly or overestimate them at most by one, hence we prove that no overflow occurs. The overestimation may come from the fact that the value of the MSB is determined using approximated quantities (WCPG measures). In these rare cases we might need to increase the accuracy of the approximations. However,

this includes, amongst other ideas, an instance of the Table Maker's Dilemma [35]. Finally, using the WCPG theorem we determine a tight bound on the output error of the filter. We never use any simulations, our algorithms are exclusively based on mathematical proofs.

Overall, our algorithm has reasonable timings that permit to use it extensively during the exploration of a large design space in search of the trade-off between implementation constraints. We can also look at the problem from another angle and instead of fixing the wordlengths and determining the errors this choice yields to, determine the least wordlengths necessary to guarantee a certain fixed error bound.

With this approach we enabled the kernel functionality of the automatic code generator. However, our approach of reliable computation of the bound on the implementation error can be used as an independent methodology.

Being sure – verification of an implemented filter against frequency specifications

We presented reliable algorithms, based on multiple precision floating-point and interval arithmetic with proven error bounds, for the determination of the Fixed-Point parameters of the implementations. However, guaranteed error bounds are not worth anything if the implemented filter does not demonstrate desired frequency domain behavior. We proposed an algorithm [9] for rigorous verification of filter realizations against frequency constraints. Our algorithm must be rigorous in the sense that it never returns a false positive answer.

First, we proposed a basic brick verification algorithm for the transfer functions. This algorithm relies on the combination of different techniques and our implementation makes use of interval and rational arithmetic in the Sollya tool. Our algorithm guarantees that no false positives occur. In the case of a negative answer, a list of "problematic" frequencies is returned.

Further, using SIF we extend this verification upon any filter realization. We can compute an approximation of the transfer function of any realization through the SIF and take the approximation error into account during the verification. We proposed a heuristic algorithm that performs such a verification. Again, we guarantee that no false positive answers occur while false negatives are potentially possible, even though we never observed any.

Our verification algorithm opens numerous possibilities apart from a rigorous verification of an existing implementation. It can be used extensively during the design of a filter: via the list of "problematic" frequencies we obtain an indication on how we can redesign the filter such that its implementation passes the verification. Finally, we can compare the quality of different filter design tools.

Doing things in practice – applications for the hardware implementation

After providing basic bricks for the generation of Fixed-Point algorithms we did a few steps towards implementation of recursive filters on Field Programmable Gate Arrays (FPGAs). The designer of a hardware implementation must make numerous choices concerning the implementation: quantize the structure's coefficients, choose intermediate and input/output formats, etc. We show that we can automatize a large part of this process. For this, we exploited the possibilities of the hardware code generator called FloPoCo [15] which provides tools for the generation of arithmetic cores "computed just right". We used their fixed-point core for the computation of Sums of Products by Constants (SOPC) as a basic brick for the implementation of filters in the Direct Form I (DFI) structure. We justify our choice of structure by the fact that DFI is usually represented as SOPC and, therefore, is a good candidate for the first steps towards a more general approach on the implementation on FPGAs.

As a result, we provided an open-source "push-button" tool[4] that generates a hardware architecture the output of which is guaranteed to be faithfully rounded. Using the WCPG theorem we deduced the least number of guard bits with which the SOPC in the DFI must be computed. In collaboration with the authors of FloPoCo we adapted the SOPC generator to our needs. More precisely, we built an SOPC generator that takes *real* coefficients as inputs and optimally truncates them. Optimality is in the sense that the SOPC determines internally the least number of bits for the coefficients such that the output accuracy is guaranteed. This is a significant improvement that frees the filter designer from the difficult choice of quantization scheme. On top of that, we provided code generation specific for Field Programmable Gate Arrays (FPGAs) based only on Look-Up Tables (LUTs). These logic-only architectures are suited even to low-end FPGAs.

Even though dedicated to a particular structure, DFI, this work brings us one step closer to a general approach for the reliable implementation of any LTI filter on FPGAs. In fact, any filter described with SIF representation is just a set of SOPCs. Thus, we can proceed analogously to the case of the Direct Form I structure: determine the least required number of guard bits for the computation of SOPCs such that the output error bound is satisfied. We have a strong belief that the resulting tool will interest industrial partners from the domain where reliability of the systems is of great importance (e.g. aerospace industry, automotive industry, etc.).

[4] Implemented as `FixIIR` module in the FloPoCo tool.

What is left to do – over the horizon

With this thesis we paved the road towards reliable implementation of digital filters and now we are on the crossroad of perspectives.

Short term In the short term we will need to improve the obtained results. First we propose to consider the spectrum of the input signal in our methodology. Indeed, often in real-life applications a filter designer has more precise information on the frequency behavior of the input signal than just an interval. I believe that it is possible to adapt the WCPG theorem to satisfy this setting [14].

Another problem that might require our immediate attention is the potential overestimation of the MSB positions by one bit. This issue might, however, require some deeper investigation of the set of reachable states of an *implemented* LTI system and risks to become rather a mid-term than a short-term perspective. While probably being less important from practical point of view (in the end, 1 bit more or less might not matter), the off-by-one problem is interesting as a phenomenon itself: it is an instance of the Table Maker's Dilemma in the design of digital filters.

Finally, we should generalize our approach for the generation of reliable hardware architectures. Once the generalization is provided, it will be interesting to provide an extensive comparison of hardware implementations of different structures for realization of different types of filters.

Mid term In the mid-term I propose to first address the problem of a multiple precision eigendecomposition. We have seen that the majority of our algorithms rely on the computation of eigenvalues. Using the eigenvalues we determine the location of filter's poles and, most importantly, check the stability. To guarantee the stability of a filter all these poles must be in the unit circle. Moreover, a rigorous verification of filter's stability is often a key step during the exploration of a large design space and it must be performed quickly. In order to ensure a reliable and fast verification, I propose to first provide a fast multiprecision algorithm for the computation of the spectral radius of a non-symmetric matrix. Furthermore, a varying-precision eigendecomposition with a rigorous output error bound will improve our algorithms, help with a rigorous evaluation of the ℓ_2-norm of a filter and, consequently, sensitivity measures [34] of LTI systems.

In our work we have come closer to the explanation of the link between errors that occur in the time domain with the errors in the frequency domain. However, some effort is still required to clearly explain the influence of rounding errors upon the frequency response of the filter. This

work will have impact upon both theoretical understanding of digital filters and practical solutions for reliable finite-precision (not only Fixed-Point but Floating-Point as well) implementations.

Another important remark concerning the behavior of filter's frequency response is that sometimes a violation of the specifications may be tolerated. For example, if the violation has a small energy. To take this into account, we would need to revise our verification algorithm from Chapter 8. However, finding preliminarily some real-life applications for such an improvement is crucial.

It also remains plug all the steps of filter implementation into optimization routines that permit to choose the Pareto-optimal structure according to the user's needs and to prove the solution. We can start by first formalizing the possible user criteria and defining relationships between them. For example using the WCPG we can easily obtain a relationship between the wordlengths and the output error [6]. Other criteria, like the relationship between area and power consumption, can be more application specific. I am convinced that using these relationships we can formulate a generic optimization problem that will encompass the majority of the implementation process. Once the optimization procedure of a filter in SIF representation is done, it will be interesting to compare different structures between them. This study may confirm existing empiric observations as well as show reveal new trends and properties of different families of structures.

Long term In the long-term I see several major research axes.

The first one concerns the quantization of structure's coefficients. An interesting point to investigate may be the degree of liberty with which we can "move" the coefficients while maintaining the desired behavior of the frequency response. The goal is to obtain coefficients with the least number of ones and hence faster multiplications by this number. Take for example a binary coefficient $1.494140625_{10} = 1.011111101_2$ multiplying by this coefficient is more complicated than by $1.5_{10} = 1.1_2$. Of course, this is just a general idea and in hardware multipliers [158, 159] the techniques are more complicated. The question is whether we can predict which coefficients can be subject to such optimization of the binary expression and which cannot. I believe we can use the techniques of transfer function coefficient sensitivity as the first hint and get inspiration in the works of Muller and Torres [170]. However, the idea is to investigate such optimization for an arbitrary structure and not only for a transfer function.

In the same spirit, a second prominent subject would be the study of new techniques for the design of rational transfer functions whose goal is to find the best *quantized* to low-precision coefficients. We could join forces with the authors of new robust techniques [171] for the design of polynomial transfer functions. I believe that our transfer function verification algorithm can be of great assistance in this study.

The third direction is towards the unification of code generation for filters and mathematical functions. Numerous common points have been invoked by the French ANR project “Metalibm”. On the one hand, a filter is not a mathematical function, it processes signals and therefore depends on history. On the other hand, in both filter and function implementation the final code or circuit consist of the same primitives (addition, multiplication and pre-computed constant values). In case of non-recursive filters both filters and functions often benefit from the Remez approximation algorithm [24]. It is not surprising that there are similarities between the evaluation schemes: Direct Form structure can be interpreted as Horner scheme, polyphase filters as Estrin’s scheme, etc. However, the study of analogies is not fairly possible without a rational Remez algorithm available for the design of IIR filters.

Massive implementations of filtering algorithms for big data and artificial neural networks are often performed on general purpose CPUs or GPUs. These targets possess a Floating-Point unit but the robustness requirements still dictate short wordlength formats. Therefore, the problem of the error-analysis of short Floating-Point implementations rises. I believe that this study might greatly benefit from the methodology proposed in this thesis: Worst-Case Peak Gain measure, transfer function verification algorithm, the error model, etc. What is non-trivial but promising is to address artificial neural network algorithms themselves. In fact, for some neural networks, the computations on each layer may be seen as (complicated) linear filters. Thus, the fourth main research axis concerns expanding the area of application to machine learning algorithms.

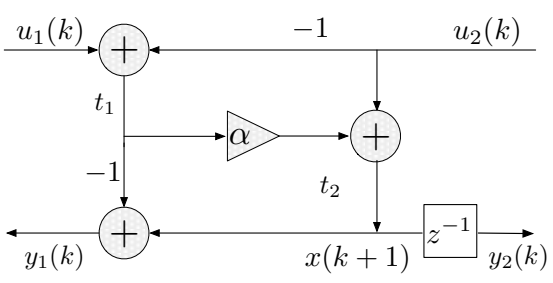
So many possibilities open up when signal processing
and computer arithmetic meet.

CHAPTER A

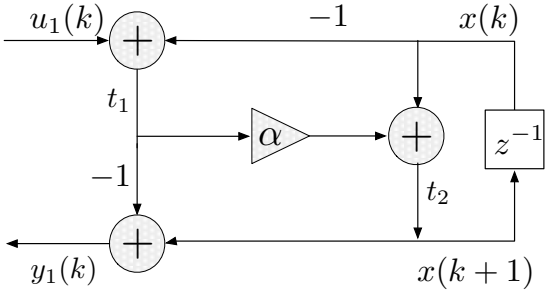
APPENDIX

1 Lattice Wave Digital Filter basic brick data-flows

In Part II Chapter 4 we presented a conversion algorithm for the Lattice Wave Digital filters to the SIF. Figures A.1-A.4 illustrate the annotated data-flow graphs that correspond to the subsystems of type A and B in Lattice Wave Digital filters.

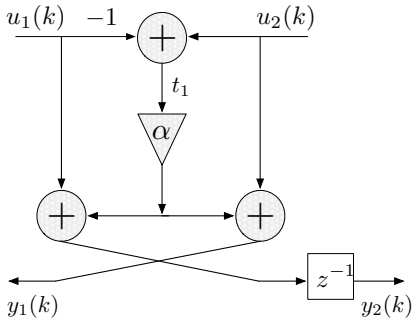


(a) Type 1-A

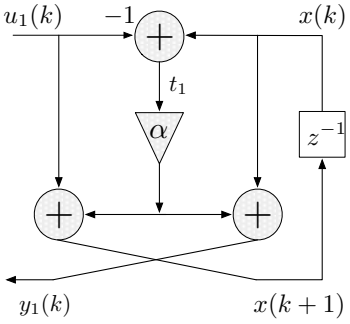


(b) Type 1-B

Figure A.1: Subsystems with adaptors of Type 1.

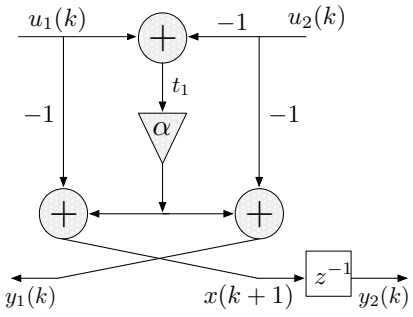


(a) Type 2-A

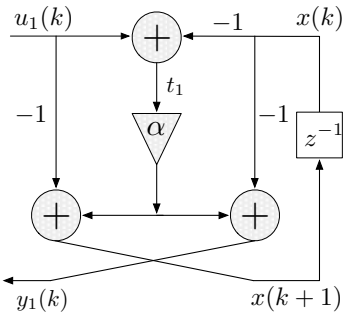


(b) Type 2-B

Figure A.2: Subsystems with adaptors of Type 2.

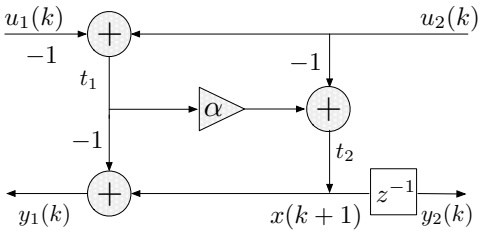


(a) Type 3-A

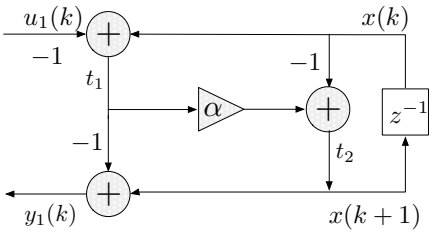


(b) Type 3-B

Figure A.3: Subsystems with adaptors of Type 3.



(a) Type 4-A



(b) Type 4-B

Figure A.4: Subsystems with adaptors of Type 4.

2 Error analysis behind the Multiple Precision basic bricks

In Chapter 6 our analysis is based on multiple precision basic brick algorithms. One of the core algorithms is the sum of the elements of a real vector with an a priori given bound on the absolute error. Here we give the reader an idea of the error analysis behind this basic brick. Let $\boldsymbol{v}$ contain n multiple precision floating-point numbers[1]. Denote by

$$s = \sum_{i=1}^{n} \boldsymbol{v}_i$$

the sum that needs to be computed. Our goal is to compute and approximation accurate to k bits, i.e. an $\widehat{s}$ such that

$$\widehat{s} = s + \delta,$$

where the overall absolute error $|\delta| \leq 2^k$ for a k given as the input to our algorithm. If necessary, we will adapt the precision p_{sum} of the result variable to satisfy the absolute error bound.

[1] If $\boldsymbol{v}$ contains Infinities or NaNs we return an error.

Denote by $e_{\max}$ the maximum exponent of non-zero real entries in the vector $\boldsymbol{v}_i$ such that $|\boldsymbol{v}_i| \leq 2^{e_{\max}}$. Hence, the overall sum is bounded in magnitude:

$$|s| \leq n \cdot 2^{e_{\max}} \leq 2^{e_{\max}+\left\lceil \log_2 n \right\rceil},$$

since $n \leq 2^{\left\lceil \log_2 n \right\rceil}$.

Obviously, if the sum itself is strictly smaller than the error bound, i.e. $2^{e_{\max}+\left\lceil \log_2 n \right\rceil} < 2^k$, we do not need to actually compute the sum and we can simply return zero.

Let

$$p = e_{\max} + \left\lceil \log_2 n \right\rceil - k + \left\lceil \log_2 n \right\rceil + 5.$$

In our case, an addition performed on a p bit precision variable induces an absolute error smaller than

$$2^{-p}\,|s| \leq 2^{-p} \cdot 2^{e_{\max}+\left\lceil \log_2 n \right\rceil} = 2^{k-\left\lceil \log_2 n \right\rceil-5}.$$

When performing $n < 2^{\left\lceil \log_2 n \right\rceil}$ additions on at least p bit precision variable $\widehat{s}$, we get

$$|s - \widehat{s}| < n \cdot 2^{k-\left\lceil \log_2 n \right\rceil-5} \leq 2^{k-5}$$

In other words, the accumulated sum $\widehat{s}$ satisfies

$$\widehat{s} = s + \delta,$$

with $|\delta| \leq 2^{k-5}$.

If $p > p_{\text{sum}}$, we need to round $\widehat{s}$ to some precision that is closest to the original precision p_{sum}, while maintaining the overall error on the sum bounded by 2^k.
If the exponent e of the accumulated sum $\widehat{s}$ is such that $e \leq k-1$, then

$$|s| = \widehat{s} + \delta \leq |\widehat{s}| + |\delta| \leq 2^{k-1} + 2^{k-5} \leq 2^k$$

and we can simply return zero and reset its precision to p_{sum}.
Otherwise, when rounding $\widehat{s}$ to p_{sum} bits we obtain

$$\widehat{s}\,' = \widehat{s} + \delta',$$

with $\left|\delta'\right| \leq 2^{-p_{\text{sum}}}\,|\widehat{s}| \leq 2^{e-p_{\text{sum}}}$.
If $e - p_{\text{sum}} \leq k-1$, we can just perform the rounding as then we have

$$\widehat{s}\,' = \widehat{s} + \delta' = s + \delta + \delta'$$

and

$$\left|s-\widehat{s}\,'\right| \le |\delta| + \left|\delta'\right| \le 2^{k-5} + 2^{e-p_{\text{sum}}} \le 2^{k-5} + 2^{k-1} \le 2^{k}.$$

Otherwise, we round the sum to $e-k+1$ bits, inducing an additional error δ' bounded in magnitude by 2^{k-1} but still maintaining the error bound 2^k. This is the only case when the final precision of the sum variable is not equal to the original precision p_{sum} of the sum variable.

3 Coefficients for the examples

In this section we give the coefficients of the filter algorithms that we used in some of our examples.

3.1 Example from Chapter 6

In Part III Chapter 6 Example 6.1 (p. 84.) we used the following SISO state-space realization:

$$\boldsymbol{A} = \begin{pmatrix} \frac{5349797894891737}{2^{58}} & \frac{5166143083671405}{2^{53}} & \frac{-2831854438491313}{2^{54}} & \frac{-5871577021383539}{2^{57}} & \frac{-8085167575254235}{2^{59}} \\ \frac{5166143083671405}{2^{53}} & \frac{-2313123528371301}{2^{53}} & \frac{-3726321566242771}{2^{55}} & \frac{-2862105117188361}{2^{53}} & \frac{-5198006051035051}{2^{54}} \\ \frac{-2831854438491313}{2^{54}} & \frac{-3726321566242771}{2^{55}} & \frac{-6953517292263399}{2^{53}} & \frac{1536770956967001}{2^{52}} & \frac{-6533578784721267}{2^{57}} \\ \frac{-5871577021383539}{2^{57}} & \frac{-2862105117188361}{2^{53}} & \frac{1536770956967001}{2^{52}} & \frac{-4578574112815079}{2^{58}} & \frac{-6108340260993661}{2^{54}} \\ \frac{-8085167575254235}{2^{59}} & \frac{-5198006051035051}{2^{54}} & \frac{-6533578784721267}{2^{57}} & \frac{-6108340260993661}{2^{54}} & \frac{-7406762621209713}{2^{58}} \end{pmatrix},$$

$$\boldsymbol{b} = \begin{pmatrix} 0 \\ \frac{-5431455542039353}{2^{53}} \\ \frac{-4330832538465309}{2^{51}} \\ \frac{-2702522316192301}{2^{51}} \\ \frac{482362316509163}{2^{49}} \end{pmatrix}, \qquad \boldsymbol{c} = \begin{pmatrix} \frac{5441181515794623}{2^{52}} & 0 & 0 & 0 & \frac{8170739390909991}{2^{53}} \end{pmatrix},$$

$$d = \frac{-6210481900542423}{2^{52}}.$$

3.2 Example from Chapter 7

In Part III Chapter 7 Example 2 (p. 117) we used the following SISO state-space realization:

$$\boldsymbol{A} = \begin{pmatrix} \frac{6784742786136467}{2^{55}} & \frac{-4418765114923729}{2^{52}} & \frac{-664385987312541}{2^{57}} & \frac{-6424963082774909}{2^{60}} & \frac{23379087041007}{2^{54}} & \frac{7224673930044821}{2^{62}} \\ \frac{8837532051801285}{2^{53}} & \frac{6702326945697769}{2^{55}} & \frac{200758871190939}{2^{55}} & \frac{7765776886168563}{2^{60}} & \frac{-7064506506383}{2^{52}} & \frac{-1091547242537563}{2^{59}} \\ \frac{332163678765481}{2^{56}} & \frac{803109201555043}{2^{57}} & \frac{3430928543256835}{2^{54}} & \frac{-8814226732612787}{2^{53}} & \frac{-340600630723981}{2^{56}} & \frac{-51393263754677}{2^{53}} \\ \frac{-100378037454999}{2^{54}} & \frac{-3883122962210243}{2^{59}} & \frac{4407113891006265}{2^{52}} & \frac{6589648288527319}{2^{55}} & \frac{51463819022587}{2^{53}} & \frac{3975875009524179}{2^{59}} \\ \frac{23338099057403}{2^{54}} & \frac{1805668067704021}{2^{60}} & \frac{1360133999454637}{2^{58}} & \frac{1644151688384379}{2^{58}} & \frac{3482771051190523}{2^{54}} & \frac{-8782983518574175}{2^{53}} \\ \frac{-28250206934819}{2^{54}} & \frac{-4371435432060817}{2^{61}} & \frac{-205801181849963}{2^{55}} & \frac{-3980412056409463}{2^{59}} & \frac{8782803780831881}{2^{53}} & \frac{402384040765689}{2^{51}} \end{pmatrix},$$

$$\boldsymbol{b} = \begin{pmatrix} \frac{2805227146785663}{2^{56}} \\ \frac{-6781289732910279}{2^{57}} \\ \frac{7151095668232217}{2^{57}} \\ \frac{-8644087168799361}{2^{57}} \\ \frac{138530205667109}{2^{52}} \\ \frac{-670749912793553}{2^{54}} \end{pmatrix}, \quad \boldsymbol{c}^\top = \begin{pmatrix} \frac{-5609854620080605}{2^{57}} \\ \frac{-1695446456016151}{2^{55}} \\ \frac{7150962371133851}{2^{57}} \\ \frac{4322098720663427}{2^{56}} \\ \frac{-8880554827970779}{2^{58}} \\ \frac{-2679975639394807}{2^{56}} \end{pmatrix}, \quad d = \frac{7582290898298045}{2^{74}}.$$

3.3 Example from Chapter 8

In Part III Chapter 8 Example 2 (p. 139) we consider four realizations of a 9^{th} order IIR filter. Their coefficients have been subject to different optimizations specific to each structure.

Direct Form II transposed This realization has the same coefficients as the transfer function of the filter. Its numerator $\boldsymbol{b}$ and denumerator $\boldsymbol{a}$ are:

$$\boldsymbol{b} = \begin{pmatrix} \frac{729237663}{2^{53}} \\ \frac{410196191}{2^{49}} \\ \frac{410196177}{2^{47}} \\ \frac{957124445}{2^{47}} \\ \frac{1435686635}{2^{47}} \\ \frac{358921665}{2^{45}} \\ \frac{1914248853}{2^{48}} \\ \frac{3281569499}{2^{50}} \\ \frac{3281569473}{2^{52}} \\ \frac{2916950655}{2^{55}} \end{pmatrix} \quad \boldsymbol{a} = \begin{pmatrix} 1 \\ \frac{-7785613688429543}{2^{50}} \\ \frac{6036692931392875}{2^{48}} \\ \frac{-5505973960157689}{2^{47}} \\ \frac{1626405210837387}{2^{45}} \\ \frac{-645028755122265}{2^{44}} \\ \frac{5492680950051031}{2^{48}} \\ \frac{-7562215714147779}{2^{50}} \\ \frac{3053779727680745}{2^{51}} \\ \frac{-2203903366092791}{2^{54}} \end{pmatrix}$$

ρ-operator Direct Form II transposed

This realization has the same number of coefficients as the Direct Form II transposed but uses a modified delay operator. The corresponding Specialized Impli Form matrix $\boldsymbol{Z}$ is:

$$\boldsymbol{Z} = \begin{pmatrix} -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{5833901303}{2^{56}} \\ \frac{-1173742736577039}{2^{49}} & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{52505111727}{2^{55}} \\ \frac{-598564716117403}{2^{48}} & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & \frac{52505111727}{2^{52}} \\ \frac{-194752412061513}{2^{47}} & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & \frac{122511927363}{2^{51}} \\ \frac{-44340287896553}{2^{46}} & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & \frac{367535782089}{2^{51}} \\ \frac{-29194831792401}{2^{47}} & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & \frac{367535782089}{2^{50}} \\ \frac{-13858087341545}{2^{48}} & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & \frac{122511927363}{2^{48}} \\ \frac{-4562787932375}{2^{49}} & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & \frac{52505111727}{2^{47}} \\ \frac{-3775617671781}{2^{52}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \frac{52505111727}{2^{48}} \\ \frac{-93342420817}{2^{51}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{5833901303}{2^{47}} \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Balanced state-space

This structure has its coefficients that are grouped into the matrix $\boldsymbol{A}$, vectors $\boldsymbol{b}$ and $\boldsymbol{c}$, and scalar d.

$$\boldsymbol{A} = \begin{pmatrix} \frac{4433399923822815}{2^{52}} & \frac{-697104920405359}{2^{52}} & \frac{-14914874060567}{2^{56}} & \frac{1834705105773231}{2^{56}} & \frac{-227990264073995}{2^{55}} & \frac{264510032627687}{2^{56}} & \frac{71919139252021}{2^{56}} & \frac{15539762312143}{2^{56}} & \frac{3987618469973}{2^{57}} \\ \frac{2788419681621411}{2^{54}} & \frac{8511655240987669}{2^{53}} & \frac{-7241731896691307}{2^{55}} & \frac{923032429638883}{2^{55}} & \frac{-1185143818093905}{2^{55}} & \frac{91680166787919}{2^{53}} & \frac{29955451375187}{2^{53}} & \frac{24793696845359}{2^{55}} & \frac{1602266531167}{2^{54}} \\ \frac{-14914874060201}{2^{56}} & \frac{7241731896691287}{2^{55}} & \frac{4014739506825505}{2^{52}} & \frac{3956246339911157}{2^{54}} & \frac{-847871350139453}{2^{54}} & \frac{975529378369809}{2^{55}} & \frac{132472831176299}{2^{54}} & \frac{57256913063569}{2^{55}} & \frac{1836516170261}{2^{53}} \\ \frac{-1834705105773195}{2^{56}} & \frac{923032429638873}{2^{55}} & \frac{-1978123169555585}{2^{53}} & \frac{3730471156627787}{2^{52}} & \frac{4008581949275589}{2^{54}} & \frac{-1822778074078241}{2^{55}} & \frac{-648692245116269}{2^{55}} & \frac{-32932900203945}{2^{53}} & \frac{-34163067919255}{2^{56}} \\ \frac{-911961056295815}{2^{57}} & \frac{4740575272375661}{2^{57}} & \frac{-1695742700278867}{2^{55}} & \frac{-4008581949275597}{2^{54}} & \frac{431207735727907}{2^{49}} & \frac{976897242794113}{2^{52}} & \frac{3142717255227975}{2^{56}} & \frac{365093342795161}{2^{55}} & \frac{185168624492341}{2^{57}} \\ \frac{-1058040130510681}{2^{58}} & \frac{1466882668606681}{2^{57}} & \frac{-7804235026958069}{2^{58}} & \frac{-3645556148156227}{2^{56}} & \frac{-7815177942352969}{2^{55}} & \frac{797208406832299}{2^{50}} & \frac{-3640204017578879}{2^{54}} & \frac{-289129023623485}{2^{53}} & \frac{-1253582148190087}{2^{58}} \\ \frac{11507062280308995}{2^{60}} & \frac{-3834297776022405}{2^{60}} & \frac{8478261195284487}{2^{60}} & \frac{2594768980465645}{2^{57}} & \frac{785679313806937}{2^{54}} & \frac{910051004394703}{2^{52}} & \frac{366857616167601}{2^{49}} & \frac{-6305858738209341}{2^{55}} & \frac{-5362082746147945}{2^{58}} \\ \frac{-621590492495 43}{2^{58}} & \frac{1586796598112755}{2^{61}} & \frac{-1832221218029465}{2^{60}} & \frac{-2107705613046249}{2^{59}} & \frac{-2920746742362679}{2^{58}} & \frac{-2313032188988843}{2^{56}} & \frac{6305858738208987}{2^{55}} & \frac{671630402948993}{2^{50}} & \frac{-1166805156754687}{2^{53}} \\ \frac{1993809236293}{2^{56}} & \frac{-4101802318525 71}{2^{62}} & \frac{58768517426727}{2^{58}} & \frac{136652271674025}{2^{58}} & \frac{14813489959536 13}{2^{60}} & \frac{3133955370500 83}{2^{56}} & \frac{-1340520686534485}{2^{56}} & \frac{4667220627014373}{2^{55}} & \frac{4896276983317139}{2^{53}} \end{pmatrix}$$

$$\boldsymbol{b} = \begin{pmatrix} \frac{-3423531969504719}{2^{55}} \\ \frac{3879265898276791}{2^{54}} \\ \frac{-626127473878301}{2^{51}} \\ \frac{-8465367200971921}{2^{55}} \\ \frac{-4828329411714067}{2^{55}} \\ \frac{-1966428315469671}{2^{55}} \\ \frac{4709145555509341}{2^{58}} \\ \frac{-3976213184483417}{2^{60}} \\ \frac{512167188113989}{2^{60}} \end{pmatrix} \quad \boldsymbol{c}^{\top} = \begin{pmatrix} \frac{-1711765984752477}{2^{54}} \\ \frac{-3879265898276791}{2^{54}} \\ \frac{-2504509895513203}{2^{53}} \\ \frac{2116341800242985}{2^{53}} \\ \frac{-301770588232129}{2^{51}} \\ \frac{983214157734825}{2^{54}} \\ \frac{588643194438675}{2^{55}} \\ \frac{248513324030127}{2^{56}} \\ \frac{8194675009308959}{2^{64}} \end{pmatrix} \quad d = \begin{pmatrix} \frac{5833901303}{2^{56}} \end{pmatrix}$$

Lattice Wave Digital Filter Coefficients $\boldsymbol{\gamma}$ of the Lattice Wave Digital filter:

$$\boldsymbol{\gamma} = \begin{pmatrix} \frac{-2788140310368283}{2^{53}} & \frac{-292219541455085}{2^{52}} & \frac{-480744335378591}{2^{50}} & \frac{-73054885139031}{2^{50}} & \frac{-1044203411065085}{2^{53}} & \frac{-584439080672053}{2^{53}} & \frac{-4499984971038737}{2^{53}} & \frac{-584439081644825}{2^{53}} & \frac{-338912398222379}{2^{50}} \end{pmatrix}$$

4 Off-by-One problem

In Part III Chapter 7 Section 7.8 we present our ongoing work on the off-by-one problem. The solution is based on an optimization problem. In this Section we give details of the formal construction of this optimization problem.

As we said in Section 7.8.1, we are looking for $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\delta_x}, \boldsymbol{\delta_y}$ such that

$$\begin{pmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{D} \end{pmatrix} \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{u} \end{pmatrix} \leq \begin{pmatrix} \overline{\boldsymbol{x}} \\ \overline{\boldsymbol{y}} \end{pmatrix} + \begin{pmatrix} \boldsymbol{\delta_x} \\ \boldsymbol{\delta_y} \end{pmatrix} \tag{A.1}$$

$$\begin{pmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{D} \end{pmatrix} \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{u} \end{pmatrix} \geq \begin{pmatrix} \overline{\boldsymbol{x}} \\ \overline{\boldsymbol{y}} \end{pmatrix} + \begin{pmatrix} \boldsymbol{\delta_x} \\ \boldsymbol{\delta_y} \end{pmatrix} \tag{A.2}$$

$$\begin{pmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} \end{pmatrix} \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{u} \end{pmatrix} \leq \begin{pmatrix} \overline{\boldsymbol{x}} \\ \overline{\boldsymbol{u}} \end{pmatrix} \tag{A.3}$$

$$\begin{pmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} \end{pmatrix} \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{u} \end{pmatrix} \geq \begin{pmatrix} \underline{\boldsymbol{x}} \\ \underline{\boldsymbol{u}} \end{pmatrix} \tag{A.4}$$

To formalize the optimization problem, we need to bring the above inequalities to the canonical form, i.e. bring all inequalities to the direction "≤":

Denote $\boldsymbol{x} := \underline{\boldsymbol{x}} + \boldsymbol{x}'$ and $\boldsymbol{u} := \underline{\boldsymbol{u}} + \boldsymbol{u}'$. Then,

$$\begin{pmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{D} \end{pmatrix} \begin{pmatrix} \boldsymbol{x}' \\ \boldsymbol{u}' \end{pmatrix} + \begin{pmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{D} \end{pmatrix} \begin{pmatrix} \underline{\boldsymbol{x}} \\ \underline{\boldsymbol{u}} \end{pmatrix} \leq \begin{pmatrix} \overline{\boldsymbol{x}} \\ \overline{\boldsymbol{y}} \end{pmatrix} + \begin{pmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} \end{pmatrix} \begin{pmatrix} \boldsymbol{\delta_x} \\ \boldsymbol{\delta_y} \end{pmatrix} \tag{A.5}$$

$$\begin{pmatrix} -\boldsymbol{A} & -\boldsymbol{B} \\ -\boldsymbol{C} & -\boldsymbol{D} \end{pmatrix} \begin{pmatrix} \boldsymbol{x}' \\ \boldsymbol{u}' \end{pmatrix} - \begin{pmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{D} \end{pmatrix} \begin{pmatrix} \underline{\boldsymbol{x}} \\ \underline{\boldsymbol{u}} \end{pmatrix} \leq -\begin{pmatrix} \overline{\boldsymbol{x}} \\ \overline{\boldsymbol{y}} \end{pmatrix} - \begin{pmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} \end{pmatrix} \begin{pmatrix} \boldsymbol{\delta_x} \\ \boldsymbol{\delta_y} \end{pmatrix} \tag{A.6}$$

$$\begin{pmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} \end{pmatrix} \begin{pmatrix} \boldsymbol{x}' \\ \boldsymbol{u}' \end{pmatrix} + \begin{pmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} \end{pmatrix} \begin{pmatrix} \underline{\boldsymbol{x}} \\ \underline{\boldsymbol{u}} \end{pmatrix} \leq \begin{pmatrix} \overline{\boldsymbol{x}} \\ \overline{\boldsymbol{u}} \end{pmatrix} \tag{A.7}$$

$$\begin{pmatrix} -\boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & -\boldsymbol{I} \end{pmatrix} \begin{pmatrix} \boldsymbol{x}' \\ \boldsymbol{u}' \end{pmatrix} - \begin{pmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} \end{pmatrix} \begin{pmatrix} \underline{\boldsymbol{x}} \\ \underline{\boldsymbol{u}} \end{pmatrix} \leq -\begin{pmatrix} \overline{\boldsymbol{x}} \\ \overline{\boldsymbol{u}} \end{pmatrix} \tag{A.8}$$

By gathering $\boldsymbol{x}', \boldsymbol{u}'$ and $\boldsymbol{\delta}_x, \boldsymbol{\delta}_y$ on the left sides of each inequality, we obtain

$$\begin{pmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{D} \end{pmatrix} \begin{pmatrix} \boldsymbol{x}' \\ \boldsymbol{u}' \end{pmatrix} - \begin{pmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} \end{pmatrix} \begin{pmatrix} \boldsymbol{\delta_x} \\ \boldsymbol{\delta_y} \end{pmatrix} \leq \begin{pmatrix} \overline{\boldsymbol{x}} \\ \overline{\boldsymbol{y}} \end{pmatrix} - \begin{pmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{D} \end{pmatrix} \begin{pmatrix} \underline{\boldsymbol{x}} \\ \underline{\boldsymbol{u}} \end{pmatrix} \tag{A.9}$$

$$\begin{pmatrix} -\boldsymbol{A} & -\boldsymbol{B} \\ -\boldsymbol{C} & -\boldsymbol{D} \end{pmatrix} \begin{pmatrix} \boldsymbol{x}' \\ \boldsymbol{u}' \end{pmatrix} + \begin{pmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} \end{pmatrix} \begin{pmatrix} \boldsymbol{\delta_x} \\ \boldsymbol{\delta_y} \end{pmatrix} \leq \begin{pmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{D} \end{pmatrix} \begin{pmatrix} \underline{\boldsymbol{x}} \\ \underline{\boldsymbol{u}} \end{pmatrix} - \begin{pmatrix} \overline{\boldsymbol{x}} \\ \overline{\boldsymbol{y}} \end{pmatrix} \tag{A.10}$$

$$\begin{pmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} \end{pmatrix} \begin{pmatrix} \boldsymbol{x}' \\ \boldsymbol{u}' \end{pmatrix} \leq \begin{pmatrix} \overline{\boldsymbol{x}} \\ \overline{\boldsymbol{u}} \end{pmatrix} - \begin{pmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} \end{pmatrix} \begin{pmatrix} \underline{\boldsymbol{x}} \\ \underline{\boldsymbol{u}} \end{pmatrix} \tag{A.11}$$

$$\begin{pmatrix} -\boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & -\boldsymbol{I} \end{pmatrix} \begin{pmatrix} \boldsymbol{x}' \\ \boldsymbol{u}' \end{pmatrix} \leq \begin{pmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} \end{pmatrix} \begin{pmatrix} \underline{\boldsymbol{x}} \\ \underline{\boldsymbol{u}} \end{pmatrix} - \begin{pmatrix} \overline{\boldsymbol{x}} \\ \overline{\boldsymbol{u}} \end{pmatrix} \tag{A.12}$$

By gathering vectors $\boldsymbol{x}', \boldsymbol{u}'$ and $\boldsymbol{\delta}_x, \boldsymbol{\delta}_y$ into one vector $\boldsymbol{\xi} = \begin{pmatrix} \boldsymbol{x}' \\ \boldsymbol{u}' \\ \boldsymbol{\delta_x} \\ \boldsymbol{\delta_y} \end{pmatrix}$, and the left parts of (A.9)-(A.12) into one matrix, we obtain the optimization problem presented in Section 7.8.1.

BIBLIOGRAPHY

[1] R. Rocher, D. Ménard, N. Hervé, and O. Sentieys, "Fixed-point configurable hardware components," *EURASIP Signal of Embedded Systems*, no. 1, Jan 2006.

[2] D. Menard, R. Rocher, O. Sentieys, N. Simon, L.-S. Didier, T. Hilaire, B. Lopez, E. Goubault, S. Putot, F. Vedrine, A. Najahi, G. Revy, L. Fangain, C. Samoyeau, F. Lemonnier, and C. Clienti, "Design of Fixed-point Embedded Systems (DEFIS)," in *2012 Conference on Design and Architectures for Signal and Image Processing (DASIP)*. ECSI - European Electronic Chips and Systems design Initiative, 2012, pp. 365–366.

[3] G. Constantinides, P. Y. K. Cheung, and W. Luk, *Synthesis and Optimization of DSP Algorithms*. Springer US, 2007.

[4] T. Hilaire, "Analyse et synthèse de l'implémentation de lois de contrôle-commande en précision finie (Étude dans le cadre des applications automobiles sur calculateur embarquée)," Ph.D. dissertation, Université de Nantes, Jun 2006.

[5] ——, "Towards Tools and Methodology for the Fixed-Point Implementation of Linear Filters," in *Digital Signal Processing Workshop and IEEE Signal Processing Education Workshop (DSP/SPE), 2011 IEEE*, Jan 2011, pp. 488–493.

[6] B. Lopez, "Implémentation optimale de filtres linéaires en arithmétique virgule fixe," Ph.D. dissertation, UPMC, 2015.

[7] A. Volkova and T. Hilaire, "Fixed-point implementation of lattice wave digital filter: Comparison and error analysis," in *2015 23rd European Signal Processing Conference (EUSIPCO)*, Aug 2015, pp. 1118–1122.

[8] A. Volkova, C. Lauter, and T. Hilaire, "Computing the worst-case peak gain of digital filter in interval arithmetic." in *17th International Symposium on Scientific Computing, Computer Arithmetics and Verified Numerics (SCAN)*, Sep 2016, abstract.

[9] A. Volkova, T. Hilaire, and C. Lauter, “Reliable verification of digital implemented filters against frequency specifications,” in *2017 IEEE 24th Symposium on Computer Arithmetic*, July 2017.

[10] V. Balakrishnan and S. Boyd, “On Computing the Worst-Case Peak Gain of Linear Systems,” *Systems & Control Letters*, vol. 19, pp. 265–269, 1992.

[11] A. Volkova, T. Hilaire, and C. Lauter, “Reliable evaluation of the worst-case peak gain matrix in multiple precision,” in *2015 IEEE 22nd Symposium on Computer Arithmetic*, June 2015, pp. 96–103.

[12] T. Hilaire, A. Volkova, and M. Ravoson, “Reliable fixed-point implementation of linear data-flows,” in *2016 IEEE International Workshop on Signal Processing Systems (SiPS)*, Oct 2016, pp. 92–97.

[13] A. Volkova, T. Hilaire, and C. Lauter, “Determining fixed-point formats for a digital filter implementation using the worst-case peak gain measure,” in *2015 49th Asilomar Conference on Signals, Systems and Computers*, Nov 2015, pp. 737–741.

[14] A. Volkova, C. Lauter, and T. Hilaire, “Rigorous determination of recursive filter fixed-point implementation with input signal frequency specifications,” in *2017 51st Asilomar Conference on Signals, Systems and Computers*, Sep 2017, abstract accepted.

[15] F. de Dinechin and B. Pasca, “Designing custom arithmetic data paths with FloPoCo,” *IEEE Design & Test of Computers*, vol. 28, no. 4, pp. 18–27, Jul. 2011.

[16] F. de Dinechin, T. Hilaire, M. Istoan, and A. Volkova, “Hardware IIR Filters: Direct Form I Computing Just Right,” Jul. 2017, technical report available online http://hal.upmc.fr/hal-01561052.

[17] F. Qureshi, A. Volkova, J. Takala, and T. Hilaire, “Multiplierless Unified Architecture for Mixed Radix-2/3/4 FFTs,” in *2017 25th European Signal Processing Conference (EUSIPCO)*, Aug 2017.

[18] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*, 3rd ed. NJ, USA: Prentice Hall Press, 2009.

[19] P. Prandoni and M. Vetterli, *Signal Processing for Communications*. EFPL Press, 2008.

[20] R. N. Bracewell, *The Fourier Transform and Its Applications (3rd ed.)*. Boston: McGraw-Hill, 2000.

[21] E. I. Jury, *Theory and Application of the Z-Transform Method.* Krieger Pub Co, 1973.

[22] W. Chen, *The Circuits and Filters Handbook, Second Edition*. Taylor & Francis, 2002.

[23] E. Eitelberg, "Convolution invariance and corrected impulse invariance," *Signal Processing*, vol. 86, no. 5, pp. 1116–1120, 2006.

[24] I. W. Selesnick, M. Lang, and C. S. Burrus, "Magnitude squared design of recursive filters with the Chebyshev norm using a constrained rational Remez algorithm," in *Proceedings of IEEE 6th Digital Signal Processing Workshop*, Oct 1994.

[25] S.-I. Filip, "Robust tools for weighted Chebyshev approximation and applications to digital filter design," Ph.D. dissertation, Université de Lyon, Dec 2016.

[26] M. Lankarany and H. Marvi, "Noise reduction in digital iir filters by finding optimum arrangement of second-order sections," in *2008 Canadian Conference on Electrical and Computer Engineering*, May 2008, pp. 689–692.

[27] C.-W. Wu and P. Cappello, "Computer-aided design of vlsi second-order sections," in *ICASSP '87. IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 12, Apr 1987, pp. 1907–1910.

[28] H. Butterweck, A. van Meer, and G. Verkroost, "New second-order digital filter sections without limit cycles," *IEEE Transactions on Circuits and Systems*, vol. 31, no. 2, pp. 141–146, Feb 1984.

[29] C. Tsai, "Floating-point roundoff noises of first- and second-order sections in parallel form digital filters," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 44, no. 9, pp. 774–779, Sep 1997.

[30] N. Bose, *Multidimensional Systems: Theory and Applications.* IEEE Press., 1979.

[31] T. Kailath, *Linear Systems*. Prentice-Hall, 1980.

[32] J. O. S. III, "Introduction to digital audio signal processing," Lectures at Stanford University, California, USA, 2016, .

[33] S. Denis, *Matrices. Theory and Applications. Second edition*, 1971.

[34] M. Gevers and G. Li, *Parametrizations in Control, Estimations and Filtering Problems: Accuracy Aspects*. Berlin: Springer-Verlag, 1993.

[35] J.-M. Muller, N. Brisebarre, F. de Dinechin, C.-P. Jeannerod, V. Lefèvre, G. Melquiond, N. Revol, D. Stehlé, and S. Torres, *Handbook of Floating-Point Arithmetic*. Birkhäuser Boston, 2010.

[36] J. von Newmann, "First Draft of a Report on the EDVAC," Tech. Rep., 1945.

[37] W. Padgett and D. Anderson, *Fixed-Point Signal Processing*, ser. Synthesis lectures on signal processing. Morgan & Claypool, 2009.

[38] T. Finley, "Two's Complement," Cornell University lecture notes, 2000.

[39] T. Hilaire and B. Lopez, "Reliable Implementation of Linear Filters with Fixed-Point Arithmetic," in *Proc. IEEE Workshop on Signal Processing Systems (SiPS)*, 2013.

[40] R. Oshana, *DSP Software Development Techniques for Embedded and Real-Time Systems*. Elsevier Science, 2006.

[41] B. Lopez, T. Hilaire, and L.-S. Didier, "Sum-of-products evaluation schemes with fixed-point arithmetic, and their application to IIR filter implementation," in *Conf. on Design and Architectures for Signal and Image Proc. (DASIP)*, Oct 2012.

[42] F. de Dinechin, M. Istoan, and A. Massouri, "Sum-of-product architectures computing just right," in *IEEE 25th International Conference on Application-Specific Systems, Architectures and Processors, ASAP 2014, Zurich, Switzerland, June 18-20, 2014*, 2014, pp. 41–47.

[43] C. Weinstein and A. V. Oppenheim, "A comparison of roundoff noise in floating point and fixed point digital filter realizations," *Proceedings of the IEEE*, vol. 57, no. 6, pp. 1181–1183, June 1969.

[44] R. Boite, H. Xian-Liang, and J. P. Renard, "A comparison of fixed-point and floating-point realization of digital filter," in *8th European Conference on Electrotechnics, Conference Proceedings on Area Communication*, Jun 1988, pp. 142–145.

[45] "IEEE Standard for Binary Floating-Point Arithmetic," *ANSI/IEEE Std 754-1985*, 1985.

[46] N. J. Higham, *Accuracy and stability of numerical algorithms (2 ed.)*. SIAM, 2002.

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

[48] D. Goldberg, “What every computer scientist should know about floating-point arithmetic,” *ACM Comput. Surv.*, vol. 23, no. 1, pp. 5–48, 1991.

[49] S. Chevillard, J. Harrison, M. Joldes, and C. Lauter, “Efficient and accurate computation of upper bounds of approximation errors,” *Theoretical Computer Science*, vol. 412, no. 16, pp. 1523–1543, 2011.

[50] C. Daramy-Loirat, D. Defour, F. de Dinechin, M. Gallet, N. Gast, C. Lauter, and J.-M. Muller, “CR-LIBM A library of correctly rounded elementary functions in double-precision,” research report available online https://hal-ens-lyon.archives-ouvertes.fr/ensl-01529804, Dec. 2006.

[51] N. Brisebarre, G. Hanrot, and O. Robert, “Exponential sums and correctly-rounded functions,” *IEEE Transactions on Computers*, vol. PP, no. 99, pp. 1–1, 2017.

[52] F. de Dinechin, C. Lauter, and J.-M. Muller, “Fast and correctly rounded logarithms in double-precision,” *RAIRO - Theoretical Informatics and Applications*, vol. 41, no. 1, pp. 85–102, April 2007.

[53] C. Q. Lauter, “Arrondi correct de fonctions mathématiques : fonctions univariées et bivariées, certification et automatisation,” Ph.D. dissertation, ENS de Lyon, 2008.

[54] S. Chevillard, C. Lauter, and M. Joldes, *Users’ manual for the Sollya tool, Release 6.0*, LIP, LIP6, LORIA, CNRS, APICS, INRIA.

[55] N. Brunie, F. de Dinechin, O. Kupriianova, and C. Q. Lauter, “Code generators for mathematical functions,” in *22nd IEEE Symposium on Computer Arithmetic, ARITH 2015, Lyon, France, June 22-24, 2015*, 2015, pp. 66–73.

[56] U. Kulisch and V. Snyder, “The Exact Dot Product As Basic Tool for Long Interval Arithmetic,” *Computing*, vol. 91, no. 3, pp. 307–313, Mar 2011.

[57] M. Joldes, O. Marty, J. M. Muller, and V. Popescu, “Arithmetic algorithms for extended precision using floating-point expansions,” *IEEE Transactions on Computers*, vol. 65, no. 4, pp. 1197–1210, April 2016.

[58] D. A. Pope and M. L. Stein, “Multiple precision arithmetic,” *Commun. ACM*, vol. 3, no. 12, pp. 652–654, Dec. 1960.

[59] L. Fousse, G. Hanrot, V. Lefèvre, P. Pélissier, and P. Zimmermann, “MPFR: A multiple-precision binary floating-point library with correct rounding,” *ACM Transactions on Mathematical Software*, vol. 33, no. 2, Jun. 2007.

[60] R. E. Moore, R. B. Kearfott, and M. J. Cloud, *Introduction to Interval Analysis*. SIAM, 2009.

[61] T. Hickey, Q. Ju, and M. H. Van Emden, "Interval arithmetic: From principles to implementation," *J. ACM*, vol. 48, no. 5, pp. 1038–1068, Sep. 2001.

[62] H. Dawood, *Theories of Interval Arithmetic*.

[63] A. Neumaier, *Interval methods for systems of equations.* Cambridge University Press, Cambridge, UK, 1990.

[64] G. Melquiond, "De l'arithmétique d'intervalles à la certification de programmes," Ph.D. dissertation, ENS Lyon, 2006.

[65] N. Revol and F. Rouillier, "Motivations for an arbitrary precision interval arithmetic and the MPFI library," *Reliable Computing*, vol. 11, no. 4, pp. 275–290, 2005.

[66] ——, *Manual for MPFI 1.5.1*, Spaces, INRIA Lorraine, Arenaire, INRIA Rhone-Alpes, Lab. ANO, USTL (Univ. of Lille).

[67] S. M. Rump, "Guaranteed inclusions for the complex generalized eigenproblem," *Computing*, vol. 42, no. 2-3, pp. 225–238, 1989.

[68] ——, "Reliability in Computing: The Role of Interval Methods in Scientific Computing," R. E. Moore, Ed. Academic Press, 1988, ch. Algorithms for Verified Inclusions — Theory and Practice, pp. 109–126.

[69] ——, "Solution of Linear Systems with Verified Accuracy," *Applied numerical mathematics*, vol. 3, no. 3, pp. 233–241, 1987.

[70] J. Kauraniemi, T. I. Laakso, I. Hartimo, and S. J. Ovaska, "Delta operator realizations of direct-form iir filters," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 45, no. 1, pp. 41–52, Jan 1998.

[71] N. Wong and T.-S. Ng, "A generalized direct-form delta operator-based iir filter with minimum noise gain and sensitivity," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 48, no. 4, pp. 425–431, Apr 2001.

[72] N. V. Dakev, J. F. Whidborne, and A. J. Chipperfield, "$\mathscr{H}_\infty$ design of an EMS control system for a maglev vehicle using evolutionary algorithms," in *Proc. GALESIA 95*, Sheffield U.K., Sep 1995, pp. 226–231.

[73] J. F. Whidborne, I. Postlethwaite, and D.-W. Gu, "Robust Controller Design Using $\mathscr{H}_\infty$ Loop-Shaping and the Method of Inequalities," San Antonio, Texas, Dec 1993, pp. 2163–2168.

[74] S. Chen and J. Wu, "The Determination of Optimal Finite-precision Controller Realisations Using a Global Optimisation Strategy: a Pole-sensitivity Approach," in *Digital Controller Implementation and Fragility: A Modern Perspective*, R. S. H. Istepanian and J. F. Whidborne, Eds. London, UK: Springer-Verlag, 2001, ch. 6, pp. 87–104.

[75] V. Tavsanoglu and L. Thiele, "Optimal design of state - space digital filters by simultaneous minimization of sensitivity and roundoff noise," *IEEE Transactions on Circuits and Systems*, vol. 31, no. 10, pp. 884–888, Oct 1984.

[76] T. Hinamoto, S. Yokoyama, T. Inoue, W. Zeng, and W. Lu, "Analysis and Minimization of L_2 -Sensitivity for Linear Systems and Two-Dimensional State-Space Filters Using General Controllability and Observability Gramians," in *IEEE Transactions on Circuits and Systems, Fundamental Theory and Applications*, vol. 49, no. 9, Sep 2002.

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

[78] G. A. Constantinides, "Perturbation analysis for word-length optimization," in *11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 2003. FCCM 2003.*, April 2003, pp. 81–90.

[79] G. Constantinides, P. Cheung, and W. Luk, "Truncation Noise in Fixed-Point SFGs," *IEE Electronics Letters*, vol. 35, no. 23, pp. 2012–2014, Nov 1999.

[80] D. Báez-López, D. Báez-Villegas, R. Alcántara, J. J. Romero, and T. Escalante, "Package for filter design based on MATLAB," *Comp. Applic. in Engineering Education*, vol. 9, no. 4, pp. 259–264, 2001.

[81] L. Jackson, J. Kaiser, and H. McDonald, "An approach to the implementation of digital filters," *IEEE Transactions on Audio and Electroacoustics*, vol. 16, no. 3, pp. 413–421, Sep 1968.

[82] A. Kireçci, M. Topalbekiroglu, and I. Eker, "Experimental evaluation of a model reference adaptive control for a hydraulic robot: a case study," *Robotica*, vol. 21, no. 1, pp. 71–78, 2003.

[83] L. D. Coster, M. Adé, R. Lauwereins, and J. A. Peperstraete, "Code generation for compiled bit-true simulation of DSP applications," in *Proceedings of the 11th International Symposium on System Synthesis, ISSS '98, Hsinchu, Taiwan*, Dec 1998, pp. 9–14.

[84] A. Benedetti and P. Perona, "Bit-width optimization for configurable dsp's by multi-interval analysis," in *Conference Record of the Thirty-Fourth Asilomar Conference on Signals, Systems and Computers*, vol. 1, Oct 2000, pp. 355–359 vol.1.

[85] C. Carreras, J. A. Lopez, and O. Nieto-Taladriz, "Bit-width selection for data-path implementations," in *Proceedings 12th International Symposium on System Synthesis*, Nov 1999, pp. 114–119.

[86] J. A. Lopez, C. Carreras, and O. Nieto-Taladriz, "Improved interval-based characterization of fixed-point LTI systems with feedback loops," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 26, no. 11, pp. 1923–1933, 2007.

[87] J. Cong, K. Gururaj, B. Liu, C. Liu, Z. Zhang, S. Zhou, and Y. Zou, "Evaluation of static analysis techniques for fixed-point precision optimization," in *17th IEEE Symposium on Field Programmable Custom Computing Machines In Proceedings*, 2009, pp. 231–234.

[88] C. F. Fang, R. A. Rutenbar, M. Puschel, and T. Chen, "Toward efficient static analysis of finite-precision effects in dsp applications via affine arithmetic modeling," in *Proceedings 2003. Design Automation Conference*, June 2003, pp. 496–501.

[89] L. H. de Figueiredo and J. Stolfi, "Affine arithmetic: Concepts and applications," *Numerical Algorithms*, vol. 37, no. 1-4, pp. 147–158, 2004.

[90] D. Menard, R. Rocher, and O. Sentieys, "Analytical fixed-point accuracy evaluation in linear time-invariant systems," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 55, no. 10, pp. 3197–3208, Nov 2008.

[91] N. Herve, D. Menard, and O. Sentieys, "Data wordlength optimization for FPGA synthesis," in *IEEE Workshop on Signal Processing Systems Design and Implementation, 2005.*, Nov 2005, pp. 623–628.

[92] E. Parzen, "On estimation of a probability density function and mode," *The Annals of Mathematical Statistics*, vol. 33, no. 3, pp. 1065–1076, 1962.

[93] A. Banciu, “A Stochastic Approach For The Range Evaluation,” Theses, Université Rennes 1, Feb. 2012.

[94] T. Hilaire, P. Chevrel, and J.-P. Clauzel, “Low Parametric Sensitivity Realization Design for FWL Implementation of MIMO Controllers,” in *Proc. of Control Applications of Optimisation CAO'O6*, Apr 2006.

[95] T. Hilaire, D. Ménard, and O. Sentieys, “Roundoff Noise Analysis of Finite Wordlength Realizations with the Implicit State-Space Framework,” in *EUSIPCO'07*, 2007, pp. 1019–1023.

[96] T. Hilaire, P. Chevrel, and J.-P. Clauzel, “Pole Sensitivity Stability Related Measure of FWL Realization with the Implicit State-Space Formalism,” in *5th IFAC Symposium on Robust Control Design (ROCOND'06)*, Jul 2006.

[97] T. Hilaire, *FWR Toolbox User's Guide (v0.99)*, http://fwrtoolbox.gforge.inria.fr/, 2009.

[98] T. Hilaire, P. Chevrel, and J.-P. Clauzel, “Low Parametric Sensitivity Realization Design for FWL Implementation of MIMO Controllers : Theory and Application to the Active Control of Vehicle Longitudinal Oscillations,” *International Journal of Tomography & Statistics*, vol. 6, no. 7, pp. 128–133, 2007.

[99] A. Fettweiss, “Wave Digital Filters: Theory and Practice,” *Proc. of the IEEE*, vol. 74, no. 2, 1986.

[100] J. Yli-Kaakinen and T. Saramaki, “An algorithm for the design of multiplierless approximately linear-phase lattice-wave digital filters,” in *2000 IEEE International Symposium on Circuits and Systems In Proceedings*, vol. 2, 2000, pp. 77–80 vol.2.

[101] L. Gazsi, “Explicit formulas for lattice wave digital filters,” *IEEE Trans. Circuits & Systems*, vol. 32, no. 1, 1985.

[102] B. Friedlander, “Lattice filters for adaptive processing,” *Proceedings of the IEEE*, vol. 70, no. 8, pp. 829–867, Aug 1982.

[103] H. Johansson and L. Wanharomar, “Digital Hilbert transformers composed of identical allpass subfilters,” in *ISCAS 1998. Proceedings of*, vol. 5, pp. 437–440 vol.5.

[104] A. Fettweis, H. Levin and A. Sedlmeyer “Wave digital lattice filters,” *International Journal on Circuit Theory and Applications*, vol. 2, pp. 203–211, Jun 1974.

[105] H. Johansson and L. Wanharomar, “Design of linear-phase Lattice Wave Digital Filters.”

[106] T. S. J. Yli-Kaakinen, “A systematic algorithm for the design of multiplierless lattice wave digital filters,” in *First International Symposium on Control, Communications and Signal Processing*, 2004, pp. 393–396.

[107] H. Ohlsson, O. Gustafsson, W. Li, and L. Wanhammar, “An environment for design and implementation of energy efficient digital filters,” in *Swedish System-on-Chip Conference*, Apr 2003.

[108] J. Yli-Kaakinen and T. Saramäki, “A Systematic Algorithm for the Design of Lattice Wave Digital Filters With Short-Coefficient Wordlength,” *IEEE Trans. on Circuits & Systems*, 2007.

[109] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction To Algorithms*. MIT Press, 2001.

[110] J. Kuriakose, S. Ristic, and G. de Cremoux, “An automated toolbox for the design and analysis of lattice wave digital filters using matlab,” in *IEE Colloquium on DSP enabled Radio*, Sep 2003, pp. 1–8.

[111] J. C. Vold, Havard and G. T. Rocklin., “New ways of estimating frequency response functions.” *Sound and Vibration*, vol. 18, pp. 34–38, Nov 1984.

[112] S. Pillai and T. Shim, *Spectrum estimation and system identification*. Springer-Verlag, 1993.

[113] E. Jones, T. Oliphant, P. Peterson *et al.*, “SciPy: Open source scientific tools for Python,” 2001, http://www.scipy.org/.

[114] S. Boyd and J. Doyle, “Comparison of peak and RMS gains for discrete-time systems,” *System & Control Letters*, vol. 9, pp. 1–6, 1987.

[115] F. Johansson *et al.*, *mpmath: a Python library for arbitrary-precision floating-point arithmetic (version 0.18)*, December 2013, http://mpmath.org/.

[116] J. Carletta, R. Veillette, F. Krach, and Z. F., “Determining appropriate precisions for signals in fixed-point IIR filters,” in *Design Automation Conference, 2003. Proceedings*, 2003, pp. 656–661.

[117] R. H. Bartels and G. W. Stewart, “Solution of the matrix equation AX + XB = C,” *Commun. ACM*, vol. 15, no. 9, pp. 820–826, Sep. 1972.

[118] V. Simoncini, “Computational methods for linear matrix equations,” *SIAM Review*, vol. 58, no. 3, pp. 377–441, 2016.

[119] S. Hammarling, “Numerical solution of the discrete-time, convergent, non-negative definite lyapunov equation,” *Syst. Control Lett.*, vol. 17, no. 2, pp. 137–139, Aug. 1991.

[120] M. Abramowitz and I. Stegun, *Handbook of Mathematical Functions: With Formulas, Graphs, and Mathematical Tables*. Dover Publications, 1964.

[121] H. Dawood, *Theories of Interval Arithmetic: Mathematical Foundations and Applications*. LAP Lambert Academic Publishing, 2011.

[122] S. M. Rump, “New Results on Verified Inclusions,” in *Accurate Scientific Computations, Symposium, 1985, Proceedings*, 1985, pp. 31–69.

[123] S. Gershgorin, “Über die Abgrenzung der Eigenwerte einer Matrix.” *Bull. Acad. Sci. URSS*, vol. 1931, no. 6, pp. 749–754, 1931.

[124] L. Fousse, G. Hanrot, V. Lefèvre, P. Pélissier, and P. Zimmermann, “ MPFR : A Multiple-Precision Binary Floating-Point Library with Correct Rounding,” *ACM Transactions on Mathematical Software*, vol. 33, no. 2, pp. 13:1—-13:15, 2007.

[125] V. Pan and J. Reif, “Efficient Parallel Solution of Linear Systems,” in *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*. ACM, 1985, pp. 143–152.

[126] D. Lefebvre, P. Chevrel, and S. Richard, “An $\mathscr{H}_\infty$ based control design methodology dedicated to the active control of longitudinal oscillations,” *IEEE Trans. on Control Systems Tec hnology*, vol. 11, no. 6, pp. 948–956, 2003.

[127] Z. Zhao and G. Li, “Roundoff noise analysis of two efficient digital filter structures,” *IEEE Trans. on Signal Processing*, vol. 54, no. 2, pp. 790–795, 2006.

[128] K. Aström and R. Murray, *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton University Press, 2010.

[129] A. Neumaier, *Interval Methods for Systems of Equations*. Cambridge, UK: Cambridge University Press, 1990, vol. 37.

[130] ——, “A distributive interval arithmetic,” *Freiburger Intervall-Berichte*, vol. 10, no. 82, pp. 31–38, 1982.

[131] S. J. Xu and A. Rachid, “Generalized Gerschgorin disc and stability analysis of dynamic interval systems,” in *Control ’96, UKACC International Conference on*, vol. 1, Sept 1996, pp. 276–280 vol.1.

[132] S. Corsaro and M. Marino, “Interval linear systems: the state of the art,” *Computational Statistics*, vol. 21, no. 2, pp. 365–384, Jun 2006.

[133] A. Neumaier, “New techniques for the analysis of linear interval equations,” *Linear Algebra and its Applications*, vol. 58, pp. 273 – 325, 1984.

[134] J. Rohn, “Inverse interval matrix,” *SIAM Journal on Numerical Analysis*, vol. 30, no. 3, pp. 864–870, 1993.

[135] V. Kreinovich, “Solving equations (and systems of equations) under uncertainty: how different practical problems lead to different mathematical and computational formulations,” *Granular Computing*, vol. 1, no. 3, pp. 171–179, Sep 2016.

[136] E. R. Hansen, “Bounding the solution of interval linear equations,” *SIAM Journal on Numerical Analysis*, vol. 29, no. 5, pp. 1493–1503, 1992.

[137] M. Christensen and F. J. Taylor, “Fixed-Point IIR filter challenges,” *EDN Networks*, vol. 51, no. 23, 2006.

[138] B. Lopez, T. Hilaire, and L.-S. Didier, “ Formatting bits to better implement signal processing algorithms,” in *4th Int. conf. PECCS , proceedings of*, 2014.

[139] L. Tsoeunyane, S. Winberg, and M. Inggs, “Software-defined radio FPGA cores: Building towards a domain-specific language,” *International Journal of Reconfigurable Computing*, Apr 2017.

[140] A. Ziv, “Fast evaluation of elementary mathematical functions with correctly rounded last bit,” *ACM Trans. Math. Softw.*, vol. 17, no. 3, pp. 410–423, Sep. 1991.

[141] L. Hogben, *Handbook of Linear Algebra*. CRC Press, 2006.

[142] I. Zelinka, V. Snasel, and A. Abraham, *Handbook of Optimization: From Classical to Modern Approach*. Springer Berlin Heidelberg, 2012.

[143] G. Cornuejols, “Valid inequalities for mixed integer linear programs.” *Mathematical Programming B,*, vol. 112, pp. 3–44, apr 2008.

[144] D. E. S. William Cook, Thorsten Koch and K. Wolter, "A hybrid branch-and-bound approach for exact rational mixed-integer programming," Konrad-Zuse-Zentrum fur Informationstechnik Berlin, Tech. Rep., 2012.

[145] W. J. Cook, T. Koch, D. E. Steffy, and K. Wolter, "An exact rational mixed-integer programming solver," in *Integer Programming and Combinatoral Optimization - 15th International Conference, IPCO 2011, New York, NY, USA, June 15-17, 2011. Proceedings*, 2011, pp. 104–116.

[146] D. E. Steffy and K. Wolter, "Valid linear programming bounds for exact mixed-integer programming," *INFORMS Journal on Computing*, vol. 25, no. 2, pp. 271–284, 2013.

[147] M. Hamza and M. Rasmy, "A simple method for determining the reachable set for linear discrete systems," *IEEE Transactions on Automatic Control*, vol. 16, no. 3, pp. 281–282, June 1971.

[148] B. Xue, Z. She, and A. Easwaran, "Under-approximating backward reachable sets by semialgebraic sets," *IEEE Transactions on Automatic Control*, vol. PP, no. 99, pp. 1–1, 2017.

[149] D. P. Bertsekas and I. B. Rhodes, "On the minimax reachability of target sets and target tubes," *Automatica*, vol. 7, no. 2, pp. 233–247, Mar. 1971.

[150] A. Barvinok, *Integer Points in Polyhedra*. European Mathematical Society, 2008.

[151] C. Barrett, R. Sebastiani, S. Seshia, and C. Tinelli, "Satisfiability Modulo Theories," in *Handbook of Satisfiability*. IOS Press, Feb 2009, vol. 185, ch. 26, pp. 825–885.

[152] T. W. Parks and J. H. McClellan, "Chebyshev Approximation for Nonrecursive Digital Filters with Linear Phase," *IEEE Transactions on Circuit Theory*, vol. 19, no. 2, pp. 189–194, Mar 1972.

[153] S.-I. Filip, "A Robust and Scalable Implementation of the Parks-McClellan Algorithm for Designing FIR Filters," *ACM Trans. Math. Softw.*, vol. 43, no. 1, pp. 7:1—-7:24, Aug 2016.

[154] B. W. Char, K. O. Geddes, and G. H. Gonnet, "GCDHEU: Heuristic polynomial GCD algorithm based on integer GCD computation," *Journal of Symbolic Computation*, vol. 7, no. 1, pp. 31–48, 1989.

[155] M.-F. Roy, *Basic algorithms in real algebraic geometry and their complexity: from Sturm's theorem to the existential theory of reals*. de Gruyter, 1996, vol. 23.

[156] G. Li and Z. Zhao, "On the Generalized DFIIt Structure and its State-Space Realization in Digital Filter Implementation," *IEEE Trans. on Circuits and Systems*, vol. 51, no. 4, pp. 769–778, Apr 2004.

[157] M. Gevers and G. Li, *Parametrizations in Control, Estimation and Filtering Probems*. Springer-Verlag, 1993.

[158] J. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach*. Elsevier Science, 2011.

[159] K. D. Chapman, "Fast integer multipliers fit in FPGAs (EDN 1993 design idea winner)," *EDN magazine*, no. 10, p. 80, May 1993.

[160] M. J. Wirthlin, "Constant Coefficient Multiplication Using Look-Up Tables," *Journal of VLSI Signal Processing*, vol. 36, no. 1, pp. 7–15, 2004.

[161] F. de Dinechin, H. Takeugming, and J.-M. Tanguy, "A 128-Tap Complex FIR Filter Processing 20 Giga-Samples/s in a Single FPGA ," in *44th Asilomar Conference on Signals, Systems & Computers*, 2010.

[162] N. Brunie, F. de Dinechin, M. Istoan, G. Sergent, K. Illyes, and B. Popa, "Arithmetic core generation using bit heaps," in *Field-Programmable Logic and Applications*, Sep 2013.

[163] IEEE Std 802.15.4-2006, *IEEE Standard for Information technology– Telecommunications and information exchange between systems– Local and metropolitan area networks– Specific requirements– Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs)*.

[164] The MathWorks, Inc., "Matlab signal processing toolbox."

[165] F. de Dinechin, M. Istoan, and A. Massouri, "Sum-of-product architectures computing just right," in *2014 IEEE 25th International Conference on Application-Specific Systems, Architectures and Processors*, June 2014, pp. 41–47.

[166] M. D. Ercegovac and T. Lang, *Digital Arithmetic*. Morgan Kaufmann, 2003.

[167] H. Parendeh-Afshar, A. Neogy, P. Brisk, and P. Ienne, “Compressor Tree Synthesis on Commercial High-Performance FPGAs,” *ACM Transactions on Reconfigurable Technology and Systems*, vol. 4, no. 4, 2011.

[168] R. Kumar, A. Mandal, and S. P. Khatri, “An efficient arithmetic Sum-of-Product (SOP) based multiplication approach for FIR filters and DFT,” in *International Conference on Computer Design (ICCD)*. IEEE, Sep 2012, pp. 195–200.

[169] T. Parks and J. McClellan, “Chebyshev approximation for nonrecursive digital filters with linear phase,” *IEEE Transactions on Circuit Theory*, vol. 19, no. 2, pp. 189–194, March 1972.

[170] S. Torres, “Tools for the Design of Reliable and Efficient Functions Evaluation Libraries,” Theses, Université de Lyon, Sep. 2016.

[171] S. Filip, “A robust and scalable implementation of the parks-mcclellan algorithm for designing FIR filters,” *ACM Trans. Math. Softw.*, vol. 43, no. 1, pp. 7:1–7:24, 2016.

Towards reliable implementation of digital filters

Abstract: In this thesis we develop approaches for improvement of the numerical behavior of digital filters with focus on the impact of accuracy of the computations. This work is done in the context of a reliable hardware/software code generator for Linear Time-Invariant (LTI) digital filters, in particular with Infinite Impulse Response (IIR). With this work we consider problems related to the implementation of LTI filters in Fixed-Point arithmetic while taking into account finite precision of the computations necessary for the transformation from filter to code. This point is important in the context of filters used in embedded critical systems such as autonomous vehicles. We provide a new methodology for the error analysis when linear filter algorithms are investigated from a computer arithmetic aspect. In the heart of this methodology lies the reliable evaluation of the Worst-Case Peak Gain measure of a filter, which is the l1 norm of its impulse response. The proposed error analysis is based on a combination of techniques such as rigorous Floating-Point error analysis, interval arithmetic and multiple precision implementations. This thesis also investigates the problematic of compromise between hardware cost (e.g. area) and the precision of computations during the implementation on FPGA. We provide basic brick algorithms for an automatic solution of this problem. Finally, we integrate our approaches into an open-source unifying framework to enable automatic and reliable implementation of any LTI digital filter algorithm.

Keywords: computer arithmetic; rigorous and reliable algorithms; error analysis; code generator; Fixed-Point, Floating-Point, Interval and Multiple Precision arithmetics; linear digital filters.

Algorithmique de l'implémentation fiable de filtres numériques

Résumé : Dans cette thèse nous essayons d'améliorer l'évaluation des filtres numériques en nous concentrant sur la précision de calcul nécessaire. Ce travail est réalisé dans le contexte d'un générateur de code matériel/logiciel fiable pour des filtres numériques linéaires, en particulier les filtres à Réponse Impulsionnelle Infinie (IIR). Dans ce travail, nous mettons en avant les problèmes liés à l'implémentation de filtres linéaires en arithmétique Virgule Fixe, tout en prenant en compte la précision finie des calculs nécessaires à la transformation des filtres vers du code. Ce point est important dans le cadre de filtres utilisés dans des systèmes embarqués critiques comme les véhicules autonomes, l'aéronautique, etc. Nous fournissons une nouvelle méthodologie pour l'analyse d'erreur lors de l'étude d'algorithmes de filtres linéaires du point de vue de l'arithmétique des ordinateurs. Au cœur de cette méthodologie se trouve le calcul fiable de la mesure *Worst Case Peak Gain* d'un filtre, qui est la norme L1 de sa réponse impulsionnelle. L'analyse d'erreur proposée est basée sur la combinaison de techniques telles que l'analyse d'erreur en Virgule Flottante, l'arithmétique d'intervalles et les implémentations multi-précisions. Cette thèse expose également la problématique de compromis entre les coûts du matériel (e.g. la surface) et la précision de calcul lors de l'implémentation de filtres numériques sur FPGA. Nous fournissons des briques algorithmiques de bases pour une solution automatique de ce problème. Finalement, nous intégrons nos approches dans un générateur de code pour les filtres afin de permettre l'implémentation automatique et fiable de tout algorithme de filtre linéaire numérique (outil open-source).

Mots clés : arithmétique des ordinateurs; algorithmes rigoureux et fiables; arithmétiques virgule fixe, virgule flottante, d'intervalles, multi-précision; analyse d'erreur; générateur du code; filtres linéaires.