DAVID LAMB

EFFICIENT ALGORITHMS AND HARDWARE STRUCTURES FOR FRACTIONAL DELAY FILTERING AND SAMPLE RATE CONVERSION

São Paulo
2015

DAVID LAMB

EFFICIENT ALGORITHMS AND HARDWARE STRUCTURES FOR FRACTIONAL DELAY FILTERING AND SAMPLE RATE CONVERSION

Tese apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Doutor em Engenharia Elétrica.

São Paulo
2015

DAVID LAMB

EFFICIENT ALGORITHMS AND HARDWARE STRUCTURES FOR FRACTIONAL DELAY FILTERING AND SAMPLE RATE CONVERSION

Tese apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Doutor em Engenharia Elétrica.

Área de Concentração:

Sistemas Eletrônicos

Orientador:

Vítor Heloiz Nascimento

São Paulo
2015

ACKNOWLEDGMENTS

This thesis is the fruit of my work at the University of São Paulo in the Electronic Systems Department (Departamento de Engenharia de Sistemas Eletrônicos). Years spent at the University were clearly the best of my academic life. I would like to thank everyone from the University, especially my colleagues from the PSI lab, for their constant support.

I'm particularly grateful to my supervisor, Dr. Vítor Heloiz Nascimento, for his valuable guidance, unwavering patience and flexibility, and for believing in me during all these years.

Many people contributed through encouragements and support. First and foremost I want to thank my parents, Monique Beaudoin and Claude Lamb, and my brother, Jean-Philippe Lamb, for teaching me from a very young age that overcoming challenges is it's own reward. A very special thank goes to Dr. David Hossack for his constant motivation to pursue this work and to, as he would say, *get it done*, as well as to my friend and fellow student Luiz F. O. Chamon, for always listening to my embryonic ideas and pushing me to make them flourish. I also want to thank Doctor Bruno Capron for his constant help with logistics.

Last but not least, I want to thank my girlfriend Erika Dias da Silva for her love, support and patience, and her family for welcoming me to their country with open arms and making it a place that I now call *home*.

"Our future success is only limited by the scale of our ambition"

-Jerald G. Fishman

"Our future success is only limited by the scale of our ambition"

-Jerald G. Fishman

RESUMO

Nesta pesquisa, são propostas novas estruturas de filtragem para a conversão da taxa de amostragem de sinais digitais. São consideradas taxas de conversão inteiras e fracionárias e ênfase é dada ao desenvolvimento de estruturas com baixa complexidade em hardware para uso em circuitos integrados dedicados (ASIC).

Primeiramente, os dois principais desafios da conversão assíncrona de taxa de amostragem são abordados: a geração em tempo real de sinais de *clock* que rastreiam a taxa fracionária e a implementação eficiente de filtros com atraso fracionário. Uma nova técnica totalmente digital para a geração de *clocks* é introduzida. Ela difere do DPLL clássico pois o rastreamento é baseado no período do *clock* de referência ao invés de sua fase e/ou frequência, o que permite seu uso em faixas mais largas de freqüências e sincronização mais rápida. Em seguida, uma nova estrutura para efetuar filtragem com atrasos fracionários utilizando polinômios *spline* é derivada usando a relação entre o filtro Farrow e a estrutura de Newton. A complexidade computacional do filtro é consideravelmente reduzida, se tornando comparável à de filtros com polinômios de Lagrange.

Em seguida, o problema de conversão de taxa de amostragem com taxas inteiras é considerado, especificamente do ponto de vista de redução da complexidade do primeiro estágio de uma cascata de filtros de decimação, tipicamente implementado usando um filtro de cascata integrador-diferenciador (CIC). A área e o consumo de energia do primeiro filtro é um fator em muitas aplicações, especialmente em conversores A/D Σ-Δ, devido às altas frequências de amostragem envolvidas. A implementação em ponto fixo de uma variante com espaço de estado reduzido do filtro CIC introduzida anteriormente mostra que esta pode ser uma alternativa interessante para filtros de ordem menor com fator de decimação em potências de dois. Por fim, uma nova técnica é desenvolvida e integrada no filtro CIC, mantendo suas vantagens, melhorando seu desempenho e reduzindo a área da implementação. A estrutura baseia-se na introdução de um multiplicador com coeficientes variáveis no tempo capaz de aproveitar a eficiência de filtros FIR esparsos.

Palavras-Chave – Conversão da Taxa de Amostragem, Conversão Fracionárias, Farrow, Interpolação, Decimação.

ABSTRACT

In this work, new filtering structures for the sampling rate conversion of digital signals are proposed. Both integer and fractional rate change are considered, and the emphasis is put on developing hardware structures with low complexity for use in application specific integrated circuits (ASIC).

First, the two main challenges of fractional sample rate conversion are addressed, namely the real time generation of clock signals that properly track the fractional ratio, and the efficient implementation of the polynomial-based filter. A new technique for all-digital clock generation circuit is introduced. It differs from classical DPLLs in that the locking mechanism is based on the period of the reference clock instead of its phase and/or frequency which allows for wide bandwidths and fast locking times. Then a new structure for performing the fractional filtering operation using spline polynomials is derived, based on the relationship of the well-known Farrow structure and the relatively newly introduced Newton structure. The computational complexity of implementing the spline polynomial is greatly reduced and approaches that of the Lagrange polynomial.

Then, the problem of integer sample rate conversion is considered, mainly from the perspective of reducing the complexity of the first filter of a multistage decimating chain, typically implemented using a Cascaded-Integrator-Comb (CIC) filter. The area and power consumption of the first filter is a concern in many applications, notably Σ-Δ A/D converters, because of the high sampling frequencies involved. The fixed-point implementation of a previously introduced reduced state-space variant of the CIC filter is considered and proven to be an interesting alternative for lower order filter with power of two decimating ratios. Finally, a novel technique is developed that integrates seamlessly within the CIC filter, keeping all of its advantages, while improving performance and reducing area. The structure is based on the inclusion of a time-varying multiplier as part of the original filter, leveraging the computational efficiency of sparse FIR filters.

Keywords – Sample Rate Conversion, Interpolation, Decimation, CIC filters, Fractional Delay Filtering, Farrow Structure, All-Digital Phase Locked Loop (DPLL), Noise Shaping.

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1 INTRODUCTION

Sampling rate conversion is at the very heart of digital signal processing (DSP). As soon as two digital systems running at different rates are exchanging data, there is a need for sampling rate conversion (SRC). Also, many applications which on surface seem to deal with a single sample rate are relying on some form of sample rate conversion to happen behind the scenes. For example, one only has to look at any oversampling Σ-Δ A/D converters which are ubiquitous in interfacing the digital and analog worlds to find specialized circuits performing sample rate conversion. The design of such circuits is particularly complex due to extreme constraints put on silicon area and power consumption. This paved the way to enormous research efforts on the derivation of efficient filtering structures to perform the surprisingly difficult task of altering the sampling rate of digital signals. This dissertation is one more effort at trying to improve the many algorithms and filtering structures available for SRC, with a unique focus on the cost of dedicated hardware implementations.

The first part of the text is focused on the problem of asynchronous sample rate conversion (ASRC). ASRC arises in many systems where a discrete time sequence needs to be converted to a new sequence with different sampling rate where the rate conversion is fractional, or even irrational. In some systems, the ratio might even be slowly varying and needs to be properly tracked. In the end, an ASRC algorithm has to be able to evaluate the value of a discrete time sequence at *any* time instant between input samples. Software Defined Radio is one important application where ASRC is required; different communication standards use different symbol rates that have to be recovered in the digital domain after being sampled by an unrelated fixed rate A/D converter [1,2]. The sampling rate of the the signal thus has to be converted to the original symbol rate entirely in the digital domain. ASRC is also found in many audio systems, where signals coming from different sources running at completely asynchronous sampling rates have to be converted to the rate of the local digital signal processor before further processing [3,4]. A less obvious application is in audio power amplifiers where the ASRC is used to convert the signal to a sampling rate where the switching frequency of the amplifier does not

interfere with an AM band [5].

The problem of ASRC can be divided in two distinct parts: (1) the calculation of the ratio between the two sampling rates, and (2), the digital filter used to calculate new data points at the required time instant. The first problem is analogous to the one faced by digital phase-locked loop (DPLL) designers and similar techniques are used. The second problem is typically solved by using FIR filters where the impulse response is built in real time using piecewise polynomial basis functions [6]. Both topics will be explored in this work and circuits with reduced complexity are proposed.

The second part of the text is concerned with the mathematically simpler problem of integer sampling rate conversion, where the ratio is known to be a fixed integer. Although the problem is easier to grasp, it is generally required in extremely demanding applications where silicon area and power consumption are tightly constrained, such as Σ-Δ A/D converters. For example, efficient decimating filters are constantly being researched and new structures regularly proposed [7–10]. It is well known that the problem of decimation is better solved using a multistage approach, where the performance requirements of the first filters of the cascade, running at higher sampling rates, can be reduced without affecting the overall system performance [11]. The design of the first stage filter will be explored in greater detail leading to a novel implementation structure.

1.1 Contributions of this work

The main contributions of this dissertation are as follow:

1. A new locking mechanism is proposed for the design of an all-digital clock generator circuit which is based on the period of the reference clock instead of its phase and/or frequency. This leads to the proposed period-locked Numerically Controlled Oscillator (NCO) which decouples various trade-offs inherent to feedback loops, such as input jitter rejection, bandwidth, and settling time

2. Noise-shaping techniques previously used in Direct Digital Synthesizers (DDS) are shown to have a straightforward relationship with the period-locked NCO and can be directly integrated into the new circuit to reduce the phase noise of the generated clock signal.

3. A matrix form of the Farrow transfer function is put forward and used to derive state-space transformations between two otherwise completely different implementation

structures of the Lagrange interpolator. These transformations are then applied to the spline polynomial giving rise to an efficient spline filtering method. The methods proposed here form the basis for developing other efficient structures based on other polynomials.

4. The fixed-point implementation of reduced state-space CIC filters is considered and it is demonstrated that stability can be guaranteed if proper wordlength is used throughout the filter. A technique is derived to calculate the maximum filter gain at each integrator stage, leading to a reduced wordlength implementation while preserving the desired finite impulse response intact.

5. A new filtering structure is proposed that systematically outperforms the classical CIC filter response, while reducing the hardware and computational expenditure. It is shown that the last integrator of standard CIC filters can be transformed into a fully programmable decimating FIR filter at very low cost and that a simple sparse filter with small integer coefficients is typically enough to increase performance.

6. A Mixed Integer Linear Programming framework for the design of the new sparse CIC filter is proposed.

1.2 List of publications and patent

1.2.1 Publications

The publications that resulted from the research presented in this dissertation are:

1. The Period-Locked NCO presented in Chapter 2 was submitted to IEEE Transactions on Circuits and Systems II (TCAS-II).

2. The new structure for fractional delay filtering using the spline polynomial of Chapter 3 was submitted to Electronics Letters (IET).

3. The fixed-point design and implementation of reduced state-space CIC filters from Chapter 4 was submitted to Electronics Letters (IET).

4. The new sparse CIC filtering structure of Chapter 5 will be submitted to IEEE Transactions on Circuits and Systems I (TCAS-I) as soon as the patent application is filed.

1.2.2 Patent

A patent application is also being prepared for the new sparse CIC filtering structure of Chapter 5.

1.3 Organization of the text

The dissertation is divided in two main parts; Part I is concerned with fractional sample rate conversion, touching on both the fractional interval ratio calculation and timing generation as well as the polynomial filter structure used to process the data. Then, in Part II, integer sample rate conversion is discussed, mainly proposing improvements to the first filter of a multistage decimating chain.

The remainder of this dissertation is structured as follow:

Chapter 2 presents a novel all-digital clock generator circuit where the period of the reference signal is measured and used as way to enforce lock.

Chapter 3 introduces an efficient structure for spline-based fractional delay filtering for interpolation/decimation.

Chapter 4 investigates the feasibility of the fixed-point implementation of the so-called reduced state-space CIC filter structure.

Chapter 5 proposes a novel technique to improve the performance of standard CIC filters while reducing complexity.

Chapter 6 concludes the thesis with a summary of the results and provides directions for future research.

PART I

FRACTIONAL SAMPLE RATE CONVERSION

2 FRACTIONAL DELAY FILTERING: TIMING OF THE INTERSAMPLE POSITION

2.1 Introduction

Many digital systems use a stable master clock (*mclk*) to produce a secondary clock signal frequency-locked onto an incoming asynchronous reference clock (*refclk*). Often times, this *mclk* is fixed so that neither its frequency nor phase can be controlled. This seriously constrains the system performance since the edges of the generated clock must be aligned with those of the *mclk* to keep the design fully synchronous and synthesis-friendly. This problem statement is analogous to the one faced by digital phase-locked loop (DPLL) designers, so that the same design methodologies are commonly used. A good review of these techniques is available in [12].

Despite their ubiquity, classical DPLLs (e.g., [13–15]) are intricate to design, typically display long locking times, and do not take advantage of the stable high frequency clock available [12]. To address these issues, this work puts forward the *period-locked NCO*, which relies on measurements of the *refclk* period instead of its frequency/phase. Although similar circuits can be found in [16, 17], they rely on an idle period at the end of each *refclk* cycle to guarantee lock and provide jitter immunity. This idle period introduces distortions in the spectrum of the output clock that can hinder its use in some applications, such as high fidelity analog to digital and digital to analog conversion (ADC/DAC).

The novel solution is developed by first reviewing how Numerically Controlled Oscillators (NCOs) can be used as flexible open-loop clock generators. Then, this circuit is extended to ensure that the generated clock is *locked* onto *refclk* despite possible disturbances, giving rise to the *period-locked NCO*. Contrary to standard closed-loop DPLL techniques, this circuit is locked *by design*. Finally, simulations are used to illustrate the performance of this novel approach and two methods are presented to improve the quality of the generated clock.

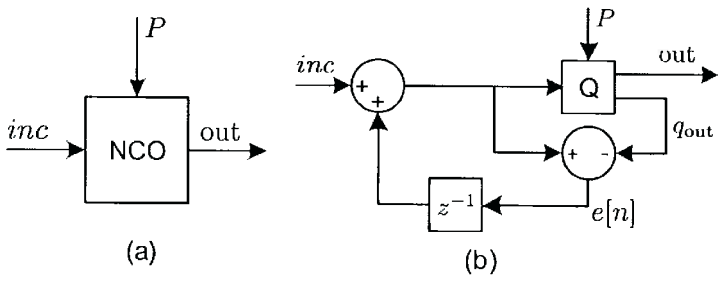

Figure 1: Typical NCO: (a) block diagram; (b) implementation, with $q_{\text{out}} = 0$ when the input of the quantizer Q is less than P and $q_{\text{out}} = P$, otherwise.

2.2 The NCO as an Open Loop Clock Generator

It is useful to first consider how an NCO can be used as a flexible open-loop clock generator. A typical NCO is simply a modulo-P overflowing accumulator that increments by *inc* each *mclk* cycle (Figure 1). The generated clock frequency (f_{out}) is given by the overflow rate of the accumulator, as in

$$f_{\text{out}} = f_{\text{mclk}} \cdot \frac{inc}{P}, \tag{2.1}$$

where f_{mclk} is the *mclk* frequency. Take, for example, $P = 660$ and $inc = 128$, so that the NCO generates a clock with frequency $\frac{128}{660} f_{\text{mclk}}$. Note that the fraction is never explicitly evaluated and that the division is exact even if it cannot be represented in the system's precision. Indeed, it is straightforward to see that the period of the output clock is precisely $\frac{P}{inc}$ *mclk* cycles *on average*. In the previous example, the output clock period in terms of *mclk* cycles would look like $[6, 5, 5, 6, 5, 5, 5, 6, \ldots]$, such that the average is exactly $\frac{660}{128}$.

Constraining *inc* and P to be integers, the resolution of f_{out} is obtained by letting $inc = 1$ in (2.1), which yields

$$\Delta f_{\text{out}} = \frac{f_{\text{mclk}}}{P}. \tag{2.2}$$

Hence the frequency resolution can be increased for a fixed f_{mclk} by adding more bits to P. Naturally, if the value of $\frac{P}{inc}$ is not an integer, the period of the generated clock will not be constant: its peak-to-peak jitter will be one full *mclk* cycle, as illustrated in the previous example.

It is worth noting that NCOs are also at the heart of Direct Digital Synthesizers (DDS), where they are commonly referred to as a *phase accumulators* [18]. In the DDS literature, it is common to use $P = 2^B - 1$, where B is the wordlength of the accumulator, in order to maximize the resolution of f_{out} as well as simplify the hardware. However,

by noticing that NCOs are systems that perform the exact division of the integers *inc* and P [see (2.1)], allowing P to take on other values increases the set of achievable exact ratios.

2.3 The Period-Locked NCO

2.3.1 The closed-loop clock generator

The output of the NCO in Section 2.2 is free-running. However, it is usually required that the frequency of the generated clock be a multiple of the frequency of an incoming asynchronous *refclk* (f_{ref}), i.e., $f_{\text{out}} = N \cdot f_{\text{ref}}$, $N \in \mathbb{N}$. Even though the resolution of f_{out} can be made arbitrarily high by increasing the number of accumulator bits, the previous circuit remains *open-loop*, i.e., it cannot track *refclk*. Generally, a control loop is designed to adjust the *inc* value of the NCO to make the output clock (or its divided version) follow *refclk* [12, 14]. In contrast, the Period-Locked NCO tracks the reference clock by adjusting the other free parameter of the NCO, P, which can be done in a considerably simpler way.

To do so, solve (2.1) for P to get

$$P = \frac{f_{\text{mclk}} \cdot inc}{f_{\text{ref}} \cdot N} = \frac{p_{\text{ref}}}{p_{\text{mclk}}} \cdot \frac{inc}{N}, \tag{2.3}$$

where p_{ref} and p_{mclk} are the periods of the reference and master clocks, respectively. Notice that by letting $inc = N$, P becomes a ratio of periods, i.e., it represents the number of *mclk* cycles per *refclk* cycle, which can be measured using a simple counter. A remarkable consequence of using this approach is that it only takes one *refclk* cycle to obtain a correct measurement of P, i.e., this circuit locks onto the frequency of *refclk* in a single period (see Section 2.3.2). Note that this is only possible because P is not constrained to be $2^B - 1$, as is usually done in DDS or DPLL designs [18].

It is straightforward to see from (2.1) and (2.3) that the ratio between p_{ref} and p_{mclk} measured by the counter ($\hat{P}$) must track the value of P for this circuit to stay locked. Yet, $\hat{P}$ can only take integer values and therefore introduces quantization errors. It can be shown, however, that these errors are always compensated on the following *refclk* cycle and that, as long as every *mclk* cycle is accounted for, the average value of the $\hat{P}$ sequence converges to P.

Theorem: The average value of $\hat{P}$, computed as described above, is P. Proof: $\hat{P}$

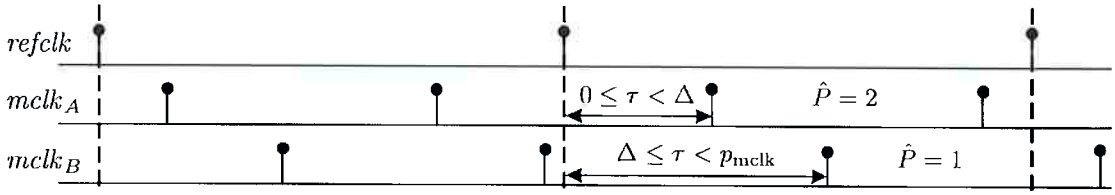

Figure 2: The values of $\hat{P}$ ($f_{\text{mclk,A}} = f_{\text{mclk,B}} = 1.7 f_{\text{ref}}$).

can take one of two values depending on the relative phase between *mclk* and *refclk*: $T_0 = \lfloor p_{\text{ref}}/p_{\text{mclk}} \rfloor$ and $T_1 = \lceil p_{\text{ref}}/p_{\text{mclk}} \rceil$, where $\lfloor \cdot \rfloor$ and $\lceil \cdot \rceil$ are the floor and ceil operators respectively. To see this, consider the delay $0 \leq \tau < p_{\text{mclk}}$ between the first *mclk* edge after a *refclk* edge (Figure 2). Define $\Delta = p_{\text{ref}} - T_0 p_{\text{mclk}} = (P - T_0) p_{\text{mclk}}$, the duration of the fractional part of P in time units. When $0 \leq \tau < \Delta$, we have $\hat{P} = T_1$; otherwise when $\Delta \leq \tau < p_{\text{mclk}}$ we have $\hat{P} = T_0$ cycles. The average of n consecutive values of $\hat{P}$ can therefore be expressed as

$$\bar{P}_n = \frac{1}{n} \left\lfloor n \cdot \frac{p_{\text{ref}}}{p_{\text{mclk}}} \right\rfloor + \frac{k_n}{n}, \tag{2.4}$$

where $k_n = 0$ or 1 depending on the initial phase between the clock signals. Notice that, since k_n is bounded, the second term in (2.4) vanishes as $n \to \infty$, whereas the first term converges to $p_{\text{ref}}/p_{\text{mclk}}$. Therefore, $\lim_{n\to\infty} \bar{P}_n = P$, which shows that the period-locked NCO is able to compensate the quantization errors introduced in $\hat{P}$ and lock onto the frequency of *refclk*. Note that the measurement of $\hat{P}$ is accurate to $\log_2(P)$ after only one reference cycle, so the generated clock can be considered *locked* right away - the theorem above is ensuring that no small error accumulates over time.

2.3.2 Lock acquisition and detection

Although the basic period-locked NCO introduced in Section 2.2 can track P variations, care must be taken when updating $\hat{P}$ to avoid accumulating errors that would cause the generated clock to drift with respect to *refclk*. Properly updating $\hat{P}$ plays the role of the feedback loop of the system ensuring that no error can accumulate. An intuitive solution is to ensure that each $\hat{P}$ value (i.e., *refclk* period) is used to output exactly N cycles. This is, indeed, one of the simplest definitions of lock: for each *refclk* cycle, produce exactly N output cycles. In [16, 17], the number of cycles generated between each *refclk* edge is tracked by a counter. However, to deal with possible jitter or changes in P, an *idle* time is inserted at the end of each N output cycle to guarantee that exactly N cycles are synthesized. Though it accommodates variations in *refclk*, the generated clock

becomes somewhat discontinuous and *bursty*. This is not desirable if the clock is to be used to drive a DAC, for example.

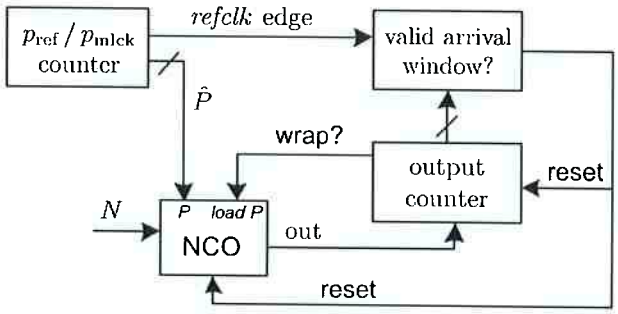

Figure 3: Period-Locked NCO with lock-detection logic.

The period-locked NCO addresses this issue by using a two-step locking mechanism, i.e., by decoupling the measurement of $\hat{P}$ and the update of the NCO (Figure 3). To do so, the modulo-N output counter used to ensure lock counts between mid-periods of *refclk* instead of between edges. This can easily be implemented by always resetting the output counter to $N/2$ (Figure 4). Notice that, for any given *refclk* pulse, the NCO generates exactly N cycles independent of the arrival time of the next pulse. Hence, this scheme can maintain lock under disturbances up to half a *refclk* cycle.

This process also embeds a lock detection mechanism into the period-locked NCO: suffices to check the value of the output counter upon a *refclk* edge. If it is outside a guard window around $N/2$, lock is assumed to be lost and the whole circuit is reset. The choice of window length depends on the desired jitter tolerance. A hardware-friendly option is to choose the range $[N/4, 3N/4]$ (constants that can be calculated with simple shift-and-adds), thus providing a jitter tolerance of $p_{ref}/2$ (Figure 4). This detection mechanism is a considerable simplification over those used in standard DPLL, whose designs typically rely on ad-hoc techniques and even Monte Carlo simulations [12].

The lock acquisition and detection process of the period-locked NCO is substantially different from that of standard feedback loops and is therefore worth illustrating. Figure 4 shows a worst case locking process, that can be split into the four sections:

1. The circuit is coming out of reset and $\hat{P} \ll P$ since a full *refclk* cycle has not yet passed. The slope of the output counter shows that the output clock is too fast.

2. A *refclk* edge is detected within a valid window of the output counter, so that the new $\hat{P}$ value is accepted (the circuit considers it is locked). The NCO gets updated with the new $\hat{P}$ when the output counter wraps-around, at which point the slope of

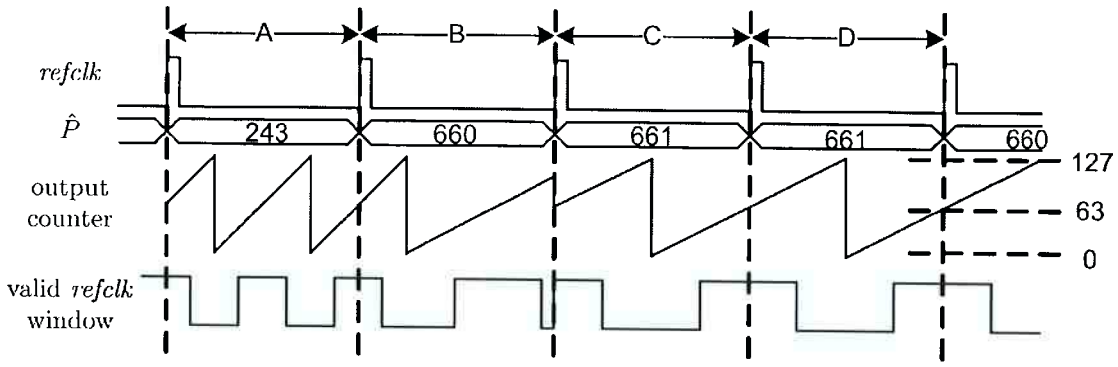

Figure 4: Period-Locked NCO with lock-detection logic: $f_{\text{mclk}} = 31.7\text{MHz}$, $f_{\text{ref}} = 48\text{kHz}$, $N = 128$, and valid P arrival window $[N/4, 3N/4]$.

the output counter changes. Notice that the output clock is already at the correct frequency.

3. Another edge is detected on *refclk*, but it is now outside a valid window. Loss of lock is detected, the NCO is reset, and the output counter is set to $N/2$. Since the NCO was already running with a correct $\hat{P}$ value, the output counter slope remains unchanged. The new $\hat{P}$ is passed to the NCO when the output counter wraps-around.

4. The circuit is now fully locked with a half *refclk* cycle of jitter tolerance as shown by the *valid* refclk *arrival* signal. The output counter now wraps close to the middle of the *refclk* cycle.

Note that in this worst case scenario the period-locked NCO tracked the *refclk* frequency in one cycle and only required two additional cycles to become fully locked. When operating with a known startup condition, such as coming out of reset, the circuit can achieve full lock in a single *refclk* period by ensuring that the NCO does not start before the arrival of the first valid $\hat{P}$ value.

2.4 Performance Analysis and Improvements

The performance of the basic period-locked NCO is limited by the precision of $\hat{P}$. Although it is guaranteed that the long time average of $\hat{P}$ tracks P, cycle to cycle variations translate to jitter and phase noise of the output clock. These effects can present themselves as random noise, harmonics, or *skirts* in the frequency domain, making the analytical study of this circuit intricate. Moreover, jitter on both *refclk* and *mclk* can affect the $\hat{P}$

sequence, further complicating the analysis. One simple yet insightful method to evaluate the quality of a clock signal is to consider its effects on the sampling process, i.e., as if it was used to drive an ADC/DAC. The reader is referred to [19] for a comprehensive analysis of this case.

Figure 5a shows the amplitude spectrum of a simulated 20kHz sine wave sampled at 6.144MHz using a period-locked NCO with the following parameters: $f_{\text{ref}} = 48\text{kHz}$, $N = 128$ and $f_{\text{mclk}} = 31.7\text{MHz}$. Assuming the signal will be ultimately downsampled by N, the SNR from 0 to 24kHz can be used as a good figure of merit. As expected, the performance in this case is quite poor, since $P = \frac{f_{\text{mclk}}}{f_{\text{ref}}}$ is measured to a precision close to 9 bits $[\log_2(P)]$. This precision can be treated as jitter in the derivations presented in [19] to provide an analytical evaluation of the sampling system. For comparison, the idle time scheme from [17] is shown in Figure 5b for an idle period of $0.62\mu\text{s}$, about 3% of the *refclk* period. Notice that since this idle time is periodic it has the effect of substantially raising the harmonics of the fundamental, thus worsening the SNR.

The period-locked NCO performance can be improved by using a faster *mclk* providing both a better estimate of the ratio P and a reduction in the peak-to-peak jitter attainable. A fast *mclk*, however, is not always available and may even be undesirable, as it would increase power consumption and sensitivity to the *refclk* jitter. This issue is addressed in the next sections by providing low complexity solutions that significantly improve the performance of the basic period-locked NCO without relying on a faster *mclk*.

2.4.1 Filtering the $\hat{P}$ sequence

One straightforward yet effective improvement to the basic period-locked NCO is to use linear filtering to increase the precision of the P measurements. Not only does this reduce the low frequency phase noise of the output clock, but it also increases immunity against *refclk* jitter. Even though this filter plays a similar role to that of the loop filter of a standard DPLL, its design is significantly simplified as it can be approached independently from the rest of the system. In fact, this filter is not embedded in a typical feedback loop as in standard DPLLs. Moreover, simple low order filters are usually sufficient, e.g., a moving average or a CIC filter [20].

Not every filter, however, is suited for this application. Indeed, recall that the $\hat{P}$ sequence must on average track the exact value of P for the system to stay locked. In other words, the filter must be DC-accurate. This was guaranteed by design when using a counter as described earlier, but now care must be taken when designing the filter so

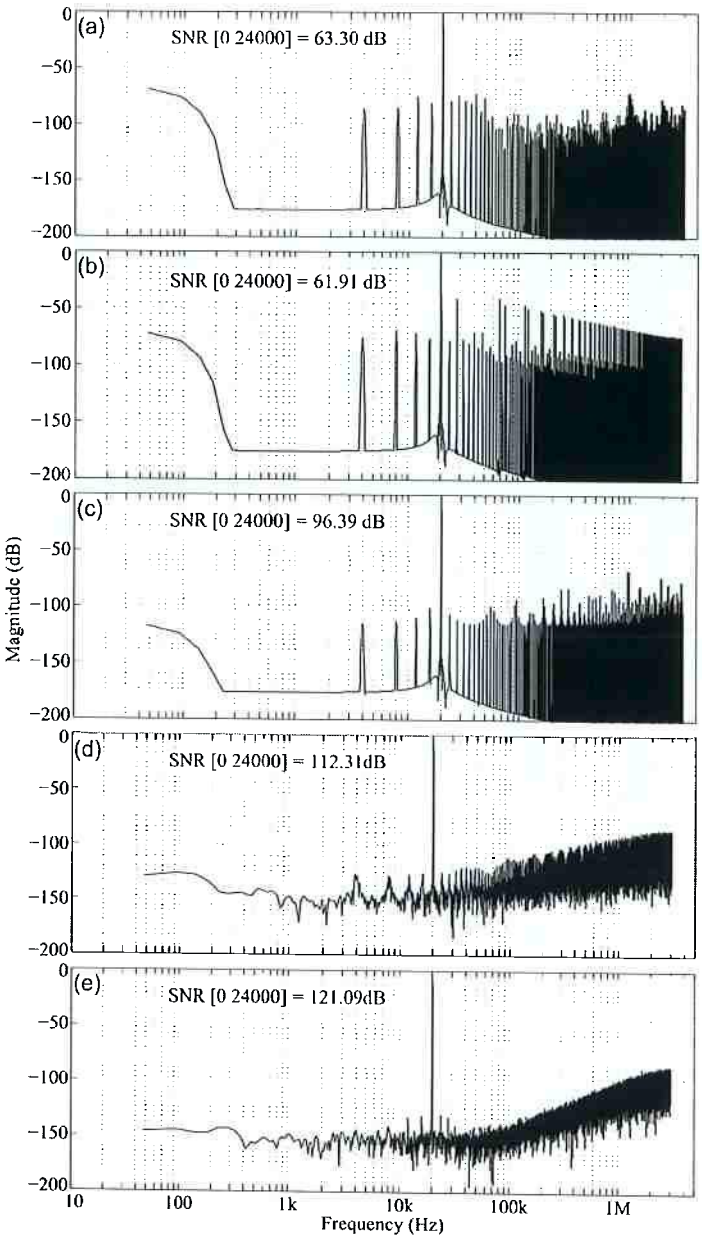

Figure 5: Spectrum of 20kHz sine wave sampled at 6.144MHz using $f_{\text{ref}} = 48\text{kHz}$, $f_{\text{mclk}} = 31.7\text{MHz}$, and $N = 128$: (a) basic period-locked NCO; (b) system from [17] with idle period of 20 *mclk* cycles ($\approx 0.03\,p_{\text{ref}}$); (c) period-locked NCO with leaky integrator filtering; (d) period-locked NCO with leaky integrator filtering and 2^{nd} order noise shaping; (e) period-locked NCO with leaky integrator filtering and 3^{rd} order noise shaping;

that errors such as bit-truncation of the output, do not accumulate over time. Error feedback can be used to circumvent this problem [21]. Take, for instance, the 1^{st} order lowpass IIR filter sometimes called *leaky integrator* (Figure 6) where efficient hardware implementation is possible by constraining α to be a power-of-two. The quantizer is

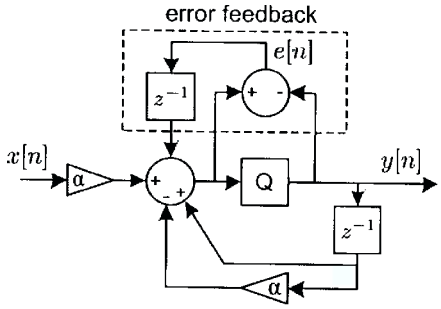

Figure 6: Leaky integrator with error feedback.

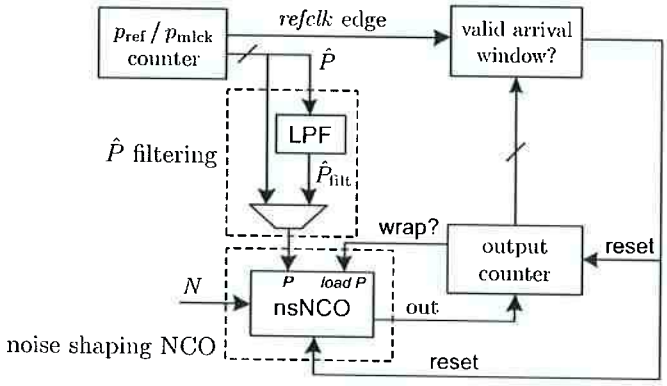

Figure 7: Period-locked NCO with low-pass filtering of $\hat{P}$.

required to avoid infinite bit-growth around the loop. Modeling it as a noise source $e[n]$, its transfer function to the output reads

$$\frac{Y(z)}{E(z)} = \frac{1}{1-(1-\alpha)z^{-1}}, \tag{2.5}$$

which is obviously not 0 at DC. To avoid having this error build up over time, it is fed-back into the system as in the dashed section of Figure 6. The result is that $e[n]$ is high-pass filtered by $1-z^{-1}$, thus adding a zero at DC. The number of extra bits required after the quantizer is a design parameter better evaluated through simulations.

The diagram of the period-locked NCO with low-pass filtering of $\hat{P}$ is shown in Figure 7. Since the output of the low-pass filter takes some time to settle, the value of $\hat{P}$ is initially used directly so as not to affect the lock time. The result of using this improved period-locked NCO is illustrated in Figure 5c. The leaky integrator uses $\alpha = 2^{-10}$ (-3dB corner at 10Hz) and 10 extra bits after the quantizer. Notice that, although the minimum achievable jitter in the output clock remains limited to one *mclk* cycle peak-to-peak, the performance of the system has improved considerably. This is due to the fact that the $\hat{P}$ used by the NCO is now much more accurate and stable, essentially reducing low frequency phase noise caused by the coarse update of the ratio at each *refclk* cycle.

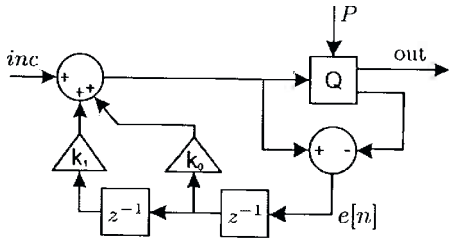

Figure 8: NCO with 2nd order noise shaping ($k_0 = 2$, $k_1 = -1$).

2.4.2 Noise shaping the NCO error

The spectrum in Figure 5c still displays undesirable high amplitude spikes and spurs. To mitigate this issue, notice that the NCO in Figure 1b can be seen as a first order Σ-Δ modulator with a DC input *inc*. A valid question is therefore: *can the modulator order be increased to improve performance while keeping the same structural advantages of the NCO*? Indeed it can and this is easily seen by comparing the NCO to a first order Σ-Δ modulator in the error-feedback topology [22]. In fact, this is one of the spur-reduction techniques used in the DDS literature, where it is referred to as *phase-error feedback* [18]. Note that the theory is similar to the error-feedback filtering from Section 2.4.1, except the shaping function is applied to the phase error instead of the amplitude error. An implementation of the 2nd order noise shaped NCO (nsNCO) from [18] is shown in Figure 8.

The performance of the complete system shown in Figure 7 is evaluated in Figure 5d and Figure 5e for 2nd and 3rd order nsNCOs. The high-pass filtering of the phase error of the generated clock drastically reduces the spurs and tones, as predicted by the analysis of the jitter effect on sampling from [19].

2.5 Comparison with Standard DPLLs

The most basic DPLL structure is presented in Figure 9. Although an astonishing number of variations of this loop have been put forward, their fundamental functioning remains the same [12]. Similar to DDS, virtually all DPLL architectures fix $P = 2^B - 1$ in the NCO. Thus, in order for the steady-state output frequency to be $f_{\text{out}} = N f_{\text{ref}}$, the value of *inc* must be, as per (2.1),

$$inc = N \cdot P \cdot \frac{f_{\text{ref}}}{f_{\text{mclk}}} = N \cdot P \cdot \frac{p_{\text{mclk}}}{p_{\text{ref}}}. \tag{2.6}$$

This value for *inc* is now difficult to find; there is no straightforward relationship with the available signals at hand and there is no simple circuit that can easily provide an accurate estimate. In fact, this value of *inc* is based on the ratio of the frequencies of the *refclk* and *mclk* signal, as opposed to their periods as proposed in the period-locked NCO. Only the period ratio is directly measurable from a simple counter; whereas frequency ratio requires a division to calculate the inverse of the period ratio.

This difficulty in evaluating the increment word of the NCO leads to lock acquisition issues in traditional DPLLs. Though various methods have been put forward to reduce their lock time, e.g., [23,24], these usually address the lack of a direct relationship between *inc* and the incoming clock signals by implicitly evaluating a division of their period measurements. On the other hand, lock acquisition and detection mechanism are embedded in the period-locked NCO, which can lock onto the *refclk* frequency in a single cycle.

To achieve good performance, both the period-lock NCO and traditional DPLLs use a loop filter for jitter reduction, leading to a well-known trade-off between tracking performance and lock time. The period-locked NCO has the same fundamental tradeoff, however the jitter attenuation filter is not deeply embedded in the feedback loop, making the design much easier. Furthermore, the Phase-Frequency Detector, a circuit that has its roots in analog design and is difficult to design, model and analyze in the z-domain as discussed in [12,25,26] is completely avoided.

Finally, noise shaping techniques similar to those in Section 2.4.2 can be integrated in the classical DPLL loop, as in [14]. Yet, it is not as straightforward as in the period-locked NCO case. Indeed, the period-locked NCO is already based on the incoming clock periods, so that noise shaping is a simple extension, whereas standard DPLLs need to deal with both frequency and period.

It is important to note that either DPLLs or period-locked NCOs can achieve the same performance if designed under the same specifications. In fact, given that the output clock edges need to align to *mclk* edges, both designs are limited to one *mclk* cycle of jitter on the generated clock. Nevertheless, the period-locked NCO is easier to design as it decouples lock time and jitter rejection trade-offs.

2.6 Conclusion

A novel circuit for all-digital clock generation was introduced using an approach that substantially differs from classical DPLLs. By adjusting the threshold value of the NCO

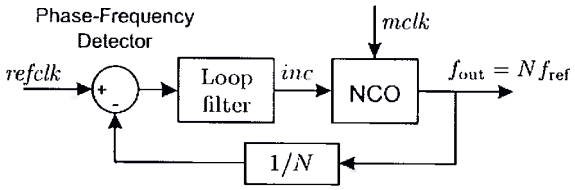

Figure 9: General topology of a DPLL.

instead of its increment value, a relationship that can be accurately measured with a simple counter arises. Jitter reduction, which requires long time-constant filters are now implemented *outside* the main loop, freeing the designer from usual closed-loop design trade-offs (e.g., lock time/disturbance rejection). Noise shaping techniques can also be integrated seamlessly in the period-locked NCO, as in open-loop DDS designs. High quality clock signals for demanding applications such as audio ADC/DAC can be generated using this low complexity and simple to design hardware.

3 AN EFFICIENT FILTERING STRUCTURE FOR SPLINE INTERPOLATION AND DECIMATION

3.1 Introduction

Fractional delay (FD) filtering is a technique to evaluate a discrete-time signal at arbitrary—possibly non-integer multiple of the sampling rate—delays. FD filters are at the heart of many digital signal processing solutions such as asynchronous sample rate conversion (ASRC) [27], timing recovery in all-digital receivers for software-defined radio [28], and wave field synthesis [29]. A thorough review of FD filtering and applications can be found in [30].

Several structures have been presented in the literature to implement different polynomial FD filters. One of the most celebrated is the *Farrow structure* that can be used to efficiently implement any polynomial response [6]. Many improvements and modifications of this structure are available, most notably the *modified Farrow structure* that exploits coefficient symmetry to reduce the number of multipliers. Further optimizations are possible by constraining the response to a single class of polynomials. For instance, when considering only Lagrange polynomials, the *Newton structure* from [31–33] is by far the least computationally expensive. Nevertheless, limitations in the frequency response of Lagrange FD filters entail the use of higher order polynomials to meet requirements, leading designers to use polynomials with better characteristics such as splines. For this reason, the structure developed in this work aims to combine the performance of spline FD filters with the reduced complexity of the Newton structure.

Before proceeding, note that any interpolation structures can be used for decimation (and vice-versa) by means of network transposition [2, 33]. All structures described in this part are therefore suitable for both interpolation and decimation. Thus, due to space constraints and without loss of generality, only interpolation is discussed in the sequel.

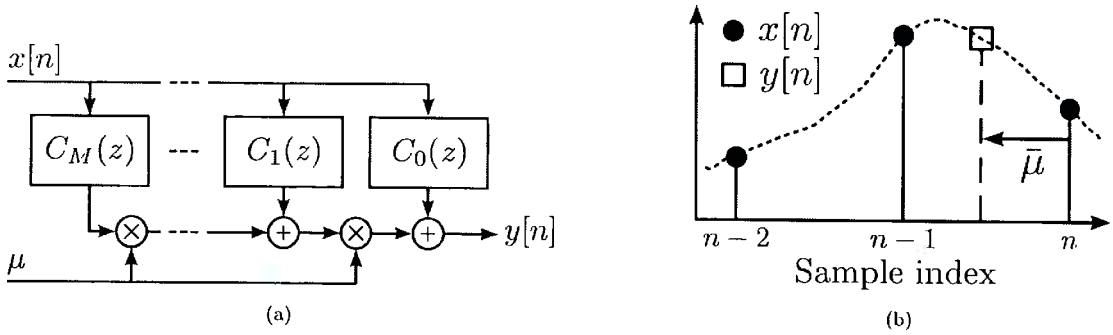

Figure 10: The Farrow structure: (a) block diagram; (b) intersample position (μ)

3.2 The Farrow structure

The Farrow structure (Fig. 10a) was introduced in [6] as a general implementation for arbitrary polynomial FD filters. Its transfer function can be written as

$$H(z,\mu) = \sum_{m=0}^{M} \left(\sum_{n=0}^{N} c_{mn}\, z^{-n} \right) \bar{\mu}^m, \tag{3.1}$$

where M is the polynomial order and N is the subfilter order—usually, $M = N$. The transfer function (3.1) is also parametrized by $\bar{\mu} \in [-1, 0)$, the intersample position, which controls the fractional delay of the filter as illustrated in Fig. 10b. In fact, one of the most important features of the Farrow structure is that it can implement variable delays. The $\{c_{mn}\}$ are coefficients of the filters $C_m(z) = \sum_{n=0}^{N} c_{mn} z^{-n}$ (see Fig. 10a) that uniquely define the polynomial being implemented. For clarity, they are typically collected in a matrix $\boldsymbol{C}$ that can be evaluated for classical polynomials such as Lagrange and Hermite using techniques from [34].

The modified Farrow structure reduces complexity using instead of $\bar{\mu}$ the transformed value $\mu = \bar{\mu} + 0.5 \in [-0.5, 0.5)$, taking advantage of the resulting symmetry in the $C_m(z)$ coefficients [35] (note the symmetry in the rows of (3.3) further ahead). However, the modified Farrow structure still requires $\mathcal{O}(M^2)$ multiplications for $M = N$.

3.3 The Newton structure

The Newton structure (Fig. 11) introduced in [31] and refined in [32, 33] is based on Newton's backward difference formula, an efficient algorithm for Lagrange polynomial interpolation. Of all optimized implementations of the Lagrange polynomial surveyed in [36], the Newton structure has the lowest complexity of only $\mathcal{O}(M)$ operations. How-

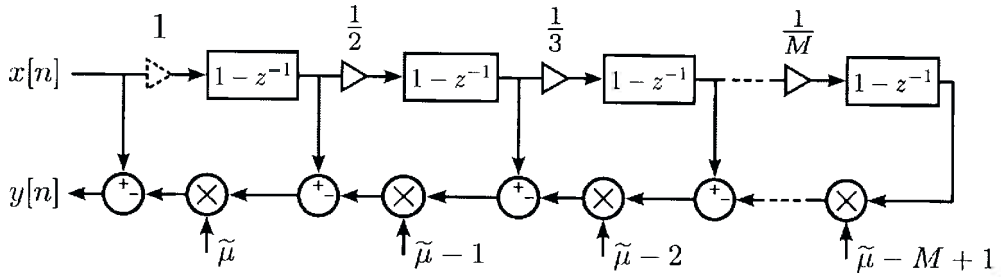

Figure 11: The Newton structure for Lagrange interpolation

ever, it is restricted to the Lagrange polynomial, so that the only way to improve its frequency response is by increasing the order M, undermining the computational advantages and adding delay [33].

3.4 Farrow state-space transformations and the Newton structure

Since the Newton structure implements a Lagrange FD filter, it is clearly equivalent to a Farrow implementation of that same polynomial. However, the Newton structure has only been motivated so far as a direct implementation of Newton's backward difference formula [31–33]. To formalize the relation between these two structures, this section shows how the Newton structure can be derived directly from a Farrow-Lagrange filter. The motivation is that the same steps might lead to efficient structures when applied to other polynomials in the Farrow structure. For the sake of clarity, the following derivations are carried with $M = N = 3$, although they are valid for arbitrary values. Furthermore, 3rd order polynomials are widely used and often times considered to be offering very good performance and complexity tradeoffs [36]. When higher order polynomial filters are required to meet specifications, it is typical to use a cascade of an integer rate conversion filter which can be very efficiently implemented, followed by a polynomial filter of lower order.

First, express the Farrow transfer function (3.1) in matrix form as

$$H_{\text{Farrow}}(z,\mu) = \boldsymbol{\mu}^T \boldsymbol{C} \boldsymbol{z}, \tag{3.2}$$

where $\boldsymbol{\mu} = [\ 1 \quad \mu \quad \mu^2 \quad \mu^3\ ]^T$; $\boldsymbol{z} = [\ 1 \quad z^{-1} \quad z^{-2} \quad z^{-3}\ ]^T$; and $\boldsymbol{C}$ is chosen to implement

a Lagrange polynomial [35]:

$$\boldsymbol{C}_{\text{Lagrange}} = \begin{bmatrix} -3 & 27 & 27 & -3 \\ 2 & -54 & 54 & -2 \\ 12 & -12 & -12 & 12 \\ -8 & 24 & -24 & 8 \end{bmatrix}. \tag{3.3}$$

Recall that $\mu \in [-0.5, 0.5)$. Then, for $\widetilde{\mu} = \mu - 1.5 = \bar{\mu} - 1$, the Newton structure in Fig. 11 can be written in the same form as (3.2), yielding

$$H_{\text{Newton}}(z, \widetilde{\mu}) = \widetilde{\boldsymbol{\mu}}^T \widetilde{\boldsymbol{C}} \widetilde{\boldsymbol{z}}, \tag{3.4}$$

where $\widetilde{\boldsymbol{\mu}} = [\ 1 \quad \widetilde{\mu} \quad \widetilde{\mu}(\widetilde{\mu}-1) \quad \widetilde{\mu}(\widetilde{\mu}-1)(\widetilde{\mu}-2)\ ]^T$, $\widetilde{\mu} \in [-2, -1)$; $\widetilde{\boldsymbol{C}}$ is a diagonal matrix whose elements are $\{1, -1, 1/2, -1/6\}$; and $\widetilde{\boldsymbol{z}} = [\ 1 \quad 1 - z^{-1} \quad (1 - z^{-1})^2 \quad (1 - z^{-1})^3\ ]^T$.

To obtain (3.4) from (3.2), suffices to find two transformations $\boldsymbol{T}_\mu$ and $\boldsymbol{T}_z$ such that $\widetilde{\boldsymbol{\mu}} = \boldsymbol{T}_\mu \boldsymbol{\mu}$, $\widetilde{\boldsymbol{z}} = \boldsymbol{T}_z \boldsymbol{z}$, and $\widetilde{\boldsymbol{C}} = \boldsymbol{T}_\mu^{-T} \boldsymbol{C} \boldsymbol{T}_z^{-1}$, with $\boldsymbol{A}^{-T} = (\boldsymbol{A}^T)^{-1}$, for invertible $\boldsymbol{A}$. Given these transformations, one would have

$$\begin{aligned} H_{\text{Farrow}}(z, \mu) &= \boldsymbol{\mu}^T \boldsymbol{C} \boldsymbol{z} = \boldsymbol{\mu}^T (\boldsymbol{T}_\mu^T \boldsymbol{T}_\mu^{-T}) \boldsymbol{C} (\boldsymbol{T}_z^{-1} \boldsymbol{T}_z) \boldsymbol{z} \\ &= (\boldsymbol{T}_\mu \boldsymbol{\mu})^T (\boldsymbol{T}_\mu^{-T} \boldsymbol{C} \boldsymbol{T}_z^{-1}) (\boldsymbol{T}_z \boldsymbol{z}) = \widetilde{\boldsymbol{\mu}}^T \widetilde{\boldsymbol{C}} \widetilde{\boldsymbol{z}} = H_{\text{Newton}}(z, \widetilde{\mu}). \end{aligned}$$

These transformations can be derived in three steps: (i) find $\boldsymbol{T}_\mu$; (ii) find $\boldsymbol{T}_z$; (iii) check that $\boldsymbol{T}_\mu$ and $\boldsymbol{T}_z$ indeed transform $\boldsymbol{C}$ into $\widetilde{\boldsymbol{C}}$.

(i) The fractional delay transformation is derived in two parts. First, the fractional interval range is made identical among structures. Changes in the range of the intersample position are common and have been used, for instance, as a means to reduce complexity in the derivation of modified Farrow structures [35,37]. Since $\mu \in [-0.5, 0.5)$ and $\widetilde{\mu} \in [-2, -1)$, $\boldsymbol{T}'_\mu$ is obtained from the relation $\widetilde{\mu} = \mu - 1.5$ as

$$\begin{bmatrix} 1 \\ \widetilde{\mu} \\ \widetilde{\mu}^2 \\ \widetilde{\mu}^3 \end{bmatrix} = \frac{1}{8} \begin{bmatrix} 8 & 0 & 0 & 0 \\ -12 & 8 & 0 & 0 \\ 18 & -24 & 8 & 0 \\ -27 & 54 & -36 & 8 \end{bmatrix} \boldsymbol{\mu}. \tag{3.5}$$

Notice that the n-th row of $\boldsymbol{T}'_\mu$ collects the coefficients of the polynomial $(\mu - 1.5)^{n-1}$. Second, the vector on the left-hand side of (3.5) must become $\widetilde{\boldsymbol{\mu}}$, where each element is a polynomial of $\widetilde{\mu}$. Once again, the transformation is based on the coefficients of

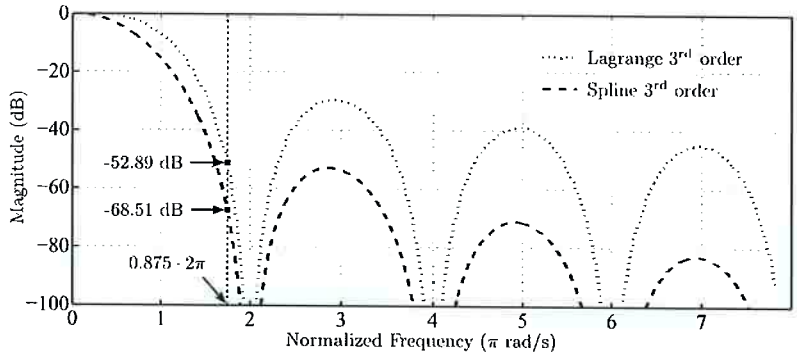

Figure 12: Frequency response of 3rd order Lagrange and spline filters

these polynomials as in

$$\boldsymbol{T}''_{\mu} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 2 & -3 & 1 \end{bmatrix}. \tag{3.6}$$

Finally, $\widetilde{\boldsymbol{\mu}} = \boldsymbol{T}''_{\mu}\boldsymbol{T}'_{\mu}\boldsymbol{\mu}$, so that the intersample position transformation is chosen as $\boldsymbol{T}_{\mu} = \boldsymbol{T}''_{\mu}\boldsymbol{T}'_{\mu}$.

(ii) The filter basis z^{-1} of the Farrow structure must be changed into the differentiator basis $1 - z^{-1}$ used by the Newton structure. This can be done using the matrix

$$\boldsymbol{T}_{z} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 1 & -3 & 3 & -1 \end{bmatrix}, \tag{3.7}$$

whose n-th row represents the coefficients of $(1 - z^{-1})^{n-1}$.

(iii) The matrices $\boldsymbol{T}_{\mu}$ and $\boldsymbol{T}_{z}$ derived in items (i) and (ii) are designed to perform the transformations $\boldsymbol{\mu} \to \widetilde{\boldsymbol{\mu}}$ and $\boldsymbol{z} \to \widetilde{\boldsymbol{z}}$, respectively. It is straightforward to see by direct evaluation that they also fulfill $\widetilde{\boldsymbol{C}} = \boldsymbol{T}_{\mu}^{-T}\boldsymbol{C}\boldsymbol{T}_{z}^{-1}$. By interpreting these transformations as changes in the bases of the interpolation operator $\boldsymbol{C}$, the low complexity of the Newton structure is explained by the fact that it uses bases in which the coefficient matrix is diagonal, reducing the number of operations required to evaluate the weighted inner product in (3.2).

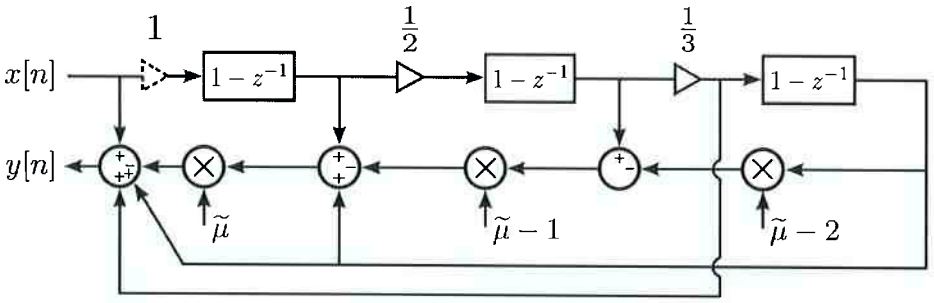

Figure 13: Novel Newton-like spline interpolation structure

3.5 Novel structure for spline interpolation

As mentioned before, the main disadvantage of the Newton structure is that it can only implement the Lagrange polynomial, which has poor frequency response. It is well known that splines have better properties for signal processing applications and converge to the ideal interpolator as their order goes to infinity [38]. Indeed, Fig. 12 compares the frequency response of 3rd order Lagrange and spline interpolators. It shows the latter displays an extra 16dB of attenuation at the $0.875 \cdot 2\pi$ normalized band edge, which corresponds to the worst-case image attenuation when interpolating a signal oversampled by 4. Spline polynomials can naturally be implemented using the Farrow structure by deriving $\boldsymbol{C}$ in (3.2) similar to [39]:

$$\boldsymbol{C}_{\text{spline}} = \begin{bmatrix} 1 & 23 & 23 & 1 \\ -6 & -30 & 30 & 6 \\ 12 & -12 & -12 & 12 \\ -8 & -4 & -24 & 8 \end{bmatrix}. \tag{3.8}$$

The proposed structure is derived by applying the same transformations from the previous section to $\boldsymbol{C}_{\text{spline}}$, yielding a Newton-like structure for spline interpolation. Explicitly,

$$\boldsymbol{T}_{\mu}^{-T}\boldsymbol{C}_{\text{spline}}\boldsymbol{T}_{z}^{-1} = \begin{bmatrix} 1 & 0 & \frac{1}{6} & \frac{1}{6} \\ 0 & -1 & 0 & \frac{1}{6} \\ 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & -\frac{1}{6} \end{bmatrix} = \boldsymbol{C}_{\text{LCN}}. \tag{3.9}$$

Notice that $\boldsymbol{C}_{\text{LCN}}$ is quasi-diagonal and that its coefficients have trivial hardware implementations. The full structure is depicted in Fig. 13 and its computational complexity is compared in Table 1 to that of the modified Farrow (suitable for Lagrange and spline)

and the Newton structure (Lagrange only). Only three additional adders are necessary to turn a 3rd order Lagrange-only Newton structure into a spline interpolation structure that is largely simpler than its Farrow counterpart.

Table 1: Computational complexity

	Modified Farrow		Newton		Proposed	
	Add	Mult	Add	Mult	Add	Mult
Lagrange	11	11	6	4	–	–
Spline	11	11	–	–	9	4

3.6 Conclusion

A novel structure for spline interpolation/decimation was proposed. First, the Newton structure was derived using a series of transformations of the Farrow-Lagrange structure. These transformations were applied to the spline coefficient matrix yielding a novel Newton-like structure for spline interpolation. The transformations were applied to 3rd order polynomials only, but more general results using this matrix formulation as well as direct optimization of the coefficients in the new structure will be addressed in future works.

PART II

INTEGER SAMPLE RATE CONVERSION

4 ON THE FIXED-POINT IMPLEMENTATION OF REDUCED STATE-SPACE CIC FILTERS

4.1 Introduction

Cascaded-integrator-comb (CIC) filters introduced in [40] are used in an array of applications where efficient sampling rate conversion is required. Their multiplier-less structure and low hardware complexity makes them particularly attractive when used as the front end decimation filter of Σ-Δ A/D converters, reducing the sampling rate before entering a more expensive second stage filter. Their regular structure, coupled with a trivial design procedure makes them the go-to choice for a myriad of applications where data needs to be averaged. For many applications, their relatively poor frequency response, namely narrow stopband and large passband droop, has been the main challenge to overcome. Hence a lot of attention has been dedicated to improve their frequency response using techniques such as polyphase decomposition [41], filter sharpening [8, 42] and zero-rotation [43].

However, there are many cases where the frequency response is adequate, but reducing the hardware area of a given filter is paramount for cost-effective solutions. This problem was tackled in [10] and further generalized in [44, 45] but the potential area savings were never quantified and the fixed point implementation not investigated. It is shown here that for some combinations of filter order and decimation ratio, the generalized structure of [44] can yield silicon area savings of up to 50% compared to a traditional CIC filter implementation, while being a bit-accurate design.

4.2 The classical CIC filter

The recursive CIC filter structure shown in Figure 14 was first introduced in [40] and is the simplest and most common way to implement the following transfer function

followed by a decimation by N

$$H(z) = \left(\sum_{i=0}^{N-1} z^{-i}\right)^R = \left(\frac{1-z^{-N}}{1-z^{-1}}\right)^R, \tag{4.1}$$

where R is the filter order. CIC filters are extensively covered in the literature and the reader is referred to [40,46] for more details. The main particularity of CIC filters is that they exhibit exact pole-zero cancellation, such that the recursive structure is implementing exactly the underlying FIR filter described on the left side of (4.1). To ensure stability, the wordlength at all nodes should be made equal to $B_{in} + \lceil R \log_2(N) \rceil$, where B_{in} is the input wordlength [40]. One way to understand how this works is that the filter needs enough bits at all nodes to accommodate for the complete gain of the underlying FIR filter. The modulo properties of two's complement arithmetic ensures that the final result will belong to the right modulo quadrant.

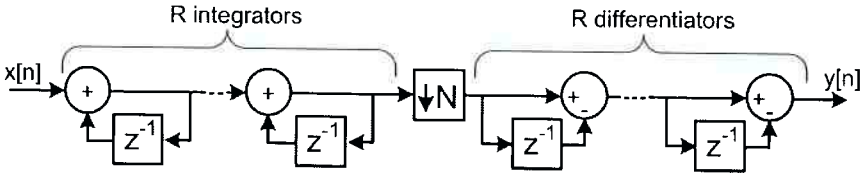

Figure 14: Standard recursive structure for CIC filters

4.3 Elimination of the first differentiator

It is well known that a the first differentiator of a CIC filter is not necessary as it is simply removing the initial condition at the preceding integrator N cycles before. This is obvious when looking at a first order filter: the integrator output at time N is the running sum of the input plus the initial condition, so the differentiator simply removes the initial condition. The pair thus works as an integrate&dump circuit. This is always the case for the innermost integrator/differentiator pair of CIC filters of any order, and is depicted in Figure 15 for a 2nd order filter. The reset operation of the integrator is virtually free in hardware so this optimization saves $\frac{1}{2R}$ the area of the corresponding CIC filter.

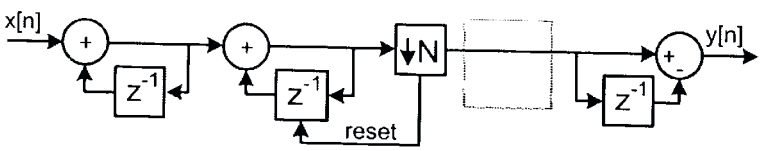

Figure 15: Removal of first differentiator using integrate&dump

4.4 Higher order integrate and dump

The idea of removing the need for differentiators by altering the state of the integrators was generalized in [44]. The task is substantially more complicated for higher order filters and the technique is based on the *Raising Procedure* presented in [47] where minimal realization of periodic systems is investigated. In the case of CIC filters, the minimal state-space representation yields a system where all the differentiators are removed, at the expense of a set of time-varying coefficient multipliers feeding back the output to each integrator. The resulting system is shown in Figure 16 for a 3rd order filter and can be extended to any order by feeding back the output to each integrator through a time-varying multiplier. Those systems can be intuitively thought of as higher order integrate&dump circuits, meaning that the *dump* operation is done over multiple periods of the output rate for $R-1$ integrators. The c_i coefficients for filters of order $1-4$ are derived in [48] and repeated in Table 2 for convenience.

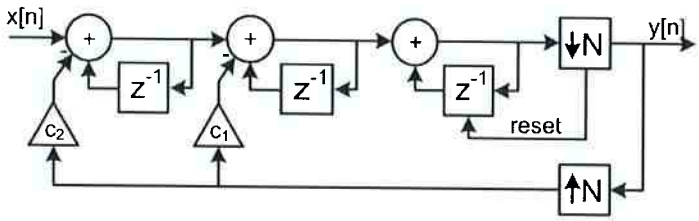

Figure 16: 3rd order reduced state-space CIC filter

CIC order (R)	c_0	c_1	c_2	c_{R-1}
1	1	–	–	–
2	1	$\frac{1}{N}$	–	–
3	1	$\frac{3N-1}{2N^2}$	$\frac{1}{N^2}$	–
4	1	$\frac{11N^2-6N+1}{6N^3}$	$\frac{2N-1}{N^3}$	$\frac{1}{N^3}$

Table 2: Feedback coefficients of reduced state-space CIC filters

A few comments are in order regarding this structure:

(i) All the states of the differentiators are completely gone, whereas the subtractors were moved to the input of each integrator as part of the feedback path.

(ii) The feedback branches only take non-zero values every N cycles, limiting switching activity in the multipliers and feedback subtractors.

(iii) The c_i coefficients are dependent on N, making the structure less flexible than its recursive counterpart if multiple decimation ratios are to be supported.

(iv) The c_0 coefficient is always 1 and is best implemented as an integrate&dump circuit as previously discussed.

(v) Up to 3rd order, all other c_i coefficients are realizable with few non-zero CSD digits when N is a power of 2.

(vi) Any decimation ratio N leading to c_i not *exactly* representable in two's complement arithmetic cannot be implemented in this structure.

(vii) Even if implementing an exact FIR response, this structure will never come out of a bad state even if given enough time (as opposed to an FIR filter, or even the classical recursive structure). This is a risk that designers need to take into account. Examples of entering a bad state would be an incorrect reset condition, or the result of the release of alpha particles, altering the states of the filter.

(viii) This structure does *not* rely on the modulo arithmetic trick of CIC filters: integrators *never* overflow if proper wordlength are selected, which is a necessary condition for proper operation.

That being said, the reduced state-space structure is a good fit for applications with a tight area constraint where the decimation ratio N is fixed. Power of two ratios are also preferred since the resulting c_i coefficients are easier to implement. It should be noted that many applications fall in this category so those restrictions should not be considered as a fatal setback.

4.5 Fixed point implementation for guaranteed stability

The reduced state-space structure derived in [44] is implementing exactly the same transfer function as its traditional recursive CIC counterpart, which is finite in duration as shown in (4.1). As mentioned before, exact pole-zero cancellation is required so care must thus be taken during the fixed point realization of either system as exact arithmetic has to be used. The problem is well known for CIC filters and is dealt with by ensuring that every signal node has enough bits to accommodate for the maximum gain of the filter (N^R) applied to the input width. Overflows in the integrators can then be ignored if

two's complement arithmetic is used, see [40,46] for a thorough explanation. The problem is slightly different in the reduced state-space structure as the multipliers will generate fractional bits that have to be kept. At first sight, it might seem that the reduced state-space structure requires truncation along the feedback path to avoid infinite bit-growth, but this is not necessary since the impulse response being implemented is the *same* as the one from the standard CIC filter, where all coefficients are integers. Indeed, the feedback coefficients were chosen so that the output of the filter *never* has non-zero fractional bits at each multiple of N cycles.

4.6 Wordlength optimization of each integrator stage

The simplest way to choose the wordlength of the reduced state-space filter is to take a similar approach to the classical CIC filter: add enough integer bits to accommodate for the worst case filter gain, while also keeping all fractional bits generated by the feedback coefficients. This is a conservative approach since only the last integrator is subject to the worst case gain; all other integrators are damped by a feedback coefficient smaller than one, so that the signal gain from the input is guaranteed to be less than the overall filter gain. This is similar to the design of sigma-delta modulators, although in this case bit exactness is still achieved. Using the techniques of [44], the *raised* state space matrices A, B, C and D of the filter can be found and the gain from input to each integrator, G_i, can be calculated with

$$\begin{bmatrix} G_1 \\ G_2 \\ \vdots \\ G_R \end{bmatrix} = \begin{bmatrix} B & AB & A^2B & \dots & A^{R-1}B \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}, \tag{4.2}$$

where the last integrator gain, G_R, is the total gain of the filter. The number of extra bits required at each integrator stage is then $\lceil \log_2(G_i) \rceil$. Figure 17 shows the optimal wordlength of each node for a 3$^{\text{rd}}$ order filter with N=32 and a 8 bit input. Using (4.2), the maximum gain for each integrator is found to be 63, 1845.5 and 32768 respectively, each requiring 6, 11, and 15 MSB bits. The number of fractional bits required are directly calculated from the feedback c_i coefficient shown in Table 2.

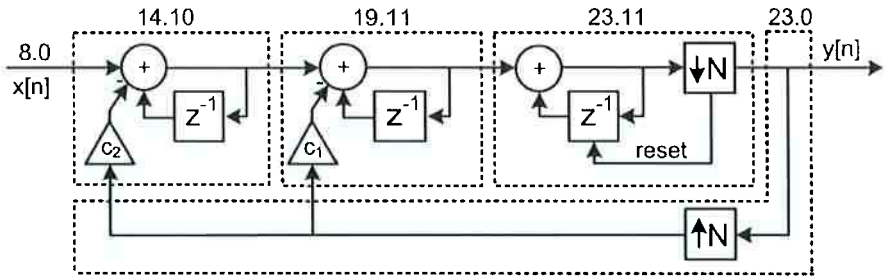

Figure 17: Reduced state-space filter with optimized wordlength, R=3, N=32

4.7 Area comparison

Many different filters were designed in both the standard recursive structure and reduced state-space. Clearly, the coefficient complexity of the reduced state-space structure goes up rapidly with the order so only filters of order 2 and 3 were considered, each for two relatively common decimation ratios of N=16 and 32. The input wordlength was set to 8 bits in all cases. The designs were coded in Verilog, verified for bit-exactness with the recursive structure counterpart, and synthesized in 0.18um using Synopsys Design Compiler. Results are shown in Table 3.

Order	N	Recursive CIC			Reduced State-Space		
		Total	Logic	Flop	Total	Logic	Flop
2	16	8947	3993	4917	5380	3044	2335
	32	10025	4456	5531	6107	3464	2643
3	16	15691	7048	8605	14071	9338	4732
	32	18057	8124	9895	16411	10879	5531

Table 3: Area comparison for 0.18um (square microns)

Area savings close to 50% are achieved for 2nd order filters, while both 3rd order filters are about 10% smaller. The reported area was broken down in *Flop* and *Combinatorial* logic since the latter can be potentially reused across channels if a higher frequency clock is available [49], while the *Flop* elements of each channel are needed. The ratio of combinatorial logic area over total area is around 50% for the typical recursive structure, while closer to 65% for the reduced state-space structure, enabling the design of extremely compact multichannel systems.

4.8 Conclusion

The design and fixed-point implementation of reduced state-space CIC filters first introduced in [48] was presented. It is shown that substantial area savings are possible when compared to the traditional recursive CIC filter structure. Those savings come at the price of flexibility: the dependency of the feedback coefficients on the decimation ratio N complicate the design if N must be made programmable. The complexity of the coefficients also increases quickly with N, making the structure suitable mostly for low order designs (R$<$ 4), where the decimation ratio is a fixed power of 2. The area is dominated by the combinatorial logic enabling efficient multichannel implementation.

5 SPARSE CIC FILTERS - A HARDWARE-EFFICIENT CLASS OF DIGITAL FILTERS FOR DECIMATION AND INTERPOLATION

5.1 Introduction

Cascaded-Integrator-Comb (CIC) filters are ubiquitous in Digital Signal Processing applications where efficient interpolation and decimation of oversampled signals is required. Since their introduction in Eugene Hogenauer's seminal paper in 1981 [40], a plethora of research has been dedicated to the improvement of their major weakness: limited worstcase stopband attenuation caused by the fact that all zeros at each stopband null are at the same location instead of being optimally distributed. Most of the literature can be categorized into two main lines of research: (a) the zero-rotating approach [9, 50, 51] where structural changes are incorporated to the classical filter with the aim to widen the stopbands by spreading the zeros closer to their optimal location, and (b), filter sharpening theory [8,52–55], where a sharpening polynomial is applied to the stopbands of the filter. The concepts of polyphase decomposition, multistage factoring and non-recursive implementation of the underlying FIR filter have been also applied to the original Hogenauer filter [41], as well as to both lines of research mentioned above [43,56]. An excellent survey of most relevant techniques is presented in [57] where the redundancy in the impressive body of work regarding CIC filters is pointed out and slightly criticized.

It is paramount to not lose sight of what makes CIC filters so widely used: simplicity, flexibility, trivial design procedure and, even, elegance. Simple because the structure consists solely of integrators and differentiators, without coefficients, in a regular arrangement. Flexible because any integer decimating ratio can be supported with *essentially the same hardware*, enabling straightforward support of programmable decimation ratios, a crucial feature for many systems such as software defined radio [48]. Trivial design procedure is almost an understatement: the order of the filter is increased until performance

is met; no more design work is required, no coefficient quantization to worry about and overflows can - and should - be left undetected. The wordlength of *all* nodes is identical and a direct consequence of the filter order and decimation ratio. Elegance - mostly as a way to encompass all aforementioned features - is the perfect term that defines CIC filters, and for any hardware design engineer, translates to *'the bug-less filter'*.

Although effective, most of the techniques proposed in previous research give up on one or more of these *features*, hindering their use in practice. In fact, any improvement of the CIC filter is up against a tenacious contender: increasing the filter order and moving on with the rest of the design. It is dangerous to confuse the quest for an optimal filter in the mathematical sense with the primary goal at hand: improving the response of a filter of order R without adding more hardware than a filter of order $R+1$ would require. The dual goal is also true and sometimes more appropriate: reduce the amount of hardware and computational expenditure of a filter of order R without compromising performance. CIC filters play in the territory of *good enough*, which is typically the enemy of *optimal*.

A new technique is put forward here that integrates seamlessly within a CIC filter, has a simple hardware implementation and improves the filter response at the bottleneck - the first null - while trading off attenuation at the higher frequency nulls that are overdesigned anyway. Both the hardware area and computational complexity are reduced, while *preserving the elegance* of the original CIC filter. The proposed technique is vast and flexible; it provides many *knobs* to designers who are faced with different design constraints and restrictions. Finally, design examples are presented and compared against previously published results.

Before proceeding, as was mentioned in Chapter 3, interpolation structures can be used for decimation (and vice-versa) by means of network transposition [2,33]. Once again, all structures described in this Chapter can therefore be used for both interpolation and decimation. Only decimation will be considered here to be consistent with most of the literature on CIC filter.

5.2 Brief CIC Filter review

The standard recursive CIC filter structure shown in Figure 18 was first introduced in [40] and was previously discussed in Chapter 4. It is a recursive way to implement

exactly the following transfer function

$$H(z) = \left(\sum_{i=0}^{N-1} z^{-i}\right)^{R} = \left(\frac{1-z^{-N}}{1-z^{-1}}\right)^{R}, \tag{5.1}$$

where R is the filter order and N is the decimation ratio. A very thorough description can be found in [40], while [46] presents a more intuitive introduction. Exact pole-zero cancellation and stability is guaranteed if the wordlength of all nodes is made equal to $B_{in} + B_{growth}$, where B_{in} is the input wordlength and $B_{growth} = \lceil R \log_2(N) \rceil$ [40]. The impulse response of a K^{th} order CIC filter is that of K boxcar filters of length-N convolved with themselves; each CIC integrator/differentiator pair is responsible for generating each such boxcar impulse.

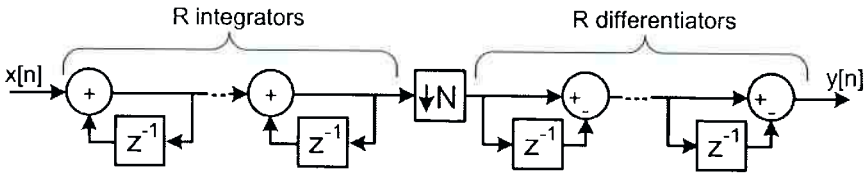

Figure 18: Standard recursive structure for CIC filters

Note that the CIC filter is not an aggressive filter by any measure, but is extremely well suited to decimation or interpolation of oversampled signals. In what is now considered a classic paper, Crochiere and Rabiner have shown in [11] that decimation is optimally performed in multiple stages where the filter order of the initial stages can be substantially reduced. Multistage partitioning is possible as long as the overall decimation ratio, osr, is expressible as a product of integers. Typically, two stages are used, so that $osr = N \cdot M$. CIC filters play the role of the first stage filter ($\downarrow N$) and are designed to decimate the signal as much as possible (since it is a cheap filter to implement), while providing enough anti-aliasing attenuation for the frequency bands that will alias into the baseband. For a sampling rate of fs, the baseband, or passband, is given by the region

$$B_p = \left[0, \frac{\frac{f_s}{2}}{osr}\right] \text{Hz}, \tag{5.2a}$$

and in the same manner as in [53], the aliasing bands, or stopbands, are given by:

$$B_s = \bigcup_{n=1}^{N/2} \left[\frac{f_s}{2}\left(\frac{2n}{N} - \frac{1}{osr}\right), \frac{f_s}{2}\left(\frac{2n}{N} + \frac{1}{osr}\right)\right] \text{Hz}, \tag{5.2b}$$

and A_{min}, the worst case attenuation in the stopbands, is given by

$$A_{min} = \max_{\omega \in B_s} |H(\omega)| . \tag{5.2c}$$

Figure 19 shows the response of 4^{th}, 5^{th} and 6^{th} order CIC filters with corresponding A_{min}, assuming a passband $f_p = \alpha \frac{\text{fs}}{2osr}$ where $\alpha = 0.907$. This value of α is a reasonable assumption and corresponds to standard audio signal specifications: passband edge at 20kHz for an output sampling rate of $\frac{\text{fs}}{osr} = 44.1\text{kHz}$ and $osr = 64$. The same values were used in [53, 57] and will serve as a comparison point later on.

Clearly, the bottleneck of the frequency response is located at the first aliasing edge of the first null. Increasing the order to $R+1$ in (5.1) adds N extra zeros across the unit circle at *exactly* the same frequency points as the filter of order R, which is not optimal from a mathematical perspective. Also notice that the other nulls, except for maybe the second one, provide unnecessary attenuation as the order is increased. This has led researchers to investigate ways to move, offset or rotate the zeros closer to their optimal locations. It would clearly be desirable to tradeoff some of the extra attenuation at those higher frequency aliasing bands for more attenuation at the first and maybe the second bands.

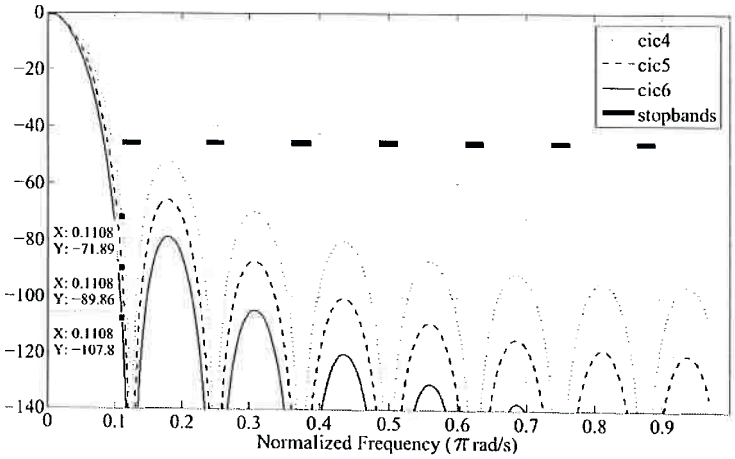

Figure 19: Frequency Response of CIC filters of order 4, 5 and 6.

5.3 Proposed structure

5.3.1 Elimination of the first differentiator and FIR equivalency

The first step in the derivation of the proposed structure is to make use of the well known fact that the first differentiator of CIC filters is redundant as it is simply removing

the initial condition at the preceding integrator N cycles before. This was explained in Chapter 4 and formed the basis for the reduced state-space structure, where higher order CIC filters were shown to be higher order integrate&dump circuits. Although the higher order filters were difficult to design, the innermost integrator/differentiator pair of any order CIC filter can always make use of this as depicted in Figure 20 for a 2nd order filter. This optimization saves roughly $\frac{1}{2R}$ the area of the corresponding CIC filter since the reset operation of the integrator is virtually free in hardware.

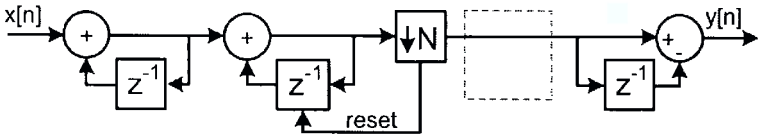

Figure 20: Removal of first differentiator using int&dump

This observation brings us back to the original idea of a CIC filter: each section is implementing an N-tap FIR filter whose coefficients are one. In fact, the diagram of Figure 20 can be redrawn as in Figure 21, where all the $h[n]$ coefficients are 1. Those two circuits are equivalent, and the int&dump version is simply a hardware-efficient way to implement the FIR filter. We show next that we can modify the original CIC idea to allow the last section to implement FIR filters with different sets of coefficients, using a time-varying coefficient in the input of the last integrator. Restricting this time-varying coefficient to be "simple" (i.e. 0 or 1, or a small integer), we retain the low complexity of the CIC filter.

5.3.2 Controllable coefficient through time-varying multiplier

Having full control over the coefficients of the last stage of a CIC filter would be a tremendous advantage when it comes to frequency response improvement, but unfortunately if the structure of Figure 21 is to be used, the amount of storage elements would quickly become prohibitive for even modest decimation ratios N. The proposed structure keeps the int&dump circuit in place, but precedes it with a time-varying multiplier, giving full control over the $h[n]$ coefficients, as depicted in Figure 22.

The transfer function of the proposed structure can be written as

$$H_{prop}(z) = \left(\frac{1-z^{-N}}{1-z^{-1}}\right)^{R-1} \cdot \left(\sum_{n=0}^{N-1} h_n z^{-n}\right) \tag{5.3}$$

where the h_n are free parameters to be designed. Note that the FIR filter is still im-

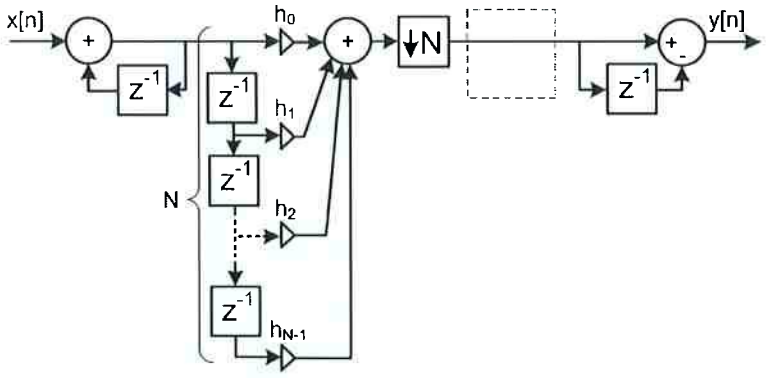

Figure 21: FIR interpretation of last integrator of a CIC filter

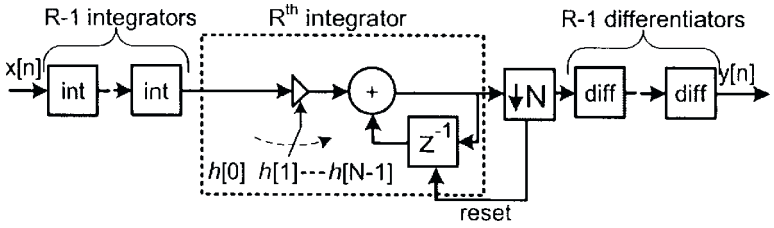

Figure 22: Proposed CIC filter variant

plemented efficiently through an integrator, but now with a time-varying input gain. In simple terms, for a system with a total of R integrators, the response of an $R-1$ CIC filter is convolved with the response of an N-tap FIR filter for which we have full control.

At first sight, the FIR filter should be optimized so that it places its zeros at or near their optimal locations, at least for the first null or two, improving the response at the bottleneck. The problem with this approach is that the optimal location *will* require precise coefficients, undermining most of the advantages of the CIC filter. However, as we show next, much improvement can be obtained even if we use suboptimal coefficients, chosen so that the hardware remains simple. In particular, a coefficient equal to zero has very interesting properties: it means *do not integrate*, reducing the amount of high speed operations performed by the filter, and lessens the overall filter gain, reducing the number of bits required for *all the nodes* of the filter. This naturally leads to the idea of constraining the FIR coefficients to a set of *small* integers, ideally as sparse as possible.

The new structure can be thought of as a hybrid recursive/polyphase approach to CIC filters. The recursive part places zeros in the middle of the aliasing bands, reasonable, albeit not optimal locations, for cheap. On the other hand, the polyphase part gives full control of its zero locations, at the cost of increased coefficient complexity. It is shown

next that once enough recursive sections are used, better performance is achieved by using a sparse polyphase FIR with trivial coefficients. The use of a time-varying multiplier in front of the int&dump circuit keeps the memory requirements of the polyphase filter to a minimum.

5.3.3 Previous work on the CIC-FIR combination

The idea of embedding an FIR filter inside a CIC filter is not new and was first introduced in [58], where it was proposed to embed the subsequent decimating stage filter inside the CIC filter. However this idea did not get investigated further as the FIR filter, which has by definition more stringent performance specifications, requires coefficients with a relatively large dynamic range, increasing the number of bits at the output. All the bits have to be kept since the FIR has to work with the same modulo arithmetic trick as the CIC filter, thus increasing the wordlength of all nodes of the filter [58].

The use of an embedded FIR filter was also proposed in [59], this time in the context of CIC filter improvement. However, the efficient implementation using a time-varying coefficient was not considered, and the design of the FIR filter not approached in a way that would elegantly embed in the CIC filter structure - see Section 5.4. In fact, the FIR filter design was approached from the angle of optimum zero placement, demanding for precise coefficients - read large integers - contributing to bit-growth for all the nodes of the overall filter. To circumvent this, it was suggested to implement the coefficients outside the non-modulo arithmetic zone, requiring separate differentiator paths for each non-zero FIR filter tap. Even if the filter is made sparse, this is clearly adding substantial hardware, as each comb path amounts to half the original filter.

5.4 Design and Optimization of the FIR filter

It is thus desired to design an FIR filter with sparse coefficients, taken from a subset of small integers, that would complement the response of the CIC filter in a better way then the conventional boxcar filter. Mixed-Integer Linear Programming (MILP) solvers are now able to solve this kind of problem quickly on most modern computers. Optimization and quantization of FIR filter coefficients using (MILP) is a powerful technique thoroughly described in [60, 61]. Now widely available, current MILP solvers do not require knowledge of the inner working of the algorithms. Frameworks are available that automatically convert the problem to a suitable form for the SOLVER and check for vi-

olations of constraints and even convexity, see [62–64] for examples. We now show how these MILP solvers can be used to design hardware-efficient modified CIC filters, using appropriate constraints on the coefficients.

First, recall from [61] that the zero-phase frequency response of a length-N FIR filter can be written as

$$H(\omega) = \sum_{n=0}^{M} b[n]\Phi(\omega, n), \tag{5.4a}$$

where

$$\Phi(\omega, n) = \begin{cases} 1 & \text{for Type I; n=0,} \\ 2\cos(n\omega) & \text{for Type I; n} > 0, \\ 2\cos[(n+\frac{1}{2})\omega] & \text{for Type II,} \\ 2\sin[(n+1)\omega)] & \text{for Type III,} \\ 2\sin[(n+\frac{1}{2})\omega] & \text{for Type IV,} \end{cases} \tag{5.4b}$$

$$b[n] = \begin{cases} h[\frac{N-1}{2} - n] & \text{for Type I,} \\ h[\frac{N-2}{2} - n] & \text{for Type II and IV,} \\ h[\frac{N-1}{2} - 1 - n] & \text{for Type III,} \end{cases} \tag{5.4c}$$

$$M = \begin{cases} \frac{N-1}{2} & \text{for Type I,} \\ \frac{N-2}{2} & \text{for Type II and IV,} \\ \frac{N-3}{2} & \text{for Type III,} \end{cases} \tag{5.4d}$$

where h[n] are the coefficients of the filter. This is all that is needed to formulate the MILP problem: find the b[n] that minimizes

$$\delta_1 = \max_{\omega \in \Omega_s} \left| H_{prop}(\omega) - D(\omega) \right|, \tag{5.5a}$$

subject to

$$\max_{\omega \in \Omega_p} \left| H_{prop}(\omega) - D(\omega) \right| < \delta_2, \tag{5.5b}$$

and

$$b[n] \in \mathbb{B}, \tag{5.5c}$$

where ω_p and ω_s are the passband and stopband regions respectively as in (5.2a) and (5.2b), δ_2 the maximum allowable passband droop, and $D(\omega)$ the ideal response, in this case obviously 1 in the passband and 0 in the stopband. The coefficients $b[n]$ are taken from a subset of the integers $\mathbb{B} \subset \mathbb{Z}$ and in many cases, $\mathbb{B}$ can be just $\{0,1\}$. Since H_{prop} shown in 5.3 consists of the cascade of a fixed CIC filter and the FIR filter to be designed,

the CIC filter response can be pre-calculated and used as a weighting function, reducing the number of variables to optimize. The problem formulation above is only one example, leading to a minimax design. Other cost functions can be used, such as minimizing the number of non-zero coefficients, while meeting a given passband droop and stopband attenuation. The full flexibility of MILP for FIR filter design can be leveraged and help designers accurately meet specific design constraints without reverting to an overdesigned CIC filter. Complete code using the *YALMIP* [62] framework is shown in Appendix A.

5.4.1 Efficient implementation of the proposed structure

Figure 23 shows an efficient implementation of the int&dump circuit with time varying coefficients. When $h[n] = 0$, the flip-flop is disabled or *clock-gated* and the adder is also *data gated* - i.e. presented with the same two input values as in the previous cycle, avoiding unnecessary calculations. If $h[n]$ can take values other than 0 or 1, then the proper multiplier (or more appropriately shift&add network) can be designed as part of the multiplexer logic.

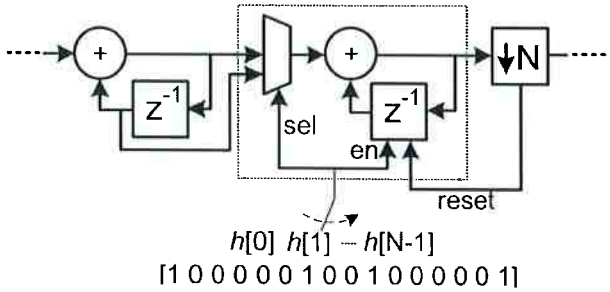

Figure 23: Efficient low-power implementation of the integrator with time varying coefficients (restricted to 0 or 1 in this example).

5.4.2 Longer FIR filters

So far only FIR filters with length equal to the decimation ratio N were considered. It is natural to ask if filters with longer impulse responses could also be implemented, ideally in a similar fashion so that few extra memory elements are required. Because the FIR filter is directly followed by the downsampler, only every other N outputs are used by the chain of differentiators, so a polyphase implementation is possible. Polyphase decomposition of FIR filters for decimation is extensively covered in [46], and when each FIR phase is implemented using the transposed form, the efficient implementation shown in Figure 24 arises. The added cost for each increase of N taps of the FIR filter is one

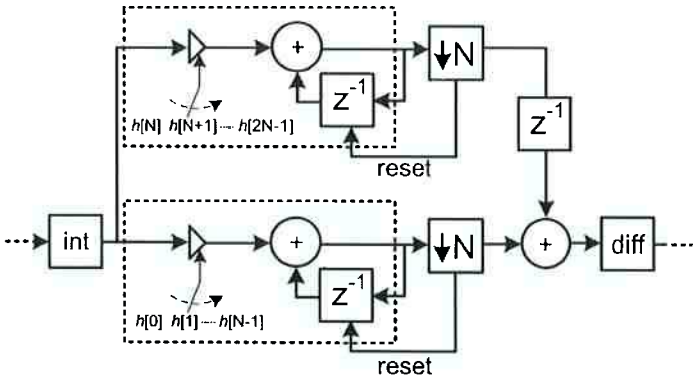

Figure 24: Implementation of FIR filter with more than N taps

integrator at the high rate that implements the second half of the impulse response $h[n]$, one memory element at the low rate delaying the integrator result by one output cycle, and one adder at the low rate. Using this implementation strategy, the hardware cost to increase the FIR length above N is thus *identical* as increasing the CIC filter order by 1 (one integrator and one differentiator). However it is possible to make use of the clever technique of [65] to reduce the cost by half, as shown in Figure 25 for a length-$2N$ FIR. In this polyphase structure, both integrators are accumulating the complete impulse response of the FIR filter, and being reset at half the output rate. The reset for the top integrator, denoted by $2N^*$ has a phase offset of half the impulse response length, so that the output can be taken from each integrator at the decimated rate using the commuter switch model. Typically such an implementation of decimating polyphase FIR filters is not preferred as the integrators have to accommodate for the complete gain of the filter instead of only a part of the impulse response, making the delay and adder larger. However, here all the nodes have to accommodate for the complete filter gain anyway as we are operating in the modulo-arithmetic zone, so it can be concluded that each new *sparse* integrator requires more or less half the resources of a standard CIC stage.

It is worth illustrating this with an example comparing a few possible implementation:

1. A standard 4^{th} order CIC filter: 4 integrators and 3 differentiators.
2. A 4^{th} order CIC filter where the last integrator is made sparse with length-N FIR: 4 integrators and 3 differentiators.
3. Using the technique of Figure 24, a 4^{th} order CIC filter where the last integrator is made sparse with length-2N FIR: 5 integrators, 3 differentiators, and one low rate delay and adder (similar to the complexity of a differentiator).

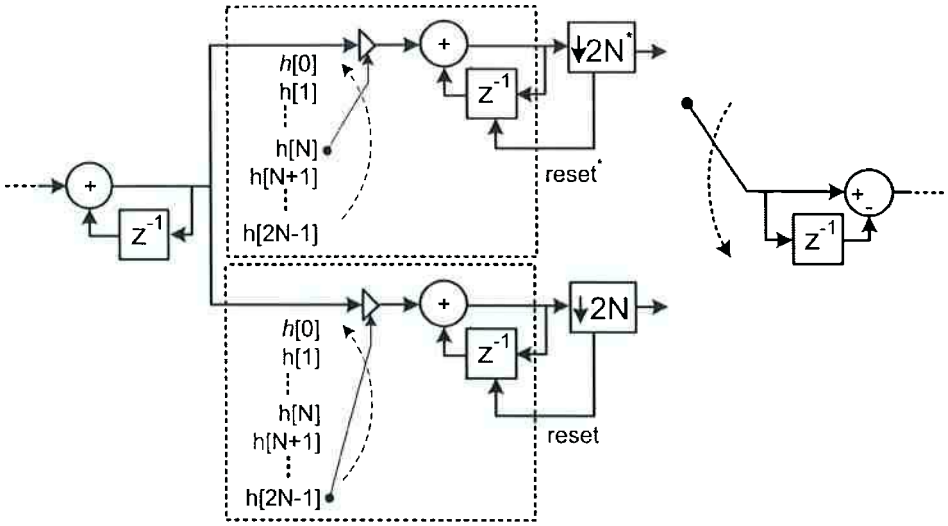

Figure 25: Efficient implementation of FIR filter with more than N taps ($2N$ shown here)

4. Using the technique of Figure 25, a 4^{th} order CIC filter where the last integrator is made sparse with length-2N FIR: 5 integrators, 3 differentiators.

5. A 5^{th} order CIC filter: 5 integrators and 4 differentiators.

As will be shown in Section 5.5.3 the *real* cost will depend on the performance improvement that a longer FIR can provide, the choice of coefficient set constraints and the overall gain of the filter. Different designs might call for the optimization of different parameters, and the added flexibility is one extra tool for designers to use.

5.4.3 Delay only path

Another observation worth noting is that if the FIR filter length is extended to $N+1$, and $h[N+1] \neq 0$, the extra integrator path reduces to a simple delay (or a gain and a delay if $h[N+1] \neq 1$). This structure is identical to Figure 8 of [58] where it was observed that using a length-(N+1) impulse for the last stage typically helps improving attenuation at the first null. The structure presented here can be seen as a generalization of this idea, giving full control of all the coefficients for any length FIR by using an integrator with time-varying coefficient.

5.4.4 Generalized structure

The same idea of an embedded FIR filter can be applied to all integrators of a CIC filter, as long as the output of each FIR is added back to the appropriate differentiator.

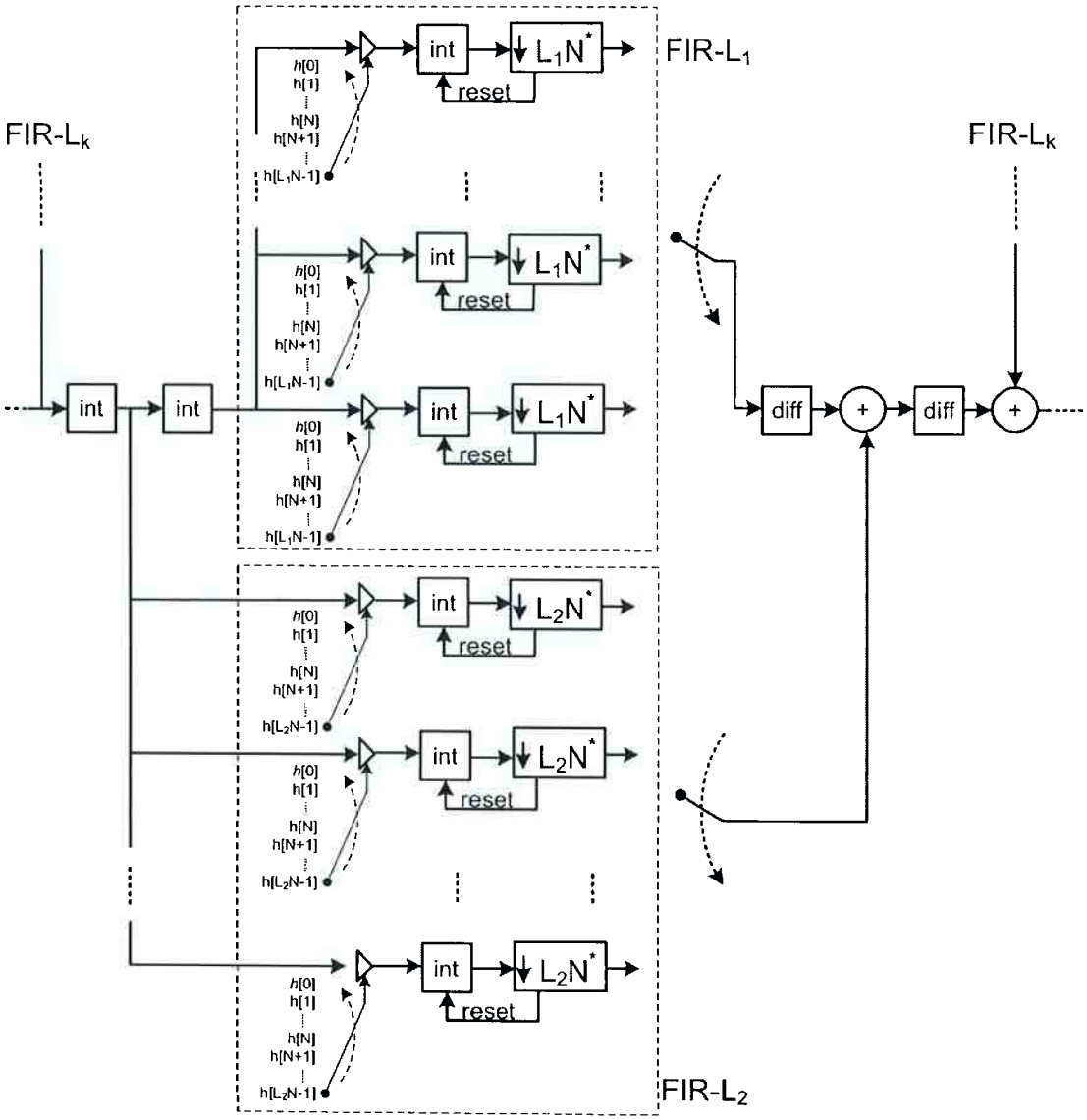

Figure 26: Generalized structure

This leads to the generalized structure shown in Figure 26. This structure is of similar shape as the one shown in [53], although here, a branch originates from each integrator as opposed to every second ones, and there is a fully controllable polyphase FIR filter of any length for each branch. The analysis of this structure is beyond the scope of this work, but it is believed that it might be an extremely efficient general purpose FIR filter structure, leveraging the unique property of integrators to generate integer coefficients economically, with added control provided by using time-varying simple/trivial multipliers.

5.5 Design Examples

5.5.1 Example from Coleman and Saramäki-Ritoniemi

An interesting comparison point is the design example from [53], which was later used as a comparison point with the Chebyshev stopband sharpening techniques in [57]. The filter specifications are $N = 16$, $osr = 64$, $\alpha = 0.907$, with a worst case attenuation of -90dB across all stopbands, see Section 5.2 for more details on each of these parameters. First, recall the performance of the standard CIC filters of order 4, 5 and 6 from Figure 19: $[A_{min}, B_{growth}]$ = [-71.8dB, 16bits], [-89.8dB, 20bits] and [-107.8dB, 24bits] respectively. A 6^{th} order standard CIC filter is thus required to meet $A_{min} = -90$dB, since a 5^{th} order filter barely falls short by -0.2dB, making one suspicious about the choice of specifications.

Sparse CIC filters of the same orders were designed, constraining the coefficients to the set {0,1}, and limiting the passband droop to be no worse than $-6dB$, as droop can always be compensated later at the lower rate. The optimal FIR filter coefficients, $h[n]$, are identical for all three filters and given by [1000001001000001], which has a high level of sparsity. Performance results are shown in Figure 27, along with a zoomed section of the first null, showing the effective zero rotation provided by the $h[n]$. Using the same notation as before, $[A_{min}, B_{growth}]$ = [-77.6dB, 14bits], [-100.1dB, 18bits] and [-120.1dB, 22bits] for the three filters. A_{min} is reduced by around 6, 11, and 13dB for each filter while B_{growth} is reduced by 2bits in all of them. Recall that B_{growth} is the wordlength of *all* nodes in the filter, so the area savings are not negligible. These results are encouraging, as the performance is *increased* at the expense of doing *less* operations, each *less* costly, in the same elegant hardware structure.

The same design specifications were achieved in [57], with block diagram and frequency response shown in their Figure 5 and 9 respectively. A fair comparison is with the 5^{th} order sparse filter, as both have a total of 5 integrators. However, the design from [57] requires 15 extra memory elements (z^{-1}) along with all the adders required to implement the integer coefficients. To be fair, the sharpening techniques and design ideology used were after *equiripple* stopbands, clearly interesting from a mathematical standpoint, but less so from the engineering perspective seeking a hardware efficient realization. The design from [53] is less storage intensive, but after having cancelled out comparable structural adders and memory elements, their design requires 3 extra storage locations and 2 extra adders, as well as 2 extra bits to account for bit growth. Considering that the sparse CIC filter has a total of 9 memory elements and 9 adders, the extra hardware needed is

significant.

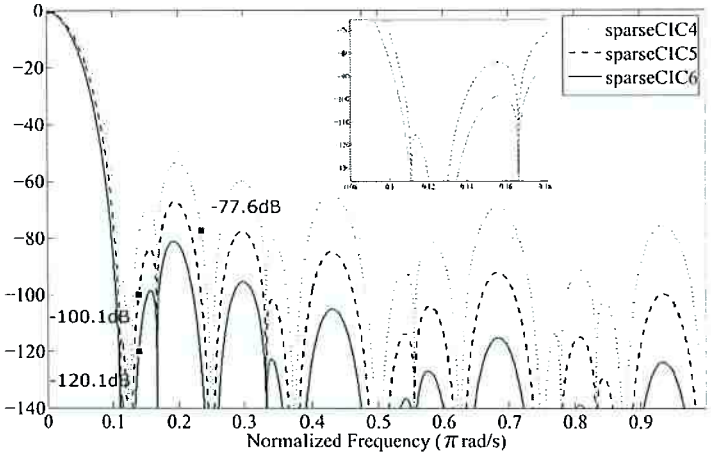

Figure 27: Frequency Response of Sparse CIC filters of order 4, 5, and 6 where $h[n] \in \{0, 1\}$

5.5.2 Non-binary coefficients

As mentioned before, the FIR coefficients need not be constrained to the {0,1} subset. Better performance can be achieved if this constraint is relaxed - however the designer has to be careful monitoring the benefits of doing so. Larger coefficients might require structural adders, and generate more gain, making all the nodes of the filter larger. Depending on the performance gain, it might be more economical to add a standard CIC stage. Nevertheless, the same three filters as above were redesigned, this time constraining the coefficients to integers in the set [-4,4]. Results are shown in Figure 28. Improvements of 5, 7, and 14dB are possible over the same filters constrained to use binary coefficients, while B_{growth} is *reduced* by 1bit. Only for the 5^{th} order filter was the coefficient ± 3 used, costing one extra adder, while the set $\{0, \pm 1, \pm 2, \pm 4\}$ is optimal for the other two filters, implementable with a trivial multiplexer, sign change and a shift. Other set of trivial coefficients could be used and worth investigating.

5.5.3 FIR filter longer than N

In Section 5.4.2, it was shown how FIR filters longer than N can also be efficiently implemented using the proposed structure by making use of polyphase decomposition. Combined with optimizing for different subsets of admissible coefficient values, this opens up a wide array of design options which cannot be covered here. It is however enlightening to compare filters with the same total amount of integrators, as this is a good measure

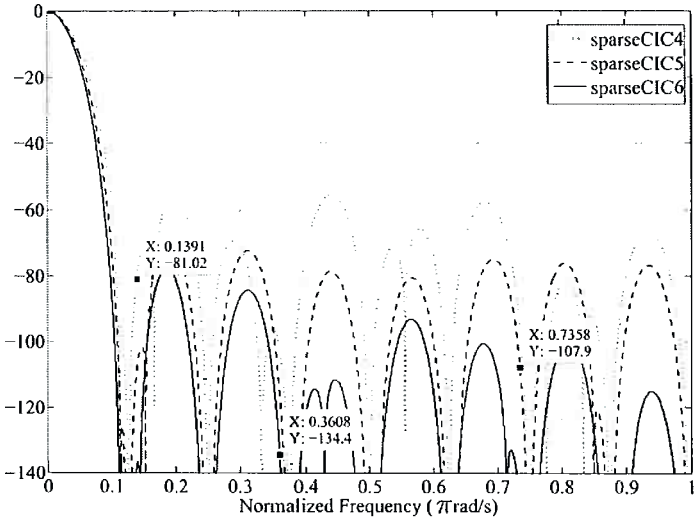

Figure 28: Frequency Response of Sparse CIC filters of order 4, 5, and 6 where $h[n] \in \{0, \pm 1, \pm 2, \pm 3, \pm 4\}$

of overall complexity. We will compare three filters with a total of 5 integrators: the standard 5^{th} CIC, a 4^{th} order sparse CIC with length-N FIR, and a 3^{rd} order sparse CIC with length-2N FIR. Performance is [−89.8dB, 20bits], [−105.3dB, 17bits] and [−108.1dB, 16bits] for the three filters respectively and the frequency response shown in Figure 29.

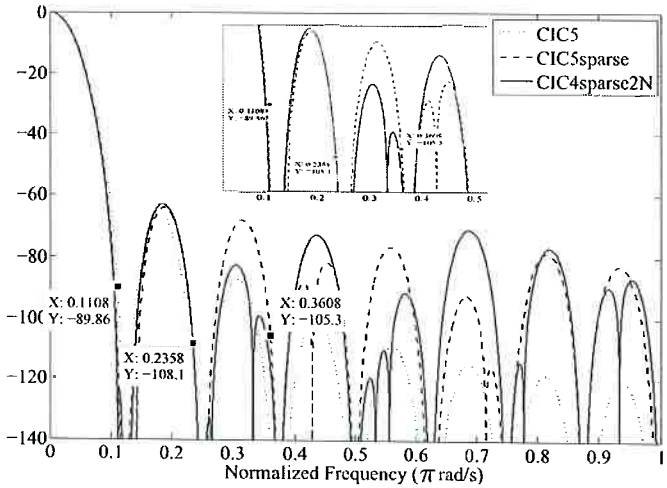

Figure 29: Frequency Response of CIC5, CIC4 with lengthN FIR and CIC3 with length2N FIR, where $h[n] \in 0, \pm 1, \pm 2$

The 3^{rd} order sparse CIC with length-2N FIR has similar performance to the 6^{th} order standard CIC filter previously shown in Figure 19 ($A_{min} \approx -107$dB). Both filters are drawn side by side in Figure 30, where B_{growth} is 24bits for the CIC and merely 16 bits for the sparse CIC. The benefits of using a length-2N FIR (i.e. extra sparse integrators) as opposed to adding standard CIC stages is clearly illustrated: not only performance is

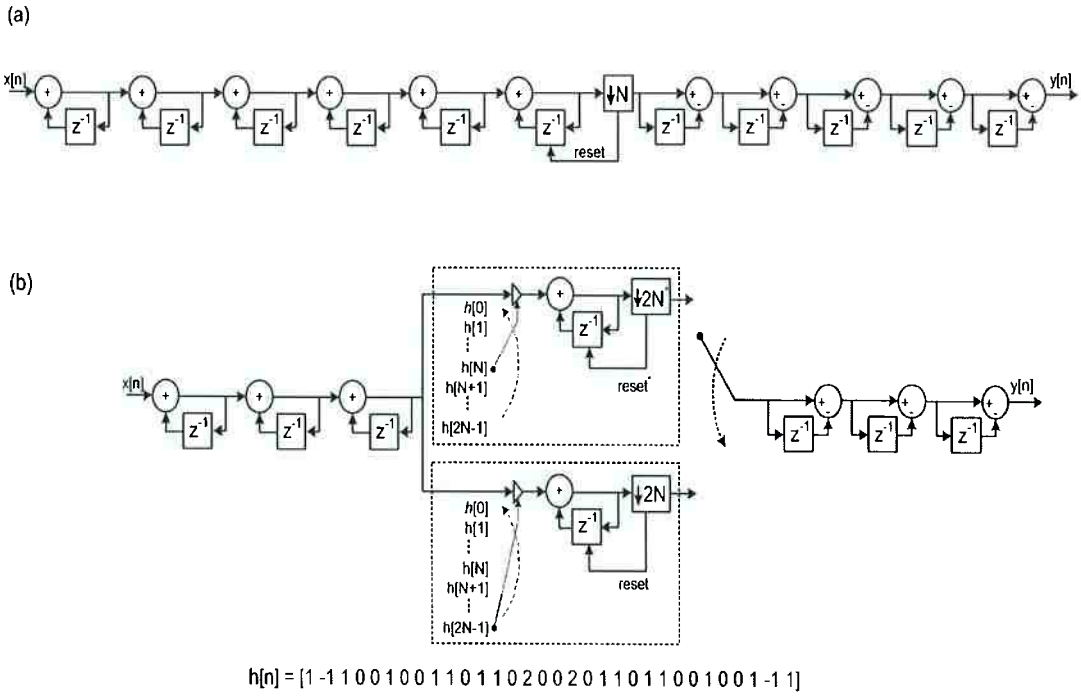

Figure 30: Frequency response of (a) 6^{th} order CIC filter (b) 3^{rd} order sparse CIC with length-2N FIR

increased, but the number of bits to do so is reduced, thanks to the sparsity of the FIR filter. Extra sparse integrators also do not require a differentiator. Area of both circuits can be estimated using

$$A = \sum_{i=0}^{L} NP_i \cdot W_i \tag{5.6}$$

which was proposed in [41] for estimating silicon area of different polyphase implementations of CIC filters. A in (5.6) is unit-less but proportional to silicon area. It was concluded in [41] that equation-based area estimation is very close to real implementation results. NP_i is the number of 1bit partial products to be added in stage i and W_i is the wordlength of stage i. Here, decimation is performed in one stage so $L = 1$. The area for both circuits of Figure 30 is 616 for the classical 6^{th} order CIC and 256 for the 3^{rd} order sparse CIC with length-2N FIR, showing savings of more than 50%.

5.6 Bit accurate model and verification methodology

Proving that hardware structures are actually implementing the desired transfer function is not a trivial task. This is further complicated here because the complete filter is

working with modulo data and overflows are not detected. A complete cycle and bit-accurate fixed-point model of the proposed structure was built and all of the filter examples designed here were tested with various types of input signals such as sine waves, square waves, full scale DC inputs, and the response was compared with that of the underlying FIR filter predicted by the transfer function. An example of a simulated frequency response sweep test is shown in Figure 31 for the 5^{th} order filter of Figure 27. Each point of the simulated response line is calculated by measuring the signal level of a sinusoidal test tone after being filtered by the model, accounting for aliasing, and adjusting for gain. 1024 such simulations were run to generate this curve. Matlab code of the model is shown in Appendix B.

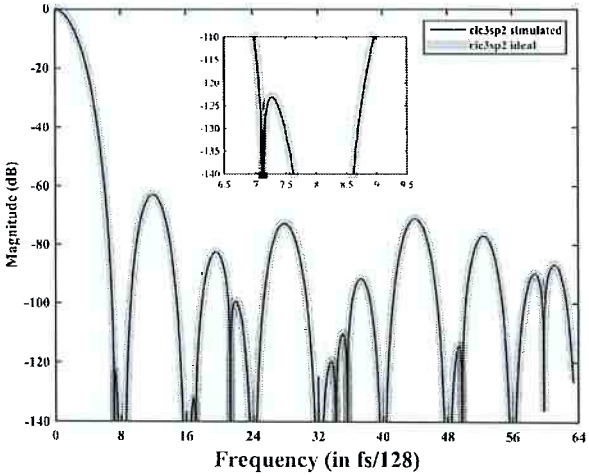

Figure 31: Simulated frequency response of the 5^{th} order filter of Figure 27

5.7 Conclusion

A novel structure for improving the performance of the well known CIC filter while preserving their main qualities and characteristics was proposed. It was shown that the last integrator stage of CIC filters can be modified in order to implement any FIR filter impulse response of any length. Substantial performance improvement is possible with sparse FIR filters constrained to small integer coefficients, which leads to efficient hardware implementation. When compared to standard CIC filters, the proposed structure can improve the worst case stopband attenuation, while reducing the overall gain of the filter and the amount of high rate operations performed. The structure allows to leverage the advantages of both the recursive and polyphase implementation of CIC filters, giving more flexibility to the designers confronted with different design challenges.

6 CONCLUSIONS AND FURTHER RESEARCH

6.1 Summary

In this dissertation, algorithms and efficient filtering structures for the problem of sampling rate conversion of digital signals were investigated. The first part of the text was concerned with the two distinct challenges faced in asynchronous sample rate conversion (ASRC), namely the generation of clock signals that properly track the fractional ratio, and the implementation of efficient polynomial-based filtering structures for fractional rate change. The second part of the work covered integer sample rate conversion, investigating potential ways to improve the first filter of a decimating chain.

In Chapter 2, an all-digital clock generator circuit was introduced. It differs from classical DPLLs in that the locking mechanism is based on the period of the reference clock instead of its phase and/or frequency. This allows for wide bandwidths and fast locking times without relying on dividers, mode switching, and/or tuning word presetting. Moreover, various trade-offs inherent to feedback loops, such as input jitter rejection, bandwidth, and settling time, are decoupled and made transparent to the designer making the overall design more intuitive. Noise shaping techniques can also be seamlessly integrated into the period-locked NCO to reduce phase noise in a desired frequency band. The resulting low complexity hardware can be used to generate high quality clock signals for demanding applications such as audio ADC/DAC.

Then, in Chapter 3, an efficient structure for spline-based fractional delay filtering was introduced. Inspired by the Newton structures for Lagrange interpolation, it requires less than half the number of operations of a typical Farrow implementation. Moreover, it displays better frequency response characteristics than Lagrange-based filters. To obtain this structure, a matrix form of the Farrow transfer function is put forward and used to derive state-space transformations between the Lagrange-Farrow structure and its Newton counterpart. These transformations are then applied to the spline polynomial giving rise to the efficient Newton-like spline filtering method.

Chapter 4 analyzed a previously published structure for reduced complexity CIC filters [44]. The reduced state-space structure has been mostly ignored due to its apparent complexity caused by the need for feedback multipliers. It was shown that for filters of order 3 and below along with even decimation ratios, the multipliers can be exactly implemented with only a few non-zero canonical-signed-digit(CSD) terms, making the structure very attractive. Savings of up to 50% silicon area compared to a traditional recursive implementation are possible. Furthermore, most of the area comes from combinatorial logic which can be reused across channels, leading to compact multichannel implementations.

Finally, a novel structure for improving the classical CIC filter was proposed in Chapter 5. The structure is based on the observation that the last integrator of a standard CIC filter is in fact implementing a polyphase FIR filter where the coefficients are all 1. By making the coefficients controllable through a time-varying multiplier in front of the integrator, the frequency response of the CIC filter can be improved while simultaneously reducing computational complexity, given a certain level of sparsity. A MILP framework was suggested to optimize the coefficients of the polyphase FIR filter by constraining them to small integers. The length of the polyphase FIR filter can also be extended above the decimating ratio.

6.2 Promising lines of research

6.2.1 Reduced complexity implementation structures for polynomial-based filters

The new structure proposed in Section 3.5 was derived using a Matrix representation of the Farrow transfer function leading to identifying state-space transformations between the Lagrange-Farrow structure and its Newton counterpart. These transformations were then applied to the spline polynomial giving rise to the efficient Newton-like spline filtering method. First, those same transformations could be applied to other polynomials, or different orders, in order to generate other potentially efficient structures for classical time-domain polynomial functions. However, with a slight change of basis function from $z^{-1} \rightarrow 1 - z^{-1}$, the mixed-integer linear programming (MILP) techniques proposed in [66] for direct optimization of the coefficients of the Farrow structure could be applied to optimization directly in the new structure. Furthermore, computational complexity might be reduced even more by incorporating the fractional interval state-space transformation

as part of the optimization process.

6.2.2 Sparse CIC filters

6.2.2.1 Non-symmetric sparse FIR filters

So far only symmetric sparse FIR filters were considered in the design of the proposed sparse CIC filters. It would be worthwhile investigating design techniques where the symmetry of the sparse FIR filter is not enforced, opening up the design space to include non-linear phase filters. Non-linear phase filter could be used to either improve performance, or increase the sparsity of the FIR, effectively reducing the overall gain and the wordlength required. Genetic algorithms come to mind, such as Differential Evolution [67], previously successfully applied to digital filter optimization problems.

6.2.2.2 Droop Compensation using the sparse FIR filter

Only stopband performance improvement was considered in the work presented here, but preliminary work has shown that the sparse CIC filter can be used to effectively reduce the undesired droop caused by higher order CIC filters. This opens up room for the design of a more encompassing set of MILP constraints, where designers could balance desired stopband attenuation, maximum droop allowed, and hardware complexity.

REFERENCES

[1] GRAYVER, E. *Implementing Software Defined Radio.* [S.l.]: Springer Science & Business Media, 2012.

[2] HENTSCHEL, T.; FETTWEIS, G. Continuous-time digital filters for sample-rate conversion in reconfigurable radio terminals. *Frequenz*, v. 55[5/6], p. 185–188, 2001.

[3] PARK, J.-S. et al. An asynchronous sample-rate converter from CD to DAT. In: *Acoustics, Speech, and Signal Processing, 2003. Proceedings. (ICASSP '03). 2003 IEEE International Conference on.* [S.l.: s.n.], 2003. v. 2, p. II–509–12 vol.2. ISSN 1520-6149.

[4] ADAMS, R.; KWAN, T. Theory and VLSI architectures for asynchronous sample-rate converters. *J. Audio Eng. Soc.*, v. 41[7/8], p. 539–555, 1993.

[5] MIDYA, P.; ROECKNER, B.; SCHOOLER, T. Asynchronous sample rate converter for digital audio amplifiers. In: AUDIO ENGINEERING SOCIETY. *Audio Engineering Society Convention 121.* [S.l.], 2006.

[6] FARROW, C. A continuously variable digital delay element. In: *IEEE Int. Symp. on Circuits and Syst.* [S.l.: s.n.], 1988. p. 2641–2645.

[7] ABED, K.; NERURKAR, S. Low power and hardware efficient decimation filter. In: *Wireless Communications and Networking, 2003. WCNC 2003. 2003 IEEE.* [S.l.: s.n.], 2003. v. 1, p. 454–459 vol.1. ISSN 1525-3511.

[8] CANDAN, C. Optimal sharpening of CIC filters and an efficient implementation through Saramaki-Ritoniemi decimation filter structure. 2013.

[9] DOLECEK, G. J.; LADDOMADA, M. An economical class of droop-compensated generalized comb filters: Analysis and design. *Circuits and Systems II: Express Briefs, IEEE Transactions on*, IEEE, v. 57, n. 4, p. 275–279, 2010.

[10] LOSADA, R. A.; LYONS, R. Reducing CIC filter complexity. *Signal Processing Magazine, IEEE*, IEEE, v. 23, n. 4, p. 124–126, 2006.

[11] CROCHIERE, R. E.; RABINER, L. Optimum FIR digital filter implementations for decimation, interpolation, and narrow-band filtering. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, IEEE, v. 23, n. 5, p. 444–456, 1975.

[12] BEST, R. *Phase-locked loops: theory, design, and applications.* [S.l.]: McGraw-Hill, 1993.

[13] RILEY, T.; COPELAND, M.; KWASNIEWSKI, T. Delta-sigma modulation in fractional-N frequency synthesis. *IEEE Journal of Solid-State Circuits*, v. 28[5], p. 553–559, 1993.

[14] KWAN, T.; ADAMS, R.; LIBERT, R. A stereo multibit Σ-Δ DAC with asynchronous master-clock interface. *IEEE Journal of Solid-State Circuits*, v. 31[12], p. 1881–1887, 1996.

[15] BRANDONISIO, F. et al. Modelling and implementation of an accumulator-based ADPLL on a Virtex-5. IET, 2012.

[16] NILSSON, P.; TORKELSON, M. A monolithic digital clock-generator for on-chip clocking of custom DSP's. *IEEE Journal of Solid-State Circuits*, v. 31[5], p. 700–706, 1996.

[17] OLSSON, T.; NILSSON, P. Portable digital clock generator for digital signal processing applications. *IET Electronics Letters*, v. 39[19], p. 1372–1374, 2003.

[18] VANKKA, J.; HALONEN, K. *Direct digital synthesizers: Theory, design, and applications*. [S.l.]: Springer, 2001.

[19] VITAL, J. C.; TEMES, G. C. Clock generation system with reduced jitter noise in the baseband. In: *International Circuits and Systems Symposium*. [S.l.: s.n.], 1991. p. 2621–2624.

[20] HOGENAUER, E. An economical class of digital filters for decimation and interpolation. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, v. 29[2], p. 155–162, 1981.

[21] DATTORRO, J. The implementation of recursive digital filters for high-fidelity audio. *Journal of the Audio Engineering Society*, v. 36[11], p. 851–878, 1988.

[22] KISS, P. et al. Stable high-order delta-sigma digital-to-analog converters. *IEEE Transactions on Circuits and Systems I*, v. 51[1], p. 200–205, 2004.

[23] DUNNING, J. et al. An all-digital phase-locked loop with 50-cycle lock time suitable for high-performance microprocessors. *IEICE Transactions on Electronics*, v. 78[6], p. 660–670, 1995.

[24] MENDEL, S.; VOGEL, C. A z-domain model and analysis of phase-domain all-digital phase-locked loops. In: *Norchip*. [S.l.: s.n.], 2007. p. 1–6.

[25] ABRAMOVITCH, D. Phase-locked loops: A control centric tutorial. In: *American Control Conference*. [S.l.: s.n.], 2002. p. 1–15.

[26] MENDEL, S.; VOGEL, C. Improved lock-time in all-digital phase-locked loops due to binary search acquisition. In: *International Conference on Electronics, Circuits and Systems*. [S.l.: s.n.], 2008. p. 384–387.

[27] KLEIDER, J.; STEENHOEK, C. A new filter implementation strategy for Lagrange interpolation. In: *IEEE Military Commun. Conf.* [S.l.: s.n.], 2014. p. 724–730.

[28] HUANG, X.; GUO, Y.; ZHANG, J. A. Sample rate conversion using B-spline interpolation for OFDM based software defined radios. *IEEE Trans. on Commun.*, v. 60[8], p. 2113–2122, 2012.

[29] FRANCK, A. et al. Reproduction of moving sound sources by wave field synthesis: An analysis of artifacts. In: *Int. Conf. Audio Eng. Soc.* [S.l.: s.n.], 2007.

[30] LAAKSO, T. et al. Splitting the unit delay [FIR/all pass filters design]. *IEEE Signal Process. Mag.*, v. 13[1], p. 30–60, 1996.

[31] TASSART, S.; DEPALLE, P. Fractional delays using Lagrange interpolators. In: *Int. Comput. Music Conf.* [S.l.: s.n.], 1996.

[32] CANDAN, C. An efficient filtering structure for Lagrange interpolation. *IEEE Signal Proc. Lett.*, v. 14[1], p. 17–19, 2007.

[33] LEHTINEN, V.; RENFORS, M. Structures for interpolation, decimation, and nonuniform sampling based on Newton's interpolation formula. In: *Int. Conf. on Sampling Theory and Applicat.* [S.l.: s.n.], 2009.

[34] WISE, D.; BRISTOW-JOHNSON, R. Performance of low-order polynomial interpolators in the presence of oversampled input. *J. Audio Eng. Soc.*, 1999.

[35] VALIMAKI, V. A new filter implementation strategy for Lagrange interpolation. In: *IEEE Int. Symp. on Circuits and Syst.* [S.l.: s.n.], 1995. p. 361–364.

[36] FRANCK, A. Efficient algorithms and structures for fractional delay filtering based on Lagrange interpolation. *J. Audio Eng. Soc.*, v. 56[12], p. 1036–1056, 2009.

[37] HUNTER, M.; MIKHAEL, W. A novel Farrow structure with reduced complexity. In: *IEEE Int. Midwest Symp. on Circuits and Syst.* [S.l.: s.n.], 2009. p. 581–585.

[38] UNSER, M. Splines: A perfect fit for signal and image processing. *IEEE Signal Process. Mag.*, v. 16[6], p. 22–38, 1999.

[39] DOOLEY, S.; STEWART, R.; DURRANI, T. Fast on-line B-spline interpolation. *IET Electron. Lett.*, v. 35[14], p. 1130–1131, 1999.

[40] HOGENAUER, E. An economical class of digital filters for decimation and interpolation. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, IEEE, v. 29, n. 2, p. 155–162, 1981.

[41] ABOUSHADY, H. et al. Efficient polyphase decomposition of comb decimation filters in Σ-Δ analog-to-digital converters. In: IEEE. *Circuits and Systems, 2000. Proceedings of the 43rd IEEE Midwest Symposium on.* [S.l.], 2000. v. 1, p. 432–435.

[42] JOVANOVIC-DOLECEK, G.; MITRA, S. K. Efficient sharpening of CIC decimation filter. In: IEEE. *Acoustics, Speech, and Signal Processing, 2003. Proceedings.(ICASSP'03). 2003 IEEE International Conference on.* [S.l.], 2003. v. 6, p. VI–385.

[43] JOVANOVIC-DOLECEK, G.; MITRA, S. Efficient multistage comb-modified rotated sinc (RS) decimator. In: *Signal Processing Conference, 2004 12th European.* [S.l.: s.n.], 2004. p. 1425–1428.

[44] HENTSCHEL, T.; FETTWEIS, G. Reduced complexity comb-filters for decimation and interpolation in mobile communications terminals. In: IEEE. *Electronics, Circuits and Systems, 1999. Proceedings of ICECS'99. The 6th IEEE International Conference on.* [S.l.], 1999. v. 1, p. 81–84.

[45] HENTSCHEL, T.; FETTWEIS, G. Time-varying recursive filters for decimation and interpolation. In: *Proc. 10th Euro. Sig. Processing Conf.* [S.l.: s.n.], 2000. p. 5–8.

[46] LYONS, R. G. *Understanding Digital Signal Processing.* [S.l.]: Pearson Education, 2010.

[47] LIN, C.-A.; KING, C.-W. Minimal periodic realizations of transfer matrices. *Automatic Control, IEEE Transactions on*, IEEE, v. 38, n. 3, p. 462–466, 1993.

[48] HENTSCHEL, T. *Sample Rate Conversion in Software Configurable Radios.* [S.l.]: Artech House, 2002.

[49] PARHI, K. K. *VLSI Digital Signal Processing Systems: Design and Implementation.* [S.l.]: John Wiley & Sons, 2007.

[50] PRESTI, L. L.; AKHDAR, A. Efficient antialiasing decimation filter for $\Delta\Sigma$ converters. In: IEEE. *Electronics, Circuits and Systems, 1998 IEEE International Conference on.* [S.l.], 1998. v. 1, p. 367–370.

[51] PRESTI, L. L. Efficient modified-sinc filters for sigma-delta A/D converters. *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, IEEE, v. 47, n. 11, p. 1204–1213, 2000.

[52] ENGELBERG, S. A more general approach to the filter sharpening technique of kaiser and hamming. *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS PART 2 EXPRESS BRIEFS*, IEEE, v. 53, n. 7, p. 538, 2006.

[53] SARAMAKI, T.; RITONIEMI, T. A modified comb filter structure for decimation. In: IEEE. *Circuits and Systems, 1997. ISCAS'97., Proceedings of 1997 IEEE International Symposium on.* [S.l.], 1997. v. 4, p. 2353–2356.

[54] Tapio Saramaki, Tapani Ritoniemi, Ville Eerola, Timo Husu, Eero Pajarre, Seppo Ingalsuo et al. *Decimation Filter.* 1997. US Patent 5,689,449.

[55] KWENTUS, A. Y.; JIANG, Z.; JR, A. N. W. Application of filter sharpening to cascaded integrator-comb decimation filters. *Signal Processing, IEEE Transactions on*, IEEE, v. 45, n. 2, p. 457–467, 1997.

[56] STEPHEN, G.; STEWART, R. High-speed sharpening of decimating cic filter. *Electronics Letters*, IET, v. 40, n. 21, p. 1383–1384, 2004.

[57] COLEMAN, J. O. Chebyshev stopbands for cic decimation filters and cic-implemented array tapers in 1d and 2d. *Circuits and Systems I: Regular Papers, IEEE Transactions on*, IEEE, v. 59, n. 12, p. 2956–2968, 2012.

[58] CHU, S.; BURRUS, C. S. Multirate filter designs using comb filters. *Circuits and Systems, IEEE Transactions on*, IEEE, v. 31, n. 11, p. 913–924, 1984.

[59] LEHTINEN, V.; RENFORS, M. On cic decimator variants - from shifting zeros to the sparse fir-cic structure. In: IEEE. *Signal Processing and Its Applications, 2007. ISSPA 2007. 9th International Symposium on.* [S.l.], 2007. p. 1–4.

[60] LIM, Y. Design of discrete-coefficient-value linear phase FIR filters with optimum normalized peak ripple magnitude. *Circuits and Systems, IEEE Transactions on*, v. 37, n. 12, p. 1480–1486, Dec 1990. ISSN 0098-4094.

[61] MITRA, S. K.; KAISER, J. F. *Handbook for Digital Signal Processing.* [S.l.]: John Wiley & Sons, Inc., 1993.

[62] LOFBERG, J. YALMIP: A toolbox for modeling and optimization in MATLAB. In: IEEE. *Computer Aided Control Systems Design, 2004 IEEE International Symposium on.* [S.l.], 2004. p. 284–289.

[63] GRANT, M.; BOYD, S.; YE, Y. *CVX: Matlab Software for Disciplined Convex Programming.* 2008.

[64] DIAMOND, S.; CHU, E.; BOYD, S. *CVXPY: A Python-Embedded Modeling Language for Convex Optimization, version 0.2.* 2014. Urlhttp://cvxpy.org/.

[65] Koji Kawamoto, Toru Kengaku, Eiichi Teraoka, Tetsuaki Oga e Hiroichi Ishida. *Decimating digital finite impulse response filter.* 1993. US Patent 5,191,547.

[66] FRANCK, A. *Efficient Algorithms for Arbitrary Sample Rate Conversion with Application to Wave Field Synthesis.* Tese (Doutorado) — Universitätsbibliothek Ilmenau, 2011.

[67] PRICE, K.; STORN, R. M.; LAMPINEN, J. A. *Differential Evolution: A Practical Approach to Global Optimization.* [S.l.]: Springer Science & Business Media, 2006.

APPENDIX A – MATLAB CODE FOR THE DESIGN OF SPARSE CIC FILTERS

The following Matlab script can be used to design sparse CIC filters. The design specifications are the same as presented in Chapter 5. A suitable MILP solver has to be installed. Support functions are also listed.

```
clear all
format long
%-------------------------------------------------------------
%Take general form of FIR filter
%about the symmetric/odd/even stuff
%          N
%H(w) = sum(h[n]*trig(w,n))
%         n=0
%type and length of FIR filter
%type1 Neven-sym, cyc=2 Nodd-sym, type3 Neven-antisym, type4 Nodd-antisym
%Remember that N means Nil TAP, and that M means M+1 distinct numbers
%-------------------------------------------------------------

npoints = 2^14;
cicR = 16;
showresp = 1;

%sparse filter settings
mincoeff = 0;
maxcoeff = 1;
sparseN = cicR-1;
if (mod(sparseN,2)==0) sparseType = 1; else sparseType = 2; end
sparseM = getM(sparseN, sparseType);
%standard cic stages prior to sparse filter
```

```
cicsofar = makeCIC(4, cicR);

%The final filter N is the convolution of cicsofar and sparseN
N = length(cicsofar)+(sparseN+1)-1; %convolution length
N = N-1;%actual N
if (mod(N,2)==0) type = 1; else type = 2; end

w = linspace(0,pi,npoints);
[trig,M] = getTrig(N,w,type);
%trig(abs(trig)<1e-15) = 0;
%----------------------------------------------------------
%cic to compare against
cicorder = 5;
hcic = makeCIC(cicorder, cicR);
[tmp, w] = freqz(hcic/sum(hcic), 1, npoints);  HcicdB = ...
    20*log10(abs(tmp));
%----------------------------------------------------------
%----------------------------------------------------------
%set passband and stopband indexes-----------------
fsin = 3072e3;
osr = 64;
% pEdge = 20e3/fsin;
% pEdgeosr = pEdge/osr;
alpha = 0.907;
fp = alpha*fsin/(2*osr);
pEdgeosr = fp/fsin;
sbwidth = round(pEdgeosr*2*npoints);
sbcenter = round((2*npoints/cicR):(2*npoints/cicR):npoints);
sbidx = sbcenter;
for i=1:length(sbcenter)
    sbidx = [sbidx, ((sbcenter(i)-sbwidth) : (sbcenter(i)+sbwidth))];
end
sbidx = sort(unique(sbidx));
sbidx = sbidx(sbidx>0);
sbidx = sbidx(sbidx<npoints);
pbidx = 1:sbwidth;
wcare = [pbidx,sbidx];
wdontcare = setdiff((1:npoints),wcare);
Hideal = zeros(1,npoints);
Hideal(pbidx) = 1;
Hideal(sbidx) = 0;
Hidealcare = Hideal(wcare)';
Hidealpass = Hideal(pbidx)';
```

```
67  Hidealstop = Hideal(sbidx)';
68  Gcare = trig(wcare,:);
69  Gpass = trig(pbidx,:);
70  Gstop = trig(sbidx,:);
71  %-----------------------------------------------------------
72
73  %droop constraint if desired-
74  maxdroop = HcicdB(sbwidth)-3;
75  minstop = -80;%-Smax(HdB(sbidx));
76  %-----------------------------------------------------------
77
78  %using yalmip
79  all1 = ones(1,cicR);
80  tvc = intvar(1,sparseM+1); %or binvar
81  symtvc = btoh(tvc, sparseType);
82  hyal = conv(cicsofar, symtvc);
83  byal = htob(hyal, type);
84  s = sdpvar;
85  C1 = [s*(1 - (1-10^(maxdroop/20))) ≤ (Gpass*byal') ≤ s*(1 + ...
        (10^(-maxdroop/20)-1))];
86  C2 = 10 ≤ s ;
87  C4 = [mincoeff ≤ tvc ≤ maxcoeff];
88  Constraints = [C1, C2, C4];
89  Objective = norm(Gstop*byal', 2);
90  options = sdpsettings('solver','gurobi');%sedpt3
91  sol = solvesdp(Constraints, Objective, options);
92
93  if (sol.problem == 0)
94   byal = double(byal)
95  else
96   display('Hmm, something went wrongin Yalmip solving...!');
97   sol.info
98   yalmiperror(sol.problem)
99  end
100 hyal = btoh(byal, type);
101 Hyal = 20*log10(abs(trig*byal'));
102 gain = max(Hyal);
103 Hyal = Hyal-gain;
104 %-----------------------------------------------------------
105
106 showtip=1;
107 figure
108 freqs = (fsin/2)*w/pi;
```

```
109 dePlot = plot(freqs, Hyal, freqs, HcicdB);
110 hold all;
111 xlabel('Freq normalized'); ylabel('mag dB'); ylim([-140 0]); grid on;
112 Hidealplot = Hideal; Hidealplot(sbidx) = 0.003;
113 plot(freqs,20*log10(Hidealplot), 'LineWidth', 5);
114 legend('Hsparse',sprintf('Hcic order %d', cicorder), 'Stopbands');
115
116 figure
117 stem(hyal);hold all; stem(hcic)
118 legend('hsparse', sprintf('hcic%d',cicorder))
119
120 display(sprintf('CIC worst case attenuation %gdB',max(HcicdB(sbidx))))
121 display(sprintf('CIC-Sparse worst case attenuation ...
       %gdB',max(Hyal(sbidx))))
122 display(sprintf('CIC bitgrowth %gbits',ceil(log2(sum(hcic)))));
123 display(sprintf('CIC-Sparse bitgrowth ...
       %gbits',ceil(log2(sum(abs(hyal))))));
```

```
%Return this function for calculation of the frequency response
%of an FIR filter for a given order N and type, along a provided
%normalized frequency axis w.
function [trig, M] = getTrig(N, w, type)
    M = getM(N, type);
    switch type
        case 1
            n=0:1:M;
            trig = 2*cos(w'*n);
            trig(:,1) = 1;
        case 2
            n=0:1:M;
            trig = 2*cos(w'*(n+0.5));
        case 3
            n=0:1:M;
            trig = 2*sin(w'*(n+1));
        case 4
            n=0:1:M;
            trig = 2*sin(w'*(n+0.5));
    end
end
```

```
%Return M, the number of unique coefficients for a
%given order N and type.
function [M] = getM(N, type)
    switch type
        case 1
            M=N/2;
        case 2
            M=(N-1)/2;
        case 3
            M=(N-2)/2;
        case 4
            M=(N-1)/2;
    end
end
```

```
% Get standard impulse response coefficients from the b[] vector
function b = htob(h, type)

m = length(h);
switch type
    case 1
        b = fliplr(h(1:(m+1)/2));
    case 2
        b = fliplr(h(1:m/2));
    case 3
        M=(N-2)/2; n=0:1:M;
        b(n+1) = 2*sin(w'*(n+1));
    case 4
        M=(N-1)/2; n=0:1:M;
        b(n+1) = 2*sin(w'*(n+0.5));
end

end
```

```
% Get standard impulse response coefficients from the b[] vector
function h = btoh(b, type)

b = b';

switch type
    case 1
        h=[fliplr(b), b(2:end)];
    case 2
        h=[fliplr(b), b(1:end)];

    case 3
        h=[fliplr(b), 0, -b];
    case 4
        h=[fliplr(b), b(1:end)];

end

end
```

```
1 %Return the impulse response of a cic filter for a given order and
2 %decimation ratio.
3 function [hcic] = makeCIC(order, cicR)
4     hcic = ones(1,cicR);
5     for i=1:(order-1)
6         hcic = conv(hcic, ones(1,cicR));
7     end
8 end
```

APPENDIX B – MATLAB CODE FOR SPARSE CIC FILTER MODEL

```
1  %cycle accurate sparse cic filter function
2  function dout = dosparse(y, tvc, sparseorder, order, cicW, cicR, ...
       numcycles)
3
4       int = zeros(1,order);
5       intN = zeros(1,order);
6       diff = zeros(1,order);
7       diffN = zeros(1,order);
8
9       intsparseN = zeros(1,sparseorder-1);
10      intsparse = zeros(1,sparseorder-1);
11      diffsparseN = zeros(1,sparseorder);
12      diffsparse = zeros(1,sparseorder);
13
14      k = 1;
15      for i=1:numcycles
16          coeff = tvc(1,mod(i-1,cicR)+1);
17          intN(1) = addsubq(int(1), y(i), cicW, 'add');
18          for j=2:order-1
19              intN(j) = addsubq(int(j), intN(j-1), cicW,'add');
20          end
21          intN(order) = addsubq(int(order), coeff*intN(order-1), ...
                cicW,'add');
22          for j=1:sparseorder-1
23              intsparseN(j) = addsubq(intsparse(j), ...
                    tvc(j+1,mod(i-1,cicR)+1)*intN(order-1), cicW,'add');
24          end
25
```

```
26          if (mod(i,cicR)==0)
27              for j=1:sparseorder-1
28                  diffsparseN(j) = ...
                        addsubq(intsparseN(j),diffsparse(j+1), cicW,'add');
29              end
30              diffN(1) = addsubq(intN(order),diffsparse(1), cicW,'add');
31              intN(order) = 0;
32              for j=1:sparseorder-1
33                  intsparseN(j) = 0;
34              end
35              for j=2:order
36                  diffN(j) = addsubq(diffN(j-1), diff(j-1), cicW, 'sub');
37              end
38              out(k) = diffN(order);
39              k = k+1;
40
41              for j=1:order
42                 diff(j) = diffN(j);
43              end
44              for j=1:sparseorder-1
45                  diffsparse(j) = diffsparseN(j);
46              end
47          end
48          for j=1:order
49             int(j) = intN(j);
50          end
51          for j=1:sparseorder-1
52             intsparse(j) = intsparseN(j);
53          end
54      end
55      dout = out / 2^(cicW-1);
56  end
```